UNIVERSIDAD DE OVIEDO

Departamento de Informática

Universidad de Oviedo

Tesis Doctoral

Programa de Doctorado de Ingeniería Informática

Mención Internacional

«MIDGAR: interoperabilidad de objetos en el marco de Internet de las Cosas mediante el uso de Ingeniería Dirigida por Modelos»

AUTOR: Cristian González García

DIRECTORES

Dr. Juan Manuel Cueva Lovelle

Dra. B. Cristina Pelayo García-Bustelo

Oviedo, 2017

«Fue el tiempo que pasaste con tu rosa lo que la hizo tan importante»
Antoine de Saint-Exupéry, El principito

«Escribe un artículo y da una charla sobre cualquier idea, no importa cuán insignificante sea para ti»
Symon Peyton Jones en «How to write a great research paper»

«Un hombre que no arriesga nada por sus ideas, o no valen nada sus ideas, o no vale nada el hombre»
Platón

«Vale más saber algo acerca de todo, que saberlo todo acerca de una sola cosa»
Blaise Pascal

«Si no conozco una cosa, la investigaré»
Louis Pasteur

«Despacito y buena letra, que el hacer las cosas bien, importa más que el hacerlas»
Antonio Machado

«En mi tarjeta de visita, soy un presidente de empresa.
En mi mente soy un programador de juegos.
Pero en mi corazón soy un jugador»
Satoru Iwata

Agradecimientos

«Dime y lo olvido, enséñame y lo recuerdo, involúcrame y lo aprendo»
Benjamín Franklin

«Ninguno de nosotros es tan inteligente como todos nosotros»
Ken Blanchard

He de agradecer este trabajo a todas aquellas maravillosas personas que me apoyaron durante estos tres años, casi cuatro, siempre, sin pedir nada a cambio, sino todo lo contrario, aportando todo lo posible. Sin ellas esto no hubiera sido posible.

A Cristina y Cueva, que han sido mis directores, consejeros y «jefes». Gracias por ofrecerme la oportunidad de trabajar con vosotros y, desde el primer día que empecé, haberme dado todo el apoyo que he necesitado, primero para el Máster y después para esta tesis, así como durante todo el tiempo que he trabajo para el grupo de investigación *MDE Research Group* durante estos años. Siempre habéis estado ahí y me habéis permitido todo lo que quise y deseé, apoyándome en todo momento. Gracias por estar ahí, ayudarme y apoyarme. Y, como dijo Benjamin Franklin, gracias por involucrarme.

Expresar mis más profundos y sinceros agradecimientos a Vicente García Díaz. Siempre has estado ahí para cualquier duda, tanto de mi tesis, como ideas de posibles artículos, solucionarme dudas, ayudarme con cualquier cosa y aconsejarme. Además, gracias a ti, pude realizar la estancia en Manchester, en la cual tuviste un papel muy importante. Muchísimas gracias, pues me allanaste mucho el camino y sin ti todo hubiera sido mucho más laborioso.

Agradecer también a mis compañeros de MDE y de proyectos, Dani Meana y Gonzalo, por poder contar con ellos para todo en cualquier momento, toda su ayuda prestada que no ha sido poca y haber pasado tan buenos ratos mientras avanzábamos juntos en muy buen ambiente.

A toda mi familia. Porque siempre estuvisteis ahí para todo lo que he necesitado.

A mis queridos caribeños: Andrea, Edward, Gio y Guille. Todo empezó con vosotros y me ayudasteis con todo desde el Máster. Tras esto, y especialmente a Edward y Guille, y ya continuando con la tesis, fuisteis un apoyo en todo momento, para lo bueno y lo malo, sin pedir nada a cambio. Gracias por todo vuestro apoyo condicional y por ofrecerme vuestro tiempo y vuestra amistad. Siempre que os necesité habéis estado allí. Muchas Gracias.

Mil gracias a todos mis compañeros de laboratorio, a los que están y a los que ya se movieron a otra etapa de su vida. Todos los que habéis pasado habéis sido grandísimos compañeros. Gracias por todas las partidas de tenis de mesa, los cafés, las cervezas, compartir la mesa, todos los buenos momentos y risas que nos echamos, y todas esas procrastinaciones. Si conseguís lo que buscáis, se os echará en falta, pues el laboratorio

ya nunca será igual sin vosotros. Gracias Borja Garrido, Claudio, Dani Fernández, Dani Meana, Edward, Francesco, Gabriel, Gonzalo, Guille, Herminio, Javi, Leticia, Manuel, Nadeem, Óscar, Ricardo y Yagüe.

A todos mis amigos, compañeros de fútbol, pádel y compañeros de la Ingeniería y del Máster. Sois muchos para nombrar, pero os veo a menudo, casi todas las semanas. Sabéis quienes sois. Siempre estuvisteis y estáis ahí. Lleváis muchos años a mi lado y espero que sigáis por muchos más. Aunque suene avaricioso, espero que me sigáis cediendo vuestro valioso tiempo, pues, se os quiere. Gracias a todos, y muy especialmente a los más cercanos como Alex y Lucía, Alex Montes, Antonio Mon, Bécares, César, Cornel y Cris, Enol, Eva, Gerardo, Gonzalo, Jandro, Óscar Prieto, Michu y Sonia, Luis, Toni, Sandra e Iris y a todos los miembros del Olimpic de Avilés.

A todos los componentes del grupo de investigación WESO en los últimos 4 años: Alex, César, Labra, Miguel, Nacho y Juan. Además, muy especialmente, a Dani y Herminio. Gracias por todos esos grandes momentos, charlas, descansos, risas, consejos y apoyo. Nada hubiera sido lo mismo sin vosotros. Gracias.

Gracias especiales a Miguel y a Quiroga, por tener siempre la puerta de vuestro despacho abierta y ofrecerme vuestra ayuda, darme consejos y vuestra experiencia en todo momento.

Muchas gracias a todos mis alumnos, y, sobre todo, a mis proyectantes, quienes confiaron en mí y aportaron su granito de arena, ya sea para mi vida o mi tesis doctoral. Especialmente a Adrián González, Dani Meana, Fernando Rodríguez, Gabriel, Gonzalo, Marco, Michu, Lara y Ricardo.

A todos los profesores que me disteis clase, tanto en la Ingeniería Técnica como en el Máster. De alguna manera u otra aprendí algo de todos vosotros. Gracias por vuestra dedicación y esfuerzo. Gracias especiales a aquellos que en los últimos años os preocupasteis y me aconsejasteis, gracias Alberto, Benja, Candi, Covadonga, Darío, Elías, Labra, Lanvín, Ortín y Redondo.

A todos los participantes de todas las pruebas. Gracias por hacerlas, pues sin vosotros, no tendría evaluaciones.

Special thanks to the University of Manchester for my stay. I have to thank my supervisor, the Dr Liping Zhao, for guiding and helping me at all moment, and my colleagues in that research group: Shupeng, Syafiq, and Waad. Here, I have to express more special thanks to Oscar Florez-Vargas and Simone. You both are great, thanks a lot for those Fridays and those beers! And thanks Manchester!

Many thanks to Bran Selic and Monique Snoeck, who answered my doubts about Model-Driven Engineering and helped me to find the right way in the history of this field.

Y para finalizar, gracias a todos, pues, como dijo Ken Blanchard, «Ninguno de nosotros es tan inteligente como todos nosotros», donde todos vosotros habéis aportado vuestro granito de arena en muchas cosas. Gracias.

Abstract

In the last years has appeared a revolution in the world of objects thanks to *Smart Objects*, sensors, and actuators, and the integration of these ones with the Internet of Things. This has given the opportunity of leveraging the whole collected information by the sensors and using this information in the Cloud (Cloud Computing), making there the required computation, and using the Cloud to interconnect the objects between themselves. The classic and indefatigable example about this case is the smart fridge, which will call our smartphone when we are in the supermarket to notify us about what we have to buy. Or even, if we move forward in the future, our own servant robot will do this type of labour, while it talks with its friends, who will be, no more no less, the remainder of the smart electrical appliances that will live in our house.

However, the Internet of Things is a field still barely exploited, very recent, and which needs a lot of investigation. It is necessary to connect the own objects to allow interacting automatically amongst themselves at the same time that these interconnections will be easily definable by any type of person. These interconnections can facilitate the daily life of different fields. It can improve our life at home, in the so-called Smart Homes; it can make better the cities livability creating in this way Smart Cities; it can protect the culture and heritage of towns creating Smart Towns; it can improve the security, the productivity, and the labour conditions in the industry through the Industrial Internet of Things, also named as Industry 4.0; finally, it can enhance our communication with The Earth to be able to understand it better through the Smart Earth. These improvements are obtained through the interconnections of different *Smart Objects*, sensors, and actuators.

To facilitate this task and obtain an abstraction which allows to any type of user create these interconnections, and at the same time that avoid the classic and so known problems of the Software Crisis, it has been decided to use Model-Driven Engineering, in order to be able to create Domain-Specific Languages. These Domain-Specific Languages will allow offering an abstraction that moves the creation of the necessary software of the objects that are in the Internet of Things closer to the users, as well as its maintenance.

To interconnect these objects there are different methods: The objects can be interconnected between themselves through a public cloud, using different private or own clouds, or as it occurs in some researches, using Online Social Networks as the point of interconnection amongst objects and between objects and people.

Notwithstanding, not everything in the Internet of Things is based on the objects, although the objects are closely connected. An important point in the Internet of Things is the security, exactly the cryptology, above everything after the last problems and leaks that have appeared in the last years. This remembers us how important is the privacy in our lives and how dangerous is the Internet, and therefore the Internet of Things, for our life. This is why the investigation in security is necessary to be able to create a secure Internet of Things.

Another point that appears is the level of intelligence of these objects, because they may not have intelligence or have Artificial Intelligence to do different types of actions, as can be to take decisions using Fuzzy Logic, learning with Machine Learning, distinguish objects applying Computer Vision, or understand

people language through Natural Language Processing. Each of these things can allow to the objects have more or less intelligence, or even allows creating or improving the robots, which field is known as Robotics.

The problem of all this, as it has been happening in recent years and it has been incrementing with the appearance of the Internet of Things, is the huge quantity of data necessary to be processed, being many times required in real time. This implies that it is essential the use of Big Data tools to be able to keep an adequate flow and constant of data processing to be able to offer a proper service in an Internet of Things platform and their connected objects.

All of the above is part of the problems and the possible solutions to the existing problems in the Internet of Things, being explained respectively in the 'Bloque III' and 'Bloque IV' of this doctoral dissertation. Thus, it has been contemplated a series of objectives that are itemised in the hypothesis of this doctoral dissertation with the purpose of facilitating their verification. This has led to create different prototypes that have allowed obtaining results and conclusions that verify the objectives of this doctoral dissertation and, as a whole, the hypothesis, as this is presented in the final conclusions.

Clearly, in this doctoral dissertation is impossible to resolve every problem of the Internet of Things world. Besides, from this doctoral dissertation appear other new ways to continue this research that are presented as future work to continue the research path.

Keywords

Internet of Things; IoT; Smart Home; Smart Town; Smart City; Smart Earth; *Smart Objects*; Sensors; Actuators; Sensor Networks; Model-Driven Engineering; MDE; Domain-Specific Language; DSL; Security; Cryptography; Online Social Networks; The Cloud; Cloud Computing; Artificial Intelligence; Machine Learning; Computer Vision; Fuzzy Logic; Natural Language Processing; NLP; Big Data; Robotics; Midgar; Doctoral thesis;

Resumen

En los últimos años ha surgido una revolución en el mundo de los objetos gracias a los Objetos inteligentes, sensores y actuadores y a la integración de estos en Internet de las Cosas. Esto ha dado la oportunidad de aprovechar toda la información recogida por los objetos y utilizarla en La Nube (Cloud Computing), realizar allí la computación requerida y utilizar La Nube para poder interconectar los objetos entre ellos. El clásico e incansable ejemplo de esto es la nevera inteligente que nos llamará a nuestro smartphone cuando nos encontremos en el supermercado para avisarnos de que tenemos que comprar, o, incluso, si avanzamos mucho en el futuro, el propio robot sirviente que nos haga todo este trabajo, mientras habla con sus amigos, que serán ni más ni menos que el resto de electrodomésticos inteligentes que habitarán nuestra casa.

Sin embargo, Internet de las Cosas es un campo aún apenas sin explotar, muy reciente, y que necesita mucha investigación. Se necesitan conectar los propios objetos permitiendo que interactúen automáticamente entre ellos al mismo tiempo que estas interconexiones sean fácilmente definibles por cualquier tipo de persona. Estas interconexiones pueden facilitar la vida diaria en diferentes ámbitos. Puede ser para mejorar nuestra vida en casa, en lo conocido como Smart Homes, puede ser para mejor la habitabilidad de las ciudades creando así las famosas Smart Cities, proteger la cultura y la herencia de los pueblos creando así Smart Towns, mejorar la seguridad, la productividad y las condiciones laborales en la industria por medio de Industrial Internet of Things, también conocida como Industria 4.0, y mejorar la comunicación con la Tierra para poder entenderla mejor por medio de Smart Earth. Todo esto se consigue mediante la interconexión de diferentes *Smart Objects*, sensores y actuadores.

Para facilitar estas tareas y conseguir una abstracción que permita a cualquier tipo de usuario crear estas interconexiones, así como para evitar los problemas clásicos y tan conocidos de la Crisis del Software, se ha decidido utilizar Ingeniería Dirigida por Modelos, con el fin de poder crear Lenguajes de Dominio Específico. Estos Lenguajes de Dominio Específico permitirán ofrecer una abstracción que acerque la creación del software necesario por los objetos de Internet de las Cosas a los usuarios, así como su mantenimiento.

Para interconectar estos objetos hay diferentes métodos: se pueden interconectar los objetos entre sí mismos mediante una nube pública, usando de diferentes plataformas propietarias o propias, o bien, como ocurre en algunas investigaciones, utilizando las Redes Sociales Online como punto de interconexión entre objetos y entre objetos y personas.

No obstante, no todo en Internet de las Cosas está basado en los objetos, aunque si estén estrechamente relacionados. Un punto muy importante en IoT es la Seguridad: criptología, sobre todo después de todos los problemas y filtraciones que han surgido en los últimos años. Estas nos han recordado cuán importante es la privacidad en nuestras vidas y lo peligroso que es Internet, y por consiguiente Internet de las Cosas, para ella. Por este motivo, investigar en seguridad se hace muy necesario para poder crear un Internet de las Cosas seguro.

Otro punto que surge es el nivel de inteligencia de estos objetos, pues pueden no disponer de inteligencia o bien tener una cierta Inteligencia Artificial para poder realizar diferente tipos de acciones, como puede ser

la toma de decisiones mediante Lógica Difusa, aprender mediante Machine Learning, diferenciar objetos aplicandp Visión por Computador, o entender nuestro idioma mediante Procesamiento de Lenguaje Natural. Todo esto permite que los objetos sean más o menos inteligentes, e incluso permite el crear o mejorar los robots, siendo esto conocido como Robótica.

El problema de todo esto, como lleva ocurriendo también en los últimos años, y que se ha incrementado con la aparición de Internet de las Cosas es la enorme cantidad de datos que se deben de tratar, siendo muchas veces necesario que sea en un tiempo cercano al real. Esto implica que sea necesario el uso de herramientas Big Data para poder mantener un flujo adecuado y constante de procesamiento de los datos para poder ofrecer un servicio adecuado en una plataforma de Internet de las Cosas y a los objetos que están interconectados a esta.

Todo lo anterior forma parte tanto de los problemas como de las posibles soluciones a los problemas existentes en Internet de las Cosas, siendo explicados respectivamente en el Bloque III y el Bloque IV de esta tesis doctoral. Por ello, se han planteado una serie de Objetivos de forma que desglosan la Hipótesis de esta tesis doctoral con el fin de facilitar su verificación. Esto ha dado lugar a diferentes prototipos que han permitido obtener unos resultados y conclusiones que verificasen los objetivos de esta tesis doctoral y, en su conjunto, la hipótesis como un todo en las Conclusiones finales.

Claramente, en esta tesis doctoral se hace imposible solucionar todos los problemas del mundo de Internet de las Cosas, sino que además de aquí surgen otros nuevos caminos por los que continuar la investigación y que se presentan como un Trabajo futuro de cara a continuar el camino de la investigación.

Palabras Clave

Internet de las Cosas; *Internet of Things*; IoT; *Smart Home*; *Smart Town*; *Smart City*; *Smart Earth*; *Smart Objects*; Objeto Inteligente; Sensores; Actuadores; Redes de sensores; Ingeniería Dirigida por Modelos; MDE; Lenguaje de Dominio Específico; DSL; Seguridad; Criptografía; Redes Sociales; La nube; *Cloud Computing*; Inteligencia Artificial; *Machine Learning*; Visión por Computador; Lógica Difusa; Procesamiento de Lenguaje Natural; NLP; *Big Data*; Robótica; Midgar; Tesis;

Tabla de contenido general resumido

Tabla de contenido general

Tabla de ilustraciones

Índice de tablas

Tabla de archivos de código fuente

«Un aspecto de hacer ciencia consiste en elegir la línea divisoria entre lo relevante y lo irrelevante»
Edsger. W. Dijkstra y Carel S. Scholten [1]

«Si supiéramos lo que estamos haciendo, no sería investigación»
Albert Einstein

«Lo último que uno sabe es por dónde empezar»
Blaise Pascal

Bloque I Planteamiento del problema

Contenidos de bloque

1. INTRODUCCIÓN

«El que sube una escalera debe empezar por el primer peldaño»
Sir Walter Scott

«Caminante no hay camino, se hace camino al andar»
Antonio Machado en «Campos de Castilla»

«No es natural que un solo hombre pueda hacer un descubrimiento repentino, la ciencia va paso a paso y cada hombre depende de la obra de sus predecesores. Cuando usted oye hablar de un descubrimiento repentino e inesperado - un rayo caído del cielo - puede estar seguro siempre que ha crecido por la influencia de un hombre o de otro, y es esa influencia mutua lo que produce la enorme posibilidad del avance científico. Los científicos no dependen de las ideas de un solo hombre, sino de la sabiduría combinada de miles de hombres, todos pensando sobre el mismo problema y cada uno de ellos haciendo su pequeña aportación a añadir a la gran estructura de conocimiento que se va construyendo poco a poco»
Ernest Rutherford

El primer paso de toda investigación es asentar bien las bases. Por eso, la parte principal es tener una motivación que permita saber bien el por qué se ha elegido ese camino en la tesis y saber si tiene sentido realizar esta tesis. La motivación contiene la explicación de las razones que motivaron a su realización.

A partir de la motivación, surgieron diferentes preguntas de investigación que fueron planteadas y que gracias a ellas se formó la hipótesis de esta tesis. De acuerdo a conseguir lograr la demostración de la hipótesis, se especificaron una serie de objetivos que se debían cumplir y que están explicados en un subcapítulo. El último subcapítulo está dedicado a las actividades principales en las que desembocaron dichos objetivos y contiene un resumen de cada actividad, pues están explicadas más exhaustamente en sus propios capítulos.

Introducción

1.1 *MOTIVACIÓN*

La gran mayoría de la interacción realizada en Internet es del tipo humano-humano (H2H). No obstante, cada vez más objetos heterogéneos y ubicuos pueden conectarse a Internet. En un futuro se espera que haya más objetos conectados que personas [2]. Estos pueden interactuar entre ellos, enviarse datos y realizar determinadas acciones según cierta información. Esto es **Internet de las Cosas**: **objetos heterogéneos y ubicuos interconectados entre ellos y comunicándose entre sí a través de Internet** [3].

Internet de las Cosas (IoT) surgió en base a la necesidad de las cadenas de suministro y la identificación de objetos, personas y animales mediante el uso de etiquetas inteligentes *Radio Frequency IDentification* (RFID) [3], [4]. Con ellas se consiguió otorgar de un identificador único al objeto deseado. No obstante, para la existencia de Internet de las Cosas, son necesarias tres cosas: inteligencia integrada en los objetos, la conectividad de los objetos a Internet y la interacción entre los propios objetos. Para lograr este fin se necesitan componentes atómicos que unan el mundo real con el mundo digital. Entre estos destacan los sistemas informáticos embebidos en objetos, los sistemas avanzados de etiquetado como son *Near Field Communication* (NFC) y RFID, y las redes de sensores (WSN) y las redes de sensores y actuadores (WSAN) [5]–[7]. Si combinamos varios de estos componentes, podemos hacer que un objeto concreto realice una acción en base a un evento captado por otro, como puede ser un sensor u otro objeto. Para lograr esto, se necesita de una infraestructura que sea capaz de conectarlos y dote a los objetos de la inteligencia necesaria para comunicarse entre ellos, incluso, en algunos casos, indicándoles la decisión que deben tomar. Esto se debe a que muchos objetos no tienen la capacidad suficiente y necesaria para poder tomar decisiones por sí mismos [8].

Internet de las Cosas puede producir un gran impacto en el cambio de la vida diaria, en el ámbito doméstico y comercial. Algunas de las actuales investigaciones en este ámbito son el de neveras que analizan los hábitos alimenticios y pueden ser útiles para gente con enfermedades, diversos ejemplos de sistemas inteligentes para dotar de inteligencia a casas, o sistemas para obtener ahorro energético en un campus universitario y en casas [9], [10]. Por otro lado, otras investigaciones se centran en el control y prevención de desastres, como es el caso del *Deepwater Horizon* en el año 2010 [11].

IoT está ganando tanta importancia que el *United States National Intelligence Council* lo considera como una de las seis tecnologías con mayor impacto en los intereses de los Estados Unidos hasta 2025 [12]. Por otro lado, el gobierno del Reino Unido la engloba dentro de su informe del año 2014 acerca de las ocho grandes tecnologías que han identificado que podrán propulsar el Reino Unido [13], además de clasificarla como la segunda revolución de la segunda revolución digital en [14]. Otro organismo importante es la Organización de las Naciones Unidas (ONU), que en la reunión de Túnez del año 2005 dio notabilidad a Internet de las Cosas [2], prediciendo una nueva era de ubicuidad en donde el tráfico de datos generado por seres humanos en Internet será pequeño en comparación con el de los objetos cotidianos. Sin embargo, estos no son los únicos organismos que así lo piensan. Muchas otras organizaciones, así como países, están interesados en un estándar para IoT debido al gran impacto que este puede tener en la economía. No obstante, no se centran en IoT, sino en diferentes aspectos que contiene IoT. Algunos de estos son la Comisión Europea

[15], el Comité Europeo de Normalización (CEN) en RFID [16], la Organización Internacional de Normalización (ISO) que se centra las frecuencias utilizadas [17] y el Instituto Europeo de Normas de Telecomunicaciones (ETSI) en IoT [18]. En el caso de países [19], el gobierno del Reino Unido aportó 5 millones de libras, el de China 800 millones de dólares y el de Japón estuvo promoviendo iniciativas como u-Japan [20] e i-Japan.

Para realizar estas interconexiones entre objetos, así como los propios objetos inteligentes, hay que desarrollar el software necesario. En esta parte, el problema reside en que existen graves problemas durante el desarrollo del software. Estos llevan dándose desde los años 60 [21]. Se dieron a conocer por la Organización del Tratado del Atlántico Norte (OTAN) en 1968, en donde los acuñó bajo el término Crisis del Software [22]. Algunos de estos problemas se pueden minimizar mediante la automatización de tareas y procesos. De esta manera, se puede lograr reducir la complejidad, lo que repercute en conseguir un software más fiable con funcionalidades más sofisticadas [23], [24]. Para esto mismo surgió la Ingeniería Dirigida por Modelos y por este motivo, se planteó la introducción de Ingeniería Dirigida por Modelos en el marco de esta tesis.

1.2 *HIPÓTESIS*

El desarrollo de software es una tarea ardua que requiere un aprendizaje previo de los diferentes lenguajes de programación que se necesitarán para programar en una determinada plataforma o para una plataforma específica. Además, en algunos casos, se necesita aprender a utilizar diferentes librerías de terceros o *frameworks* [1], e incluso, a veces, el sistema operativo influye en el propio software a desarrollar.

En Internet de las Cosas uno de los problemas que surge es debido a que la idea de IoT es conectar todos los objetos entre ellos, lo que implica la existencia de muchos objetos heterogéneos y ubicuos interactuando entre ellos [25]. Estos objetos van desde objetos sin ningún tipo de interfaz ni capacidad para procesar como son los sensores y los actuadores, hasta objetos inteligentes, conocidos como ***Smart Objects*** [26], que permiten tratar todos los tipos de datos y poseen diferentes protocolos de envío de datos, sistema operativo propio, librerías y *frameworks*. Esto requiere de diferentes implementaciones para cada tipo dispositivo, pues entre estos puede diferir el sistema operativo (Android, GNU/Linux, iOS, Windows) y el o los Lenguajes de programación a utilizar (C#, Java, Objective-C o Swift), así como el tipo de acceso a la *Application Programming Interface* (API) nativa o los protocolos y tecnologías disponibles en los dispositivos para conectarse con los otros objetos, como puede ser por WI-FI, usando el *Universal Serial Bus* (USB) o el Bluetooth. Esto implica que la comunicación directa entre objetos no siempre puede ser posible debido al hecho de la falta de entendimiento y a la ausencia de interfaces o protocolos estándar [27]. Además, en el caso de los sensores, hay que hacer una implementación para que el sistema al que estén conectados pueda recoger los datos de los sensores y se los comunique a Internet. En el caso de los actuadores, hace falta crear una interfaz que permita comunicarlos con Internet para así recibir órdenes. Esto conlleva a que muchas veces,

[1] Glosario: ***Framework***

cuando se utilizan microcontroladores o microordenadores para trabajar con los sensores, las posibilidades hardware sean casi infinitas, debido a que pueden llevar muchos y muy diversas combinaciones de sensores y actuadores enchufados a ellos.

Una posible solución podría ser el uso de un servidor y un lenguaje de comunicación común, como es el caso de algunas aplicaciones multiplataforma como son los servicios de mensajería móvil. No obstante, esta solución ofrece otros problemas como es que el propio usuario tenga que desarrollar su propia aplicación siguiendo un estándar y utilizando una API, lo que implica que el usuario deba tener conocimiento en el desarrollo de software y tiempo para realizarlo, en el caso de que fuesen servidores públicos de cara al usuario. En el caso de que fuesen servidores privados, el usuario debería de descargarse el servidor, instalarlo y después desarrollar las aplicaciones.

Otra posible solución sería utilizar soluciones IoT privativas, como es el caso de LG, Samsung y Telefónica. Esta solución obliga al usuario a utilizar hardware privativo de una determinada empresa y que no es interoperable con hardware de otras empresas, y que en muchos casos, no permite el uso de otros componentes de terceros debido a la falta de entendimiento entre los mensajes de los ***Smart Objects*** de las diferentes empresas [25], o al uso de un protocolo de mensajes privados. Esto hace que se pierda heterogeneidad en IoT al no poder conectar cualquier tipo de objeto que el usuario desea y depender únicamente de una marca en todos los aspectos: objetos y servidor. En otras ocasiones, como sugirieron los científicos de Ericsson, se podrían utilizar las Redes Sociales Online para facilitar la familiarización con los objetos de una mejor manera por parte de los usuarios de dichos objetos [2].

Así, en ambos casos, se limita a los usuarios a contratar a alguien con los conocimientos necesarios, a aprender por sí mismos, o bien, a comprar hardware de una única empresa, sin poder conectar dispositivos u objetos de otras marcas, o a cambiar todo el sistema si desean utilizar otra marca diferente.

Una posible solución para estos problemas sería utilizar una red que permitiese conectar diferentes dispositivos [10]. Esta debería de poder procesar todos los datos recibidos, utilizando uno o varios formatos, y dar la inteligencia necesaria y entendimiento a cada dispositivo a través de la red. En este caso, el problema principal es la heterogeneidad, el dinamismo y la evolución de estos contenidos [25]. De estos tres principales problemas, esta tesis se enfoca en el primero de ellos, en la heterogeneidad de los objetos en una red IoT. Así, la primera pregunta de dónde surgió esta tesis fue:

¿Se podría crear un sistema que permitiese la interconexión de objetos heterogéneos y ubicuos?

No obstante, como se comentó previamente, la heterogeneidad es un problema a la hora de crear aplicaciones, sobre todo si se quiere que estas sean creadas por los propios usuarios, de los cuáles puede haber muchos sin conocimientos de programación y/o sin conocimientos sobre la plataforma sobre la que han de desarrollar. Otro problema relacionado es que estos usuarios deben de tener conocimientos acerca del dominio que desean resolver, en este caso, la interconexión de objetos heterogéneos en IoT. Así, uno de los problemas principales es hacer posible, para gente sin conocimientos de programación, el poder crear este tipo de aplicaciones de una manera rápida y sencilla [27].

Introducción

Para solucionar este tipo de problemas surgieron hace unos años la Arquitectura Dirigida por Modelos (MDA) [24], [28] y la Ingeniería Dirigida por Modelos (MDE) [29], de tal manera que surgieron los **Lenguajes de Dominio Específico**. Los DSLs ayudaron a simplificar la creación de lenguajes de alto nivel que se utilizaron para mejorar y gestionar los problemas de desarrollo de software de una manera fácil, gracias al poder de expresión que estos lenguajes otorgan cuando se trata de resolver un problema de un determinado dominio específico [30]. Por el contrario, lo que ofrecen los Lenguajes de Propósito General (GPLs) es el poder crear «cualquier tipo de aplicación». Gracias a esta diferencia, se consigue una mejora en la comprensión de los modelos para personas sin experiencia al acotar las posibilidades que estos ofrecen, limita los errores que se pueden cometer y permiten una mayor facilidad para migrar entre diferentes tecnologías.

De acuerdo a esto, una posible solución al problema anterior podría ser aplicar MDE para así obtener un DSL que permita obtener el nivel de abstracción necesario para encapsular el dominio del problema [30]. Este DSL podría facilitar la creación de este tipo de aplicaciones permitiendo a la gente solo necesitar conocimientos acerca del dominio explicado. Claramente, deberían de aprender a utilizar el DSL creado que, no obstante, es mucho más sencillo de aprender que un GPL y solo da las opciones necesarias para resolver este problema. Otra característica que ofrecen los DSLs es que se conseguiría obtener un nivel de abstracción que permitiría hacer este lenguaje portable para que así este sirviese para cualquier plataforma. Esto conllevó a una segunda pregunta:

¿Se podría utilizar la Ingeniería Dirigida por Modelos para crear un lenguaje de dominio específico que permitiese crear e interconectar objetos heterogéneos y ubicuos?

Sin embargo, hay que tener otra consideración que afecta a todo este proceso de envío de mensajes, que es la seguridad, pues los mensajes deben de ser privados y la comunicación debe de mantener la privacidad de sus usuarios. Esto es algo esencial y básico, y más después de las últimas filtraciones que han surgido en la actualidad.

¿Se podría securizar la comunicación entre objetos para mantener la privacidad de sus usuarios?

Teniendo estas tres preguntas de investigación, surgió la hipótesis de esta tesis doctoral:

Aplicar la Ingeniería Dirigida por Modelos para lograr la creación de la interacción humano-máquina y máquina-máquina de una forma fácil y rápida mediante la automatización o semiautomatización de diferentes procesos para desarrollar las aplicaciones de una manera ágil y reduciendo errores y costes en el marco de Internet de las Cosas, mientras que se mantiene la privacidad del usuario.

1.3 *OBJETIVOS*

Las preguntas de investigación anteriores que conformaron la hipótesis de esta tesis doctoral dieron lugar a una serie de objetivos que debían de cumplirse para comprobar si la hipótesis planteada era factible. Así, el **objetivo principal** de esta tesis es:

> Desarrollar una investigación en el marco de Internet de las Cosas junto con la utilización de la Ingeniería Dirigida por Modelos para permitir a los usuarios sin conocimientos de desarrollo de software construir aplicaciones para IoT de forma fácil, ágil y segura.

Para la consecución del objetivo principal, este se subdividido en varios **objetivos parciales** más específicos:

1. **Diseño de objetos inteligentes**: un usuario sin conocimientos en el desarrollo de software podrá crear tantos objetos inteligentes como desee sin necesidad de tener conocimientos técnicos y conectarlos a la red y con otros objetos de una forma fácil y rápida.
2. **Seguridad y privacidad**: los datos enviados y recibidos de los objetos podrían ser interceptados, bien para conseguir información o bien para modificarlos y entorpecer una actividad o conseguir el control del objeto. Por lo tanto, este es un tema muy delicado. Por ello, hay que realizar un estudio acerca de la seguridad existentes cuando se envían mensajes entre objetos para intentar conseguir un sistema fiable, seguro y que sea adecuado para el correcto rendimiento de la red, a la vez que sea transparente o muy fácil de usar por cualquier tipo de usuario.
3. **Objetos asistentes**: con la aparición de diferentes módulos para los objetos y las posibilidades de combinación que surgieron entre diferentes objetos, se consiguieron crear objetos mucho más inteligentes. Este es el caso de los robots. En algunos casos, estos se usan para ayudar a personas con discapacidades y problemas psicomotrices. Así, surge la necesidad de permitir a estas personas manejarlos y configurarlos. Por esto, se investigarán diferentes formas de facilitar la configuración y el uso de robots con diferentes tipos de objetos para conectarlos a Internet y/o redes sociales para dotarlos y adaptarlos de todo lo necesario para que sean más accesibles y usables por personas cualquier tipo de persona.

2. METODOLOGÍA Y DESARROLLO DE LA INVESTIGACIÓN

«El éxito no se logra solo con cualidades especiales.
Es sobre todo un trabajo de constancia, de método y de organización»
J.P. Sergent

«Investigar es ver lo que todo el mundo ya ha visto y pensar lo que nadie ha pensado todavía»
Albert Szent-Györgyi

«[En historia natural,] un gran descubrimiento a menudo requiere un mapa
hacia una mina llena de tesoros ocultos entonces fácilmente recogidos con las herramientas convencionales,
no una nueva y brillante máquina de la era espacial para penetrar en los mundos que antes eran inaccesibles»
Stephen Jay Gould (1941 – 2002)

Este segundo capítulo presenta la metodología utilizada en el desarrollo de esta tesis doctoral. Para ello, se expone la metodología del trabajo, que incluye las líneas de investigación de la tesis, el ciclo de vida de trabajo seguido en la tesis y los diferentes métodos científicos utilizados. En este capítulo también se describe el desarrollo temporal de toda la investigación, la cual incluye los hitos y eventos más importantes del desarrollo de esta tesis en orden cronológico. Finalmente, se explica la organización del resto del documento, que se encuentra estructurado en bloques y capítulos, con el fin de mejorar la visión general de lo que el lector puede encontrarse en ella.

*****

2.1 *METODOLOGÍA DEL TRABAJO*

Esta tesis tuvo su punto de partida en los estudios del **Máster Universitario de Ingeniería Web** en la Escuela de Ingeniería Informática de Oviedo, perteneciente a la Universidad de Oviedo. En este máster cursé [2] la rama de investigación, que es la que da acceso al programa de doctorado y en la que se cursan asignaturas orientadas a la investigación, que son aquellas que ponen los primeros cimientos del conocimiento requerido para la realización de una tesis. Además, como trabajo final de máster, esta rama exige un trabajo de investigación con un artículo *Journal Citation Reports* (JCR) enviado. Así, en esta parte, senté las bases de lo que sería mi doctorado al realización mi Trabajo Final de Máster (TFM) titulado «*MIDGAR: Plataforma para la generación dinámica de aplicaciones distribuidas basadas en la integración de redes de sensores y dispositivos electrónicos IoT*» [31] y presentado en julio de 2013.

No obstante, antes de comenzar el segundo curso, en agosto de 2012, surgió algo que sentó las bases previas a la solución de los problemas del desarrollo de software. Esto fue mi incorporación en el grupo de **Ingeniería Dirigida por Modelos MDE-RG** de la Universidad de Oviedo. Desde entonces, estuve trabajando en diferentes proyectos de investigación, los cuáles contenían siempre como componente la Ingeniería Dirigida por Modelos y la investigación. Por ello, ya desde los comienzos, como algo que influyó en el trabajo final de máster y en la investigación, fueron los conocimientos adquiridos en estos proyectos y la intención de aplicar soluciones MDE cuando fueran necesarias.

Así, con los conocimientos adquiridos previos acerca de IoT y MDE con la realización del TFM y de MDE con los proyectos de investigación, comencé el doctorado como continuación del TFM, siempre paralelamente a los proyectos de investigación en los que trabajaba.

De esta forma, con una línea de investigación bien marcada y ya comenzada, conocida y con un trabajo futuro posible, tuve que investigar y realizar la investigación y estudio de qué problemas existían en Internet de las Cosas, como podría resolverlos y si sería posible abarcarlos. A partir de estos problemas, tuve que definir la hipótesis de esta tesis a partir de las preguntas de investigación que se me plantearon y definir una serie de objetivos que tuviese que cumplir para así demostrar la hipótesis. Además de las conclusiones y el trabajo futuro del TFM, surgieron varias ramas a investigar, las cuáles se fueron ampliando debido a diferentes ideas y proyectos. En la Ilustración 1 muestro las diferentes líneas que fueron surgiendo. En esta ilustración se pueden ver las ocho líneas de investigación finales existentes y las posibles investigaciones dentro de estas, en caso de haberlas. Subrayadas y en cursiva se encuentran las partes ya investigadas y publicadas. Solo subrayadas están las partes que se investigaron, se tienen resultados, e incluso, en algunos casos, ya han sido enviadas. No obstante, todas estas diferentes líneas han sido investigadas a lo largo de la tesis. Claramente, unas con más profundidad que otras, pero todas han tenido su presencia en el desarrollo de la tesis.

[2] Esta subsección se encuentra redactada en primera persona a modo de diario de la tesis doctoral.

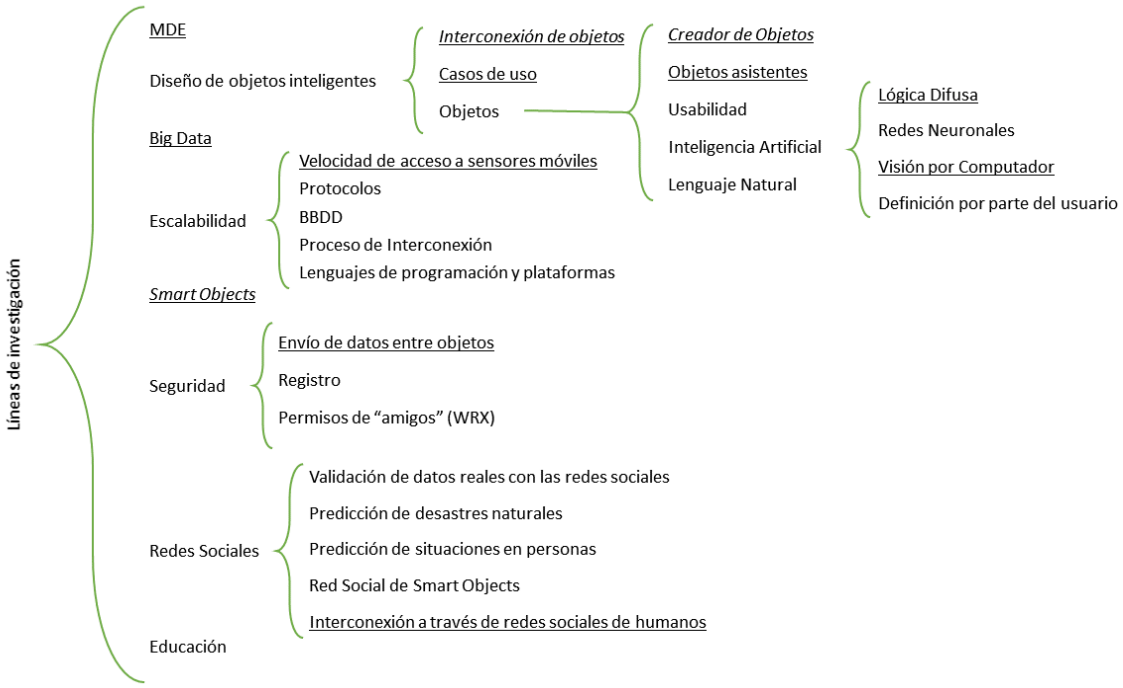


Ilustración 1 Líneas de investigación

Como complemento, a lo largo de los tres años de tesis, también fui profesor en el Máster Universitario de Ingeniería Web. Esto contribuyó con la especialización en ciertas asignaturas y ayudó a profundizar en ciertos aspectos relacionados con IoT y MDE, debido al hecho de participar en asignaturas como *Big Data*, *Modelado de Software Adaptable Dirigido por Modelos*, *Sistemas de Seguridad en la Web*, *Administración de Servidores Web* y *Modelos de negocio y comercio electrónico en la web*. Además de codirigir algunos proyectos de grado y máster que añadieron pequeñas contribuciones a mis investigaciones y a esta tesis.

Para desarrollar esta tesis, opté por una **metodología de desarrollo ágil** basada en el **método científico** y con una **ejecución en paralelo** de los diferentes subcampos que identifiqué dentro del campo principal, Internet de las Cosas. Cada iteración se planeó de acuerdo a que fuese un pequeño módulo completo que desembocase en una investigación finalizada, con su propio estudio del estado del arte, prototipo, evaluación y resultados publicables. Estos resultados fueron enviados a congresos y revistas, de entre las cuales la mayoría eran del índice JCR. La retroalimentación obtenida en estos envíos sirvió para mejorar los artículos enviados, así como los posteriores, la tesis, e incluso, en algunos casos, abrir nuevos objetivos, subobjetivos o ideas de cara a la creación de investigaciones paralelas dentro de la tesis.

En la Ilustración 2 muestro un gráfico acerca de la metodología de esta investigación, la cual fue explicada anteriormente, junto con todas las tareas adicionales realizadas durante esta tesis, siendo el punto de partida el inicio del Máster Universitario en Ingeniería Web.

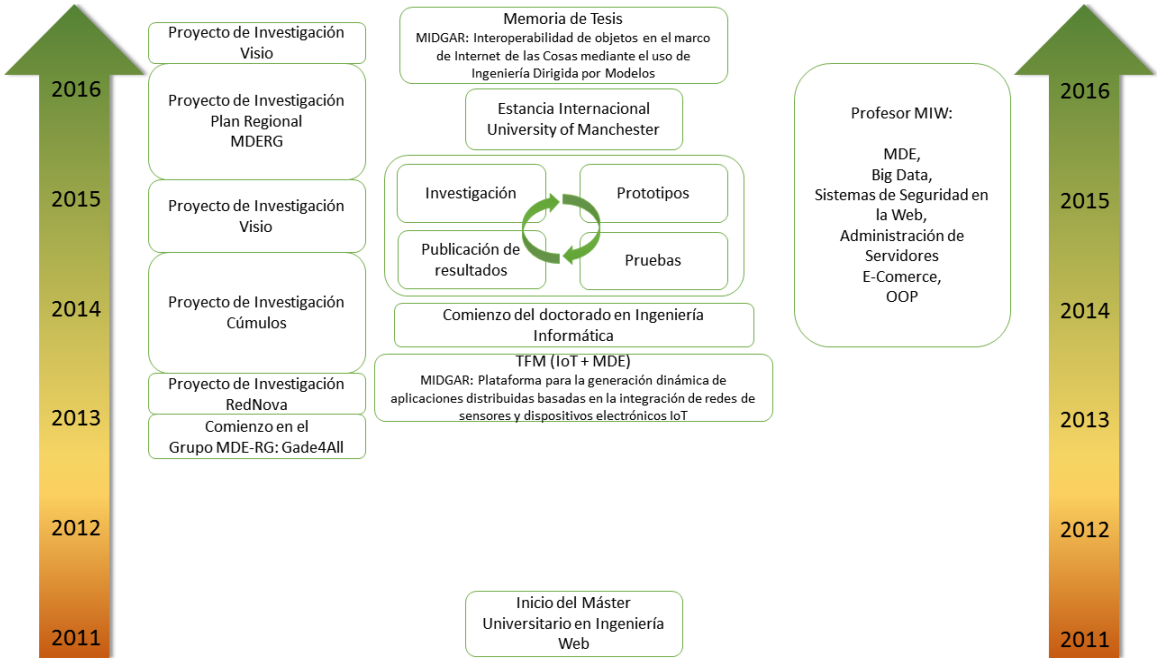


Ilustración 2 Metodología utilizada en la investigación

2.1.1 CICLO DE TRABAJO

Previamente, se dijo que se optó por una **metodología de desarrollo ágil** basada en el **método científico** y con una **ejecución en paralelo**. En esta sección se explica cómo fue todo el ciclo de trabajo para ayudar a los lectores a entender cómo fue la forma de trabajar en esta tesis doctoral.

2.1.1.1 METODOLOGÍA DE DESARROLLO ÁGIL

La elección de usar una **metodología de desarrollo ágil** viene dada por la realización y partición de las diferentes tareas, de forma que cada una de ellas fuese un módulo ejecutable y finalizado. Ejemplos de estas metodologías son **SCRUM** [32], [33] y **eXtreme Programming** (**XP**) [34], [35]. Claramente, esta metodología fue adaptada a la tesis, luego no se usan SCRUM y XP como tal, pero sí una metodología de desarrollo ágil basada en ambas. Las causas de esto se deben a que los diferentes proyectos que componen esta tesis han sido desarrollados por una única persona, el autor, o por el autor junto a otras personas en forma de colaboradores.

No obstante, los pros de usar una metodología ágil son muchos, como la forma ágil de evolucionar los diferentes proyectos y prototipos y adaptarse a los cambios que fueran surgiendo, así como la modularización en productos ya acabados. Como fue debido, esto se adaptó al estilo de investigación. Así, cada módulo se componía de submódulos, los cuáles se correspondían con los diferentes pasos del método científico a la vez que se acoplaba muy bien a este, debido a las diferentes etapas bien marcadas.

2.1.1.2 MÉTODO CIENTÍFICO

En esta tesis se ha utilizado el modelo clásico del **método científico** basado en el trabajo de Aristóteles, introductor de este método [36]. Al utilizar este método, toda la investigación de la tesis se basa en el **empirismo**, que es aquel que se adquiere con la observación y la experiencia del tiempo y de las situaciones vividas, y en la **medición** de los hechos, siendo a su vez **reproducibles** y **no refutados**. Sin embargo, existen diferentes formas de aplicar el método científico, el cual posee varias ramas, y que son aplicables según el tipo de investigación que se quiere llevar a cabo y/o según la ciencia que se investigue.

Sin embargo, no existe una clasificación de todas las formas de realizar el método científico ni tampoco una clasificación clara de ellas, pues no hay consenso dentro de la propia comunidad científica. Luego, a continuación, se expone una posible clasificación que contiene las principales ramas de los métodos científicos utilizados en esta tesis, junto a sus posibles vertientes, subrayando los utilizados:

- Método empírico-analítico
    - Método experimental
        - Método hipotético-deductivo
    - Método de la observación científica: se basa en la observación del fenómeno mediante la percepción y es propio de las ciencias que tratan de describir como es el mundo o las cosas.
    - Método de la medición
- Método lógico: una de las grandes ramas del método científico, más clásica y de menor fiabilidad.
    - Método lógico-deductivo: consiste en aplicar los principios descubiertos a casos particulares a partir de un enlace de juicios, destacando en su aplicación el método de extrapolación.
    - Método lógico-inductivo: es el razonamiento que a partir de casos particulares se eleva a conocimientos generales.
        - Método inductivo de inducción completa: la conclusión se saca tras estudiar todos los elementos que forman parte de la investigación.
        - Método inductivo de inducción incompleta: no se pueden numerar e investigar todos los elementos de la investigación y se recurre a muestras representativas que permitan generalizar las conclusiones.
            - Método de inducción incompleta por simple enumeración
            - Método de inducción científica: estudia los caracteres y/o conexiones necesarias del objeto a investigar con otros objetos.

2.1.1.2.1 MÉTODO EMPÍRICO-ANALÍTICO

Método empírico-analítico: este método se basa en la experimentación y en la lógica empírica. Es un método que se vale de la verificación empírica, pues pone a prueba la **hipótesis** a una cuidadosa contrastación en vez de someterla al sentido común o dogmatismo filosófico. Además, es **autocorrectivo y progresivo** al no considerar sus conclusiones como infalibles o finales. Esto permite que se le puedan incorporar nuevos conocimientos y procedimientos con el fin de asegurar una mayor veracidad. Estas pruebas se basan en las **muestras** (representativas o sesgadas), es decir, en un subconjunto de casos o individuos de una población estadística o universo de discurso. Esto puede representar que, si la muestra es incorrecta debido a que se escogió mal, los resultados podrían ser erróneos. Dentro de este método se encuentran el Método experimental, que a su vez tiene el Método hipotético-deductivo, el método de la observación científica y el Método de la medición.

Así, este método se basa en la observación a partir de la **experiencia** para establecer deducciones al analizar los datos obtenidos en la observación, pues según este método, **los hechos son observables, cuantificables y medibles**. Por ello, contrasta las hipótesis rigurosamente a través de los experimentos científicos que determinan si la hipótesis es verdadera o falsa.

El proceso de este método, de acuerdo a Adriaan de Groot [37], comienza con la **observación del problema**. En base al estudio de este, se procede a establecer una **hipótesis por inducción** sobre la que se trabajará y que ofrecerá una explicación del fenómeno. Tras esto, se formularán diferentes **experimentos deductivos** que tratarán de confirmar o refutar la hipótesis. El siguiente paso es la recogida de datos al **testear** la hipótesis por medio de los experimentos deductivos anteriormente formulados. El último paso es **evaluar los datos** recogidos para poder formular una teoría, refutar la hipótesis, o modificar la hipótesis y realizar de nuevos experimentos deductivos.

Claramente, este método solo es aplicable a aquellos objetos de estudio que puedan ser observables, cuantificables y medibles matemáticamente.

2.1.1.2.2 MÉTODO EXPERIMENTAL

Método experimental, perteneciente al método empírico-analítico: este método es el más complejo y eficaz de los métodos empíricos debido a que consiste en intervenir directamente sobre el objeto de estudio modificándolo directa o indirectamente para crear las condiciones necesarias que permitan revelar sus características fundamentales y sus relaciones esenciales. Hay algunos autores que debido a la relevancia que tiene este método lo consideran independiente del método empírico-analítico. Este método hace que se obtengan los datos a partir de la manipulación sistemática de las variables del experimento. Esta manipulación sistemática se puede hacer de tres formas:

- Aislando el objeto y las propiedades a estudiar de la influencia de otros factores.
- Reproduciendo el objeto de estudio en condiciones controladas.
- Modificando las condiciones bajo las cuales tiene lugar el proceso.

2.1.1.2.3 MÉTODO HIPOTÉTICO-DEDUCTIVO

Método hipotético-deductivo, perteneciente al método experimental del método empírico-analítico: este método es muy similar al método experimental, pues es un submétodo del método experimental, con la diferencia de que el método hipotético-deductivo se basa en los métodos deductivos e inductivos. Las fases son [38]:

- **Observación**: en esta fase se observa el hecho sobre el que se desea encontrar una explicación o resolverlo.
- **Hipótesis**: se define la hipótesis a partir de los elementos observados previamente y que permite explicar o resolver dicho fenómeno.
- **Deducción**: con la hipótesis ya elaborada, se sacan de ella consecuencias empíricas y teóricas tratando de predecir que ocurre en caso de que esa hipótesis sea verdad.
- **Test** o **experimento**: en esta fase se realizan los experimentos empíricos mediante inducción, de los que se obtendrán los datos para así contrastarlos con las deducciones hechas a partir de la hipótesis y saber si las deducciones fueron correctas, lo que conduciría a una hipótesis cierta. En caso de no cumplirse las deducciones, la hipótesis no se cumpliría y quedaría refutada.

Esto hace que, a partir de una idea inicial, conocida como hipótesis, sobre la que se aplica deducción para tratar de inferir las consecuencias de la hipótesis y poder realizar un experimento para obtener los datos necesarios para crear un proceso de inducción y saber si la hipótesis no ha sido refutada, o, en caso contrario, modificarla de nuevo iniciando un nuevo ciclo. Luego, este método combina la formación de la hipótesis y la deducción con la observación y la verificación, es decir, deducción e inducción.

Este método científico fue aplicado en los prototipos presentados en los capítulos 22 y 23, en donde se dedujo a partir de la hipótesis que podría suceder y después se comprobó inductivamente. Además, este método se uso en estos prototipos en conjunto a otro método científico.

2.1.1.2.4 MÉTODO DE LA MEDICIÓN

Método de la medición, perteneciente al método empírico-analítico: este método se aplica en aquellas investigaciones en las que se necesitaba obtener información numérica acerca de las cualidades del objeto o proceso para así poder comparar cualidades medibles. De esta manera, se asignan valores numéricos a determinadas propiedades y relaciones para poder evaluarlas y representarlas, permitiendo apoyarse en procedimientos estadísticos.

Es por ello, que esté método científico fue aplicado en diferentes prototipos con el fin de analizar los datos obtenidos de manera estadística consiguiendo así datos medibles y evaluables que se pudierna comparar. Este método fué utilizado en los siguientes capítulos: 20, 21, 22, 23 y 24. Cabe aclarar, que este método se uso en estos prototipos en conjunto a otro método científico.

2.1.1.2.5 MÉTODO DE INDUCCIÓN INCOMPLETA POR SIMPLE ENUMERACIÓN

Método inductivo de inducción incompleta por simple enumeración o conclusión probable, perteneciente el método lógico-inductivo de la rama del método lógico: este método se aplica en aquellas investigaciones en las que los objetos de investigación no pueden ser estudiados en su totalidad dada la imposibilidad de estudiar todo el conjunto, lo que obligaba a recurrir a una muestra representativa (por esto es incompleto y no completo) de acuerdo a hacer la generalización. Por ello, las conclusiones y los resultados deben tomarse como posibilidades de veracidad y no como demostraciones de algo, pues si se da un caso negando la conclusión, esta puede ser refutada.

- Las 4 etapas básicas del método **lógico-inductivo** son:
    1. **Observación** y **registro** de todos los hechos
    2. **Análisis** y **clasificación** de los hechos registrados.
    3. **Razonamiento inductivo y lógico** de los datos para así obtener una **hipótesis**, basada en un análisis lógico de los datos procesados, que solucione el problema.
    4. **Contrastación** de la hipótesis para comprobar si esta es verdadera, en el caso de que deba de serlo.
        - En caso de que no se pueda deducir una implicación contrastadora de la hipótesis, en el caso de que deba ser verdadera, la hipótesis no será valida.

Este método científico fue aplicado en los prototipos que uso de encuestas con el fin de obtener valores cualitativos a partir de la opinión de los usuarios tras contestar las encuestas. Así, los prototipos en los que se usó este método son los correpsondientes a los siguientes capítulos: 20, 21 y 24. En estos prototipos se usó este método en conjunto a otro método científico.

2.1.1.3 EJECUCIÓN EN PARALELO

La **ejecución en paralelo** de esta tesis vino dada por dos casos fundamentales. Como se ha explicado previamente, y se puede observar en la Ilustración 1, a lo largo de esta tesis doctoral han surgido muchas líneas de investigación dentro de Internet de las Cosas. En algunas de estas investigaciones se colaboró con otra gente, ya fueran compañeros de trabajo, proyectantes o gente de otras universidades. Esto implicó que se realizasen investigaciones en paralelo, desarrollando y colaborando en varias al mismo tiempo con diferentes personas.

2.2 *DESARROLLO TEMPORAL DE LA INVESTIGACIÓN*

A continuación, presento [3] los hitos y eventos más relevantes surgidos en el desarrollo de esta tesis:

- **Septiembre 2011**: comienzo de los estudios en el **Máster Universitario en Ingeniería Web** en la Universidad de Oviedo. Esto inició todos mis intereses en Internet y me dio todos los conocimientos necesarios para poder trabajar con IoT.
- **Agosto 2012**: comienzo en el grupo de investigación **MDE-RG** de la Universidad de Oviedo. En esta primera etapa comienzo a trabajar en el proyecto de investigación **Gade4All**. Este proyecto trata sobre la creación de una plataforma genérica para facilitar el desarrollo de videojuegos y software de entretenimiento multiplataforma. Para conseguirlo, se aplicó una solución basada **MDE**. Así, en este proyecto adquirí mis primeros conocimientos sobre MDE, sobre cómo hacer artículos de investigación y el desarrollo multiplataforma con Android, iOS y JavaScript.
- **Marzo 2013**: cambio al proyecto de investigación **Cúmulos** en colaboración con ZED WORLDWIDE S.A. Este proyecto consistió en la realización de un navegador web para dispositivos Android. Este navegador incorporaba una API JavaScript para poder desarrollar aplicaciones en JavaScript que llamasen a casi cualquier función del sistema operativo Android. En este proyecto profundice mucho en la API de Android y en JavaScript, los cuáles fueron pilares fundamentales de mis prototipos, así como en el desarrollo móvil y sus optimizaciones para usar los sensores de forma óptima.
- **Julio 2013**: lectura del **Trabajo Final de Máster** titulado «*MIDGAR: Plataforma para la generación dinámica de aplicaciones distribuidas basadas en la integración de redes de sensores y dispositivos electrónicos IoT*» [31], cuya nota fue de 10, Matrícula de Honor. Este trabajo presenta las bases de una plataforma IoT que permite interconectar los objetos y que sirve de ayuda para seguir realizando investigaciones en el campo de **IoT**.
- **Agosto 2013**: publicación del **capítulo** 4 titulado «*Using Model-Driven Architecture Principles to Generate Applications based on Interconnecting Smart Objects and Sensors*» en el libro *Advances and Applications in Model-Driven Engineering* de la editorial *IGI Global* [39]. Este capítulo trata sobre una explicación de qué es Internet de las Cosas y el posible uso del *Graphical Modeling Framework* (GMF) para crear un editor gráfico para interconectar los objetos.
- **Septiembre 2013**: comienzo del doctorado en Ingeniería Informática, en la línea de investigación de Ingeniería Web, redes sociales y dispositivos móviles. En este, continué con las bases que senté en mi TFM, siendo por ello mi línea principal IoT usando MDE. El título de la tesis es «*MIDGAR: interoperabilidad de objetos en el marco de Internet de las Cosas mediante el uso de Ingeniería Dirigida por Modelos*».
- **Septiembre 2013**: primer curso como **Profesor Invitado** en el Máster Universitario de Ingeniería Web impartiendo asignaturas referentes a los temas de mi tesis y que me ayudan a

[3] Esta subsección se encuentra redactada en primera persona a modo de diario de la tesis doctoral.

profundizar y aprender cosas útiles de cara a la investigación. Algunos ejemplos son las asignaturas de *Modelado de Software Adaptable Dirigido por Modelos*, *Sistemas de Seguridad en la Web* y *Modelos de negocio y comercio electrónico en la web*.

- **Febrero 2014**: publicación del **artículo** «*Midgar: Generation of heterogeneous objects interconnecting applications. A Domain Specific Language proposal for Internet of Things scenarios*» en la revista *Computer Networks* perteneciente al índice JCR [40]. En este artículo se presenta la plataforma IoT Midgar. A esta se la acompañó de un DSL gráfico para crear la interconexión entre objetos en una red IoT de una forma fácil y rápida para usuarios sin conocimientos de programación.
- **Marzo 2014**: publicación del **artículo** «*Vitruvius: An expert system for vehicle sensor tracking and managing application generation*» en la revista *Journal of Network and Computer Applications* perteneciente al índice JCR [41]. En este artículo se presenta una plataforma para que usuarios sin conocimientos de programación puedan diseñar rápidamente aplicaciones web basadas en el consumo de datos que generan los vehículos y sus sensores en tiempo real.
- **Julio 2014**: nuevo proyecto de investigación en colaboración con ZED WORLDWIDE S.A. llamado **VISIO**. El objetivo fue el desarrollo de una plataforma que permitiese que las aplicaciones en la nube generasen automáticamente la interfaz de usuario en función de las capacidades de interacción del dispositivo y se adaptasen al contexto en que se encontrasen para hacerlas realmente accesibles, ubicuas e independientes de los dispositivos. En este proyecto, profundicé más mis conocimientos **multiplataforma**, en **iOS** con **Swift**, **MDE** al crear un **DSL**, y la publicación de artículos científicos.
- **Julio 2014**: publicación del artículo «*Midgar: Domain-Specific Language to generate Smart Objects for an Internet of Things platform*» en el **congreso** *Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing* [42]. En este artículo se presenta la idea de un DSL gráfico para crear objetos inteligentes por usuarios sin conocimientos de programación.
- **Agosto 2014**: publicación del **capítulo** 15 titulado «*MUSPEL: Generation of Applications to Interconnect Heterogeneous Objects Using Model-Driven Engineering*» en el libro *Handbook of Research on Innovations in Systems and Software Engineering* de la editorial *IGI Global* [43]. Este capítulo es una actualización del capítulo anterior, en el que se renueva la literatura, se mejora el editor realizado con GMF y se actualiza a su nueva versión y, además, se incorpora una versión textual del editor creada con la herramienta Xtext.
- **Septiembre 2014**: segundo curso como **Profesor Invitado** en el Máster Universitario de Ingeniería Web impartiendo: *Big Data*, *Modelado de Software Adaptable Dirigido por Modelos*, *Sistemas de Seguridad en la Web*, *Administración de Servidores Web*, *Modelos de negocio y comercio electrónico en la web*, y P*rogramación Orientada a Objetos*.
- **Abril 2015**: cambio de proyecto para pasar a la parte de investigación de la Universidad de Oviedo de la mano del grupo MDE-RG para llevar a cabo el **Plan Regional de Investigación**,

llamado igual que el grupo de investigación, **Ingeniería Dirigida por Modelos MDE-RG**, y que trata sobre la investigación y publicación en **IoT** y **MDE**.

- **Mayo 2015**: publicación del **artículo** «*Swift vs. Objective-C: A New Programming Language*» en la revista *International Journal of Artificial Intelligence and Interactive Multimedia* [44]. Este artículo hace una segunda revisión del lenguaje de programación Swift, exactamente en su versión 1.2, su tercera versión y que conllevó bastantes cambios, frente a Objective-C.
- **Julio – noviembre 2015**: estancia internacional de 3 meses y 2 semanas en *The University of Manchester* en el grupo de investigación *Software Systems*. Este grupo trabaja con Desarrollo Dirigido por Modelos (MDD), Arquitectura Dirigida por Modelos (MDA) y X-as-a-Service (XaaS). Colaboré en una combinación de mi tesis con sus investigaciones, es decir, en IoT y MDD bajo la supervisión de la jefa del grupo, la Doctora Liping Zhao.
- **Noviembre 2015**: publicación del artículo «*El futuro de Apple: Swift versus Objective-C*» en la **revista** *Redes de Ingeniería* [45]. Este artículo hace una primera revisión del nuevo lenguaje de Apple, Swift 1.0 GM, contra Objective-C.
- **Noviembre 2015 – enero 2016**: tercer curso como **Profesor Laboral de Sustitución** en la Universidad de Oviedo impartiendo clases en el Máster Universitario en Ingeniería Web de *Modelado de Software Adaptable Dirigido por Modelos*.
- **Enero 2016**: vuelta al proyecto **Visio**.
- **Enero 2016**: comienzo de escritura de la memoria de esta tesis.
- **Junio 2016**: publicación del artículo «*IntelliSenses: Sintiendo Internet de las Cosas*» en el **congreso** *Proceedings of the 2016 11th Iberian Conference on Information Systems and Technologies (CISTI)* [46]. En este artículo se presenta la idea de crear la simulación de los sentidos del cuerpo humano mediante diferentes tipos de sensores a través del uso de Internet de las Cosas.
- **Agosto 2016**: publicación del artículo «*SenseQ: Creating relationships between objects to answer questions of humans by using Social Networks*» en el **congreso** *Proceedings of the 3rd Multidisciplinary International Social Networks Conference on SocialInformatics 2016, Data Science 2016 - MISNC, SI, DS 2016* [47]. En este artículo se presenta la idea de utilizar las Redes Sociales Online junto a Internet de las Cosas para responder preguntas realizadas por los seres humanos.
- **Septiembre 2016**: aceptación del **artículo** «*A review about Smart Objects, Sensors, and Actuators*» en la **revista** *International Journal of Artificial Intelligence and Interactive Multimedia* [48], con su próxima publicación en el volumen 4 tirada 3 del año 2017.
- **Noviembre 2016**: aceptación del **artículo** «*IoFClime: The fuzzy logic and the Internet of Things to control indoor temperature regarding the outdoor ambient conditions*» en la **revista** *Future Generation Computer Systems* [49], con su próxima publicación en 2017.

2.3 *ORGANIZACIÓN DEL DOCUMENTO*

Esta memoria de tesis está dividida en varios bloques relacionados para agrupar los diferentes capítulos que están relacionados. A continuación, se describe la organización de esta tesis con el motivo de facilitar su lectura:

- **Bloque I – Planteamiento del problema**: este bloque contiene los dos capítulos introductorios de esta tesis. El primer capítulo es el de la Introducción, donde se explica la motivación de realizar esta tesis, se plantea la hipótesis y se marcan los objetivos. Por otro lado, el segundo capítulo, titulado Metodología y desarrollo de la investigación, presenta la metodología del trabajo seguida en la realización de la presente tesis, el desarrollo temporal de la investigación y la organización de este documento.
- **Bloque II – Marco teórico**: en este segundo bloque se presenta todo el estado del arte y el trabajo relacionado de acuerdo con todo lo estudiado en esta tesis. Estos son, respectivamente con cada capítulo, los siguientes temas tratados: Internet de las Cosas, Objetos inteligentes, sensores y actuadores, Ingeniería Dirigida por Modelos, Lenguajes de programación, Seguridad: criptología, Redes Sociales Online, La Nube (Cloud Computing), Inteligencia Artificial, Big Data y Robótica.
- **Bloque III – ¿Qué problemas tratamos de resolver?**: en el tercer bloque se presentan los problemas principales que se han resuelto en esta tesis, estando estos divididos en tres grandes grupos: Internet de las Cosas, Desarrollo de Software para Internet de las Cosas y Seguridad.
- **Bloque IV – Solución general**: el cuarto bloque contiene la solución general propuesta para tratar de solucionar los problemas presentados en el bloque III, y que se presentan bajo los capítulos de Internet de las Cosas, Ingeniería Dirigida por Modelos y Seguridad.
- **Bloque V – Prototipos desarrollados**: el bloque número cinco presenta todos los prototipos desarrollados a lo largo de la tesis, y que han servido para probar los diferentes objetivos y verificar la hipótesis de esta tesis. Cada prototipo contiene los objetivos y la funcionalidad de este, la arquitectura que se necesitó desarrollar para montar el prototipo y obtener resultados, la metodología y los resultados obtenidos para realizar las pruebas, salvo en el último, y las conclusiones de cada prototipo. En total hay 7 prototipos, a saber: Plataforma hardware y software Midgar, Generación de aplicaciones para interconectar objetos heterogéneos y ubicuos utilizando un lenguaje de dominio específico, Generación de *Smart Objects* para Internet de las Cosas mediante el uso de Ingeniería Dirigida por Modelos, Detección de personas mediante el uso de Visión por Computador para mejorar la seguridad en escenarios de Internet de las Cosas, Seguridad en la comunicación de los *Smart Objects* a través de una plataforma de Internet de las Cosas, Un lenguaje orientado al usuario para especificar interconexiones entre objetos heterogéneos y ubicuos en Internet de las Cosas y MiBot: asistente personal automatizado programable mediante el uso de Internet de las Cosas.
- **Bloque VI – Conclusiones y trabajo futuro**: aquí se presentan, primeramente, las Conclusiones finales, tanto en inglés (Final Conclusions) como en español debido a que esta tesis es mención

internacional, que se pueden extraer de esta tesis doctoral. Estas incluyen la Verificación de los objetivos, demostrando que se han cumplido, las Aportaciones de esta tesis a la comunidad científica, todas las Publicaciones derivadas de esta tesis doctoral, y las Conclusiones generales de esta tesis doctoral desde mi punto de vista. Este bloque también contiene el Trabajo futuro que se puede realizar a partir de esta tesis dividido en varios bloques, según su temática.

- **Bloque VII – Anexos**: por último, el séptimo bloque incluye todos los anexos de la tesis, entre los cuales se encuentran las siglas y acrónimos, la conversión de términos inglés-español, que contiene la traducción libre adoptada de palabras no existentes en español y cuyos términos en inglés había que traducir, el glosario con los términos necesarios de comprender para entender la tesis, y las referencias, conteniendo todas las referencias utilizadas en esta tesis doctoral.

«Si conoces a los demás y te conoces a ti mismo, ni en cien batallas correrás peligro;
si no conoces a los demás, pero te conoces a ti mismo, perderás una batalla y ganarás otra;
si no conoces a los demás ni te conoces a ti mismo, correrás peligro en cada batalla»
Sun Tzu, El arte de la guerra, capítulo 3

«Lo poco que sé se lo debo a mi ignorancia»
Platón

«Cuanto más sabes, más te das cuenta de que no sabes nada»
«Este hombre, por una parte, cree que sabe algo, mientras que no sabe [nada].
Por otra parte, yo, que igualmente no sé [nada], tampoco creo [saber algo]»
Sócrates

Bloque II
Marco teórico

Contenidos de bloque

3. INTERNET DE LAS COSAS

«Internet de las Cosas tiene el potencial de cambiar el mundo, al igual que lo hizo Internet»
Kevin Ashton [4]

«La creatividad simplemente consiste en conectar las cosas. Cuando le preguntas a personas creativas cómo hicieron algo, se sienten un poco culpables porque en realidad no crearon nada, simplemente vieron algo. Les fue obvio después de un tiempo. Eso es porque fueron capaces de conectar las experiencias que habían tenido y las sintetizaron de formas nuevas»
Steve Jobs

Internet de las Cosas, o ***the Internet of Things*** (IoT), es la tecnología del futuro y que ya está tomando parte de nuestro presente. IoT es la interconexión de objetos heterogéneos y ubicuos a través de Internet. IoT es una forma de hacer más inteligentes, si cabe, a nuestros objetos, pues su finalidad es conectar todos los objetos del mundo para facilitar la vida diaria de las personas, ya sea bien ofreciéndoles ayuda en tareas diarias o bien, automatizando ciertos aspectos repetitivos y monótonos. Conectar nuestros objetos, nuestras casas, industrias, pueblos, ciudades, el medioambiente, y la Tierra.

No obstante, no es nada trivial el interconectar dos objetos heterogéneos entre ellos, mucho menos si se mueven o no tienen conexión permanente a Internet, luego, el intentar conectar todos los objetos del mundo es algo muy ambicioso. Además, a esto, hay que sumarle lo que conlleva realizar dicha interconexión, desarrollar software, programar.

Este capítulo presentará el marco teórico y el trabajo relacionado acerca de Internet de las Cosas, explicando que es y mostrando una visión de las diferentes plataformas IoT existentes. Además, también se verán las diferentes aplicaciones existentes en los diferentes ámbitos que integran IoT.

3.1 *¿QUÉ ES INTERNET DE LAS COSAS?*

Internet de las Cosas es la **interconexión** de **objetos heterogéneos** y **ubicuos** a través de **Internet** [7], [9], [50]–[52]. Esta es una de sus definiciones, pero…¿qué significa esta definición?

El término «**Internet de las Cosas**» surgió en el año 1999 en el título de una presentación de Kevin Ashton para Procter & Gamble, también conocida como P&G. Esta presentación trataba acerca del uso de la tecnología RFID en las cadenas de suministro [4]. En ella, Kevin Ashton introdujo la palabra **Internet** ya que era el tema de moda, el «*trending topic*» en aquellos años y él buscaba llamar la atención. No obstante, él nunca llegó a definir este término y no pensó en las consecuencias que ocurrirían 10 años después cuando lo usase la Unión Europea. Sin embargo, si hace referencia a que Internet necesita datos del mundo real, datos introducidos por los humanos. Para Kevin Ashton, el problema es que los humanos no disponen de tiempo, precisión y atención, por lo que no son buenos para capturar datos del mundo real. Así pues, se necesita de un dispositivo que los capture. Por otro lado, utilizó la palabra «***things***» para referirse a todas las cosas que se podrían representar, como son nuestras cualidades, nuestro ambiente, la economía o la sociedad, pues, según Kevin Ashton, estos no están basados en ideas o información, sino en algo más importante como son las «cosas». De esta manera, las computadoras podrían saber todo y no necesitarían nuestra ayuda. Estas podrían rastrear cosas, reducir el tiempo y los costes, saber cuándo algo necesita ser reparado o recolocado, y saber si algo está fresco o caducado. Pero para conseguir que los ordenadores consigan sentir y comprender el mundo, se necesitan dos cosas: RFID y sensores.

Continuando con la definición nos encontramos con las palabras «**objetos heterogéneos**». Esto se refiere a que **Internet de las Cosas** está compuesto de objetos diferentes. Internet de las Cosas pretende conectar millones de objetos de cualquier tipo a Internet, lo que repercutirá en posibilidades de grandes cambios en nuestras vidas, proporcionando una mayor productividad, mejorando la salud, mejorando la eficiencia, reduciendo energía y haciendo confortables nuestras casas [14]. Estos pueden ser, como ya se demostró en ciertas investigaciones, microcontroladores como los Arduino [7], [53]–[55], microordenadores como la Raspberry Pi [56], neveras [57], smartphones [58], [59], sensores [60], actuadores, Smart TVs, coches, etiquetas inteligentes como son RFID [61], NFC y los códigos de barra [62], los códigos QR (*Quick Response Code*, QR Code), entre muchos otros tipos de objetos [63]–[65].

La sentencia «**objetos ubicuos**» se refiera a que estos dispositivos pueden estar en cualquier sitio y en continuo movimiento mientras están conectados a la red. Esto implica que los objetos pueden estar en continuo movimiento y tal vez sufrir desconexiones. Además, estos se pueden encontrar en cualquier parte del mundo, ya sea dentro de nuestra casa, dentro de máquinas inaccesibles o moviéndose continuamente por que sean móviles o vehículos. También se pueden encontrar en el fondo de los océanos para ayudar a detectar el cambio climático [66]. Otros pueden estar dentro de centrales nucleares o habitaciones selladas llenas de productos químicos para detectar posibles cambios de temperatura o riesgos que puedan surgir al manipular estos productos, o se encuentren detrás de la líneas enemigas [60]. También podrían estar localizados en otros lugares inaccesibles tal vez debido a su toxicidad, localización o riesgo [67].

Internet de las Cosas

Internet es la red mundial de computadoras interconectadas mediante el Protocolo de Control de Transmisión (TCP) [68] y el Protocolo de Internet (IP) [69], conocido como TCP/IP. Esto permite que las redes que la componen se comporten como una red global para poder servir de mecanismo de diseminación, de medio de comunicación y medio de interacción entre individuos y sus computadores sin tener en cuenta su localización geográfica [70].

IoT abarca un conjunto de **tecnologías** que buscan permitir la interconexión e interoperabilidad [4] de objetos heterogéneos y ubicuos a través de diferentes redes. Así, los dispositivos de una red IoT dependen de tecnologías sensoriales, de comunicación y de procesamiento de la información para conectarse a esta infraestructura de red global [10]. Actualmente, según Gartner, más del 50% de las conexiones en Internet son de objetos, además de situar IoT como una de las 10 tecnologías más de moda en el año 2013 [71]. Entre las tecnologías existentes en IoT existen diferentes protocolos, que van desde los clásicos como TCP/IP, hasta protocolos construidos específicamente para IoT, como es *Message Queue Telemetry Transport* (MQTT) [72]. Lo mismo ocurre con el hardware, pues hay desde el más típico computador o smartphone hasta las tarjetas inteligentes o procesadores especiales diseñados para funcionar en entornos que necesiten condiciones especiales, como puede ser, pequeño espacio, gran duración de la batería o poco consumo eléctrico, y conexión a Internet. Un ejemplo actual de estos es el chip Intel Edison [73] o el microcontrolador Arduino [55]. Además, en los últimos años, esto mejoró notablemente debido al abaratamiento de los sensores y la mejora de las redes inalámbricas [60]. Esto nos permite poder dispersar todo tipo de sensores alrededor de todo el mundo.

Dentro de IoT existe una clara diferenciación en donde IoT es aplicado. Estos son las casas, conocidas como ***Smart Homes***, la industria, llamada ***Industrial IoT*** **(**IIoT), los pueblos, los cuáles fueron agrupados en el concepto ***Smart Towns***, las ciudades también llamadas ***Smart Cities*** y el medio ambiente que se sitúa bajo el concepto ***Smart Earth***. Cada uno de estas divisiones tiene sus propias aplicaciones orientadas a una finalidad claramente diferente frente a las otras, aunque todas se basan en IoT. Las casas buscan priorizar la automatización, la industria mejorar el proceso industrial, los pueblos el no olvido de su cultura, las ciudades su habitabilidad, y el medioambiente la comunicación con los edificios y la naturaleza. En base a esto, la finalidad es crear un mundo inteligente, a ***Smart World*** [13].

Debido a estas razones, cobran también mucha importancia los **objetos inteligentes** o ***Smart Objects***, que son objetos físicos con un sistema embebido que le permite procesar información y comunicarse con otros dispositivos y realizar acciones en base a una acción o evento determinado [7]. Estos pueden ser los smartphones, un micro-controlador como es Arduino [7], [53]–[55] o cualquier otro objeto que tenga conectividad a la red [9], y sea capaz, como mínimo, de gestionar información [26]. Así, en combinación con Internet de las Cosas, se consigue que la red pase de solo transportar datos, a dotarla de inteligencia y acciones según los datos que recogen estos objetos y los servicios que estos permiten realizar.

[4] Glosario: **Interoperabilidad**

Por todos estos motivos, entre muchos otros, la finalidad de **Internet de las Cosas** es interconectar todos los objetos, sea cuál sea su finalidad, tipo y emplazamiento, con todos los demás objetos. Esto ayudará a crear un Internet más inteligente y con más datos, que nos ayude con nuestra vida y problemas diarios. Por ello, de lo que se trata es de interconectar a través de Internet, las personas con las máquinas (H2M) [74], mejorar la comunicación entre personas (H2H) [75], o conectar los propios objetos entre ellos bajo lo que se conoce como *Machine-to-Machine* (M2M) [10], [75], [76]. O, dicho de otra forma, conectar todo el mundo físico con el mundo virtual a través de diferentes lugares inteligentes para automatizar, mejorar y facilitar la vida diaria. Además de poder mantener estas interconexiones a cualquier hora en cualquier lugar del mundo, mientras los objetos o las personas se están moviendo, ya estén fuera o dentro de los edificios. El único requisito necesario para estos objetos es tener acceso a **Internet** [9].

3.2 *PLATAFORMAS DE INTERNET DE LAS COSAS*

Internet de las Cosas es la interconexión de objetos a través de Internet. Para ello, se necesita de uno o más sistemas centralizados que permitan dicha conexión. En algunos casos, incluso los objetos necesitan consumir datos de algún sitio para realizar su función o conectarse con otros objetos.

Actualmente, hay muchas plataformas de Internet de las Cosas para interconectar nuestros objetos, las cuáles nos ofrecen grandes, muchas y muy diversas funcionalidades. Sin embargo, el propósito final de todas estas y el como un usuario define las interconexiones entre sus objetos, difiere bastante. Por eso, se pueden clasificar estas plataformas en cuatro grupos [42]: plataformas de negocio, plataformas de investigación, plataformas en estado beta y plataformas de código abierto.

En el primer grupo, **plataformas IoT de negocios**, nosotros hemos clasificado las plataformas IoT con una clara orientación al mundo de los negocios. En otras palabras, las plataformas IoT que solo venden servicios. En este grupo podemos encontrar nueve plataformas: Xively [77], Carriots [78], Exosite [79], SensorCloud [80], Etherios [81], ThingWorx [82], Azure IoT Suit [83], Amazon Web Services [84] e IBM *Internet of Things* [85].

En segundo lugar, están las **plataformas de investigación**. Este grupo contiene aquellas plataformas que son usadas por investigadores para investigar en el campo de Internet de las Cosas. Alguna plataforma está todavía en estado beta, pero todas ellas tienen información publicada en diferentes artículos y conferencias. Estas plataformas IoT son: Paraimpu [86], SIoT [87], SenseWeb [88], [89] y QuadraSpace [90].

El tercer grupo está conformado por las **plataformas en estado beta**. La diferencia con las otras plataformas reside en que las plataformas en estado beta ofrecen la posibilidad de usarlas, pero necesitas de una invitación para ello debido a que, según lo que explican, se encuentran en estado beta, o bien, solo permiten ver la plataforma sin poder interaccionar con ella. Este es el caso de las siguientes plataformas IoT: Open.Sen.se [91], Sensorpedia [92], [93] y Evrythng [94].

Finalmente, las **plataformas de IoT de código abierto** constituido el cuarto grupo. Este grupo contiene las plataformas IoT que permiten crear una red IoT basada en su software por un usuario cualquier en su propio servidor. Estas plataformas son: ThingSpeak [95], Nimbits [96] y Kaa [97].

A continuación, se hablará de las plataformas IoT más relevantes de cada grupo.

Ilustración 3 Logotipo de Xively

Xively, previamente conocida como COSM y antes como Pachube, es una plataforma IoT de negocios que ofrece muchos ajustes de configuración acerca de la privacidad. Además, provee acceso utilizando REST (*Representational State Transfer*). **Xively** también tiene un sistema para hacer más fácil la conexión entre objetos creando disparadores. Estos disparados están pensados para que usuarios sin conocimientos de programación puedan crearlo. No obstante, estos disparados tienen como límite una acción de tipo REST por cada dispositivo, debido a que el usuario tiene que poner una petición HTTP usando el método POST para enviar las notificaciones cuando el disparador se activa en el objeto elegido.

Ilustración 4 Isologo de Carriots

Carriots es otra plataforma IoT de negocios que ofrece una solución tipo Plataforma como un Servicio (PaaS) para construir aplicaciones para IoT. Carriots provee de una API REST para manejar casi toda interacción con el servicio como es crear, mostrar, actualizar y borrar dispositivos. Así, el usuario tiene que conectar sus dispositivos y recolectar sus datos, para, después, construir la aplicación y ejecutarla. Además, Carriots soporta muchos microcontroladores, pero no otros tipos de objetos como los smartphones. Otra cosa que el usuario debe hacer es programar el código fuente que el dispositivo debe de ejecutar para conectarlo a la API REST de Carriots y después crear la interconexión entre los objetos utilizando un formulario web que ofrece la propia plataforma.

Ilustración 5 Imagotipo de Paraimpu

La siguiente plataforma de IoT de investigación es **Paraimpu**, aunque ahora vende servicios. Esta plataforma fue utilizada previamente en muchas investigaciones como se puede ver en [54], [98]–[100]. Paraimpu ofrece una API cURL (*Client for URLs*, *Client URL Request Library*, o *see URL*) para interconectar los dispositivos con la plataforma. Para trabajar con Paraimpu, el usuario debe de enviar una petición para indicar que servicio envía datos y que servicio los recibe. Después de esto, hay que analizar los datos en el segundo dispositivo, lo que nos obliga a bajarlos a este, y hacer las acciones deseadas para los datos recibidos. Por ello, se debe de añadir toda la lógica en los diferentes dispositivos.

Ilustración 6 Imagotipo de SIoT

SIoT es una red IoT enmarcada en lo que se conoce como IoT Social, en inglés *Social IoT*. SIoT fue utilizada anteriormente en diferentes propuestas de investigación [51], [52], [101] y fue construida sobre el núcleo de **ThingSpeak**. SIoT es una aplicación RESTful y ofrece soporte para los formatos *Comma-Separated Values* (CSV), JSON y *eXtensible Markup Language* (XML) usando métodos GET y POST. La principal meta de SIoT es permitir la creación de listas de amigos con otros objetos para crear sus relaciones. Actualmente, se permite la descarga de esta red para poder montarte una red IoT en tu propio servidor.

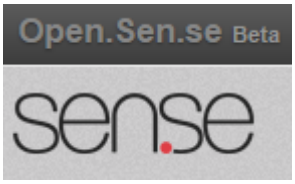


Ilustración 7 Logotipo de Open.Sen.se

Una plataforma IoT en beta es **Open.Sen.se**, la cual necesita una invitación para permitir usarla. Esta plataforma ofrece soporte para diferentes protocolos como *Hypertext Transfer Protocol* (HTTP) [102], *Extensible Messaging and Presence Protocol* (XMPP) [103] y *Constrained Application Protocol* (CoAP) [104], [105]. Además, ofrece diferentes gráficas sobre los valores de los objetos, que se pueden visualizar en la página del dispositivo en el portal web o compartir en otras páginas. Sin embargo, Open.Sen.se solo permite crear interconexiones entre objetos y servicios web.

ThingSpeak es una plataforma IoT de código abierto que ofrece la posibilidad de subir los datos de diferentes objetos. Una vez los datos están subidos, se puede visualizar en diferentes gráficas. No obstante, ThingSpeak nos obliga a bajar estos datos en un fichero en bruto y procesarlo en el dispositivo en el que deseemos hacer acciones en base a esos datos. Esto implica la descarga continua de los datos y el tener programadas las diferentes condiciones en el propio dispositivo, así como a realizar el procesamiento en él.

Ilustración 8 Imagotipo de ThingSpeak

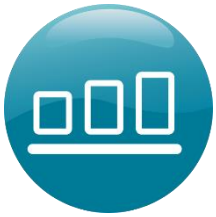


Ilustración 9 Isotipo de Nimbits

Otra posibilidad de software abierto es la plataforma IoT **Nimbits**. Esta dice que soluciona el dilema de *Edge Computing* [5] aplicado a IoT. Para desplegar Nimbits, solo hace falta descargar el fichero WAR (*Web Application Archive*) de la aplicación y desplegarlo en un servidor. La funcionalidad que tiene se basa en el uso de formularios para establecer alarmas, filtros y cálculos entre otras opciones. Como contra, Nimbits limita los disparadores a solo tres parámetros.

Kaa es otra plataforma IoT de código abierto. Esta red IoT permite conectar tu aplicación con otros objetos utilizando como *middleware* [6] el servidor Kaa para crear los *endpoints* [7]. Además, tiene una integración con sistemas de gestión de datos y sistemas de análisis. No obstante, como ocurre con otras plataformas, hay que programar la aplicación que debe funcionar en los diferentes objetos y conectarse con el servidor Kaa.

Ilustración 10 Imagotipo de Kaa

3.3 *Campos de Internet de las Cosas*

Internet de las Cosas abre la puerta a muchas posibilidades y por ello quiere estar presente en todos los ámbitos de nuestra vida. Por ello, existen diferentes tipos de representaciones de IoT acotadas a un determinado dominio específico. Todas estas buscan fines similares como son la mejora de nuestra vida diaria, pero se aplican de diferente forma. En la literatura actual se encuentran muchos ejemplos de los diferentes

[5] Glosario: ***Edge Computing***
[6] Glosario: ***Middleware***
[7] Glosario: ***Endpoint***

tipos, cada uno con su propio nombre, aunque a veces este difiere según el autor, pero que coincide en el uso y/o finalidad: crear un mundo más inteligente, fácil y habitable.

La **habitabilidad** depende de la situación, pues, según el *Departamento de Transporte de los Estados Unidos* (*U.S. Department of Transportation*) existen seis principios de habitabilidad [106], pero no todos los principios son aplicables a cada los campos de aplicación de IoT. El primero de ellos es ofrecer más alternativas de transporte para reducir los costes y la dependencia del petróleo y mejorar la salud pública. El segundo es ampliar las opciones de vivienda y su eficiencia energética para todas las personas e incrementar de esta manera la movilidad y reducir los costes de los hogares y del transporte. El siguiente es mejorar la competitividad económica de los barrios con el fin de hacer más accesible las necesidades de trabajo, educación y otras necesidades básicas. En el cuarto punto se habla de la revitalización de comunidades orientando el tráfico y el reciclaje de la tierra, reduciendo así costes y salvaguardando los paisajes. El siguiente punto trata la planificación del crecimiento del futuro por medio del alineamiento de las políticas federales y los fondos públicos. Por último, se habla sobre la mejora de las características únicas de las comunidades invirtiendo en vecindarios seguros y transitables.

Por ello, cada campo de IoT necesita habitabilidad, pero no todos los campos cumplen los seis puntos presentados anteriormente.

3.3.1 SMART HOMES

Las ***Smart Homes***, también conocidas como *Intelligent Homes* [7], o en español como Casas inteligentes, son las redes IoT más cercanas a nosotros. El propósito de los diferentes trabajos en esta área es mejorar la habitabilidad de las casas usando la monitorización de algunos elementos, la creación de ciertas automatizaciones y permitiendo controlar casi toda la casa desde casi cualquier dispositivo, como puede ser un smartphone o un control remoto. Así, el propósito de las ***Smart Homes*** es equipar una casa con tecnología para proveer servicios [107]. De esta forma, esta habitabilidad mejora nuestra vida diaria por la posibilidad de crear eventos automáticos para ciertas condiciones, ofreciéndonos un control remoto de la casa, ahorrándonos dinero en electricidad o ayudándonos con diferentes cosas. Normalmente estos sistemas están basados en redes WSAN para interconectar los diferentes objetos. Algunas veces, los investigadores utilizan microcontroladores para crear los *Smart Objects* para interconectar los sensores con la red WSAN. En este caso, ellos suelen usar un microcontrolador Arduino o, en el caso de que necesiten más potencia, un microordenador como es la Raspberry Pi.

Como se muestra en [7], se pueden abrir o cerrar puertas utilizando una tarjeta RFID, encender o apagar las luces automáticamente utilizando sensores fotorresistores, tener una calefacción con control automático utilizando un sensor de temperatura, abrir o cerrar automáticamente las persianas o tener un sistema de riego automático en nuestro jardín.

Otro ejemplo es el de la típica nevera inteligente, *Smart Fridge*. Esto se encuentra englobado en el campo de ***e-Health***, que es la monitorización de la salud por medio de la inclusión de tecnologías IoT [108]. Como podemos ver en [5], los autores utilizan la tecnología RFID para crear una nevera que provea servicios para ajustar los hábitos de comida del dueño. Otro ejemplo similar lo podemos encontrar en [109]. Este artículo

presenta una simulación para obtener la respuesta acerca de si la gente apreciará una nevera con conocimiento sobre su contenido. En esta simulación, los autores analizaron la comida sana y el valor económico. Mientras tanto, en [57] se presenta otra forma de gestionar la comida de un frigorífico para advertir a su dueño sobre los diferentes métodos de cocina basados en la comida almacenada en el frigorífico. Esta nevera inteligente puede sugerir recetas, hacer demostraciones multimedia, generar listas de la compra, avisar acerca de la comida que está a punto de caducar y gestionar las calorías.

Otras investigaciones buscan el ahorro de energía en nuestras casas [9]. La razón es que utilizando IoT se puede ahorrar bastante energía y dinero.

Incluso, algunos trabajos proponen utilizar las ***Smart Homes*** para ayudar a la gente mayor, a la gente con discapacidades [110] o a la gente con alguna capacidad reducida [107]. Durante la última década este tipo de uso se ha incrementado [110], [111]. Por ejemplo, un posible uso de las ***Smart Homes*** es el de mejorar la asistencia a domicilio de la gente mayor o con alguna discapacidad con el menor impacto en sus vidas [111].

Estos son algunos de los diferentes ejemplos acerca de las posibilidades que Internet de las Cosas nos ofrece para mejorar nuestra vida diaria en nuestras casas.

3.3.2 INDUSTRIAL INTERNET OF THINGS

Internet de las Cosas Industrial (**IIoT**) también conocido como Industria 4.0, es la combinación de Internet de las Cosas con la industria de acuerdo con el reporte del *Industrie 4.0 Working Group* del Ministerio Federal de Educación e Investigación Alemán [112]. IIoT surgió junto a **Internet de las Cosas**, pues de este tipo fue la aplicación que en el año 1999 Kevin Ashton utilizó en la presentación para P&G [4]. No obstante, en los últimos años han sido desarrolladas un gran número de aplicaciones de **IoT** para la industria [19]. Esto es un indicativo de que se sigue evolucionando y aún es necesario **IoT** en la industria, pues, como tecnología emergente se espera que **IoT** ofrezca soluciones para transformar los sistemas de transporte y manufacturación [19], mejorándolos para obtener una mejor eficiencia, ahorro de costes y una reducción del tiempo en los procesos.

Mirando la literatura, podemos encontrar que el interés en **IoT** sigue creciendo en diferentes tipos de industrias como son la agricultura, la alimenticia, la farmacéutica y la minera, entre muchas otras. En estos casos, la tecnología que más ha sido utilizada es RFID, pues, se ha estado utilizando para mejorar la eficiencia de las cadenas de suministro [113] y evitar el robo [114]. Lo que se trata de conseguir con esta tecnología es sustituir a los códigos de barra para optimizar y mejorar la información obtenida durante todo el proceso en tiempo real por un coste similar [115].

Como se comentó previamente, existe un gran abanico de usos en **IIoT**. Uno de los más típicos en los ejemplos es el de la monitorización de objetos. En [19] utilizan como ejemplo un sistema que es capaz de poder localizar el transporte y monitorizarlo. De esta manera, se podría predecir su localización futura y el tráfico que tendrá en la carretera.

Otro uso es en la asistencia médica como comentan en [116], donde creen que IoT puede ayudar a las personas con alguna incapacidad a lograr una vida de calidad. Esto es muy importante, pues, según la *World*

Health Organization y *The World Bank* en el informe del año 2011 [117], en el año 2010, un 15% de la población, la cual estaba estimada en 6.900 millones de personas, tenía alguna discapacidad. Mientras, en [118] proponen una plataforma IoT abierta para crear un ecosistema de salud con ciertos mecanismos de seguridad incorporados. En estos casos, **IoT** permite una identificación ubicua que permita obtener datos y comunicarse con los trabajadores, el equipo y los enfermos mediante un monitoreo constante de personas con enfermedades crónicas [119], así como un aviso temprano de casos de emergencia, pues puede ser que el paciente no pueda avisar o este viva solo. En estos casos, se podría salvar la vida a pacientes gracias a la rápida identificación de ciertos problemas debido a la puesta de sensores en el paciente y el aviso al médico o equipo médico más cercano utilizando ciertas reglas de Inteligencia Artificial, o bien, personas que revisasen esos datos previamente y dieran el aviso pertinente. Estas aplicaciones pueden ser de varios tipos, como por ejemplo el detector de actividad diaria, de detección de movimiento y caídas, de monitorización de localización, de monitorización de toma de medicación y de monitorización del estado médico. Otros posibles ejemplos serían los que se presentaron en el apartado de ***Smart Homes*** y que son conocidos como *IoT-powered in-Home Healthcare* (IHH) [118].

Las minas son ambientes complejos, con grandes cantidades de gases y muy diversos, de los cuáles algunos pueden provocar explosiones. A esto hay que sumarle el posible peligro de desprendimiento, fuego o rotura de materiales que pueden provocarse. Por estos motivos, actualmente se están incluyendo soluciones **IoT** para prevenir y reducir los accidentes mediante la detección de señales de desastres y así conseguir alertar antes de que ocurra una catástrofe [120]. De esta manera, pueden obtener datos acerca de un posiblemente movimiento de tierra, el cese de un techo o el estado de las vigas o bien obtener datos de la catástrofe para ayudar a los médicos a realizar un plan de rescate más efectivo. No obstante, el problema reside en cómo comunicar estos datos con la superficie en tiempo real pues, el uso de tecnologías inalámbricas dentro de una mina puede llevar a una explosión debido a los gases que residen en ella. Por este motivo, se necesita investigar una forma de incorporar **IoT** en las minas evitando este riesgo, pues, gracias al uso de sensores se podría comunicar la mina con la superficie e ir realizando la monitorización continua de los mineros. Otro posible uso sería para la identificación de gases mediante los sensores adecuados y así poder medir, prever y diagnosticar posibles riesgos y accidentes que tengan que ver con la fuga de gases perjudiciales para la salud o que tengan carácter explosivo.

Sin embargo, si se habla de **IIoT** hay que hablar de las cadenas de suministro, pues, como se comentó, aquí nació IoT y fue donde se aplicó la tecnología RFID con mayor énfasis. Por eso mismo, IoT tiene un rol tan importante en la industria logística [2]. En estas se utiliza **IoT** para monitorizar el movimiento de objetos desde su origen hasta su destino a través de la cadena de suministro, ya sea dentro de una fábrica o a través de su distribución mientras va en barco, tren, por carretera o en avión. Es decir, en las cadenas de suministro se suele tender a utilizar IoT para identificar productos y obtener información acerca de su almacenaje, pero si se le incluye el uso de sensores se podría obtener más datos del entorno, como la localización y las condiciones del ambiente [26]. No obstante, también hay ideas para prototipos que implementan inteligencia en las carretillas elevadoras obteniendo datos de los almacenes [7]. Por el lado de las empresas, unas intentan mejorar las cadenas de suministro por medio de la monitorización en tiempo real del envío mediante la recolección de la localización, temperatura, luz, humedad relativa y presión biométrica, como es el caso de

SenseAware [121]. Por otro lado, Cantaloupe Systems [122] ofrece un sistema para que sus usuarios puedan ver el stock que tienen remotamente en sus máquinas de ventas y así poder saber el número de productos que les quedan y si deben ir a reponer o no la máquina.

Algo vinculado con lo anterior pero también con nuestra vida diaria es lo que explican en [123]. Zhou et al. hablan de diferentes métodos de utilizar los vehículos inteligentes en IoT para mejorar la logística de la carretera, crear advertencias de seguridad, tráfico o problemas en ella, además de una posible arquitectura que soporte todo esto. Como se ve, con IoT se podría monitorear los vehículos y las condiciones externas de manera que ayudase a incrementar la ayuda que ofrecen los vehículos y así, por ejemplo, saber si hay vehículos parados en medio de la carretera. Un sistema que implementa esta idea es el sistema de inteligencia iDrive de BMW [124], asistiendo al conductor mediante la medición de los agentes externos del automóvil utilizando sensores y monitorizando el vehículo por GPS. Otro sistema similar es el propuesto por Zhang et al. en [125], quienes diseñaron un sistema de monitorización inteligente de humedad y temperatura para los camiones refrigerados por medio del uso de etiquetas RFID, sensores y comunicación inalámbrica.

Profundizando más en el transporte, la ciudad de Newcastle, en el Reino Unido, probó un sistema de prueba para enviar señales a los conductores para ajustar su velocidad cuando el semáforo va a cambiar [126]. Otra investigación del Reino Unido en vehículos tiene que ver con los coches autónomos para mejorar la conducción, y reducir el tráfico y el dióxido de carbono ($CO_{2)}$ [127]. Más ejemplos como estos se pueden encontrar en el informe [14], que además contiene más usos de IoT en el transporte para mejorar así la conducción, en sanidad de cara a prevenir enfermedades o conseguir un diagnóstico previo más rápido, y en la agricultura para monitorizar los animales y las cosechas o ciertos parámetros relativos a las cosechas o la tierra, como pueden ser la humedad y la temperatura.

En China han utilizado IoT en la lucha contra los incendios y así poder detectar anticipadamente un riesgo de incendio. Para ello, para gestionar los elementos antincendio utilizan etiquetas RFID y códigos de barra. De esta manera, presentan una forma de gestionar y mantener los diferentes elementos necesarios contra incendios [128]. Otras posibles ideas de la aplicación de IoT son las presentadas en [129], donde ilustran una posible infraestructura IoT que pudiera servir para casos de emergencia en China, como puede ser la inclusión de sensores en los bomberos para tenerlos monitorizados durante un incendio, la monitorización de materiales peligrosos para saber su estado en tiempo real y así tenerlos bajo control, la monitorización de sangre donada para avisar si ocurre algún incidente durante su transporte, la protección medioambiental mediante la detección de fugas de gases o un sistema anti intrusos para advertir ataques terroristas o la entrada de fugitivos.

No obstante, el mejor ejemplo de IIoT posiblemente sea el de Amazon [130]. La sección de *Amazon Robotics* se encarga de mejorar la cadena de suministro dentro de los propios almacenes de Amazon mediante el uso de robots e informática para así tener un servicio más eficiente dentro de la propia empresa.

3.3.3 SMART TOWNS

El concepto de ***Smart Town*** puede ser parecido al concepto de ***Smart City***, no obstante, ambos son muy diferentes.

Las ***Smart Towns*** son pueblos o ciudades pequeñas con una gran cultura y herencia, las cuáles necesitan ser preservadas y protegidas, además de necesitar mejorar su habitabilidad y sostenibilidad utilizando tecnologías IoT [131]. No obstante, en este caso, la habitabilidad es secundaria debido a que lo más importante en las ***Smart Towns*** es cuidar de su pasado para recordarlo, priorizando la preservación y la revitalización para ello [131].

Asimismo, de acuerdo con la Organización de las Naciones Unidas para la Educación y Diversificación, la Ciencia y la Cultura (UNESCO) [132], hay que tener en cuenta que: «*Heritage constitutes a source of identity and cohesion for communities disrupted by bewildering change and economic instability*». Para conseguir este objetivo, la UNESCO ha adoptado tres enfoques: como punta de lanza la defensa de la cultura para salvaguardar la herencia, fortalecer la industria creativa y fomentar al pluralismo cultural. Estos enfoques podrían revitalizar y renovar las áreas comerciales, lo cual es un trabajo muy difícil [133].

Sin embargo, las ***Smart Towns*** también necesitan la habitabilidad ya que necesitan vivir en el presente. Además, las ***Smart Towns*** necesitan de la sostenibilidad para preparar los pueblos y ciudades pequeñas para el futuro ya que es necesario el compromiso entre el presente y el futuro para satisfacer las necesidades presentes sin comprometer el futuro [134].

Otra diferencia con las ***Smart Cities*** *es* el tamaño, pues las ***Smart Towns*** son más pequeñas que las ciudades. Los datos dicen que, en 2014, el mundo tenía el 46% de su población viviendo en áreas rurales en vez de en las áreas urbanas. Esto representa un decrecimiento desde el año 1950, cuando la población de las áreas rurales representaba el 70% de la población de la Tierra. Asimismo, las Naciones Unidas prevén un decrecimiento de la población en las áreas rurales de hasta el 34% en 2050. Otra diferencia es en África y Asia, donde la población rural representa el 40% y el 48% respectivamente, mientras que en el Norte de América es del 18% [135].

Así, las ***Smart Towns*** necesitan una protección y una forma para expandir su cultura para evitar el olvido de su cultura, herencia, monumentos, paisajes y vistas, folklore, tradición, y todo aquello necesario para no olvidar su pasado [136]. Esto es muy importante, ya que es lo que compone la identidad de cada pueblo, de cada civilización, las cuales tienen que protegerse para sobrevivir a través del tiempo para las futuras generaciones.

Para lograr estos objetivos, las ***Smart Towns*** pueden compartir fotos o videos con sus paisajes, grabar su tradición y folklore, monitorizar determinados lugares que necesiten condiciones especiales como museos o bibliotecas, o proteger monumentos y edificios [136]. De esta manera, IoT puede ayudar a los pueblos y ciudades a lograr estas metas [131], [137].

Sin embargo, las tecnologías que permiten y utilizan las ***Smart Towns*** son las mismas que las ***Smart Cities***, pero las metas de estas y la forma de conseguirlas son diferentes, como se ha explicado.

3.3.4 SMART CITIES

Una ***Smart City*** es una ciudad que ofrece nuevos y diferentes servicios a los ciudadanos [138], para mejorar la habitabilidad de la ciudad [139], interconectando objetos. Así pues, lo que buscan las ***Smart Cities*** es facilitar la vida diaria de los ciudadanos usando la conexión de sensores y actuadores para proveer nuevos servicios, automatizar tareas repetitivas y crear una ciudad más inteligente que pueda tomar decisiones u ofrecer información dependiendo del contexto o usuario.

Por ello, el propósito final de estas es crear un mejor uso de los recursos públicos, incrementar y mejorar los servicios ofrecidos a los ciudadanos y reducir el coste del gobierno. Algunos ejemplos son la optimización de servicios públicos como las luces, el transporte público, la vigilancia y el mantenimiento de áreas públicas, la salud de las estructuras de los edificios, la gestión de la basura, la calidad del aire, la monitorización del ruido, la congestión del tráfico, el consumo de energía de la ciudad y la automatización y salubridad de los edificios públicos [3], [140].

Las ***Smart Cities*** son grandes ejemplos de las interacciones de una red IoT porque proveen de una enorme red de sensores distribuida alrededor de la ciudad para obtener datos sobre la propia ciudad a través de objetos heterogéneos y ubicuos. Algunas veces, las WSNs o WSANs de las ciudades están trabajando junto a sistemas inteligentes para interconectar los objetos y dotarlos de inteligencia.

Sin embargo, la discusión entre diferentes investigadores es grande debido a que cada uno usa diferentes criterios para clasificar las ***Smart Cities***. Se pueden citar como ejemplos los siguientes artículos [141]–[143]. Entre estas ciudades se puede encuentran Luxemburgo, Eindhoven, Aberdeen u Oviedo. En los rankings de estos artículos se consideran para clasificar las ciudades los diferentes servicios que cada ciudad ofrece a sus ciudadanos.

El primer ranking de todos considera seis diferentes puntos, a saber, *Smart Economy*, *Smart People*, *Smart Goverment*, Smart *Mobility* y *Smart Living* [141]. Cada uno de estos puntos se divide en 4 o 6 subpuntos para así obtener la nota final de una ciudad en ese punto. Algunas de estas características son el espíritu de innovación, el espíritu emprendedor, la productividad, la cualificación de la gente y su creatividad, el cosmopolitismo, los servicios públicos y sociales, la transparencia del gobierno, la innovación en el sistema de transporte, las condiciones naturales, la polución, las facilidades culturales, las condiciones de salud o las facilidades educativas. Como se ve, este primer ranking considera muchas y muy diferentes cualidades las que una ciudad debe tener.

El segundo ranking contiene 250 indicadores para medir acerca de temas como son la demografía, los aspectos sociales y económicos, el entrenamiento y la educación, o la cultura. En cambio, otros investigadores consideran otros indicadores como el agua, la comunicación, la seguridad pública y la energía [144].

En la literatura actual se pueden ver otros ejemplos de algunos proyectos europeos debido al hecho de que la Unión Europea esta haciendo un constante esfuerzo en mejorar el crecimiento del área inteligente de las áreas metropolitanas [142]. Uno de estos proyectos es conocido como SmartSantander [145], [146]. En estos artículos, los autores proponen una arquitectura para usar Internet de las Cosas en las ***Smart Cities***, complementándolo con algunos ejemplos de su uso.

Otro ejemplo es el de la ciudad de Ámsterdam, la cual ha definido su ***Smart City*** en base a cuatro áreas: vida sostenible, trabajo sostenible, movilidad sostenible y espacios públicos sostenibles. Además, esta ciudad está centrada en la reducción del CO_2 [147].

Padua, en Italia, es otra ***Smart City*** en construcción [140], [148]. Exactamente, la Universidad de Padua tiene una red de sensores inalámbrica experimental con más de 300 nodos [149]. El proyecto ha promocionado la adopción de soluciones de datos abiertos y tecnologías de la comunicación y de la información, en inglés *Open Data and Information and Communications Technology* (ICT), en la administración pública. Este proyecto ha recopilado datos medioambientales como son los del Monóxido de Carbono (CO), temperatura, humedad y supervisión del ruido y de las luces de la calle. Otra ciudad italiana con alguna mejora en este aspecto es Cosenza [150], la cual ha hecho comparaciones entre diferentes transportes públicos urbanos y su rendimiento energético.

Otras ***Smart City*** son la presentada por Li Hao, Xue Lei, Zhu Yan, y Yang ChunLi en [3], donde ellos hablan sobre diferentes ciudades chinas, las cuales tienen diferentes puntos de vista de cómo crear una ciudad inteligente. Un primer punto de vista es el de la ciudad de ShenZhen, ciudad que lanzó una estrategia para mejorar la infraestructura basada en sistemas de *e-commerce*, transporte e industria. Mientras tanto, la ciudad de NanJing propuso un sistema similar, pero añadió el gobierno y la humanidad. En cambio, en el caso de la ciudad de WuHan, que contó con la colaboración de International Business Machines Corp., más conocida por sus siglas, IBM, promocionó Internet de las Cosas, *Cloud Computing* y más mejoras en las industrias y en la construcción. Un caso similar es el de la ciudad de KunShan ya que propone la mejora del equipamiento y la industria. Otras dos ciudades que colaboran con IBM son KunMing y ShenYang. Ambas ciudades están buscando mejorar la gestión de servicios. Además, en el caso de KunMing, la ciudad quiere mejorar también el cuidado médico y el *e-goverment*. Mientras, la otra ciudad, ShenYang, se centra también en ser más ecológica debido a que posee industria pesada. Una ciudad parecida a esta última es HangZhou, que propone una ciudad inteligente verde para proteger el medio ambiente. No obstante, existen muchos más ejemplos de ciudades en China que buscan convertirse en ***Smart City***.

Como hemos visto, es muy importante en el caso de las ***Smart Cities*** la mejora de la habitabilidad en las ciudades debido al hecho de su continuo crecimiento, pues, como se puede ver en [135], la población urbana del mundo ha crecido entre 1950 y 2014 en más de tres mil millones. Este estudio también comenta que la población crecerá en 2.5 mil millones hasta el año 2050. Por eso, necesitamos preparar las ciudades para crear un futuro sostenible con la mejor habitabilidad posible.

3.3.5 **SMART EARTH**

En 1998, Al Gore [151], vicepresidente de los Estados Unidos de América, definió el concepto ***Digital Earth*** como «*A new era of technological innovation to capture, store, process, and display information about our planet*». Para lograr este cometido, las tecnologías necesarias de acuerdo con Al Gore son: la ciencia computacional para gestionar, simular y entender las observaciones, las imágenes por satélite para obtener fotografías con buena calidad, las redes de banda ancha para conectar los objetos con las bases de datos usando

redes de alta velocidad, Internet para interconectar los diferentes objetos, y los metadatos [8] para obtener más información acerca de las cosas.

La ***Digital Earth*** fue el comienzo de lo que se conoce en IoT como ***Smart Earth***, pues esta es la combinación entre Internet de las Cosas y la ***Digital Earth***. La razón es que con IoT se pueden obtener datos sobre el entorno que nos rodea para representarlos en los sistemas digitales, que es donde se encuentra la ***Digital Earth***. Por ello, ***Smart Earth*** es la fusión de **Internet de las Cosas** con ***Digital Earth*** [152].

Por ello, el principio de ***Smart Earth*** es tener sensores heterogéneos y ubicuos distribuidos en cualquier tipo de entorno para percibir y medir sus cualidades físicas para interconectar estos objetos con el mundo digital. Estos objetos pueden ser puentes, presas, túneles, edificios, carreteras o cualquier otro elemento que nos rodee [3]. Así pues, la ***Smart Earth*** es el concepto de observar la Tierra para mejorar su comunicación con nosotros, ya que esta es una forma de obtener las cosas que la Tierra está sintiendo, obtener datos para cuidarla y monitorizarla en busca de pistas para evitar desastres.

Un ejemplo de esto es que algunos investigadores están desarrollando sistemas de monitorización del cambio climático y de oceanografía operacional para obtener datos en tiempo real usando sensores heterogéneos [66]. Debido a este último propósito, los autores han creado un sensor inteligente para la observación del océano, donde dicho sensor es sumergido en los océanos y usa un sistema de anclaje por cable con batería inductiva y un control en tiempo real.

Mientras tanto, otros trabajos están centrados en cómo gestionar los diferentes datos que un sistema podría obtener debido al hecho de que un gran número de sensores podría producir una cantidad ingente de datos. Por eso, en [153], el autor propuso una arquitectura para solucionar este problema. Sin embargo, en este otro caso, el autor de [152] habló sobre datos espaciales, los cuáles están compuestos de la localización y los objetos identificados como carreteras, países u océanos, entre otros ejemplos, además de hablar sobre cómo estos podrían ayudar a obtener una Tierra más inteligente usando Minería de datos. De esta manera, se consigue obtener más información acerca de cada localización, exactamente que tipo de objeto hay en una localización determinada.

Otros usos de ***Smart Earth*** podrían ser los usos de los ejemplo de ***Digital Earth*** que presentó Al Gore en [151]. Uno de estos fue la gestión virtual de la diplomacia, como fue en el caso de Sarajevo, donde simularon un tour aéreo para mostrar las montañas impracticables en las fronteras propuestas. Otro caso puede ser la lucha contra el crimen, donde en este caso, la ciudad de Salinas, en California, ha reducido la delincuencia juvenil usando Sistemas de Información Geográfica (*Geographic Information System* [GIS]) para detectar patrones de criminalidad y actividad de pandillas, y así, redistribuir a la policía. También el preservar la biodiversidad es otro éxito importante utilizando datos sobre la tierra y el agua para modelar el impacto en la biodiversidad de las diferentes áreas para proteger las especies, para predecir el cambio climático o incrementar la productividad agrícola.

[8] Glosario: **Metadatos**

La naturaleza es lo que nos proporciona la vida a nosotros y a al ecosistema. Por ello, es una parte fundamental de nuestras vidas. Ejemplo de esto es la conexión de sensores para el monitoreo ambiental**. Smart Earth** podría ser crucial para la prevención o control de cierto tipo de circunstancias en las que podría jugar un papel crucial [11]. Por ejemplo, en el desastre del *Deepwater Horizon* en 2010 en el Golfo de México se podría haber utilizado IoT para ayudar a controlar el vertido de petróleo, analizar el grado de salinidad del agua y comprobar el estado de las diferentes partes de la plataforma petrolífera para evitar así problemas mayores [11]. Incluso, se podría utilizar este tipo de estructuras con el carácter de identificar catástrofes antes de que ocurran y así ayudar a prevenir ciertos tipos de desastres de este tipo mediante la monitorización de diferentes estructuras y condiciones, como pueden ser tifones, tsunamis, estructuras en mal estado o monitorizar el medio ambiente para tratar de proteger la fauna y flora o ver la evolución de terrenos peligrosos, como podría ser cerca de catástrofes nucleares como fueron Chernóbil y Fukushima.

Como mostraron estas investigaciones en ***Smart Earth***, por un lado, buscan la recopilación de datos para prever posibles accidentes o ayudar en caso de que uno de ellos ocurra. Por otro lado, se podría decir que tratan de mejorar nuestra comunicación con la Tierra para así permitir identificar los problemas o las consecuencias que ocurren a partir de nuestros actos y así poder obtener soluciones más rápidas para su cuidado.

4. OBJETOS INTELIGENTES, SENSORES Y ACTUADORES

«Deus ex machina»
Menandro

«Beethoven era un buen compositor porque utilizaba ideas nuevas en combinación con ideas antiguas. Nadie, ni siquiera Beethoven podría inventar la música desde cero. Es igual con la informática»
Richard Stallman

Los ***Smart Objects*** e **Internet de las Cosas** son dos ideas que describen el futuro y que van unidas entre sí, que se complementan. Así, la interconexión entre objetos puede hacerlos inteligentes o bien expandir su inteligencia hasta límites insospechados, pudiendo llegar a crear una red que conecte todos los objetos alrededor del mundo. No obstante, para interconectarlos hay que utilizar una red que soporte la heterogeneidad y la ubicuidad de estos, una red donde exista más tráfico entre objetos que entre humanos, pero soporte ambos, una red **IoT**. Por este motivo están tan ligados ambos conceptos.

Ciudades, casas, coches, máquinas o cualquier otro objeto que pueda sentir, responder, trabajar o hacer más fácil la vida de su dueño. Esto es parte del futuro, un futuro inmediato. Sin embargo, primero hay que resolver una serie de problemas. Entre ellos, el más importante, el de la heterogeneidad de los objetos.

Este capítulo presentará el marco teórico y el trabajo relacionado de los *Smart Objects*. Explicará que son, haciendo hincapié en las diferencias con los objetos no inteligentes y expondrá una de las clasificaciones existentes de los objetos, que se corresponde como la que se tomó como válida entre las existentes para la realización de esta tesis. Finalmente, se presentará el trabajo relacionado existente.

4.1 *¿QUÉ ES UN OBJETO?*

A lo largo de esta tesis y de las investigaciones derivadas de esta tesis, se habla de «objetos» en general, al igual que hacen muchos artículos científicos. ¿Por qué? La razón es sencilla. Con la palabra «objetos» se quiere hacer referencia a cualquier dispositivo u objeto, sea inteligente o no. Es decir, si se dice interconexión entre objetos, se hace referencia a que la interconexión puede ser entre *Smart Objects*, entre objetos sin inteligencia o entre un *Smart Object* con un objeto sin inteligencia. No obstante, hay muchos problemas en la literatura y en la propia comprensión de los humanos cuando se dice la palabra «**objeto**», y a veces «**cosa**», pues algunos autores las usan indistintamente, pues es muy ambigua, tanto por cómo se usa comúnmente como por su uso en los artículos. Lo mismo sucede en inglés con «***object***» y «***thing***». Por esta razón, en esta primera sección, se introducirá que significados tienen estas palabras y que significa en el universo de discurso de esta tesis la palabra **objeto**.

«Objeto» según la Real Academia Española (RAE) [154]

1. Cosa.

«Cosa» según la Real Academia Española [154]

1. f. Lo que tiene entidad, ya sea corporal o espiritual, natural o artificial, concreta, abstracta o virtual.

2. f. Objeto inanimado, por oposición a ser viviente.

«Objeto» según *WordReference* [155] - Español

1. m. Lo que posee carácter material e inanimado.

2. filos. Todo lo que puede ser conocido o sentido por el sujeto, incluso él mismo.

«Cosa» según *WordReference* [155] - Español

1. f. Todo lo que existe, ya sea real o irreal, concreto o abstracto.

2. Ser inanimado, en contraposición con los seres animados.

«Objeto» según Google [156]

1. Cosa material inanimada, generalmente de tamaño pequeño o mediano, que puede ser percibida por los sentidos.

«Cosa» según Google [156]

1. Palabra con que se designa todo aquello que existe o tiene entidad ya sea material o inmaterial, real o imaginario, concreto o abstracto (objetos, seres vivos, pensamientos, sensaciones, emociones, acciones, sucesos, etc.) y que puede ser concebido como una unidad independiente de otra; a menudo se usa en sustitución de una palabra que no se quiere decir o no se conoce.

2. Palabra con que se designa todo aquello que es concreto y real, por oposición a lo abstracto e ideal.

«Object» según *WordReference* [155] - Inglés

1. Anything that can be seen or touched and is for the most part stable or lasting in form, and is usually not alive.

2. A thing, person, or matter to which thought or action is directed; the cause of such thought or action.

«Thing» según *WordReference* [155] - Inglés

1. An object, usually not a person or animal.

«Object» según Oxford [157]

1. A material thing that can be seen and touched.

«Thing» según Oxford [157]

1. An object that one need not, cannot, or does not wish to give a specific name.

«Object» según Cambridge [158]

1. A thing that can be seen or felt.

«Thing» según Cambridge [158]

1. An object; something that is not living.

Como se puede ver, la definición depende mucho de donde se consulte, y, aun así, en estos casos, puede ser bastante ambigua. Las definiciones que mejor se adaptan a como se utiliza la palabra «***object***» en nuestro universo de discurso y en los artículos de la literatura son la primera definición de *WordReference* en inglés

y las definiciones de Oxford y Cambridge. Hay que tener en cuenta que los artículos debían de ser enviados a revistas en idioma inglés, por lo que esta definición toma más peso sobre las otras. Por eso, en nuestro universo de discurso, que está enmarcado en **Internet de las Cosas**, no se utiliza la palabra «**cosa**» ni «***thing***» y se toma la definición de «**objeto**» y «***object***» como:

«Objeto» en el marco de Internet de las Cosas

Cualquier dispositivo electrónico que pueda ser conectado a Internet y pueda, bien recoger datos, como puede ser un sensor, o bien ejecutar una acción que pueda ser realizada por un objeto, comúnmente llamado actuador.

En los siguientes apartados se entrará en detalle para explicar la diferencia entre objetos inteligentes y no inteligentes.

4.2 *OBJETOS NO INTELIGENTES*

En el apartado anterior se explicó que es un **objeto**. Se dijo que este término engloba tanto los objetos inteligentes, conocidos como ***Smart Objects***, como los objetos sin inteligencia. Por esta razón, se hace indispensable saber diferenciar entre los diferentes tipos de objetos existentes y la manera en que estos pueden interactuar con nosotros. Los objetos del primer grupo son aquellos conocidos como ***Smart Object***, sobre los que se profundiza en la siguiente sección. Esta sección está dedicada a los objetos del segundo grupo, los objetos no inteligentes, los cuales a su vez se dividen en dos grupos, **sensores** y **actuadores**.

Los **sensores** son dispositivos formados por células sensibles [155], que nos permiten capturar magnitudes físicas, como pueden ser la variación de luz con un fotorresistor, la temperatura con un termistor, la detección de llamas, de sonido, de movimiento o cualquier otra alteración del entorno [154]–[156]. Es decir, son elementos físicos específicos que nos permiten medir una determinada magnitud física o detectar algo del entorno que rodea a este elemento.

En cambio, los **actuadores**, en nuestro universo de discurso, son **actuadores mecánicos** que permiten realizar una acción sobre sí mismo u otro dispositivo, o bien aquellas **acciones** que permita realizar un objeto determinado. Es decir, los **actuadores** se pueden dividir en dos grupos: dispositivos mecánicos y acciones. Posibles **actuadores mecánicos** son los motores, servomotores o bombas. Mientras que posibles **acciones** serían las que permiten realizar ciertos objetos, como puede ser en un smartphone el vibrar, enviar un mensaje o encender la luz de la cámara, o bien un Diodo Emisor de Luz (LED) que peude encenderse y apagarse o la acción de moverse perteneciente a un robot, entre otras muchas acciones de las que estos puedan disponer.

Así, según estas definiciones, se pueden encontrar que ciertos objetos puedan tener sensores y actuadores. Este es el caso de ciertos objetos inteligentes, como son los smartphones. Estos dispositivos están compuestos, entre otras muchas cosas, de sensores y actuadores. Otro ejemplo similar sería el de un microcontrolador, como, por ejemplo, un Arduino. Al Arduino se le pueden conectar casi cualquier dispositivo electrónico, lo que implica que se pueda crear un sistema de solo sensores, solo actuadores o una combinación de ambos. Es decir, los ***Smart Objects*** están compuestos de objetos no inteligentes.

La Ilustración 11 muestra un esquema que explica de que se componen los «**objetos**» en nuestro universo de discurso de IoT. Este diagrama sirve para entender mejor las diferencias entre objetos no inteligentes y los ***Smart Objects***. Como se puede observar, los objetos no inteligentes pueden ser sensores o actuadores, que se dividen a su vez en **mecánicos** y **acciones**. Además, esta ilustración tiene ejemplos de que tipos de objetos se corresponden a cada grupo.

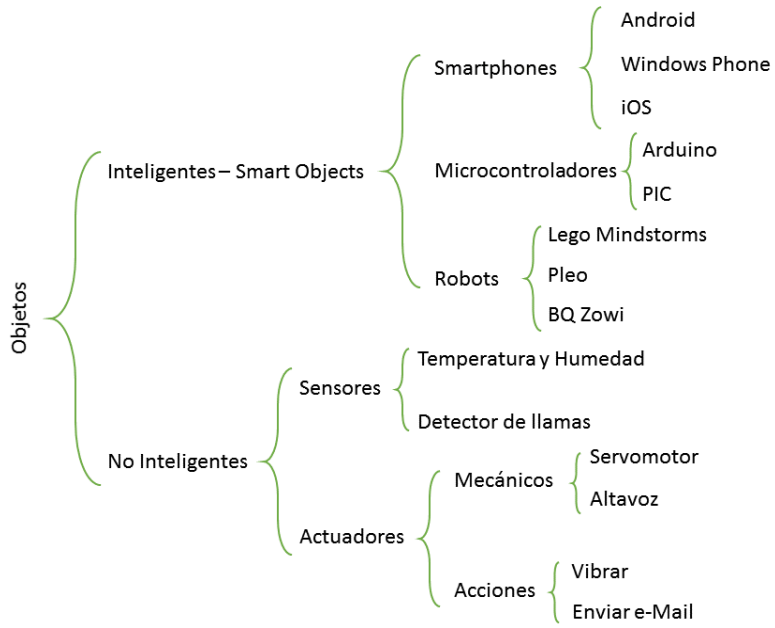


Ilustración 11 Esquema de la composición de la palabra «Objetos» con ejemplos

4.3 *SMART OBJECTS*

Como lleva ocurriendo en este capítulo, la definición de ***Smart Object*** es también dependiente del autor. No obstante, hay algunos autores con definiciones o partes de estas muy similares de las cuáles se puede sacar una premisa. En el marco de esta tesis y por cómo usamos los ***Smart Objects*** y tras revisar mucha y muy diferente literatura, se optó por utilizar la siguiente premisa obtenida de la definición de los siguientes autores [7], [159]–[162].

Definición de «*Smart Object*» en el marco de esta tesis

Un *Smart Object*, también conocido como *Intelligent Product*, es un elemento físico, identificable a lo largo de su vida útil, que interactúa con el entorno y otros objetos, y que puede actuar de manera inteligente según unas determinadas situaciones, mediante una conducta autónoma. Además, los *Smart Objects* poseen un sistema informático incrustado y frecuentemente poseen sensores o actuadores [7]. Esto les permite comunicarse con otros objetos, ser capaces de procesar su información, obtener datos del entorno o de realizar un evento.

También existen otras definiciones que difieren un poco de esta, que esta sacada de los anteriores autores. Por ejemplo, otro caso muy diferente es la definición de [163], quien solo considera como ***Intelligent Products*** aquellos objetos que están en constante monitorización, reaccionan y se adaptan con el medioambiente, tienen un rendimiento óptimo y mantienen una comunicación activa.

Nos encontramos rodeados de ejemplos de ***Smart Objects*** en nuestra vida diaria, entre nuestros objetos cotidianos. Por ejemplo, estos son los smartphones, las tablets, las Smart TVs, los microcontroladores como el Arduino [7], [53]–[55], incluso algunas cafeteras y algunos coches son objetos inteligentes, así como cualquier otro objeto que tenga conectividad a la red [9] y sea capaz, como mínimo, de gestionar información [26].

Como se puede observar, los ***Smart Objects*** pueden ser muy diferente entre ellos, pues un smartphone no tiene apenas nada que ver con un microcontrolador o con un microordenador. Lo único en común son algunos componentes electrónicos. Internamente, cada uno tiene sus propios sensores y actuadores, así como su propia inteligencia, si es que disponen de ella y su propio sistema operativo, si es que tiene.

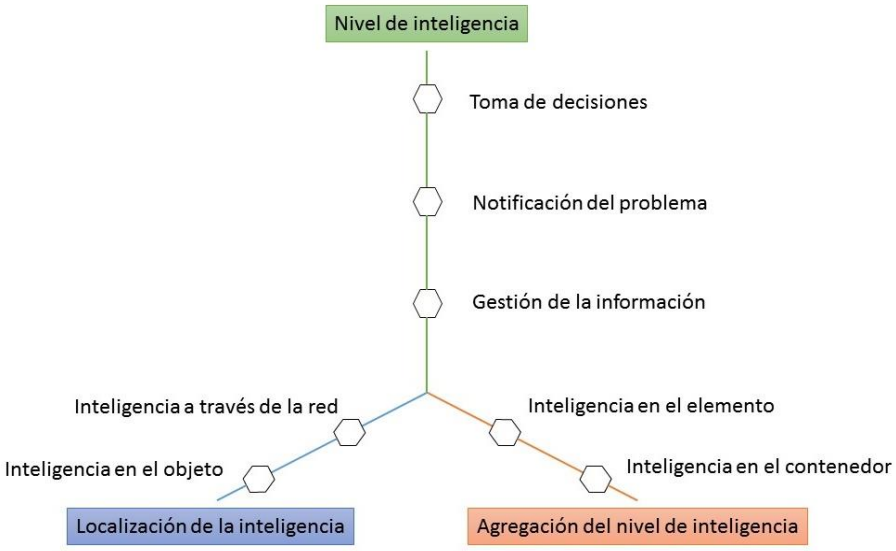


Ilustración 12 Clasificación de los niveles de inteligencia según Meyer

Los ***Smart Objects***, según [26], se pueden clasificar en base a tres dimensiones, como puede observarse en la Ilustración 12. Esta clasificación sirve para diferenciar la diferente información que nos puede dar un ***Smart Object*** acerca de su arquitectura. Cada dimensión se corresponde con una cualidad de la inteligencia. De esta manera, por medio de las tres dimensiones de un objeto, se puede determinar la inteligencia que tiene un objeto y el tipo de ***Smart Object*** que es para así compararlo con otros. Las dimensiones son las siguientes:

- La primera dimensión es el **nivel de inteligencia**. En esta se describe la capacidad de inteligencia del objeto y cuán listo puede ser. Consta de tres niveles:
    - La **gestión de la información** es la capacidad para manejar la información que recoge a través de sensores, lectores u otras técnicas. Este es el nivel más básico que debe de tener

un objeto para considerarse ***Smart Object***, es decir, ser capaz de manejar la información que recibe. Sin este nivel, simplemente sería un **objeto no inteligente**.

- La **notificación del problema** es la posibilidad de que un **objeto** sea capaz de notificar a su propietario cuando ocurre un determinado problema o evento en el propio ***Smart Object***, como puede ser la detección de bajada de temperatura. En este nivel, los **objetos** aún no poseen libre albedrío.
- La **toma de decisiones** es el nivel más inteligente que puede poseer un **objeto**. Un **objeto** posee este nivel cuando, además de los dos anteriores niveles, también posee la capacidad propia sobre la toma de decisiones sin intervención de un control externo, es decir, tiene libre albedrío.

- La segunda dimensión es la **localización de la inteligencia**, que consta de dos niveles:
  - El primero es el nivel de la **inteligencia a través de la red**, la cual consiste en que la inteligencia del objeto depende totalmente de un agente externo al propio objeto, pues este no tiene ninguna incorporada. Este agente puede ser una red a la que se encuentra conectado, comúnmente llamadas plataformas portal (*Portal platform*) [164], un servidor que tenga los agentes inteligentes corriendo u otro objeto que se dedique a tomar decisiones o contenga la inteligencia global.
  - El otro nivel es la **inteligencia en el objeto**. Los objetos que tienen este nivel de localización de la inteligencia computan todo por sí mismo, es decir, toda la inteligencia es llevada por ellos y no necesitan de agentes externos para ser inteligentes. Las plataformas que tienen **objetos** que poseen este nivel suelen llamarse plataformas integradas (*Embedded platforms*) [164].
  - Un tercer nivel, que no incluye [26] en su clasificación, pero del que si habla y muestra en un gráfico de ejemplo, es el nivel en el que el objeto tienes ambas inteligencias, es decir, tiene inteligencia propia y es capaz de usar también la inteligencia en la red. Estas plataformas se llaman plataformas de sustitución (*Surrogated platforms*) [164].

- La última dimensión es la **agregación del nivel de inteligencia** y se compone de dos categorías. Esta dimensión sirve para describir aquellos **objetos** que se componen de partes, de manera que, se pueda detectar si cada parte es individual o si el **objeto** es un todo indivisible. Por ejemplo, a una Raspberry Pi se le puede conectar un Arduino y a ambos se le pueden conectar sensores, actuadores y otros microcontroladores. Los objetos no inteligentes, es decir, actuadores y sensores, no tienen inteligencia por sí mismos, pero los microcontroladores, como el Arduino, sí. Así, si este es separado de la Raspberry Pi, ambos pueden seguir funcionando independientemente. Las categorías son:
  - La **inteligencia en el elemento**. A esta primera categoría pertenecen aquellos **objetos** que pueden manejar información, notificaciones y/o decisiones y si contienen otros componentes, estos no pueden ser distinguidos como objetos individuales. Un ejemplo de

este tipo de elementos son los smartphones, que contienen sensores y estos no pueden ser separados, ya que van integrados.

- La **inteligencia en el contenedor**. En esta categoría los objetos deben de poder manejar información, notificaciones y/o decisiones, pero, además, son conscientes de los componentes de los que están formados, permitiendo funcionar como un proxy entre ellos y la red o la inteligencia. Además, estos **objetos**, son capaces de seguir funcionando como contenedor u objeto inteligente a pesar de que se les desensamble alguna parte de él. Por ejemplo, a este grupo pertenecería una placa Arduino con como mínimo dos sensores. Si a esta se le quita un sensor, puede seguir funcionando como contenedor. Otro ejemplo es el que proponen en el artículo [26], que consiste en una estantería inteligente que notifica cuando se queda sin stock de algún producto.

Bajo el marco de esta tesis, lo más relevante en esta clasificación y en los ***Smart Objects***, es en la **localización de la inteligencia**, los objetos que tienen la **inteligencia a través de la red**, pues con este tipo de objetos, en combinación con **Internet de las Cosas**, permiten conseguir que la red pase de solo transportar datos, a dotarla de inteligencia y acciones según los datos que recogen estos objetos y los servicios que estos permiten realizar. Esto implica que los objetos que estén conectados a ella puedan tener inteligencia, o incluso, ser más inteligentes.

4.4 *CAMPOS DE USO*

Los ***Smart Objects*** están presentes desde hace mucho en nuestra vida diaria. Se mostraron algunos ejemplos de ellos en las diferentes aplicaciones existentes de **Internet de las Cosas**. No obstante, hay ejemplos mucho más precisos del uso de solo ***Smart Objects*** sin el uso de **IoT**.

Los podemos encontrar en el **ámbito comercial** en diferentes sistemas que ayuden a controlar la manufacturación como ocurre en [159]. Por otro lado, en [26], [160], utilizan ***Smart Objects*** para mejorar la distribución y gestión de productos en las cadenas de suministro, de forma que los tienen localizados durante todo el proceso de su ciclo de vida. En el primer artículo describen en que partes se podrían utilizar diferentes elementos como son los lectores, para averiguar el estado del producto, monitorizarlos o acceder a su historial. Mientras, en el segundo artículo, uno de los ejemplos es el de estanterías inteligentes que notifiquen cuando esta se ha quedado sin stock de un producto. Este tipo de aplicaciones son muy útiles de cara a las empresas, pues, les otorga ventajas para mejorar y evitar problemas de falta de stock a lo largo de toda la cadena de vida de un producto. Siguiendo con otro posible uso de los objetos inteligentes en este ámbito, es el propuesto por [50], el cual consiste en controlar el uso de objetos alquilados para cobrar la cantidad de dinero adecuada según su utilización y, si hiciera falta, añadir una sanción en el caso de que se detectase un uso incorrecto del objeto por parte del usuario. Este sistema ayuda a cliente y empresa pues, en el primer caso, se le cobra exactamente por el uso que le da y en el caso de la empresa sirve para detectar un mal uso del objeto y compensarlo.

Otro tipo de investigaciones relacionadas con los ***Smart Objects*** son las que tienen que ver con la seguridad en el trabajo. En [50], los autores propusieron un sistema para avisar a los trabajadores acerca del almacenaje incorrecto e inseguro de materiales químicos. Este sistema puede ser muy útil, ya que permitiría controlar el almacenaje de posibles sustancias peligrosas y podría evitar muchos desastres y problemas.

Referente al ámbito de la salud, también se puede encontrar literatura. Uno de estos artículos es [60], que propuso un sistema de monitorización de pacientes con problemas. Gracias a sistemas como este se podrían llegar a salvar muchas vidas humanas, pues permitiría, por ejemplo, conectar el marcapasos con un centro de vigilancia y así detectar inmediatamente un posible ataque al corazón o un fallo en el marcapasos.

5. INGENIERÍA DIRIGIDA POR MODELOS

«Usar modelos para diseñar sistemas complejos es de rigor en las disciplinas tradicionales de ingeniería. Nadie imaginaría construir un edificio tan complejo como un puente o un automóvil sin construir primero una variedad de modelos especializados del sistema. Los modelos nos ayudan a entender un problema complejo y sus soluciones potenciales a través de la abstracción»
Bran Selic [165]

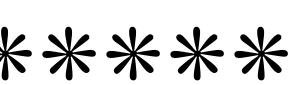
«Toda la historia de la ingeniería del software es el incremento de los niveles de abstracción»
Grady Booch en la charla «The Limits of Software» y citado por Gerry Boyd (2003) en «Executable UML: Diagrams for the Future»

«Tan pronto como comenzamos a programar encontramos, para nuestra sorpresa, que crear programas correctos no era tan fácil como habíamos pensado. Tuvo que descubrirse la depuración. Puedo recordar el instante exacto en el que me di cuenta que una gran parte de mi vida desde entonces iba a pasarla encontrando errores en mis propios programas»
Maurice Wilkes descubre la depuración, 1949

No existe ninguna bala de plata capaz de matar nuestros miedos, ni magia que los haga desaparecer, ni una tecnología para solucionar todos los problemas tecnológicos, o especificando, solucionar los problemas de la **Crisis del Software** descritos por Edsger W. Dijkstra y la OTAN. No obstante, sí que surgieron nuevos tipos de desarrollo, que, claramente, no son la panacea pero que si ayudaron a mitigar los efectos de la complejidad accidental y esencial.

La **Ingeniería Dirigida por Modelos** o **MDE** surgió para solucionar los problemas pertenecientes al desarrollo software. Estos problemas son la baja calidad del software desarrollado, el incumplimiento del presupuesto y la planificación, y un aumento en el coste de mantenimiento del proyecto. Problemas que surgían en los años 60 y que fueron documentados por Edsger W. Dijkstra, y que todavía se siguen dando hoy en día y que son una de las causas de que los proyectos de informática no terminen de buena manera. La solución aparte de estos problemas se puede obtener mediante la automatización o semiautomatización de procesos, algo en lo que es muy popular **MDE**.

Por estos motivos, este capítulo presentará la historia, el marco teórico y el trabajo relacionado acerca de **MDE**, incluyendo una breve mención al estándar del Object Management Group® (OMG), la Arquitectura Dirigida por Modelos (MDA). Además, introducirá los conceptos de ambos y de todo lo relacionado con ellos y su historia.



5.1 *¿QUÉ ES UN MODELO?*

En esta tesis se habla de **Ingeniería Dirigida por Modelos** (MDE) y se hace referencia a que se trabaja con **modelos**. No obstante, ¿qué es un **modelo**?

«Modelo» según la Real Academia Española (RAE) [154]

1. m. Arquetipo o punto de referencia para imitarlo o reproducirlo.

2. m. En las obras de ingenio y en las acciones morales, ejemplar que por su perfección se debe seguir e imitar.

3. m. Representación en pequeño de alguna cosa.

4. m. Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, como la evolución económica de un país, que se elabora para facilitar su comprensión y el estudio de su comportamiento.

8. m. Esc. Figura de barro, yeso o cera, que se ha de reproducir en madera, mármol o metal.

11. m. y f. Persona u objeto que copia el artista.

«Modelo» según *WordReference* [155] - Español

1. m. Arquetipo digno de ser imitado que se toma como pauta a seguir.

3. Representación a escala reducida de alguna cosa.

4. Objeto, aparato o construcción realizada conforme a un mismo diseño.

5. esc. Figura de barro, yeso o cera que se reproduce en un material más sólido.

7. En arte, persona u objeto que copia el artista.

«Modelo» según Google [156]

1. Cosa que sirve como pauta para ser imitada, reproducida o copiada.

2. Persona que merece ser imitada por sus buenas cualidades.

3. Producto industrial que se fabrica en serie y responde a unas características de la serie.

4. Representación de un objeto a pequeña escala.

6. Representación de una categoría o tipo de cosas definidas por ciertas características.

7. Esquema teórico que representa una realidad compleja o un proceso complicado y que sirve para facilitar su comprensión.

«Model» según *WordReference* [155] - Inglés

1. A standard or example of something that can be used for imitation or comparison.

2. A copy, usually in miniature, to show appearance of something.

3. Fine Arta person or thing that serves as a subject for an artist, etc.

4. A style of a particular product, as a car, machine, etc.

5. A simplified representation of a system or of some event or action, as in the sciences, proposed by scientists to explain or describe the event or action.

«Model» según Oxford [157]

1. A three-dimensional representation of a person or thing or of a proposed structure, typically on a smaller scale than the original.

2. (In sculpture) a figure or object made in clay or wax, to be reproduced in another more durable material.

3. A thing used as an example to follow or imitate.

«Model» según Cambridge [158]

1. A standard or example for imitation or comparison.
2. A representation, generally in miniature, to show the construction or appearance of something.
3. An image in clay, wax, or the like, to be reproduced in more durable material.
4. A person or thing that serves as a subject for an artist, sculptor, writer, etc.
5. A person whose profession is posing for artists or photographers.
6. A person employed to wear clothing or pose with a product for purposes of display and advertising.
7. A style or design of a particular product.

Como se ve, a diferencia de lo que pasaba con la palabra **objeto**, la palabra «**modelo**» y «**model**» tienen casi idénticas acepciones tanto en español como en inglés. Ejemplos de estas acepciones pueden ser vistos cuando se construye un edificio, en los planes previos realizados para diseñarlo correctamente o posteriormente en la miniatura a escala del edificio donde se muestra el resultado final. El no diseñar previamente estos modelos especializados para ver el resultado sería inimaginable, ya que su uso es de rigor para diseñar sistemas complejos en la ingeniería tradicional y ayudarnos a entenderlos a través de la abstracción [165]. De esta manera, el uso de **modelos** permite ayudar a entender los aspectos que interesan de algo complejo antes de gastar dinero y esfuerzo en su construcción, ya sea por la cantidad de recursos que necesitará o para saber si será posible llevarlo a cabo [166]. Por ello, hay que mirar como otras disciplinas utilizaron los modelos [167].

Los modelos ya eran usados en la antigua Grecia y posteriormente en Roma, donde Vitruvius discutió ampliamente sobre su uso para diseñar edificios y maquinaria [168]. Un caso en el que se pusieron en práctica los modelos fue el de la Catedral Santa María del Fiore en el año 1418 [169], la cual se muestra en la Ilustración 13. Se requería terminar de construir dicha catedral, que llevaba ya más de 100 años en construcción, exactamente le faltaba la cúpula, y para ello se creó un concurso. Está cúpula era difícil de construir, pues era muy grande y requería que no tuviera arbotantes que la sujetasen, pues la cúpula debía ser erigida en el aire para evitar que estos soportes laterales le restasen belleza. En este concurso se les pidió una maqueta a escala para así comunicar el diseño que proponían y ver si era factible. La solución ganadora la propuso Filippo Brunelleschi, orfebre y relojero, quien primero fue considerado un loco y al final un genio. Su maqueta fue realizada en madera, ladrillo y mortero y era tan grande que permitió al jurado inspeccionar la cúpula por dentro, lo que les permitió ver que la cúpula sería construida sin utilizar costosos andamios de madera que soportasen el techo que tenía una inclinación de treinta grados. Esta propuesta había sido una gran incertidumbre, pues nunca se había realizado algo similar hasta la época. No obstante, la propuesta de Filippo Brunelleschi consiguió revolucionar la arquitectura posteriormente. A día de hoy, esta cúpula es la más grande del mundo con 43 metros de diámetro.

Ilustración 13 Cúpula de la Catedral Santa María del Fiore[9]

Como se ve, siguiendo este tipo de desarrollo en el contexto de la construcción de edificios, vehículos o piezas, se consigue plasmar en los planos el cómo se quiere construir el objeto, incluyendo todas sus vistas y medidas. Esto sirve para indicar que se quiere y como debe hacerse, además de poder ver en los planos y los modelos posibles defectos o mejoras que se pueden corregir de una forma «rápida, fácil y barata», pues los planos simplemente son papel, y no son un edificio o un vehículo entero que si sale mal hay que tirar, con todo el gasto que ello conllevaría. Normalmente, la primera fase suele ser el dibujo en plano y tras esta, muchas veces se realiza un modelo a escala en un material más barato que el final, como en el caso de Filippo Brunelleschi, que fue de madera, ladrillo y mortero a escala. Claramente, en la actualidad se suelen utilizar compuestos plásticos, e incluso los modelos se imprimen con impresoras 3D, como ocurre con los edificios. No obstante, algunas casas de coches realizan algunos modelos en madera. En este modelo a escala se puede observar, de mejor manera que sobre los planos, como será el resultado final y si cumple los requisitos o si tiene algún defecto o se puede mejorar, como ocurrió con la Catedral de Santa María del Fiore. En caso contrario, hay que modificar los planos y la miniatura o bien crearlo todo de cero, algo que sigue siendo mucho más «rápido, fácil y barato» que hacerlo sobre un edificio o vehículo finalizado. Una vez los planos y el modelo a escala cumplen los requisitos sin errores, esto se traslada en la creación final.

Hay que tener en cuenta también que, una misma construcción puede requerir diferentes puntos de vista. Por ejemplo, continuando el ejemplo del edificio, podemos tener un modelo con la vista exterior del edificio, para ver cómo quedará externamente. No obstante, también se necesitará un modelo de las vistas interiores para poder observar cómo serán las habitaciones y si hay espacio suficiente para todo el material que se desee

[9] https://commons.wikimedia.org/wiki/File:Santa_Maria_del_Fiore.jpg

utilizar en su interior. Luego, para un mismo sistema, un mismo edificio, existen dos puntos de vista diferentes, el externo y el interno. Además, en el punto de vista interno, podemos tener un modelo para cada planta, pues estas pueden ser totalmente diferentes. Así pues, nos encontramos con que tenemos, además, diferentes modelos para representar diferentes vistas de un mismo nivel, el interno, exactamente uno por planta. Esta diferencia entre una vista externa y otra interna también se debe a que no es la misma gente la que crea la fachada del edificio y el interior, es decir, tenemos diferentes trabajadores para cada parte, y cada trabajador debe de conocer que debe de hacer en su parte.

Así, mediante el uso de modelos, se consigue hacer una implementación más barata para ver si el resultado final puede ser satisfactorio, sin tener que sufrir el riesgo de perder dinero al diseñar un producto incorrectamente, además de comprobar que todo sea posiblemente realizable [170], [171].

Los **modelos** aplicados a la informática siguen una idea y proceso parecido al de otras disciplinas. No obstante, en este caso, se trabaja con **modelos** digitalmente, es decir, en el ámbito informático para ayudar al desarrollo de software. Un **modelo** tiene la habilidad de resaltar los aspectos importantes de un sistema para obtener una abstracción que contenga únicamente la información relevante y ayude a afrontar la complejidad del software moderno [172], [173]. Además, este **modelo** siempre es tomado desde una perspectiva determinada, la cual hay que tener en cuenta [28], [29], pues el **modelo** puede cambiar según la perspectiva del objeto que se utilice. Esta perspectiva puede ser una parte de una función, la estructura y/o el comportamiento del sistema [174].

Por ello, las definiciones anteriores requieren ser adaptadas precisamente al universo de discurso de donde se utiliza. En este caso, no varía mucho, pero hay que asignarle una serie de matices, pues estamos hablando de modelos en el universo de discurso del software:

Definición de «Modelo Software»

Descripción en un lenguaje de modelado o lenguaje natural de una vista determinada de un objeto o sistema, perteneciente al mundo virtual o real, que abstrae sus conceptos relevantes y que sirve como punto de referencia para reproducirlo.

Al igual que ocurre en la fabricación y con diversas acepciones vistas anteriormente, parece obvio que se utilicen modelos en el desarrollo del software para así beneficiarnos de sus ventajas [165]. El uso de modelos dio lugar a la **Ingeniería Dirigida por Modelos** (MDE), la cual es considerada otro nivel de compilación [28]. De esta manera, se pretende usar los diagramas para obtener una abstracción respecto a los lenguajes de programación y así utilizarlos para especificar el sistema de cara a utilizar una herramienta de modelado que los traduzca a un lenguaje de programación convencional [175]. Esto hace que los **modelos** sean cruciales en MDE y no sean algo opcional a utilizar [165], pues nos proporcionan abstracción sobre una parte del producto, desde los requisitos, hasta el código y los test [176].

Es considerado también un modelo el propio código fuente, pues tiene las características de poder ser ejecutado por una máquina, como una especificación basada en *Unified Modeling Language*® (UML) [174].

No obstante, hay que tener en cuenta que un diagrama con cajas, líneas y flechas, que carezca de información acerca de su significado y el significado de cada componente, es decir, que significa cada caja, cada línea y cada flecha, no es un **modelo**. Un modelo no debe de ser ambiguo y debe de tener un significado inequívoco enlazado a su sintaxis y semántica de modelado [174].

Los **modelos** pueden ser tanto horizontales, es decir, que describan diferentes sistemas, como verticales, los cuáles describen diferentes niveles de abstracción. Esto ayuda a gestionar mejor un sistema complejo capturando en los modelos diferentes aspectos de la solución, siendo la más baja la implementación tecnológica [177].

El uso de **modelos**, es casi tan viejo como la propia ingeniería, pues los usaban para verificar su intención de diseño y como medio de transmitir al resto de personas su idea [166], [170]. No obstante, la tendencia de utilizar **modelos software** se podría datar en el año 1947, cuando en la Universidad de Cambridge, Wheeler y Wilkes crearon las primeras librerías prescritas para la computadora de dicha universidad, llamada *Electronic Delay Storage Automatic Calculator* (EDSAC), y así obtener una abstracción para poder utilizar subrutinas sin tener que saber exactamente como estaban implementadas [28].

Los modelos a su vez se componen de **artefactos** y **relaciones** [176]. El modelo es todo el conjunto que se desea representar. La Ilustración 14 muestra los diferentes componentes de un modelo y de los que se habla en este párrafo. Este modelo contiene los **artefactos**, que son los diferentes elementos que componen el modelo. Por ejemplo, si se utiliza UML, cada artefacto sería una clase de UML. En cambio, las **relaciones** son las encargadas de mapear los artefactos dentro de un modelo a los artefactos de otro modelo. Es decir, las **relaciones** son las encargadas de decir cómo se hace la transformación entre modelos. Las relaciones pueden incluir anotaciones en sus bordes para mejorar el entendimiento del modelo, además de ser de diferente tipo, como instancias o especializaciones, entre otras. Esta **relación** no hay que confundirla con la relación existente entre clases, que, como se puede ver en la Ilustración 14, se llamó «**Artefacto** (**Relación**)». Estas relaciones pueden tener **anotaciones**.

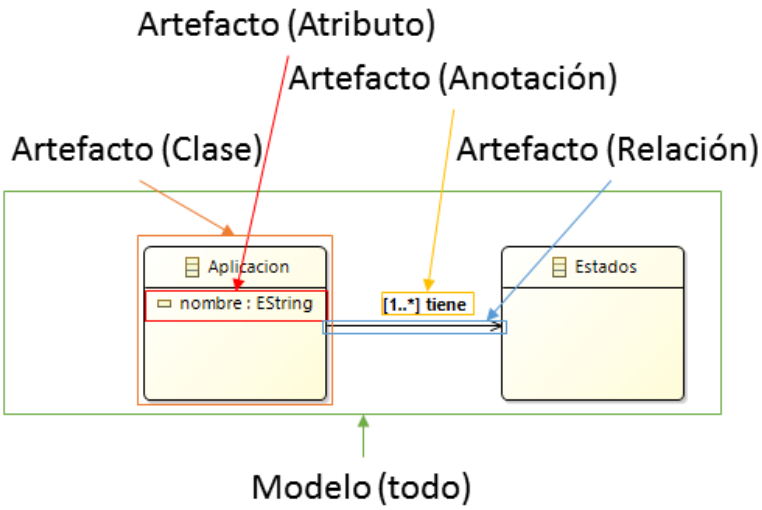


Ilustración 14 Ejemplo de modelo realizado con Ecore para explicar sus partes

5.2 *JSD, KBSA, MB, MD, MDD, MDSD, MBE, MDA, MDE, …: ¿QUÉ ES QUÉ? UN POCO DE HISTORIA* [10]

En la literatura actual nos podemos encontrar con muchos términos diferentes para expresar cosas similares o muy parecidas, como es el caso de las diferentes técnicas que se basan en el uso de **modelos**. A veces, bien porque dos investigadores diferentes investigaron en el mismo campo, pero usaron diferente nombre, o por problemas de registro, pues algunos fueron registrados y se utilizó otro nombre para poder utilizarlo libremente. Esto provoca un lío en la terminología y es lo que sucede con los términos MDE, MDD, MDSD, MBE, MBSE y MDA, entre otros [178]. También, a lo largo de la historia, surgieron diferentes vistas o enfoques implementando estas ideas, este es el caso de M.E.R.O.DE. o MDA. Otras veces, en la búsqueda de artículos, se puede ver que se encuentra el término en un artículo de modelos, pero este no se refiere al desarrollo, sino al trabajo, es decir, una persona es experta en desarrollo de software dirigido por modelos, como ocurre con el autor David W. Embley en [179], lo que hace que no se refiera a que se utiliza **Ingeniería Dirigida por Modelos**.

A todo esto, hay que sumarle que las herramientas CASE hacían ya un trabajo parecido, pues, en ellas se definía lo que se quería desde el diseñó para así obtener el código final, o bien, desde el código final para obtener el diseño [171], [180], teniendo toda la transformación necesaria oculta y sin hacer referencia a los modelos o metamodelos. O bien, se hacían modelos sin utilizar ninguna herramienta estándar utilizada hoy en día como son UML y Ecore [181], pues de aquella todavía no existían. Esto hace que sea difícil encontrar los primeros artículos en donde aparecieron por primera vez los términos. Además de esta variedad de términos, varios del tipo «Model-Driven *» y «Model-Based *» han sido registrados por el OMG [175].

Otro problema para situar la evolución de MDE a lo largo de la historia surge en que hay que tener en cuenta que en los años 80-90 era más difícil conseguir y encontrar artículos que ahora, pues ahora existe una distribución online en Internet de ellos así como buscadores específicos y herramientas especializadas, mientras que antes utilizaban casi únicamente el formato impreso. Por ello, como se verá, existen tres vertientes, la de JSD, KBSA y la de Zachman, de los cuáles a cada uno citan otros autores por separado para atribuirles que son el primer paso o la primera arquitectura de este tipo. Luego, podemos decir, que, a pesar de la diferencia en años ambos evolucionaron paralelamente sin conocimiento del otro, siempre teniendo en cuenta las buenas intenciones investigadoras.

A veces, debido a que se trabaja con modelos más allá del software, ciertos autores introducen un matiz para especificar o aclarar su «correcto uso». Como se comentó en el apartado anterior, los **modelos** pueden ser utilizados en otros campos, como es en la ingeniería mecánica, eléctrica o física. Por ello, en algunos nombres, como es en el caso de MDSD, MBSS y MBSE, se introdujo la «S» para denotar que es una solución basada en software. Sin embargo, a veces esta «S» puede significar «Sistema (*System*)», lo que provoca más ambigüedad en los términos y a veces se utiliza el mismo término para hablar de diferentes cosas. Esto es tal

[10] Nota del autor: a pesar de que este título parezca algo caótico, se ha buscado esto mismo para así representar lo mismo que sucede con su historia, un caos.

que, si se hacen búsqueda de modelos en otras áreas además de la informática, se pueden encontrar el uso de estos en áreas de la ingeniería, pero bajo el mismo nombre, en este caso, MBE. Por eso, hay que tener cuidado a la hora de trabajar con estos términos. Cabe destacar que, en esta tesis, los términos utilizados son siempre en base al software.

Por esta razón, en esta sección se hablará de estos términos, de su historia sin entrar en mucho detalle y se intentará establecer de la forma más precisa posible la línea temporal de su aparición en la historia. En la Ilustración 15, se muestra la línea temporal de los diferentes conceptos que se tratarán en esta sección, mostrando en verde las aproximaciones realizadas y en rojo y subrayado los enfoques realizados de estas aproximaciones.

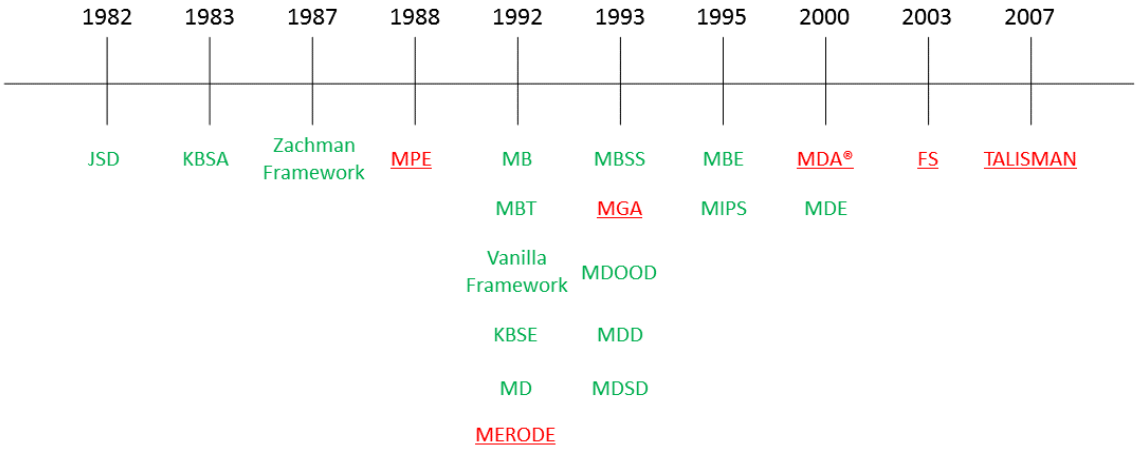


Ilustración 15 Línea temporal de las apariciones de los diferentes conceptos

En 1982, Michael A. Jackson propuso su propio sistema, el ***Jackson Systems Limited*** (**JSD**) [182]. JSD se basa en tres principios, el primero de ellos en describir y modelar el mundo real mediante la simulación de este mundo real en detalle. JSD hablaba de lo mismo que habla MDE actualmente y fue una de las primeras en hacerlo. JSD fue usado más adelante como se verá en M.E.R.O.DE. [183]. JSD es la primera vertiente.

Uno de los primeros términos de todos los existentes para referirse a la **Ingeniería Dirigida por Modelos** y que hace referencia a las metas y lo que se intenta resolver es el de ***Knowledge-Based Software Assistant*** (KBSA), publicado en el año 1983 en este reporte [184]. En este artículo, los autores presentan un paradigma[11] para desarrollar software, además de analizar los problemas en los que se basaría más adelante esta ingeniería. Unos años más adelante, se pueden encontrar autores referenciando este artículo con el fin de seguir hablando de lo que es KBSA, pero bajo otros nombres diferentes [185]. Esta es la segunda vertiente.

Otra de las referencias más antiguas que utilizan algunos autores es [186], del año 1987, perteneciente a Zachman. Algunos autores lo citan con el fin de decir que el enfoque basado en modelos está basado en la arquitectura sugerida en este artículo [173], [187], [188]. En este artículo Zachman presenta su *framework*, conocido posteriormente como ***Zachman Framework*** y habla acerca de una arquitectura con diferentes

[11] Glosario:
Paradigma

niveles, los cuales están adaptados a las diferentes personas involucradas. Ante la ausencia de ingenierías previas o artículos referenciados y en base a como lo citan, se puede considerar la base de MDE a pesar de ser un *framework*. Esta la consideramos la tercera vertiente.

Ya en el año 1988, aún sin hacer referencia directa a ningún concepto, pero haciendo uso de modelos software, Gabor Karsai publicó un *framework* llamado ***Multigraph Programming Environment*** (MPE), que permitía especificar paradigmas de modelado [189].

Buscando en la literatura, se puede ver otras aproximaciones del uso basado en modelos para dirigir el proceso software, todas ellas diferentes a las tan conocidas MDE, MDD y MDA, pero todas ellas nombradas como «***Model-Based***» (MB).

Una de las primeras apariciones bajo el uso del nombre de «**modelos**» y utilizando conceptos de MDE fue la de ***Model-Based Techniques*** (**MBT**), que data del año 1992 [190]. Aquí, los autores utilizaron modelos para el mantenimiento de sistemas de procesamiento de señales en tiempo real. En este mismo año, en 1992, David Harel publicó un artículo acerca de su ***Vanilla framework*** [191]. Este artículo explica como hicieron la aproximación «*vanilla*» de la programación mediante el uso de **modelos** para generar código automático, así como explican ciertos conceptos utilizados posteriormente por MDE. Luego, el caso de ***Vanilla Framework*** es similar al caso de ***Zachman Framework***, son *frameworks* pero que sientan las bases de lo que será conocido como MDE por medio de implementaciones. **MBT** y ***Vanilla Framework*** no fueron las únicas referencias en el uso de modelos en el año 1992, pues Michael R. Lowry publicó un artículo acerca ***Knowledge-Based Software Engineering*** (KBSE) [192], en el que se basaba en el artículo de **KBSA** comentado previamente, para así conseguir con este enfoque algo mucho más ambicioso que las herramientas CASE [193]. Además, en este mismo año también se puede encontrar alguna referencia a enfoques «***Model-Driven***» (MD) [194]. También, en el año 1992, se pueden encontrar las primeras referencias a ***Model-driven Entity-Relationship Object oriented DEvelopment*** (M.E.R.O.DE.) [195].

Otro técnica «***Model-Based***» fue ***Model-Based Software Synthesis*** (MBSS), la cual fue publicada en un artículo del año 1993 [196]. En este artículo utilizaron modelos para facilitar su trabajo por medio de la abstracción. En este mismo artículo hacen referencia también al enfoque ***Multigraph Architecture*** (MGA), no obstante, no publicaron su enfoque hasta el año 1995 [197]. En este artículo de M.E.R.O.DE. [187], describen su sistema como una implementación de ***Model-driven object-oriented development*** (MDOOD). En este mismo año también surgió lo que se conoce ahora como **Desarrollo Dirigido por Modelos**, ***Model-Driven Development*** o **MDD**. En este año es posible encontrar un artículo que utiliza MDD y que no pertenecen al OMG ni hacen referencia a este [198]. Los autores utilizaron MDD para poder construir herramientas CASE que permitiesen guardar la consistencia de las especificaciones automáticamente. Cabe destacar que se pueden encontrar otros artículos fuera del ámbito del software utilizando esta nomenclatura siendo anteriores a este artículo. Sin embargo, no fue hasta el año 2003 donde se puede encontrar una gran cantidad de artículos sobre MDD y exactamente sobre que es [165], especificando que es el término genérico para referirse a distintos paradigmas como las **Factorías de Software** o **MDA** [199]. 1993 fue un año muy importante, pues también apareció el término MDSD en este artículo [200] para hacer referencia al trabajo **M.E.R.O.DE.**, pues el artículo habla de modelos junto con programación orientada a objetos. El significado

de MDSD es el mismo que el de MDD, pero bajo el nombre de **Desarrollo Software Dirigido por Modelos**, ***Model-Driven Software Development***. Es decir, especifican que el desarrollo dirigido por modelos es para la vertiente de software. No obstante, al igual que MDE, MDSD no es un nombre registrado, lo que permite a la gente utilizarlo para hablar de ello sin tener que preocuparse de cómo lo utiliza, no como ocurre con el término MDD. Un ejemplo de ello es el que comenta Martin Fowler, quien utilizó el término MDSD para su libro por este mismo motivo en el año 2008 [175].

Posteriormente surgió el término **Ingeniería Basada en Modelos**, del inglés ***Model-Based Engineering*** (MBE), que apareció en un artículo del año 1995 [197], para explicar el funcionamiento del enfoque ***Multigraph Architecture*** (MGA) de Gabor Karsai, del que no obstante ya habían nombrado en el año 1993 en [196]. Según el artículo de Karsai, MGA fue un avance importante en el desarrollo de entornos de Síntesis de Programas Integrados en el Modelo (MIPS), que son los entornos compuestos de herramientas para desarrollar utilizando modelos. Como se ve, en el mimo artículo, Gabor Karsai hace referencia a dos posibles nombres y un enfoque.

La primera implementación estándar de MDE que surgió fue la **Arquitectura Dirigida por Modelos**, conocida en inglés como ***Model-Driven Architecture***® (**MDA**®). MDA tal vez es la vista más predominante de MDE [176]. **MDA** es una marca registrada por el grupo OMG® con el motivo de crear un estándar para el uso de modelos y de aplicaciones que se interconecten entre sí y que utilice solo tecnologías estándar. MDA surgió en el año 2000 [24] y en el año 2001 sacaron su primera versión incompleta [174]. MDA es la visión particular del OMG sobre MDE [175]. Sin embargo, hay que llevar cuidado, pues a veces, el término MDA es usado incorrectamente como sinónimo de MDD, cuando en verdad es una implementación de MDD [24]. Así, MDA fue una estrategia del OMG por crear un sistema estándar que utilizase estándares para concebir un mundo donde los modelos fuesen uno de los puntos más importantes en la producción de software [29].

Otro término que apareció en el año 2000 fue el de la **Ingeniería Dirigida por Modelos** o ***Model-Driven Engineering*** (**MDE**) [201] por la empresa SOFTEAM, el cual es verificado por el OMG en [202], donde comenta que esta empresa lleva investigando y trabajando en MDE desde el año 1991. Más adelante, en el año 2002, se encuentra un artículo de Stuart Kent [29], titulado de la misma manera que la tecnología, en el cuál comenta que MDE trata un ámbito más amplio que MDA mediante la combinación del proceso y el análisis con la arquitectura. Además, en este artículo, hace una crítica del estándar MDA y las lagunas que el estándar MDA posee. MDE es un sinónimo de MDD, pero con la diferencia de que no está registrado.

Las **Factorías de Software**, conocidas como ***Software Factories***, fueron el enfoque de Microsoft acerca de MDE, registrado en el año 2003 y presentado en el año 2004 [203], [204]. En estas, Microsoft opta por modelar utilizando lenguajes de dominio específico en vez de UML. De esta manera, consiguen mediante los DSLs abstraer diferentes sistemas de una misma taxonomía, como pueden ser para la banca, las finanzas o el e-commerce. Con ello, lo que buscaron fue industrializar el desarrollo de software basándose en los procesos de desarrollo de otras industrias ya maduras. De esta manera, las **Factorías de Software** buscaban hacer el ensamblado de aplicaciones más rentable reutilizando componentes y creando un sistema similar al de las cadenas de suministro, pero permitiendo una personalización masiva.

Más adelante, como producto de la siguiente tesis [205], surgió un nuevo enfoque de MDE, llamado TALISMAN, que se basó en MDA pero que adaptaba MDA a los doce principios del desarrollo ágil de software [206], además de resolver un problema existente en el estándar MDA: la falta de especificación de como tenían que ser los **Modelos Independientes de Cómputo** (CIM) y como se debía hacer su transformación a los **Modelos Independientes de la Plataforma** (PIM).

Como se pudo leer, muchos de estos términos se usan para describir y hablar de la misma «cosa». Este es el caso de KBSA, MPE, MB, MBT, KBSE, MD, MBSS, MDOOD, MDD, MDSD, MBE, y MDE, entre otros posibles existentes en la literatura, como es el caso de MBSE, y otras variaciones. El problema aquí reside más bien en la falta de acuerdo de qué es exactamente esa «cosa», pues se diferencian en pequeños matices que da cada autor como ya se explicó. Además, claramente, los más antiguos no representaban exactamente lo que es hoy en día MDE, que es una ingeniería bastante bien definida, pues en los primeros artículos simplemente se estaban dando los primeros pasos, pero en busca de la meta que se consiguió finalmente con MDE.

Por ello, todos estos términos vistos, nombran a MDE, que, aunque no fue de los primeros términos que salió, es el término libre más extendido en ciencias de la computación, además de ser el que se utilizó para definir exactamente todo el resultado final. Por este mismo motivo, a lo largo de esta tesis y de sus publicaciones derivadas se hace uso del término MDE en vez de utilizar alguno de los otros términos existentes.

Mientras, MPE, MGA, M.E.R.O.DE., MDA, las Factorías de Software y TALISMAN, entre otras muchas implementaciones existentes y no nombradas en esta tesis, debido a que solo se nombraron las consideradas más relevantes y sus primeras evoluciones. Todas estas son visiones particulares o las propias implementaciones de MDE de diferentes consorcios, empresas o investigadores. La relación y un resumen de los términos explicados en este apartado puede ser visto en la Ilustración 20.

En base a todo lo explicado, como nota, ni MDA, ni MDD, ni MDE fueron los primeros términos en aparecer, ni los primeros en empezar a utilizar modelos, pero si los términos más extendidos y sobre los que más se ha trabajado después.

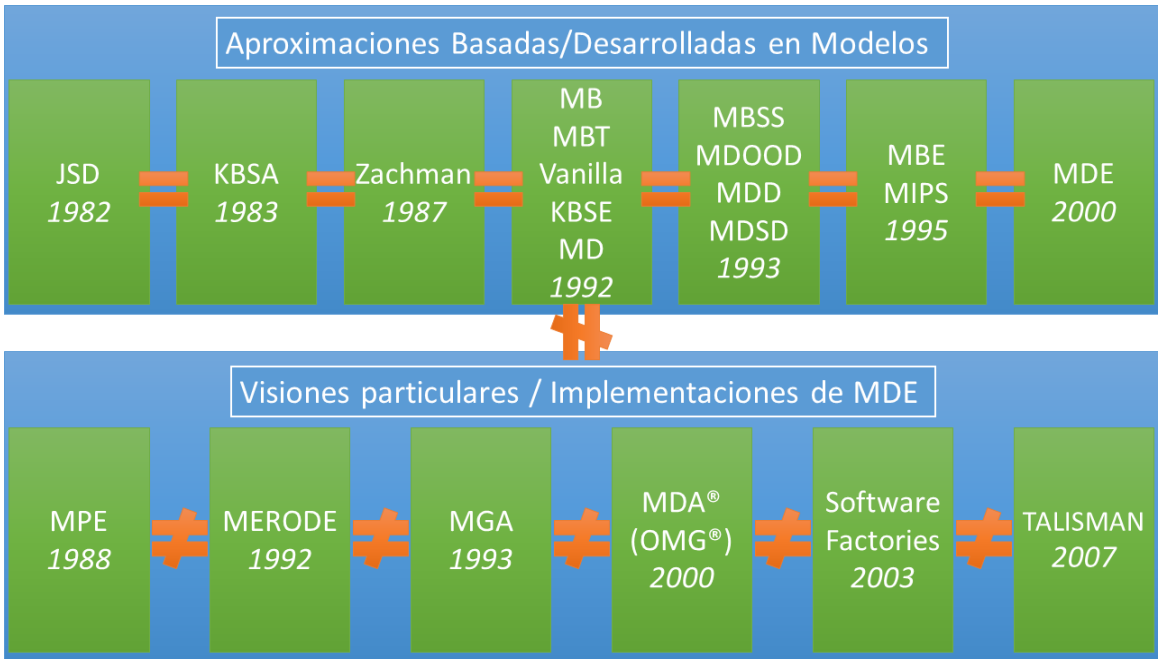


Ilustración 16 Nomenclaturas y relaciones de los diferentes usos de modelos

5.3 *INGENIERÍA DIRIGIDA POR MODELOS*

Como se introdujo en este capítulo, no existe ninguna bala de plata para solucionar los problemas de la **Crisis del Software** descritos por Edsger W. Dijkstra [21] y la OTAN [22]. Sin embargo, si hay pequeñas soluciones que ayudaron a minimizar estos problemas, mitigando los efectos de la complejidad esencial [207] y de la complejidad accidental [207].

La **complejidad esencial** se da siempre en el desarrollo de software debido a que esta complejidad depende del número de características que se desean añadir, es decir, si hay diez características, tendremos diez problemas, es decir, esta complejidad es la esencia del problema. Para ayudar a reducir la complejidad esencial surgieron nuevas metodologías de desarrollo, aquellas conocidas como metodologías ágiles, entre las cuales destacan eXtreme Programming (XP) [34], [35] y SCRUM [32], [33]. Estas metodologías ayudaron a hacer mucho más flexible el desarrollo de software manteniendo un contacto constante con el cliente y sus expectativas sobre el software, además de ayudar a la gestión de los miembros dentro del equipo y su conocimiento sobre el proyecto.

Por otro lado, la **complejidad accidental** es aquella que surge del problema de programar y escribir código. Esta se fue paliando a lo largo de los años con las nuevas generaciones de lenguajes de programación.

No obstante, ambas complejidades no está solucionadas del todo pues los problemas de la **Crisis del Software** [21], [22] se siguen dando a día de hoy todavía [208].

Un ejemplo de cómo estos problemas afectaban a la industria se puede encontrar en el ***Chaos Report*** de 1995 escrito por Standish Group. En este muestran como tan solo el 16% de los proyectos terminaron a tiempo y dentro del presupuesto acordado, el 31% fueron cancelados o se realizaron fuera de plazo o de presupuesto,

además de no ofrecer toda la funcionalidad acordada. Mientras, el 53% restante correspondía a los proyectos que fueron cancelados en algún punto durante su desarrollo [209].

La solución que surgió para paliar y resolver estos problemas fue tomar en cuenta otras industrias que tuvieron en el pasado problemas similares, los cuáles los resolvieron mediante la industrialización y automatización de procesos dentro de las propias fábricas. Así, surgió la idea de la industrialización del software. En base a esto, se esconde el reto de automatizar todo el flujo de desarrollo de software, la mejora de las herramientas, desarrollar en base a ciertos estándares y crear componentes estándar. De esta manera, podrían crear productos personalizados de una forma rápida y fácil, formando una cadena de suministro para distribuir costes y riesgos a través de diferentes proveedores [203].

Debido a estos problemas, surgieron los diferentes intentos de industrializar el software, entre los que se puede destacar la **Ingeniería Dirigida por Modelos** (**MDE**). **MDE** es un enfoque para el diseño y desarrollo de software basado en el uso de modelos software [170], [176], permitiendo una mayor abstracción para simplificar y formalizar diferentes actividades del ciclo de desarrollo de software [176]. Sin embargo, para lograr esta abstracción es necesario agrupar los diferentes conceptos y nociones fundamentales del sistema a modelar [191], es decir, hace falta un estudio previo del problema para crear el modelo con la información relevante del problema a resolver. Esta abstracción ayuda a que los usuarios finales puedan interactuar con el sistema utilizando solo los conceptos más familiares, como son la estructura y las entradas y salidas, ya sea de forma textual o gráfica [210], y eliminando toda aquella información no relevante para su propósito, como puede ser el código fuente, la programación y las variables. Por ello, los enfoques basados en modelos tratan de capturar toda la información relevante del sistema en un modelo específico del dominio de manera que todos estos conceptos se muestren de forma familiar a los usuarios del sistema, muchas veces, por medio del uso de representaciones gráficas para ayudar a disminuir la complejidad [211].

En otras palabras, **MDE** es la técnica que se basa en utilizar **modelos** para desarrollar software [167], abstrayendo las partes relevantes del problema, siendo los modelos su enfoque en vez de los programas de ordenador [165], pues el código fuente se convierte en la proyección del modelo diseñado [201] y se centra todo el desarrollo en el uso de modelos. De esta forma, **MDE** permite tomar ventaja de un **modelo** de entrada para maximizar la automatización del desarrollo de las aplicaciones, lo que repercute en una mejora general de la calidad [201].

El objetivo de la **Ingeniería Dirigida por Modelos** es incrementar la productividad y reducir el tiempo de desarrollo permitiendo que este sea más cercano al dominio del problema y con una abstracción mayor que los lenguajes de programación, haciendo que este desarrollo sea más entendible para las personas. La clave de MDE es transformar estos modelos, que están en un nivel de abstracción alto, en modelos específicos de la plataforma por medio de herramientas que puedan transformar los modelos en código fuente [176], [177], ya sea mediante un proceso automático o semiautomático [166]. Por ello, se puede decir que, MDE intenta resolver la complejidad del diseño e implementación del software moderno [24], permitiendo generar automáticamente documentación, código fuente o diagramas [201].

Algunos autores consideran MDE esencial en grandes proyectos de ingeniería [166], [170]. No obstante, MDE no es la panacea ya que a pesar de tener ciertas ventajas y características [165], no siempre puede ser

aplicado a cualquier proyecto. Hay que tener en cuenta que MDE impone el uso de ciertas estructuras y vocabulario comunes para así permitir la comunicación entre los diferentes participantes en el proyecto [176], ya sean objetos o personas.

Según comenta Selic en [166], [170], el mayor beneficio de utilizar MDE es cuando el modelo y la implementación son uno, es decir, cuando se genera el programa correspondiente automáticamente a partir del modelo definido. Por ello, en este caso, los lenguajes de modelado que usen MDE deberán de tener la misma precisión semántica que los lenguajes de programación, pero esto no significa que deban de tener el mismo nivel de detalle.

Esta generación automática de código requiere que los modelos sean ejecutables («*executable models*»). No obstante, hay que tener cuidado, pues a veces, en la literatura se entendió incorrectamente el término de ejecución de modelos, pues algunos investigadores atribuyeron este término erróneamente a la animación de diagramas, cuando en realidad el significado es el de poder llevar a cabo una operación dinámica del sistema, como puede ser la generación de eventos o actualización o la generación de código fuente [191]. Esta ejecución de los modelos es algo que permite experimentar y aprender rápidamente las propiedades del sistema de una manera barata. Así, la misma semántica que se requiere para soportar esta generación automática también permite la ejecución de los modelos [170].

5.3.1 UNA BREVE HISTORIA DEL USO DE MODELOS Y SUS INICIATIVAS

¿Cuándo surgió MDE? Esta pregunta es muy difícil de responder pues algunos autores datan el uso de modelos en el año 1947 en la en la Universidad de Cambridge cuando Wheeler y Wilkes crearon las primeras librerías prescritas para la computadora EDSAC [28]. Otros autores nombran que las primeras herramientas de programación automática surgieron en los años 50-60, las cuales fueron los ensambladores y compiladores [192]. Por otro lado, como se explicó en el apartado 5.2, las herramientas CASE de los años 80 ya hacían un trabajo parecido [171], [180]. Además, como se explicó, no hubo un consenso desde el principio y cada uno llamo a MDE de una forma diferente, además de existir tres vertientes, la de **JSD** surgida en 1982 [182], la de **KBSA** surgida en 1983 [184] y el ***Zachman Framework*** en 1987 [186]. Por tanto, el comienzo es algo difuso de definir exactamente, pero si se pueden decir esas fechas relevantes en la historia de MDE. Por ello, en este apartado, se explicará un poco de la historia más relevante de MDE y los usos que se le fueron dando y como fue evolucionando hasta nuestros días.

Primero, en 1982, apareció el ***Jackson Systems Limited*** (**JSD**) de Michael A. Jackson [182], quien propuso tres principios utilizando modelos. El primero de los principios es crear el **modelo** y después se le añaden las partes del sistema. Esta vista capturada en el modelo debe de ser bien conocida por los usuarios y ser capturada por el desarrollador para proveer una abstracción que sea más fácil de usar y provoque menos errores. El segundo principio dicta que un modelo debe de proveer de una comunicación adecuada de procesos secuenciales, de manera que el orden de los acontecimientos del mundo real esté representado en el proceso del programa. De esta manera, el modelo contendrá mapeado todo el progreso requerido en el mundo real y permitirá hacer todo de una manera sencilla. El tercer principio es que el sistema será implementado por transformaciones de un set de procesos eficientes y adaptados para correr sobre el software y hardware

disponible. En resumen, JSD hablaba de describir el mundo real mediante el modelado y así, que el modelo creado tuviese unas salidas que permitiesen comunicar procesos secuenciales y que mediante un set de transformaciones sencillo se consiguiesen muchas implementaciones diferentes a partir de una misma especificación. Como se ve, JSD hablaba de lo mismo que habla MDE actualmente, siendo así una de las primeras en proponer el uso de modelos, aquí la primera vertiente y siendo más adelante utilizado por M.E.R.O.DE. [183].

En el año 1983 se publicó este reporte [184], en el cuál se habla de un nuevo paradigma para desarrollar, llamado ***Knowledge-Based Software Assistant*** (KBSA). Esta es la segunda vertiente. Esta propuesta quiso formalizar un paradigma para el desarrollo software. Este artículo también analizó los problemas que se intentaban resolver. Para ello, propuso un cambio en el ciclo de vida de los proyectos para automatizar procesos y así reducir costes. De esta manera, pretendían mejorar la productividad, adaptabilidad y funcionalidad en los sistemas software. Algunos autores apoyaron este artículo como una primera propuesta de paradigmas de programación automática [185]. Estos paradigmas son aquellos que además de tener un compilador automático, proporcionaban medios para adquirir especificaciones de alto nivel para ser compiladas y traducían estas especificaciones de alto nivel en otras de bajo nivel [185].

Unos años más tarde, como la tercera referencia histórica a la **Ingeniería Dirigida por Modelos** pero que no hace referencia a las primeras, se encuentra este artículo [186] del año 1987 perteneciente a Zachman, la cual consideramos la tercera vertiente, que es citada como el enfoque inicial por [187], [188]. En este artículo, Zachman presenta la arquitectura de su *framework* enfocado a los sistemas de la información, al cual nos referiremos como **Zachman *Framework***. Este *framework* se compone de diferentes niveles de detalles para cada tipo de trabajo involucrado en el proceso de creación de software. Esto se debe, a que el ingeniero necesita conocer detalles a bajo nivel, detalles que no le importan al diseñador y que estos a su vez no tienen que ver con los detalles del cliente, además de que cada uno tiene su contenido, semántica y perspectiva. Es decir, una perspectiva para cada participante en el proyecto. Aunque Zachman no lo diga, estas son las diferentes vistas que un objeto puede poseer y que se pueden modelar, o como bien él dice: «*Different types of descriptions for the same product*». Estos cuatro modelos son el modelo de ámbito, el modelo de negocio, el modelo de diseño y el modelo de tecnológico. Por medio de ellas, pretendió mejorar la comunicación del equipo, entender mejor los riesgos de los proyectos, relacionar herramientas y metodologías entre ellas y mejorar el enfoque de la representación de las arquitecturas. En 1992 sacaron una actualización de este *framework* explicando mejor su funcionamiento y las diferentes vistas en mayor profundidad [212].

Un año más tarde, en 1988, Gabor Karsai redactó su primer trabajo haciendo uso de los modelos software, pero sin hacer referencia a ningún concepto específico como los conocemos hoy en día, es decir, a MDE. Karsai publicó un *framework* con representación gráfica y que utilizaba un lenguaje declarativo basado en Lisp al que llamó ***Multigraph Programming Environment*** (**MPE**). **MPE** permitía especificar paradigmas de modelado, lo que permitía modelar muchos aspectos diferentes [189].

Posteriormente, Karsai volvió a utilizar modelos en el siguiente año, en 1989. En este caso, los autores desarrollaron un editor gráfico para crear modelos que contaba además con herramientas especializadas para ayudar a la interpretación de los modelos [213].

Ya en el año 1992, Karsai junto otros nuevos autores, utilizaron modelos en esta investigación [190]. En este artículo hacen referencia a **Técnicas Basadas en Modelos** (***Model-Based Techniques***, **MBT**). Esta vez los usaron de forma similar para así ayudar al mantenimiento de sistemas de procesamiento de señales en tiempo real. No obstante, este sistema tenía un problema que se debía a que era un sistema reactivo. Así, el problema residía en que los sistemas reactivos necesitan adaptarse continuamente al entorno y esta aproximación de modelos no permitía esta posibilidad.

En este mismo año, 1992, David Harel publicó un artículo acerca del ***framework Vanilla*** [191]. Este *framework* sigue el concepto conocido como «*vanilla*». Es decir, mantener las cosas sencillas, simples y planas. Así, el artículo explica como hicieron la aproximación «*vanilla*» de la programación mediante el uso de **modelos** para generar código automáticamente, semiautomáticamente o parcialmente. Además, explica diferentes conceptos como la generación de eventos o la actualización y generación de código fuente, remarcando que este muchas veces debía ser rellenado a mano, la consistencia entre capas y la verificación del modelo. De esta manera, explicó la forma de generar directamente código Ada [214], C [215] o Modula 2 [12] [216], a partir del modelo definido. Estos conceptos que fueron después «recogidos» y utilizados posteriormente por lo que conocemos, entre otros nombres, como MDE. Este artículo también da pinceladas sobre los modelos y los diferentes términos utilizados en los modelos, incluyendo como deberían de funcionar las herramientas. Así, su investigación provee de una forma de desarrollar sistemas software complejos de una manera mucho más fácil.

El año 1992 fue muy productivo, pues las anteriores no fueron las únicas referencias en el uso de modelos, ya que Michael R. Lowry publicó un artículo acerca de ***Knowledge-Based Software Engineering*** (KBSE) [192]. Para describir KBSE se basa en lo comentado en el artículo de KBSA. La diferencia es, que KBSE fue ampliamente más utilizado posteriormente que el término KBSA. En este artículo también hace referencia al gasto extra computacional que requieren ellos debido a los diferentes editores y la diferencia de potencia existente en su época con la de los años 80, que es de cuando data KBSA. También predijo que esta tecnología tendría su auge en el año 2000. Así como que este enfoque era mucho más ambicioso que el existente con las herramientas CASE [193]. Razón no le faltó, pues en el año 2000 surgió el termino MDE, el enfoque MDA, y se comenzó utilizar estos más ampliamente en las empresas.

En 1992 [195], ampliada posteriormente en 1993 [187], 1994 [183], 1995 [217] y en 1997 [188], surgió un enfoque utilizando MDE, llamado ***Model-driven Entity-Relationship Object-oriented DEvelopment*** (**M.E.R.O.DE.**), sobre el que hay muchas más investigaciones más recientes [218]. Con ella pretendieron mejorar el mantenimiento, modularización y ahorro de costes. Para ilustrar esto, utilizaron el ejemplo de préstamos de libros de una biblioteca. No obstante, también aclararon que el impacto exacto de MDE no había quedado del todo claro según los cuestionarios que entregaron a las empresas y las respuestas de estas, pero que si advirtieron que el uso de herramientas CASE y de documentación sería crucial. Estos mismos autores publicaron también, en el año 1993 [198], un artículo en el que hicieron referencia a MDD para poder construir herramientas CASE que permitiesen guardar la consistencia de las especificaciones automáticamente. En este

[12] http://www.modula2.org/

artículo muestran métodos de desarrollo que pueden ser utilizados como lenguajes formales para describir y optimizar los procedimientos de administración de empresas.

En 1993 nos encontramos con el uso de la **Ingeniería Dirigida por Modelos** en un artículo publicado en mayo [196], titulado ***Model-Based Software Synthesis*** (MBSS), en el cuál definen estos autores como una parte de la disciplina de la ingeniería software basada en el conocimiento y que integra la Inteligencia Artificial con la ingeniería de software utilizando **modelos**. En este artículo, los autores crearon un sistema que generaba código automáticamente a partir de unos modelos declarativos definidos usando un editor gráfico que ayudaba a limitar el dominio de la aplicación. Según los autores, el uso de *Model-Based Synthesis* es bueno cuando las especificaciones software y hardware están en constante cambio. Así, lo que consiguieron, fue añadir un nuevo nivel de abstracción usando modelos para facilitar el uso de su sistema de control y procesamiento en tiempo real de señales, el cuál era bastante complejo. No obstante, como bien comentan los autores, entre los cuáles se vuelve a encontrar Gabor Karsai, ellos se basaron en el *framework* **MPE** [189]. En este artículo hacen referencia también al *framework* **MGA**, alegando que llevan años trabajando con él, pero del cuál no se encuentra un artículo que hable de él hasta el año 1995 [197].

En 1995, con la aparición del término MDD, el autor de [219] comentó los problemas de la «**Crisis del Software**», la cuál podría ser solucionada mediante la tecnología de objetos, algo que propuso los Estados Unidos de América para incrementar la productividad, flexibilidad y escalabilidad de los sistemas software. No obstante, esto se tradujo en la «crisis de los objetos» al fallar el conseguir solventar lo propuesto. La solución que propuso el autor fue hacer componentes compartibles en base a un estándar que permitiese la interoperabilidad y automatización de procesos. Estos son los principios que presentó el OMG más adelante en su arquitectura conocida como MDA.

En este mismo año, 1995, Gabor Karsai junto a otros autores sacaron un artículo sobre su arquitectura ***Multigraph***, pero esta vez mucho más refinada y bajo el nombre ***Multigraph Architecture*** (MGA) [197]. No obstante, hicieron referencia a él en un artículo previo [196]. En este artículo, los autores explicaron que MGA es una arquitectura metanivel que incluye herramientas y métodos para crear modelos específicos del dominio y soportar la integración de modelado de sistemas independientes. Con esto buscaron facilitar el diseño, la implementación y el desarrollo de aplicaciones en ambientes complejos y cambiantes. Según sus propias palabras, esto supuso un avance tremendamente importante en el desarrollo de entornos de Síntesis de Programas Integrados en el Modelo (MIPS). Los ambientes MIPS son específicos del dominio e incluyen herramientas para construir y testear modelos, ofrecen transformación de modelos en programas, tienen herramientas para extraer información de los modelos y ofrecen integración de plataformas de computación paralela y distribuida. Es decir, se podría decir que son plataformas que trabajan con modelos.

En 1997, los autores de [210], utilizaron MDE para automatizar la construcción de sistemas de instrumentación complejos y así reducir los costes de construcción y mantenimiento de estos sistemas y mejorar el desarrollo de este tipo de sistemas, permitiendo hacer una representación de este más familiar para los usuarios finales. La captura de los conceptos la realizaron mediante el uso de formularios textuales y gráficos.

También en el año 1997, los autores del artículo [173] se basaron en el ***Zachman*** *Framework* y en **M.E.R.O.DE.** para presentar un enfoque para medir la complejidad de los requisitos del negocio basada en el álgebra de **M.E.R.O.DE.** De esta manera, se basaron en la técnica de especificación orientada a objetos para garantizar la consistencia y la exactitud de los modelos.

Como se comentó previamente, MDA es un enfoque de MDE. En el apartado 5.4.3, se mostrarán ejemplos de uso exitoso de MDA.

En lo referente al enfoque de las **Factorías de Software**, este tiene cuatro implementaciones de referencia [220] que datan del año 2006. La primera implementación es *Smart Client Software Factory* (SCSF) [221], que está pensada para facilitar la creación de aplicaciones de escritorio. La segunda es *Mobile Client Software Factory* (MCSF) [222], llamado ahora *Mobile Application Blocks* [223], que se diseñó para facilitar la creación de aplicaciones móviles. Para las aplicaciones web se creó *Web Client Software Factory* (WCSF), mientras que *Web Service Software Factory* (WSSF) fue para crear servicios web [224].

Existen también otras dos implementaciones de **Factorías de Software** de relativa importancia. La primera, llamada *Application Block Software Factory* [225], está diseñada para facilitar el desarrollo de *Application Blocks* personalizados, es decir, implementaciones y guías de mejores prácticas propuestos por *Microsoft Patterns & Practice Group*. Por otro lado, la *Computer Games Software Factory* está orientada al desarrollo de videojuegos, identificando la arquitectura de estos, como son el DSL, los validadores y el generador de código, mostrando así el aumento de la productividad gracias a la abstracción [226]. Ambos son del año 2007.

Posteriormente, surgió la versión de TALISMAN basada en MDE [23], [227]. TALISMAN fue aplicado a entornos de producción reales en áreas no ligadas directamente con la informática pero que si la usan. Este es el caso de [199], en el año 2008, donde realizaron un sistema de trazabilidad inteligente para el queso Cabrales y así evitar tener que adaptar la misma aplicación para cada productor de este queso. Para esto, buscan disminuir el peso que tiene en el desarrollo la implementación y aumentar el peso del modelado. En [228], se puede encontrar una definición exacta del funcionamiento interno de TALISMAN MDA y su *XML Metadata Interchange*® (XMI) [229], junto a los cinco sistemas de trazabilidad de queso que realizaron usando este enfoque, a saber: Cabrales, Casín, Afuega'l pitu, Gamoneu y Beyos. De esta manera, eran capaces de generar el software que necesitaban que era el necesario para interactuar con impresoras, lectores RFID, bases de datos SQL Server y terminales. Mientras, en [227], se puede ver toda la arquitectura de TALISMAN MDE (TMDE), donde define TMDE como mitad **MDA** y mitad ***Software Factories***, cogiendo así lo mejor de cada implementación.

Los sistemas embebidos en tiempo real, conocidos como RTES, son sistemas que podemos encontrar distribuidos en cualquier sitio al que vayamos y que son demasiado heterogéneos. No obstante, son sistemas en los que es difícil realizar su mantenimiento y reutilizar los conocimientos. Por ello, los RTES son sistemas que encajan muy bien con la idea de MDE. Debido a estos motivos, en este artículo del año 2011 [230], utilizaron MDE, aplicando el enfoque del OMG, es decir, MDA, para conseguir reducir los costes de desarrollo de múltiples plataformas RTES mejorando la portabilidad.

En el año 2011 también surgieron artículos y tesis que intentaban mejorar MDE. Este es el caso de [220], [231], proyecto conocido como ***Model-Driven Continuous Integration*** (**MDCI**). En este, con el desarrollo de MDCI se consiguió permitir que los expertos tecnológicos aplicasen integración continua durante el desarrollo y que los expertos del dominio pudiesen modificar la aplicación usando los modelos de forma distribuida.

Incluso, MDE fue utilizado en proyectos de ámbito nacional español y financiados por el Ministerio de Industria, turismo y comercio para solucionar los problemas relacionados en el desarrollo de videojuegos. Este es el caso del proyecto Gade4All [232], código MITC-11-TSI-090302-2011-11, en donde se utilizó MDE para permitir la generación de videojuegos 2D multiplataforma de una forma rápida y fácil. Este proyecto ofrecía diferentes tipologías de juegos, como son la de táctil de habilidad, puzle, plataformas, trivial y estrategia. El punto fuerte residía en que permitía generar automáticamente el juego para tres plataformas a la vez, las cuales eran Android, iOS y HTML5, todo mediante el uso de un DSL gráfico que abstraía al usuario de la programación.

5.3.2 TERMINOLOGÍA UTILIZADA EN MDE

Como sucede con cada tecnología, hay una serie de términos que se deben entender para comprender el funcionamiento de esta. Esto sucede con MDE, la cual se compone de una diversidad muy grande de ellos y que necesitan ser entendidos en su conjunto, pues todos están relacionados. En la Ilustración 17 se muestra la relación de diferentes términos en base a la figura de [233], la cual fue modificada para mejorar su comprensión.

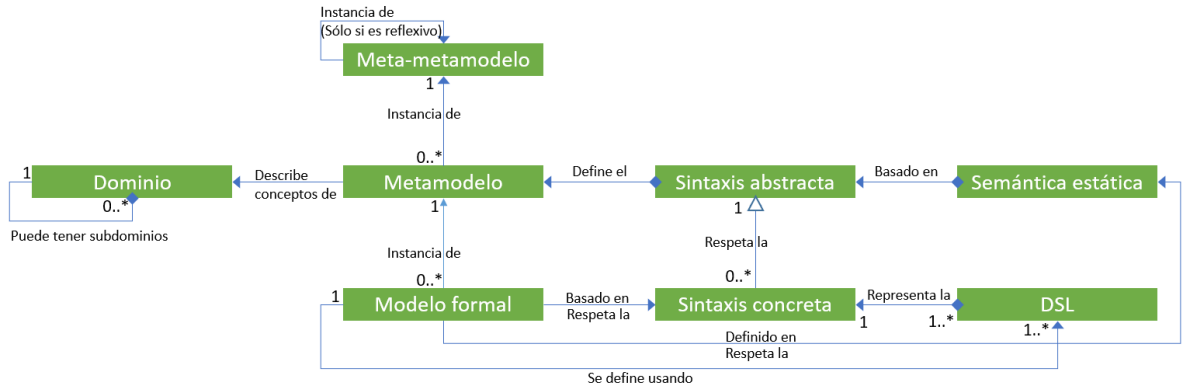


Ilustración 17 Arquitectura de los conceptos de MDE

A continuación, se describirán los diferentes conceptos incluidos en esta figura, así como otros términos relevantes.

Sistema [234]: conjunto de partes y de las relaciones entre estas partes que pueden ser organizados para lograr un propósito. Así, un sistema puede ser un sistema hardware o software, una compañía, procesos de negocio o la combinación de diferentes sistemas. Un sistema está formado por la plataforma y su aplicación.

Dominio: el **dominio** siempre es el punto inicial de MDE y delimita el campo de conocimiento [220]. Así, el dominio es el área de conocimiento sobre la que se trabaja y sobre la que se quiere resolver el problema.

Estos se dividen en dominios tecnológicos y profesionales, donde los primeros hacen referencia a la tecnología de desarrollo de software y los segundos a los conceptos que manejará la aplicación. Los dominios a su vez pueden estar compuestos de varios subdominios [220].

Metamodelo: es una especificación del lenguaje de modelado que define las características del modelo y permite verificar el modelo expresado en ese lenguaje determinado de manera formal, es decir, es un modelo del lenguaje de modelado [167]. Además, los metamodelos son fundamentales para conseguir automatizar el desarrollo de software [235]. Los metamodelos están formados por la **sintaxis abstracta** y la **sintaxis concreta**.

Metamodelo reflexivo [167] o **supermetamodelo**: es cuando el metamodelo de un lenguaje de modelado usa el mismo lenguaje de modelado, es decir, se define el metamodelo utilizando el mismo lenguaje en el que el metamodelo está descrito. Dentro de estos, el metamodelo mínimo reflexivo es aquel que usa el mínimo número de elementos del lenguaje de modelado para los propósitos de ese metamodelo, luego, si se eliminase cualquier elemento sería imposible modelar o expresar cualquier estado esencial.

Meta-metamodelo: es una especificación del metamodelo que define las características del metamodelo y permite verificar el metamodelo expresado en ese lenguaje determinado, es decir, en un modelo del metamodelo. El tener un meta-metamodelo permite que exista un metamodelo para cada dominio del conocimiento a tratar mientras se tiene un meta-metamodelo común a todos estos para así poder realizar operaciones sobre ellos, como pueden ser transformaciones automáticas, validaciones y búsquedas. Los meta-metamodelos suelen ser reflexivos.

Así, según [220], el meta-metamodelo es una abstracción superior del metamodelo que permite que los metamodelos sean reutilizables, interoperables y portables.

Los meta-metamodelos pueden tener un metamodelo, sobre todo en el caso de que no sea reflexivos. En este caso, el metamodelo padre sería llamado meta-meta-metamodelo y se correspondería con el nivel M5, y así sucesivamente, aplicando un prefijo «meta-» por cada nivel nuevo que se incrementase. No obstante, esto podría crear capas redundantes.

Lenguaje de dominio específico, **DSL** o **lenguaje de modelado** [234]: un DSL está constituido por la estructura, los términos, notaciones, sintaxis, semántica y reglas de integridad que son usadas para expresar un modelo. Algunos ejemplos de lenguajes de modelado son UML, SQL Schema, *Business Process Management and Notation* (BPMN), E/R, *Ontology Web Language* (OWL) y XML Schema. Los DSLs son los lenguajes creados específicamente para expresar los conceptos de un dominio determinado [30], [236]. Así, un DSL es un lenguaje que fue creado específicamente para solucionar un problema específico de un dominio concreto, siendo así el elemento principal de cualquier solución de dominio específico [220].

Punto de vista (***Viewpoint***): un **punto de vista** de un sistema es una técnica de abstracción en la que se seleccionan una serie de conceptos y reglas estructurales de ese sistema con el fin de centrarse en las preocupaciones importantes de ese sistema. Es decir, se crea una abstracción para suprimir detalles irrelevantes y así obtener un modelo simplificado de una parte del sistema [28], [234]. Un mismo sistema puede tener diferentes **puntos de vista** para mostrar diferentes abstracciones con diferente nivel de detalle,

pues un mismo sistema puede necesitar más de un punto de vista para mostrar sus propiedades o para poder ofrecer diferentes usos [174]. Cada **punto de vista** puede tener uno o más **modelos**, también conocidos como **vistas**.

Vista (***View***): una **vista** es la representación del sistema desde una perspectiva elegida de un **punto de vista** [237]. Por ejemplo, si tenemos un sistema para mostrar los datos de los usuarios de un videojuego online, una vista podría ser la forma en que está la información estructurada, otra vista la que muestra la información que puede ver cada rol existente, otra vista podría ser la que contiene los protocolos utilizados para transmitir la información y otra vista para saber cómo se obtiene la información de los usuarios a partir de la información del sistema.

Capas de la arquitectura de metamodelado [167]:

- M0: lo que debe de ser modelado (Elementos del mundo real)
- M1: Modelo – Un modelo realizado utilizando UML.
- M2: Metamodelos – es el modelo de sintaxis abstracta que contiene la especificación con el mapeo entre los elementos utilizados en el modelo M1 (UML, ODM)
- M3: el meta-metamodelo. Esta es usada para expresar los modelos M2. Este lenguaje de modelado es reflexivo, luego no necesita de capas superiores. En el OMG esta siempre es *Meta-Object Facility*™ (MOF) [238].

En la Ilustración 18 se muestra la arquitectura en cuatro capas y como cada capa inferior es una instancia de la superior, a excepción de la capa M1 que sirve para representar los elementos del mundo real, que se encuentran en la capa M0 y la capa M3, es decir, el meta-metamodelo, que es una instancia de sí mismo en el único caso de que se trate de un metamodelo reflexivo. A la derecha de la imagen se puede ver unos posibles ejemplos correspondientes a cada capa. En la capa M3 se encuentra MOF, que es el meta-metamodelo reflexivo del OMG, en la capa M2 estarían Ecore y UML, que son dos metamodelos basados en MOF, de los cuáles el primero fue creado como el núcleo de Eclipse Modeling Project (EMF) [239] y el segundo fue creado por el OMG para ser el núcleo de MDA [240]. En la capa M1 se encuentran diferentes modelos realizados utilizando el mismo o diferentes metamodelos y que sirven para representar sus correspondientes elementos de la capa del mundo real, M0.

Dependiendo del tipo de problema, se podrían incluir más capas, pero esto podría llegar a ser redundante.

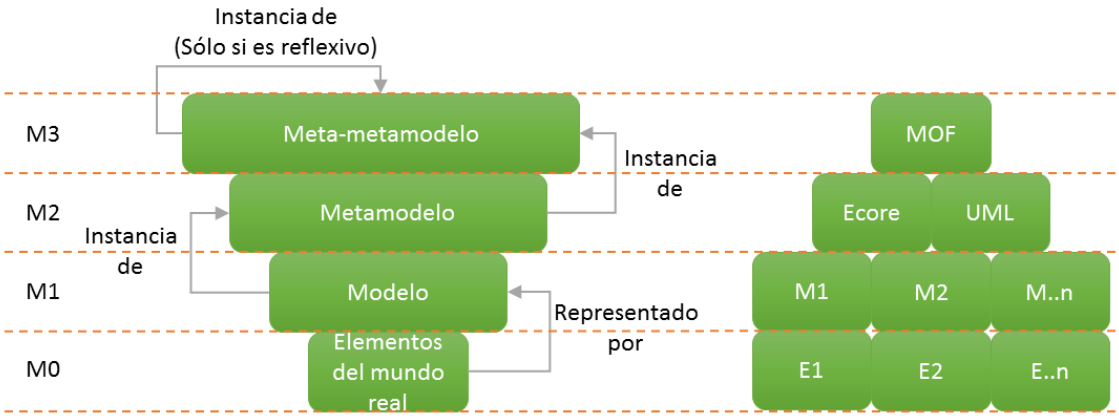


Ilustración 18 Arquitectura de cuatro capas

Sintaxis abstracta: especificación formal en la que se basa un lenguaje [174]. Así, la sintaxis abstracta de un metamodelo es básicamente un modelo de datos de como almacenar o intercambiar datos en la estructura semántica del modelo a realizar, además de servir para definir las restricciones y estructura de los elementos del modelo [167]. La sintaxis abstracta también puede poseer reglas del lenguaje metamodelado para así evitar la mala práctica de validar modelos en los generadores de código de manera que se consigan detectar las anomalías cuanto antes y simplificar el resto de componentes [220]. Es decir, la sintaxis abstracta está definida implícitamente en el modelo a nivel conceptual, luego, no tiene una sintaxis específica. La semántica estática se basa en esta sintaxis.

Sintaxis concreta: esta puede ser textual o gráfica [174]. Bajo una misma sintaxis gráfica, se pueden tener diferentes sintaxis concretas, lo que nos permite tener diferentes tipos y vistas de esta, permitiendo diferentes tipos de forma para definirla, ya sea gráfica, textual o en forma de árbol, entre otros. Es decir, la sintaxis concreta define cómo se deben representar los modelos y que notaciones deben de utilizar los usuarios para ello [220].

Semántica estática: contiene el significado del lenguaje. Esta puede ser más o menos formal. La semántica estática puede estar basada en términos de las cosas observadas en el mundo donde se describe, como puede ser el envío de mensajes, el estado de los objetos o el cambio entre estado, entre otras cosas, o bien, puede estar basada en traducciones de lenguajes de alto nivel construidos a partir de otras construcciones que tienen el significado bien definido [174]. La misión de la semántica estática es la de hacer comprobaciones semánticas en los modelos para asegurar que están bien construidos [220]. Esta semántica está basada en la sintaxis abstracta ya que al definir los conceptos en un metamodelo va a haber un significado subyacente.

Modelo o **modelo formal**: descripción en un lenguaje de modelado o lenguaje natural de una vista determinada de un objeto o sistema, perteneciente al mundo virtual o real, que abstrae sus conceptos relevantes y que sirve como punto de referencia para reproducirlo. Es una instancia del metamodelo, donde el modelo pertenece a la capa M1 y el metamodelo a la capa M2.

Modelo serializado: modelo ya definido que está codificado en un medio de almacenamiento, como puede ser un fichero o en memoria, con el fin de transmitirlo a otro lugar. Una *serialización* estándar perteneciente al OMG es el formato XML o XMI. Esta serialización contiene todos los datos relevantes del

modelo definido para que este sea reproducido en el destino. Estos datos incluyen su definición y pueden llegar a contener hasta las coordenadas de los diferentes elementos del metamodelo utilizados. La serialización del modelo se suele llevar a cabo sobre todo en los **DSLs gráficos** para así hacer persistente la construcción realizada por el usuario.

Infraestructura [174]: conjunto de piezas software o hardware presentes cuando se desarrolla un artefacto software.

Plataforma [28], [174], [234]: infraestructura de software que contiene una serie de recursos que permiten implementar o soportar un sistema determinado o implementado una tecnología específica, como pueden ser la plataforma *Common Object Request Broker Architecture*® (CORBA) [241], *Asynchronous JavaScript And XML* (AJAX), Eclipse RCP (*Rich Client Platform*), *Java Enterprise Edition* (JEE), *Open Services Gateway initiative* (OSGi) y Microsoft .NET como plataformas hardware y software, o la organización de una restructura empresarial y la organización de empleados como plataformas de negocios o de dominio. La plataforma soporta la ejecución de la aplicación y, en conjunto, constituye el sistema. Así, la aplicación provee de la parte funcional del sistema descrito en el modelo.

Aplicación [28]: funcionalidad desarrollada, las cuáles pueden ser una o más en un mismo sistema.

Arquitectura [234]: el propósito de la arquitectura es definir o mejorar los sistemas de manera que ayude a comprender el ámbito del sistema de interés y los requisitos.

Semántica del espacio del problema [220]: esto hace referencia a que, cada vez que se incluye un artefacto en el modelo, lo que se está añadiendo es significado debido a que cada concepto capturado en el modelo tiene su propio significado. Así, cada concepto del lenguaje se mapea directamente a un concepto del dominio que se modela, que es todo lo contrario de lo sucedido en los GPLs.

5.3.3 CICLO DE DESARROLLO HABITUAL UTILIZANDO MODELOS

Un objeto puede tener varias perspectivas diferentes, lo que implica tener un modelo por cada perspectiva que nos interese, de las cuales sus modelos estén expresados en diferentes lenguajes [28], [29]. Por ejemplo, estas vistas pueden ser desde la perspectiva de la base de datos hasta como se visualizan los datos. Por ello, estos requieren lenguajes con diferentes propiedades que permitan crear los modelos necesarios.

El mapeo de estos modelos no debe de depender de reglas sencillas, sino que es mejor utilizar un proceso automático, en vez de uno manual, para así poder generar, si fuese necesario, otros modelos. No obstante, para esto hace falta definir el modelo y un lenguaje para traducirlo, el cual permita mantener las coherencias automáticamente. Para saber cómo se debe de desarrollar, hay que tener en cuenta, como bien dice Kent en [29], diferentes factores. Uno de ellos es si se pretende que el resultado sea procesado por una máquina o por personas. En caso de que fuese por máquinas, habría que ver qué tipo de procesamiento se hará, es decir, si el modelo será después procesado de nuevo para crear otro modelo con el fin de mantener la coherencia entre modelos o bien si será el producto final. En cambio, si va a ser destinado para las personas, habrá que ver que sea comprensible y/o fácil de escribir. Así, depende de la audiencia al que esté destinado, se podrían usar diferentes estándares existentes como *eXtensible Stylesheet Language Transformations* (XSLT) [242] para

crear el procesado para las máquinas, en el caso de que se use XML. No obstante, esto hace que sea difícil de interpretar por seres humanos. Otra decisión a tomar es sobre si utilizar el mismo lenguaje para definir los modelos y crear los traductores, pues, así, se consigue que los desarrolladores solo deban utilizar un lenguaje y no tengan que aprender varios [29].

La arquitectura principal de los modelos y las transformaciones viene impuesta por las perspectivas elegidas del sistema. Normalmente, se suelen utilizar los casos de uso en el desarrollo de software, los cuáles pueden ser incrementales para ir añadiendo lo que se vaya necesitando en cada iteración [243]. Esta manera puede ayudar a organizar la gestión del proyecto, debido a que se adopta un acercamiento iterativo, en el cuál cada iteración podría corresponderse con un caso de uso completado o incrementado [29]. Esto podría ajustarse bien a la filosofía utilizada en el desarrollo SCRUM [32], [33] o *eXtreme Programming* (XP) [34], [35]. No obstante, en este caso, las iteraciones serían marcadas por los casos de uso en vez de que el orden de las iteraciones sean las que determinen la importancia de los casos de uso. Esto se debe a que es difícil de imaginar los modelos y los traductores sin saber a qué proceso se asocian ni a que están destinados, lo que implica que, por muy bien que estén diseñados, estos puedan variar mucho de su diseñado conceptual al modelo necesitado en la aplicación final, además, de que hay que tener en cuenta de que el contenido de los modelos puede influir en todo el proceso de desarrollo [29], [191]. Esto implica que, al depender de los modelos para realizar el proceso, se deba conocer el proceso para crear el modelo. Luego, hay que ir desarrollándolos en paralelo, dependientes el uno del otro, lo que implica el añadir esta nueva dimensión en el proceso de desarrollo para así ir refinando el modelo repetidamente [191].

Uno de los problemas existentes es que muchas veces los modelos suelen relegarse a la documentación o simplemente tienden a quedarse en el esbozo principal en el que fueron realizados para imaginar cómo sería el sistema antes de construirlo. Incluso, a veces, estos modelos no tienen todas las normas definidas. Esto se suele dar en el desarrollo orientado a objetos [29]. En cambio, en los enfoques dirigidos por modelos, los **modelos** son mantenidos junto el código. Sin embargo, hay que tener en cuenta que no siempre se puede hacer un enfoque dirigido por modelos. Debido a esto, solo se deben de utilizar los **modelos** cuando estos aporten un beneficio de mantenimiento menor o un beneficio de producción mayor que el no utilizar modelos. Los **modelos** toman mucho más poder cuanto más tangibles sean, es decir, cuando se pueden procesar, transformar y comprobar, o mejor dicho, cuando se puede realizar procesos automáticos sobre ellos para reducir la carga del sistema lo máximo posible [29].

5.3.4 APLICACIÓN DEL CICLO DE VIDA DE MDE

Para aplicar MDE se debe partir del dominio del problema que se quiere resolver y que delimita el campo de conocimientos. Estos dominios pueden ser subdivididos en unos dominios más pequeños. Por ello, el primer paso, es definir y acotar el dominio que se tratará. Por ejemplo: creación de aplicaciones que inserten datos en una base de datos, generación de videojuegos educativos de dos dimensiones en entornos móviles, etc…

Lo siguiente es definir el metamodelo. No obstante, primero se debe elegir el meta-metamodelo que se utilizará. Lo que se consigue con el meta-metamodelo es que el metamodelo creado sea reutilizable,

interoperable y portable. Esto es debido a que es una instancia de una abstracción superior que define como debe ser el metamodelo. Ejemplos de meta-metamodelos son Ecore [181] o MOF. Una vez elegido el meta-metamodelo a utilizar, se define el metamodelo. En él hay que describir de manera formal los conceptos relevantes que tiene el dominio que se desea metamodelar.

Los metamodelos están formados de una sintaxis abstracta, una semántica estática y una sintaxis concreta. La sintaxis abstracta especifica su estructura, es decir, las construcciones, propiedades y conectores que puede poseer dicho lenguaje. Esta sintaxis es representada con la sintaxis concreta, que a su vez se define en un DSL que puede ser gráfico, textual o en forma de árbol. En la Ilustración 19 se puede observar la sintaxis concreta, que respeta y representa la sintaxis abstracta, de Ecore, y que ofrece un DSL gráfico para definir los modelos. Esta posee en primer lugar las construcciones (paquetes, clases, tipos de datos, numerables y anotaciones), después las propiedades (operaciones, atributos, literales y entradas) y al final los conectores (referencia, herencia y anotación).

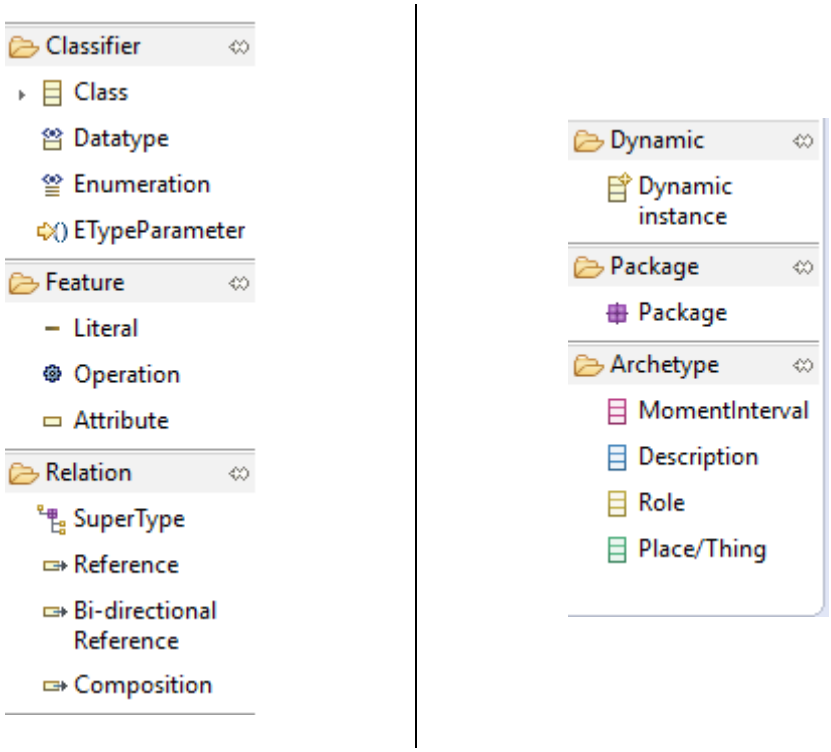


Ilustración 19 Sintaxis concreta gráfica de Ecore versión 2.11.2.v20160208-0816

Una vez creado el metamodelo con el DSL proporcionado, hay que realizar la transformación de esta a sintaxis concreta, de las cuáles puede haber más de una. Esta sintaxis concreta será la que utilicen los usuarios y se les puede presentar en diferente forma, ya sea gráfica, textual o en forma de árbol. A partir de esta sintaxis concreta, los usuarios podrán definir el modelo. Este modelo será el que se procese para crear la aplicación final.

5.3.5 TIPOS DE TRANSFORMACIONES ENTRE MODELOS

Las transformaciones entre modelos son llamadas «*model transformation*» o «*mapping*», y es algo que deben de soportar las herramientas de cara a reducir la carga de trabajo. Muchas de estas actividades se realizan automáticamente mediante el análisis de modelos existentes y transformando los modelos en modelos nuevos. No obstante, estas transformaciones requieren un entendimiento de la **sintaxis abstracta** y **semántica** tanto del **modelo** fuente como del destino. Una técnica utilizada es la de utilizar un **metamodelado** para definir la **sintaxis abstracta** y las **relaciones** entre los elementos del modelo. En este campo, las herramientas visuales ofrecen ciertas ventajas [177].

Estas transformaciones se realizan por medio de intérpretes que contienen la información de los conceptos del dominio y del entorno destino y son los encargados de sintetizar el software y el hardware de una manera determinada. Estos interpretes pueden considerarse como una implementación de un mapeador entre la representación del modelo del dominio y la plataforma de ejecución [210].

Las transformaciones son llevadas a cabo en base a las relaciones establecidas en la definición del modelo, el cual, después de la transformación puede cambiar o, simplemente, la transformación puede crear un modelo nuevo [171], [176]. Además, por medio de las transformaciones, es posible hacer ingeniería inversa, de modo que se consiga un modelo de mayor nivel a partir de uno de más bajo nivel [176]. Por ejemplo, extraer un modelo a partir del código fuente del producto software, que es algo que permiten algunas herramientas CASE.

Se pueden definir tres técnicas de modelos [177]. La primera de ellas es por medio de la **manipulación directa del modelo**, («*Direct model manipulation*» o «*pull*»). Esta técnica permite acceder a la representación interna del modelo y manipular la representación mediante el uso de una API. Esta API permite hacer transformaciones directamente desde un lenguaje de programación, como puede ser Ruby [244], [245] o Java. No obstante, debido a que estos lenguajes son GPLs, se hace más difícil escribir la transformación, comprenderlas y mantenerlas [177]. Es decir, en este caso la persona interesada en realizar una transformación entre modelos debe de elegir un GPL y programar en él todo el código necesario para traducir ese modelo utilizando una API.

La segunda técnica es la de la **representación intermedia** o «*Intermediate representation*», que permite exportar el modelo a una forma estándar, normalmente se usa XML, y transformarlo con una herramienta externa. Por ejemplo, ciertas herramientas CASE (*Computer Aided Software Engineering*), ofrecen la opción de exportar los modelos en formato XMI para hacer un procesado de este con otra herramienta cuando se exporta un UML, por ejemplo, mediante el uso de XSLT. No obstante, este tipo de transformaciones requieren experiencia con otras herramientas, como es XSLT. Este tipo de transformaciones proporcionan la posibilidad de realizar un procesamiento por lotes («*batch mode*»), lo que puede ser una ventaja por su automatización o un inconveniente por la dificultad de este tipo de procesamientos y los problemas de violación de reglas que puedan surgir [177]. Así, un ejemplo de estas transformaciones es cuando se persiste el modelo utilizando una forma estándar, como son XML, XMI o JSON y mediante otras herramientas externas podemos modificar directamente este modelo persistido o bien cargar el modelo en otra herramienta diferente para seguir trabajando con él.

Por último, el **soporte de lenguaje de transformación** o «*Transformation language support*». Esta última forma de transformación es cuando un lenguaje provee de un conjunto de construcciones para expresar, construir y aplicar transformaciones. Este tipo de transformaciones se componen de un lenguaje específico creado para realizarlas, es decir, de un lenguaje de dominio específico creado específicamente para resolver un problema determinado. Esto hace que esta sea la transformación más potente de las tres. Para lograr esto, se pueden usar varios lenguajes para especificar y ejecutar las transformaciones entre modelos. Estos lenguajes pueden contener construcciones visuales, siendo así gráficos, y suelen ser declarativos, procedurales o una combinación de ambos [177]. Un ejemplo para este tipo de transformaciones serían las herramientas que permiten modificar un determinado tipo de modelo, en algunos casos mediante el uso de interfaces gráficas.

5.3.6 HERRAMIENTAS PARA TRABAJAR CON MODELOS

Para realizar todo el proceso de trabajo con modelos es necesario disponer de herramientas que puedan trabajar con ellos y faciliten su uso, entendimiento y mantenimiento, además de ser sencillas y configurables. También, según [29], estas herramientas deberían de comprobar que las restricciones de los modelos sean cumplidas u obliguen a que se cumplan, dar soporte al trabajo con las diferentes instancias de los modelos, permitir el mapeado entre modelos, ofrecer test de pruebas de modelos, tener un control de versiones y ofrecer herramientas para gestionar el proceso software como podría ser el poder generar directamente los artefactos modelados. Por ejemplo, otros detalles son los que sugiere la tesis de [246], donde dice que los desarrolladores prefieren herramientas pequeñas y flexibles que puedan ser configuradas fácilmente y se interconecten con otras herramientas. Esto último ya lo ofrecen muchas herramientas de UML, las cuales permiten exportar e importar modelos en XMI.

Mientras, [177] comenta que las herramientas dedicas a las transformaciones entre modelos deberían de poseer la opción de ofrecer precondiciones para así comprobar que todo el proceso de transformación ha sido el correcto. También deberían de ofrecer la opción de combinar transformaciones en una nueva, ya que esto facilitaría el hacer transformaciones básicas para crear una más avanzada. Otra recomendación es el que las herramientas deban de ser gráficas, pues permite que sean más fáciles de usar que unas textuales. Por último, las herramientas deben de ser usables, no obstante, esto dependerá de la gente que las use y del lenguaje que se utilice, pues existe una gran diferencia entre si se utiliza un lenguaje declarativo o imperativo.

Algunas veces se tienen diferentes procesos que necesitan de diferentes arquitecturas de modelos, requiriendo un dominio diferente, a veces entre diferentes organizaciones e incluso, tal vez, entre diferentes proyectos. Por ello, los proyectos necesitan tener una flexibilidad propia [246]. Por ello, en MDE esto significa identificar el tipo de modelos a mantener y las herramientas para desarrollar y coordinar los modelos. Además, hay que tener en cuenta que existen diferentes lenguajes de programación y tipos de procesos. Así, el uso de herramientas que permitan facilitar esta parte es, incluso a veces, más importante que la propia abstracción del modelo. No obstante, no siempre se puede realizar una herramienta para cada proyecto, por falta de disponibilidad de recursos [29].

Una posibilidad podría ser crear estándares para MDE, pero esto podría hacer que las soluciones fuesen poco configurables y sería forzar a los desarrolladores [29]. Uno de los ejemplos es el estándar MDA, perteneciente al OMG.

5.3.7 CARACTERÍSTICAS DE LA INGENIERÍA DIRIGIDA POR MODELOS

De acuerdo con lo expuesto por Bran Selic en el artículo donde explica **MDE** en [165], y en los artículos donde habla de MDE y MDA en [24], [166], [170], los **modelos** deben de cumplir una serie de características para así conseguir que la **Ingeniería Dirigida por Modelos** sea un éxito y la gente no la rechace.

Una de las ventajas del uso de **MDE** es la **automatización**, pues es el método más efectivo para acelerar la productividad y calidad. Esto se debe a que, al **automatizar** un sistema, este puede generar programas completos a partir del modelo, así como verificar los modelos automáticamente. Esto es algo muy efectivo con el software desde que los ordenadores son capaces de crear programas complejos. No obstante, el requisito de la **automatización** viene dado de evitar que todo quede solamente documentado, pues, de ser así, es muy fácil que los resultados finales difieran de lo documentado. En cambio, si se realiza un proceso automático de creación de la aplicación a partir de la documentación, o, en este caso, de un **modelo**, se consigue que las aplicaciones sean fieles a lo documentado en el modelo. Esta automatización implica un incremento en la calidad del software desarrollado [201], algo que se lleva buscando desde los años 70 [185].

La primera técnica usada por el ser humano para superar la complejidad fue la **abstracción**. El uso de los **modelos** permite trabajar con un nivel mayor de abstracción para evitar trabajar ligados a la tecnología de implementación y a los lenguajes de programación. Esto hace que los **modelos** sean más fáciles de especificar, entender y mantener, a la vez que nos ofrecen información más comprensible y manejable. Es decir, la abstracción hace a los **modelos** independientes de la tecnología y de las evoluciones que esta tenga. Además, la **abstracción** permite ocultar detalles irrelevantes dados por un determinado punto de vista, y así, ayudar a entender el **modelo** de una forma más fácil. Incluso, con una **abstracción** suficientemente buena y bien realizada, se podría, en ciertos casos, utilizar a los expertos del dominio para realizar el trabajo en vez de utilizar a un equipo de desarrolladores. Poniendo un ejemplo, es mucho más fácil ver los diferentes estados de una máquina de un procesador en un gráfico de tipo árbol que ver los todos los posibles estados anidados en el propio código fuente. Este nivel de abstracción permite el uso de nuevos **lenguajes de dominio específico** (DSL) que permiten reflejar los conceptos del problema a la vez que nos abstraen de cómo están implementados. No obstante, como precisa Brooks [207], la abstracción del software podría abstraer su esencia, lo que puede provocar que se abstraiga la complejidad y esta se vea ocultada. Por ello, es importante cuando se realizan las abstracciones el no ocultar la complejidad esencial del sistema [191], pues esto podría desembocar en que no se subestimase el desarrollo del producto. No obstante, esta idea no es novedosa, pues la **abstracción** es algo que se lleva dando desde la introducción del lenguaje ensamblador para abstraer a los programadores de tener que programar utilizando código máquina [176], y que se verá más a fondo en el apartado 6.1.1.

La segunda característica es la **comprensibilidad**. Un buen modelo debe de ser fácil de entender para reducir el esfuerzo mental. Un **modelo** abstrae de los lenguajes de programación, que suelen ser inexpresivos

y requieren el conocimiento de ciertas reglas para así conseguir representar el contenido de una forma más sencilla y entendible. Esta característica permite que gente sin conocimiento de programación, pero con conocimiento del dominio del problema puedan reconocer el modelo al verlo.

La **precisión** es otra de las características que un **modelo** debe de poseer. Los **modelos** deben de ser precisos en su representación del sistema modelado. De nada sirve un modelo impreciso, pues con él lo que se haría serían aplicaciones que no sirviesen para el fin que deseamos.

La cuarta característica es la que permite a un **modelo ser predictivo**. Un **modelo** debe de permitir predecir correctamente el sistema modelado. Por ejemplo, para saber la carga máxima que soportará un puente es mucho mejor un modelo matemático que una maqueta. No obstante, esto depende de la **precisión** del modelo.

Finalmente, el precio del **modelo**, pues este debe de ser **económico**. Un **modelo** debe de ser bastante más **económico** de hacer que el coste de construir la aplicación final. En caso contrario, no merecería la pena crear un **modelo**, ya que resultaría más rentable realizar la aplicación final y corregirla o empezarla de nuevo.

5.3.8 TÓPICOS DE LOS PRAGMÁTICOS Y REQUISITOS DE UNA HERRAMIENTA QUE TRABAJE CON MODELOS

No todo son bondades y halagos para **MDE**. Como ocurre con todo lo bueno, siempre existen partes malas o negativas, ya que nunca pueden ser todo virtudes, sobre todo a la hora de la práctica. Estas críticas son recogidas por Bran Selic en [165], donde comenta los problemas de trabajar con **modelos** y cosas que las herramientas o los **modelos** deberían de permitir.

La primera de todas tiene que ver con el **rendimiento**. Los pragmáticos sostienen que hay dificultades técnicas al trasladar los modelos a software al preguntarse si el código será lo suficientemente rápido y compacto y si será una correcta representación del modelo. Sin embargo, estas preguntas fueron las mismas que se formularon cuando aparecieron los compiladores. Preguntas que fueron válidas en su época pero que ya casi nadie formula, pues casi nadie cuestiona los compiladores. Sin embargo, tras cierta experiencia en entornos industriales, Selic comenta que la eficiencia no es la prioridad y que, aquellas técnicas aplicadas en la construcción de compiladores pueden ser aplicadas en **MDE**. Además, es conocido que muchos compiladores superan en eficiencia a los profesionales y realizan incluso código más fiable. A esto hay que sumarle que, siempre, se vino dando la tendencia de conseguir una mayor abstracción a través de los lenguajes de programación. Esta abstracción tenía un impacto en el rendimiento, pero siempre se superpusieron a este impacto, debido a la flexibilidad que ofrecían, la reducción de la complejidad en la creación de aplicaciones y al aumento de productividad que esta abstracción suponía [247].

Continuando con el rendimiento y al igual que ocurrió con los compiladores al principio y ocurre ahora en MDE, puede que el rendimiento no sea el mejor y en algunos momentos existan aplicaciones críticas que lo necesiten. Por ello, un punto que deben de cumplir las herramientas que trabajen con **modelos** es permitir al usuario modificar el código para poder escribir o mejorar manualmente estos puntos críticos del sistema.

Otro problema surge en la **generación de modelos** a la hora de desarrollar con ellos, es cuando se debe de recibir los **errores**. Debido a la abstracción, estos son más difíciles de detectar, pues hay más capas y código que verificar. Un ejemplo de uso es la generación automática de código por medio del uso de plantillas («*templates*»). En todos estos casos, lo ideal es contar con buenas herramientas que informen de cualquier tipo de problema surgido.

Relacionado con la **generación de modelos**, surge un problema cuando se desarrollan. Las herramientas que trabajen con **modelos** deben permitir que los **modelos** puedan ser ejecutados aun cuando estos estén incompletos. Así, se puede simular como va evolucionando el **modelo** y se permite dirigirlo en la dirección adecuada sin tener que esperar a terminarlo. Es decir, permitir un **desarrollo ágil** de estos.

Además, los sistemas de **generación de modelos** deben de ser escalables. De nada sirve un sistema de estos si cuando un desarrollador modifica una parte del modelo, el sistema necesita una gran compilación, pues esto puede reducir la capacidad de desarrollo. Por ello, los **generadores de código** deben de ser capaces de minimizar la cantidad de **regeneración de código**. Según Selic, los sistemas más grandes tienen cientos de programadores y traducen los modelos en millones de líneas de lenguajes de programación estándar. Por ello, si cada vez que se realizase un cambio, hubiese que generar todo el programa, el uso de **MDE** podría ser contraproducente.

La **redundancia** obtenida en MDE debido a que uno de los principios es el tener múltiples vistas de un mismo objeto o varios niveles de abstracción para representar el mismo concepto [176].

El **mantenimiento**, pues, según [176], en verdad lo que se hace no es reducir la complejidad, sino moverla de sitio. Con esto se obtiene un programa más simple de utilizar, pero más difícil de mantener, pues se tiene el código y los procesos divididos en diferentes sitios o vistas, además de incrementar el número de artefactos creados, lo que implica tocar en varios sitios si se desea actualizar.

Otras críticas a MDE son las relacionadas a las aplicaciones en tiempo real que, a pesar de ser sistemas muy complejos, no aceptan el uso de MDE, pues creen que es inapropiado para poder representar la concurrencia y el tiempo con exactitud [172].

5.4 *ARQUITECTURA DIRIGIDA POR MODELOS*

MDA es un enfoque o acercamiento estándar de **MDE** creado por el OMG para mejorar el ciclo de desarrollo del software mediante el uso de **modelos** [28], [174], [178], y así ofrecer soporte el ciclo de vida completo de los sistemas informáticos, desde el modelado de los requisitos, hasta la implementación tecnológica, despliegue, gestión y evolución [171], [174], [234]. La promesa de MDA es permitir la definición de aplicaciones y modelos leíbles por máquinas de cara a permitir una mayor flexibilidad, integración, un mejor mantenimiento y una forma más fácil de realizar test y simulaciones [28]. Por esto, sus tres principales objetivos son la portabilidad, la interoperabilidad y la reusabilidad [28]. Todo esto desde unas especificaciones neutrales al vendedor [174]. Esto lo consiguió mediante la abstracción de las especificaciones usando diferentes modelos en diferentes **modelos** de abstracción, separándolos de la implementación final del sistema y automatizando la producción mediante su uso [178].

El objetivo principal por el que el OMG creó MDA fue para mejorar la productividad, la portabilidad, la interoperabilidad y la reutilización de los sistemas para hacerlos más manejables sin importar cuan grandes y complejos pudieran ser. Para ello, trata de mejorar la interacción entre las organizaciones, las personas, el hardware y el software mediante el uso de herramientas y procedimientos estandarizados. Así, mediante MDA se pueden representar los sistemas a cualquier nivel de abstracción y desde diferentes puntos de vista [234].

El nombre de este concepto viene de dos partes bien diferenciadas en su nombre, por las palabras «Dirigido por Modelos» y por la palabra «Arquitectura». Se usa «Dirigido por Modelos» (*Model-Driven*), debido a que se usan los **modelos** como base para dirigir todo el proceso de desarrollo [248]. Mientras, la palabra «Arquitectura», se debe a que se especifica cómo se han de hacer las cosas, como se conectan e interaccionan entre ellas, así como la definición de sus reglas [248]. Así, la idea básica de MDA es que, a partir del nivel de abstracción más alto, ir haciendo transformaciones automáticas o semiautomáticas hacia los niveles de menor abstracción. De esta forma, al final se llegará al código ejecutable por una máquina física o virtual. Este proceso de desarrollo de software hace que MDA sea considerado otra evolución en el desarrollo del software [28].

5.4.1 HISTORIA

El *Object Management Group*® (OMG®) es un consorcio internacional de empresas vendedoras de software y diferentes industrias, gobiernos y centros académicos fundado en el año 1989. El objetivo del OMG es desarrollar estándares tecnológicos para facilitar la integración entre empresas [249]. Uno de los objetivos del OMG fue el de ofrecer un *framework* conceptual para definir un conjunto de estándares que soportasen MDE [165].

El 8 de marzo del año 2000 el OMG anunció la adopción de la **Arquitectura Dirigida por Modelos**, conocida en inglés como **Model-Driven Architecture**® (**MDA**). No obstante, no fue hasta el 9 de Julio del año 2001 cuando sacaron la primera versión incompleta [174], [178], [250]. **MDA** es la iniciativa del OMG para soportar MDE a través de un conjunto de estándares abiertos [24]. Con esto pretendían desarrollar sus propios estándares de integración y recomendar una técnica de desarrollo de aplicaciones [247] mediante la interoperabilidad de estándares [251]. Por ejemplo, el núcleo de MDA está formado por varios estándares del OMG [174], como son *Unified Modeling Language*® (UML®) [240], *Meta Object Facility*™ (MOF™) [238], *XML Metadata Interchange* (XMI®) [229] y *Common Warehouse Metamodel*™ (CWM™) [251]. Hay que tener en cuenta que MDA siguió evolucionando y muestra de ello es la aparición 2.0 de MDA por parte del OMG en el año 2014 [234].

Así, con MDA, el OMG pasó de tener un enfoque basado en el middleware donde se especificaban los servicios SOA usando su estándar *Common Object Request Broker Architecture* (CORBA) [241], a un enfoque independiente de la plataforma. Con ello, consiguieron un Modelo Independiente de la Plataforma, de ahí su nombre *Platform-Independent Model* (PIM), el cuál es una de las partes más importantes de MDA [247].

No obstante, para conseguir desarrollar el estándar MDA, el OMG tuvo que acometer varias mejoras en otros estándares ya existentes. Este fue el caso del Lenguaje Unificado de Modelado, conocido como UML.

Muchos ingenieros simplemente usaban UML como ayuda para documentar el software y esto daba pie a que, en su primera versión, las herramientas automatizadas no pudiesen extraer todas las características de un diagrama UML. Luego, UML tenía una falta de precisión semántica que se requería para poder generar código [170]. Por ello, el OMG desarrollo UML2, mejorando el metamodelo que utilizaba UML para facilitar la extracción de información de un UML cuando fuera usado en un proceso MDA. Además, debido a su ubicuidad, se planeaba que UML fuese el lenguaje predominante de MDA [247].

No obstante, el OMG también tuvo que modificar otro estándar que convivía con UML, este fue MOF. MOF es un lenguaje estándar para definir DSLs sobre el que se basaban los lenguajes de modelado de MDA [24], [247]. El OMG liberó la segunda versión de MOF al mismo tiempo que UML2. MOF2 contiene también otras especificaciones adicionales, como es *Query, View & Transformation* (QVT) [252], la cual provee de mecanismos de transformación estándar entre modelos.

En el año 2003, el OMG empezó a trabajar en lo que llamaron el *Architecture Driven Modernisation* (ADM), de cara a poder recuperar los diseños originales de aplicaciones que no siguiesen el estándar MDA [247]. De esta manera el OMG buscó poder ayudar a portar aplicaciones antiguas de cuando no existía MDA o aplicaciones que se deseen portar actualmente para usar el proceso MDA.

5.4.2 LAS 4 VISTAS DE MDA

MDA se compone de tres **puntos de vista**, conocidos en inglés como ***Viewpoints***. Estos a su vez pueden tener diferentes **modelos** o **vistas**, llamadas ***Views***. Los **tres puntos de vista** de MDA son el punto de vista **independiente de computo**, el punto de vista **independiente de plataforma** y el punto de vista **específico de plataforma** [28]. Referente a las **vistas**, MDA define un total de cuatro niveles de **vistas** a partir de los **puntos de vista** que posee.

La primera **vista** es la del **Modelo Independiente de Cómputo**, conocida como ***The Computation Independent Model*** (**CIM**), y a veces llamada como modelo del dominio. Esta vista se corresponde con el punto de vista independiente de computo. Esta primera vista se desarrolla sin preocuparse del sistema operativo o hardware final. En esta capa se muestra que hará el sistema, pero sin mostrar los detalles de la estructura del sistema en que se desarrollará y sin tener en cuenta los aspectos informáticos. Esta capa se corresponde con el modelo del dominio de la aplicación, o, dicho de otra manera, los requisitos. Es decir, esta vista es independiente de cómo será implementado el sistema y evita las diferencias entre los expertos del negocio y los expertos en tecnología [28].

La segunda **vista** es la del **Modelo Independiente de Plataforma** o ***Platform Independent Model*** (**PIM**), la cual se correspondiente con el punto de vista independiente de la plataforma. Esta capa se encuentra un nivel por debajo de **CIM**, desde el punto de vista de reducción de la abstracción, y permite mostrar un modelo aplicable a diferentes plataformas. Una de las técnicas que se utilizan, de acuerdo al estándar, es serializar el PIM en XMI, que está basado en XML.

Otra técnica para lograr la independencia de plataforma es usar máquinas virtuales con neutralidad tecnológica, donde se entiende por máquina virtual un conjunto de proceso y servicios definidos para diferentes plataformas específicas, como ocurre con la *Java Virtual Machine* (JVM) de la plataforma Java o

la *Common Language Runtime* (CLR) [253], del framework Microsoft .NET [13] [254] o Mono [14]. De esta última forma, el modelo es dependiente de la plataforma específica de esa máquina virtual (JVM o CLR), pero es independiente de la plataforma de implementación de esa máquina virtual, pues no se ven afectados por la plataforma subyacente (Windows, Linux). Es decir, en vez de crear todo el resto del proceso de las 4 capas, sería utilizar una máquina virtual como son la CLR o la JVM para que obtener la dependencia de plataforma directamente, es decir, que sean independientes de si serán implementadas en Windows o Linux, debido a cómo funcionan las máquinas virtuales de los lenguajes de programación.

En la transformación de CIM a PIM, se crea un modelo pensando en un ordenador, pero independiente de la plataforma [28]. Es decir, PIM contiene las especificaciones formales de la estructura y funcionamiento del sistema [174]. Además, en esta capa y mediante el uso de estándares y por medio de diferentes reglas de transformación estándar, por ejemplo, utilizando QVT, se puede transformar código fuente de una plataforma específica a otra [248]. Ejemplos de esta capa son: *Unified Modeling Language* (UML) [240], *Meta-Object Facility* (MOF) [238], *Common Warehouse Metamodel* (CWN) [255], SysML [256] y *Ontology Definition Metamodel*™ (ODM™) [257].

La tercera **vista** es la del **Modelo Específico de Plataforma**, en inglés ***Platform-Specific Model*** (**PSM**) [28]. Esta tercera vista se corresponde con el punto de vista específico de plataforma. Esta vista añade a PIM las especificaciones necesarias para saber cómo este sistema utiliza una plataforma concreta. Si en la vista PIM el modelo era independiente de la plataforma, aquí se define exactamente que plataforma es. No obstante, esta todavía no es la implementación del sistema [29], pero debe de utilizar conceptos de la plataforma como son los mecanismos de excepción, tipos de parámetros y los componentes del modelo [174]. Al igual que ocurre con PIM, esta capa se suele serializar en XMI también. Ejemplos de esta capa son: CORBA, XML, XMI, *Common Language Infraestructure* (CLI) [15], *Java Specification Requests* (JSR) [16], *The Python Language Reference* (Python LR) [17] y los Servicios Web.

En el caso de CLI, este define tanto el código intermedio que debe ser generado por los lenguajes que quieran trabajar sobre la CLR, y que es conocido como *Common Intermediate Language* (CIL, IL - *Intermediate Language* o MSIL – *Microsoft Intermediate Language* [18]), como la máquina virtual CLR. En el caso de las JSRs estas definen las versiones y características de la plataforma Java, la cual incorpora tanto el lenguaje Java, como los *bytecodes* que deben generar todos los lenguajes que quieran trabajar sobre la JVM y las herramientas de desarrollo. En el caso de Python LR este describe la sintaxis y la semántica del lenguaje, mientras que The Python Standard Library [19] (Python SL) describe la librería estándar distribuida con Python. Así, los lenguajes que quieran trabajar sobre la CLR, como C# [258], [259], F# [20] [260], [261] y StaDyn [21]

[13] https://www.microsoft.com/net
[14] http://www.mono-project.com/
[15] http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf
[16] https://www.jcp.org/en/jsr/overview
[17] https://docs.python.org/3/reference/index.html
[18] https://weblogs.asp.net/kennykerr/introduction-to-msil-part-1-hello-world
[19] https://docs.python.org/3/library/index.html
[20] https://blogs.msdn.microsoft.com/somasegar/2007/10/17/f-a-functional-programming-language/
[21] http://www.reflection.uniovi.es/stadyn/index.shtml

[262], [263] deben de cumplir la CLI, los lenguajes que quieran trabajar sobre al JVM, como son Java [264], Scala [265], [266] y Groovy [267], [268] deben de cumplir la JSR, y las implementaciones de Python deben de cumplir las Python LR y Python SL, como es el caso de IronPython [22] [269] (cumple Python LR, Python SL y la CLI), Jython [23] (cumple Python LR, Python SL y la JSR) y CPython [270] (cumple Python LR y Python SL).

Por último, la **vista** del **Modelo Específico de Implementación**, ***Implementation-Specific Model***, ***ISM***, **Implementación**, o según el estándar MDA, ***Platform Model*** [23], [28]. Este es el nivel de abstracción más bajo. En él se define el código fuente específico del sistema para un entorno determinado creado a partir del PSM correspondiente, aunque a veces en algunas implementaciones se pasa directamente de la capa PIM a la capa ISM sin pasar por la capa PSM, ahorrándose así la generación del PSM. Algunos ejemplos de esta capa serían las implementaciones referentes a los PSM presentados antes: C#, F#, IronPython, ensamblador (ASM), del cual existen diversas versiones como pueden ser HLASM de IBM [271], x86 [24], IA-64 e IA-32 [25], o incluso para videoconsolas como la Nintendo NES con NESHLA [26], Java, Scala, Jython, CPython, Ruby [244], [245], PHP Hypertext Preprocessor (PHP) [272] y C++ [273], entre otros muchos Lenguajes de programación existentes. Es decir, aquellas implementaciones específicas existentes de los diferentes PSM.

En la Ilustración 20, se muestra un gráfico que contiene las cuatro capas del estándar MDA junto a ejemplos de las diferentes tecnologías que pertenecen a cada capa.

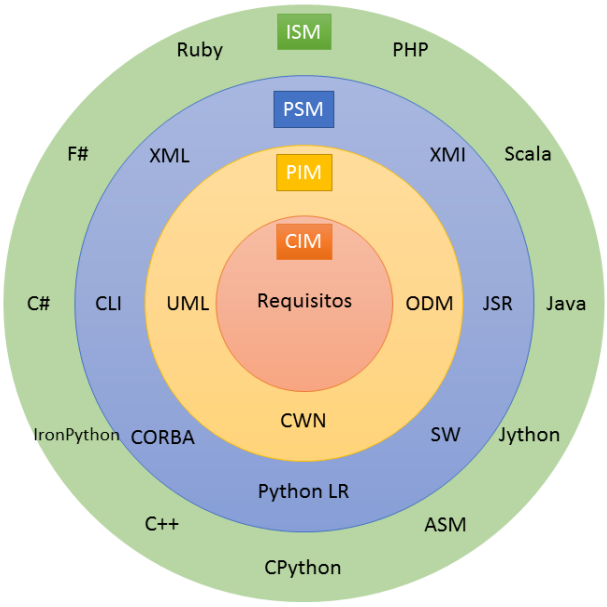


Ilustración 20 Vista de las cuatro capas que forman el estándar MDA

[22] https://blogs.msdn.microsoft.com/hugunin/2006/09/05/ironpython-1-0-released-today/
[23] http://www.jython.org/
[24] http://www.cs.virginia.edu/~evans/cs216/guides/x86.html
[25] http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf
[26] http://neshla.sourceforge.net/

En MDA, las transformaciones entre modelos son conocidas también como «*mapping*». Estas son definidas como un conjunto de reglas para modificar un modelo convirtiéndolo en otro y así realizar transformaciones de una capa a otra o entre la misma capa. Existen dos tipos de transformaciones, las verticales y las horizontales. Las verticales son todas aquellas que siguen el orden de las capas, de izquierda a derecha según la Ilustración 21, es decir, de CIM a PIM, de PIM a PSM o de PSM a ISM. Un ejemplo de esto sería transformar nuestro modelo Ecore en la plataforma JEE a Java.

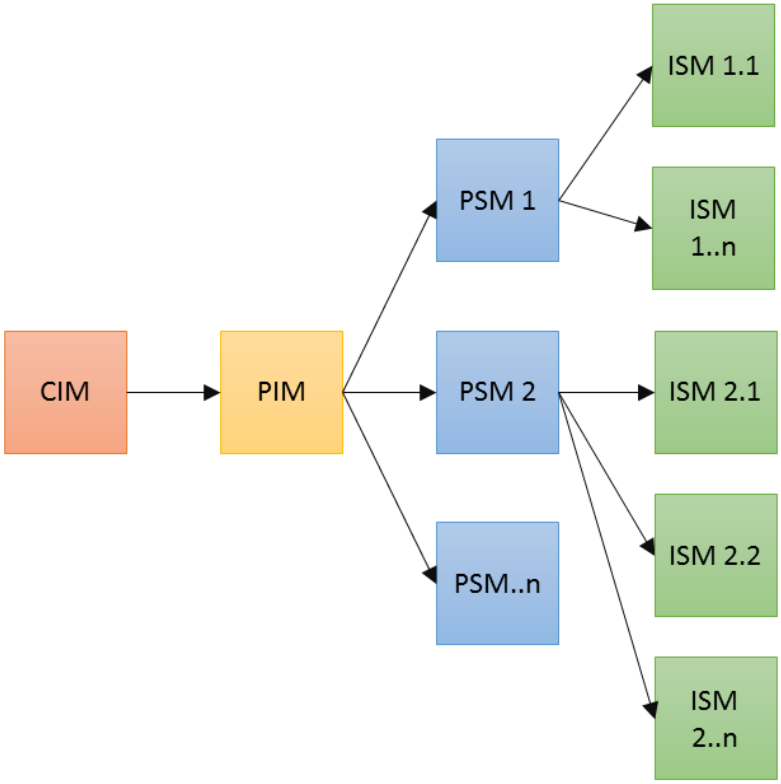


Ilustración 21 Orden de las vistas del estándar MDA

Por otro lado, tenemos las transformaciones horizontales que son aquellas que se hacen entre modelos de la misma vista. Estas son crear a partir de un PIM otro PIM, de un PSM otro PSM o de un ISM otro ISM. En este caso se transformaría un modelo que ya tuviésemos en otro modelo similar, ya sea este un cambio metamodelo, como puede ser un UML a CWN, un cambio de plataforma como podría ser de .NET a JEE o de una implementación a otra como podría ser de C# a F#.

5.4.3 ESTUDIOS DE VIABILIDAD

Un estudio informal realizado en el año 2006, demostró que, utilizando MDA en vez del uso tradicional de desarrollo de software se obtenían dos beneficios clave. Con MDA se producían tres veces más líneas de código por dólar gastado y además se cometían menos de un tercio de errores [247]. Esto representa un decremento en el coste del software o un aumento de beneficios en la empresa, depende del punto de vista, así como gran mejora en la calidad del código.

Mientras, otros estudios dicen que se gasta más de la mitad del tiempo solamente en intentar entender cómo funciona el código que se quiere arreglar [274]. Otro estudio del año 2003 y realizado sobre aplicaciones J2EE demostró que la productividad relacionada al mantenimiento del código aumentó en un 37% debido al uso de MDA [247].

En la página oficial de MDA se pueden ver los diferentes informes de empresas importantes que utilizaron MDA con éxito y que explican la historia de su experiencia de uso [275].

5.4.4 PROBLEMAS PARA SU INTRODUCCIÓN EN EL MUNDO LABORAL

A pesar de los aparentemente buenos resultados mostrados en el punto anterior, no toda la gente está a favor del uso de MDA. Los que están en contra alegan que está alejado de ser un estándar usable debido a que está alejado de la realidad práctica [24], es decir, de lo que la empresa requiere en estos momentos. Por ejemplo, en el año 2003, según las estimaciones de Bran Selic, los **modelos** estaban siendo utilizados únicamente por un 10% de las empresas [24].

Diversas causas por las que puede que MDA no esté siendo adoptado por las empresas puede ser por motivos técnicos, culturales o económicos. Un ejemplo de esto es la **usabilidad** de las herramientas, pues a pesar de ser muy potentes, suelen ser difíciles de aprender a utilizar. Esto puede ser debido a que estas no suelen ser diseñadas por gente que utiliza MDE y entonces no pueden tener en cuenta a la hora de diseñarlas lo que estas necesitan ofrecer [24]. Entre estas herramientas nos encontramos con las existente en el entorno de desarrollo integrado (IDE) y la comunidad Eclipse, exactamente en el proyecto *Eclipse Modeling Framework* (EMF) [176], [239], que se centra en trabajar con los estándares del OMG para así acercarlos a la gente [276].

Además, el estándar MDA, que es el enfoque más famoso y utilizado, presentó dos problemas que hacían difícil su o ambigua su comprensión. No especificaba como tenían que ser los CIM y como se debía hacer la transformación de CIM a PIM [199], [205].

Como **problema cultural** nos encontramos con que todavía no existen suficientes profesionales concienciados del potencial de MDE que sean capaces de explotarlo. Además, la introducción de MDE en un mercado altamente competitivo como es el actual, es difícil de aceptar debido al tiempo que se necesita para ser implementarlo en un proyecto. A esto hay que sumarle la mentalidad conservadora de algunos profesionales que prefieren seguir utilizando una tecnología en la que invirtieron mucho tiempo en aprender en vez de renovarse hacia tecnologías nuevas. En resumen, se puede decir que hay una gran resistencia al cambio tecnológico a pesar de que las nuevas tecnologías puedan ofrecer mejores soluciones [24].

Económicamente, la introducción de MDE conlleva ciertos gastos en los proyectos debido a la portabilidad de herramientas y código que esto implica hacer, así como el gasto derivado en volver a enseñanza a los miembros del equipo el uso de las nuevas tecnologías. Esto hace que la introducción de MDE haya que planificarla a largo plazo, lo que es inaceptable para muchos proyectos que son valorados a corto plazo o bien son de pequeña envergadura. Además, claro está, el uso de MDE no asegura el éxito de estos proyectos [24].

Por otro lado, Martin Fowler critica MDA diciendo que en verdad al meter tanto estándar no se consigue independencia sino dependencia de ellos. Además, estos, al no contar con bibliotecas de entrada-salida hacen que debas de programarla tú, lo que implica un desarrollo no estándar. Por ello, según él, aunque MDA no sea una pérdida de tiempo si lo es la excusa de la plataforma independiente [277].

Debido a estos problemas, se está tardando en adoptar el uso de MDE en las empresas y puede que se tarden unos años en adoptarla [24].

6. LENGUAJES DE PROGRAMACIÓN

«Hay solo dos clases de lenguajes de programación:
aquellos de los que la gente está siempre quejándose y aquellos que nadie usa»
Bjarne Stroustrup

«Un lenguaje de programación, con su sintaxis formal y las reglas de demostración que define su semántica, es un sistema formal para el cual la ejecución del programa provee solamente un modelo»
Edsger W. Dijkstra

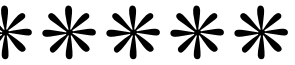
«No trabajé duro para hacer Ruby perfecto para todo el mundo, porque todos somos diferentes. Intenté hacer Ruby perfecto para mí, así que puede que a ti no te lo parezca; probablemente, el mejor lenguaje para Guido van Rossum es Python»
Yukihiro Matsumoto, aka «Matz», creador de Ruby

En nuestra vida diaria utilizamos diferentes lenguajes para comunicarnos entre las personas. Estos lenguajes se conocen como lenguajes naturales. Gracias al lenguaje natural nos podemos comunicar con cualquier persona, siempre que esta conozca el conjunto de símbolos y reglas de ese lenguaje. Estos lenguajes surgieron espontáneamente bajo el propósito de poder comunicar a la gente con sus semejantes, ya fuese de forma hablada o escrita.

Sin embargo, con el paso del tiempo, todo evolucionó. En esta evolución surgieron las máquinas, los ordenadores y, con ellos, la necesidad de comunicarnos y enviarles nuestros deseos. Por eso, junto a los ordenadores surgieron, poco a poco, lo que se conoce como lenguajes de programación. Lenguajes que fueron evolucionando y cambiando para facilitar la tarea de enviarles nuestras órdenes y hacerla de una forma más sencilla y eficaz.

Así, mediante el uso de un **lenguaje de programación**, los humanos u otros ordenadores pueden crear programas que sean ejecutados en ordenadores para obtener un determinado propósito, ya sea bien, desde crear una aplicación de gestión hasta incluso otorgar Inteligencia Artificial a la propia máquina.

Este capítulo hablará de los lenguajes de programación existentes, exactamente, explicando la diferencia entre los lenguajes de propósito general y los lenguajes de dominio específico, hablando en este transcurso de su historia y los diferentes subtipos existentes.



6.1 *LENGUAJES DE PROPÓSITO GENERAL*

Los **Lenguajes de Propósito General**, también conocidos como GPL por sus siglas del inglés *General Purpouse Language*, son lenguajes creados para poder resolver casi cualquier tipo de problema. Con los GPLs, se pueden crear soluciones para cualquier tipo de aplicación. Por ejemplo, con ellos se puede crear un sistema operativo, aplicaciones web o móviles, páginas webs, comunicaciones entre ordenadores y mucho más.

Algunos de estos ejemplos son C [215], [278], C# [258], [259], C++ [273], Java [264], JavaScript [279], Objective-C [280], [281], PHP Hypertext Preprocessor (PHP) [272], Python [270], Ruby [244], [245] y Swift [282], [283], pues hay decenas de lenguajes de diferente tipos y con diferente nivel de abstracción. Los más usados actualmente y en las diferentes épocas pueden ser consultadas en el ranking de facto del índice TIOBE [284], o bien, en el ranking creado sobre las estadísticas de los lenguajes utilizados en los repositorios de la empresa GitHub en [285].

6.1.1 GENERACIONES DE LOS LENGUAJES DE PROGRAMACIÓN

Los **GPL** pueden clasificarse por su nivel de abstracción. Estas se encuentran ligadas a las diferentes etapas de la programación. En total existen cinco generaciones.

En la **primera generación** se encuentra el **código binario**, también conocido como **código máquina**. En esta generación, los desarrolladores debían programar directamente en ceros y unos lógicos [247]. Por esto mismo también es conocido como lenguaje máquina. Esto implicaba que las instrucciones fueran muy parecidas al estar compuestas solo por la combinación de dos números, además de ser extremadamente largas. Además, para los humanos el programar en código máquina es complejo y propenso a cometer errores.

A continuación, se muestran varios ejemplos de código máquina con su respectiva traducción en decimal y la descomposición correspondiente por registros de la instrucción utilizada. Estas instrucciones pertenecen al set de instrucciones del *Microprocessor without Interlocked Pipeline Stages* (MIPS) de 32 bits. En el Código fuente 1 se muestra una suma de registros, en el Código fuente 2 la carga en el registro de un valor, y en el Código fuente 3 un salto a la dirección 1024.

	Operación	**Registro S**	**Registro T**	**Registro D**	**(Shamt) Desplazamiento**	**Función**
Decimal	0	1	2	6	0	32
Binario	000000	00001	00010	00110	00000	100000

Código fuente 1 Suma del Registro S con el Registro T y almacenamiento del resultado en el Registro D usando una instrucción de tipo R (*Register*)

	Operación	Registro S	Registro T	Dirección / Inmediato
Decimal	35	3	8	68
Binario	100011	00011	01000	00000 00001 000100

Código fuente 2 Carga un valor, en el registro 8, que se encuentra en la celda 68 después de la dirección guardada en el registro 3 usando una instrucción de tipo I (*Immediate*)

	Operación	Dirección de destino
Decimal	2	1024
Binario	000010	00000 00000 00000 10000 000000

Código fuente 3 Salto a la dirección 1024 usando una instrucción de tipo J (*Jump*)

En la **segunda generación** se encuentran los **lenguajes ensamblador** [176]. Los ensambladores son lenguajes de bajo nivel que sirven para simplificar la programación de código máquina en unas Unidades Centrales de Procesamiento (CPU) concretas. El principal objetivo de los lenguajes ensamblador fue el de liberar al programador de tener que recordar posiciones de memoria y códigos de instrucciones, otorgándoles un nivel de abstracción más alto [176], [247]. Estas instrucciones simplificaban la escritura del programa al incorporar «palabras reservadas», conocidas como mnemónicos, que en el proceso de compilación se traducían a las instrucciones en código máquina que estas representaban. Así, junto al uso de los mnemónicos para representar las instrucciones, también se incorporó el empleo de símbolos para representar los datos y direcciones. Esto simplificó mucho los programas e hizo dar un gran salto en el desarrollo de software pues permitía construir un programa estructurado y utilizar funciones para así simplificar el código y hacerlo reusable. En este grupo podemos encontrar los lenguajes x86-16 [27], x86-32 o IA-32 [28] y AMD64 [29], x86-64 o IA-64 [30]. En el Código fuente 4 se muestra un ejemplo de código ensamblador x86-32 bajo Win32.

[27] https://en.wikipedia.org/wiki/X86_instruction_listings#Original_8086.2F8088_instructions
[28] http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf
[29] http://developer.amd.com/resources/developer-guides-manuals/
[30] http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf

```
; El registro ESI se utiliza para apuntar a los elementos de 'datos'
  ; Se carga el registro ESI con el parámetro adecuado
  MOV ESI, [EBP + 12]

  ; El registro ECX se utiliza como contador del bucle
  ; Se carga ECX con el parámetro adecuado
  MOV ECX, [EBP + 8]

  ; El registro EAX se utiliza como acumulador de la suma
  ; Se resetea el registro EAX
  XOR EAX, EAX

  ; Bucle de procesamiento
bucle:
  ADD EAX, [ESI]
  ; Hacer que ESI apunte al siguiente elemento
  ADD ESI, 4; incrementa el registro ESI

    LOOP bucle
```

Código fuente 4 Ejemplo de un bucle realizado en x86-32 bajo Win32

A finales de 1950 apareció **la tercera generación** [165], la cual trajo consigo los lenguajes de alto nivel o procedimentales. Estos lenguajes simplificaron y abstrajeron más aún la programación de forma que esta fuese más «similar» al lenguaje humano y simplificase todo el proceso, como ocurrió con los compiladores, el lenguaje ensamblador IBM 704 y la aparición de Fortran [28]. Estos lenguajes poseían un código más sencillo, comprensible y menos complejo al necesitar menos líneas de programación para realizar el mismo programa. Por ejemplo, por medio de estos lenguajes de programación, el desarrollador no tenía que ocuparse de la localización de la memoria o de otras acciones a bajo nivel también otras abstracciones como la asignación de variables [176]. Además, permitían utilizar el mismo código para crear la misma aplicación para diferentes CPUs y sistemas operativos. Otra mejora en esta generación fue el poder utilizar diferentes paradigmas de programación. No obstante, estos lenguajes no eran tan eficientes, ni a nivel de rapidez ni a nivel de memoria, como los lenguajes ensamblador. Algunos ejemplos de lenguajes de esta generación son Algol (desde su nacimiento en Algol 58 [286], pasando por su versión Algol 60 [287], y que no hay que confundir con sus dos vertientes predecesoras, Algol 68 [288], [289] y Algol W [290]), C [215], [278] (Código fuente 5), Cobol [291], [292], Fortran [293], [294] y Pascal [295].

```c
#include <stdio.h>

int main(){

    int i;
    char c = ' ';

    printf("Hola Cris!\n");

    for(i = 0 ; i<14 ; i++){
        printf("%d\n %c", i, c);
    }

    for(; i>0 ; i--){
        printf("%d\n", i);
    }

    return 0;
}
```

Código fuente 5 Código realizado en C que muestra una condición, un bucle e imprime por pantalla

La **cuarta generación** trajo consigo los **lenguajes orientados a objetos**. Estos trajeron consigo en los años 1970 la **programación orientado a objetos** (**OOP**). No obstante, su usó no se popularizó hasta el año 1990, aun siendo el primer lenguaje orientado a objetos del año 1966, Simula [296]. Los lenguajes de esta generación permitieron que el código fuese más fácil de organizar y de reutilizar. Además, estos poseían una abstracción mayor respecto a las anteriores generaciones, lo que hacía que fuese más fácil y rápido programar, además de que los desarrolladores cometiesen menos errores, lo que aumentó la productividad notablemente. Algunos lenguajes OOP son C#, C++, Eiffel [297], Java, JavaScript, Objective-C, PHP a partir de la versión 5, Python, Ruby (Código fuente 6), Smalltalk [298], StaDyn [31] [262], [263] y Simula [296].

[31] http://www.reflection.uniovi.es/stadyn/index.shtml

```ruby
class DispositivosController < ApplicationController

  # Método que recibe un XML del móvil, lo registra y confirma
  def register
    result = Hash.from_xml(params[:xml])

    if result["registro"]["servicio"].kind_of?(Array)
      result["registro"]["servicio"].each do |i|
        p "Imprimo el ID: "+i["idservicio"]
        Servicio.create(:descripcion => i["descripcion"])
      end
    else
      if (result["registro"]["servicio"]).nil?
        Rails.logger.debug("Sin servicios");
      else
        Servicio.create(:descripcion =>
result["registro"]["servicio"]["descripcion"])
      end
    end

    # Llama al método confirmation
    confirmation
  end
end
```

Código fuente 6 Código realizado en Ruby que muestra una condición, un bucle e imprime por pantalla

La **quinta generación**, en cambio, es un poco polémica debido al poco consenso sobre que lenguajes pertenecen a ella. Algunos autores, sostienen que es la generación de la **programación declarativa**, ya que permite crear programas describiendo que se va a computar y no cómo se va a computar, pues lo que se hace es describir el problema detallando como debe ser la solución. Este modelo está basado en el modelo humano. El primer lenguaje funcional surgió en 1958 [299], con Lisp [300]. Otros lenguajes pertenecientes a este grupo son Clojure [301], Erlang [302]–[304], Haskell [305], [306] (Código fuente 7) y Prolog [307], [308]. Posteriormente, surgieron los DSLs, que pertenecen a este grupo y del que podemos destacar algunos ejemplos como CSS (*Cascading Style Sheets*) [309], HTML (*HyperText Markup Language*) [310] y SQL (*Structured Query Language*) [311].

```haskell
-- Criba de Eratóstenes (de una lista dada [2..n] te deja solo los
números primos)
eratostenes :: [Int] -> [Int]
eratostenes [] = []
eratostenes (x:xs) = if x^2 > head(reverse xs) && length (xs) >= 1
then (x:xs) else x: eratostenes([y | y <- xs, (mod y x /= 0)])
```

Código fuente 7 Criba de Eratóstenes en Haskell

6.1.2 FAMILIAS DE LENGUAJES DE PROGRAMACIÓN

Un **paradigma de programación** es una colección de patrones conceptuales que juntos diseñan el proceso de diseño y determinan la estructura del programa, su forma y como se llega a la solución [312]. Los lenguajes de programación se pueden englobar según el paradigma al que pertenecen. De esta manera, quedan englobados según las normas, prácticas y enfoque para diseñar soluciones, difiriendo, así, en como abstraen los elementos del problema a resolver y los pasos necesarios para dar con la solución. Se puede decir que existen dos paradigmas generales y el resto son subparadigmas de estos primeros, que son la **programación imperativa** y la **programación declarativa**.

Cabe destacar que existen **lenguajes híbridos** y que pueden **tener funcionalidades de los diferentes paradigmas**, pues es raro que un lenguaje de programación tenga solo un paradigma [312]. De esta manera, un lenguaje imperativo puede tener características del paradigma funcional o del lógico, y viceversa. Es muy común, sobre todo en los últimos años, que muchos lenguajes orientados a objetos añadan características funcionales, como es el caso de C# y Java.

Programación imperativa: este paradigma tiene en cuenta el orden de la secuencia de instrucciones del programador a la hora de resolver el problema. Es decir, describe los estados posibles por los que va a pasar el programa y utiliza diferentes sentencias para dirigir el programa según el estado en el que se encuentre. Así, lo que se crea es un algoritmo que describe los pasos necesarios para solucionar el problema. Este paradigma es el mejor representado por las máquinas Von Neumann, pues utiliza el modelo de esta máquina para conceptualizar sus soluciones [312]. Ejemplos de lenguajes de este paradigma son los lenguajes máquina, lenguajes ensamblador, C, Cobol, Fortran y Pascal, entre otros. El paradigma opuesto es el declarativo.

Programación estructurada: este subparadigma y evolución de la **programación imperativa** buscó mejorar la claridad y la calidad del desarrollo software y a reducir su tiempo. En este subparadigma se utilizan subrutinas, *identación*, variables locales, los bloques de código y diferentes estructuras de control como el *if* y *switch* para la selección de elementos y los bucles *for* y *while* para iterar [313]. Ejemplos de lenguajes de este paradigma son Ada [214], Algol, C y Pascal.

Programación Orientada a Objetos (**OOP**): actualmente es el paradigma más extendido y usado, y que además pertenece al género imperativo y ser la evolución del estructurado. La idea de OOP es construir software descomponiendo los problemas en objetos y trasladando esta separación de objetos al código fuente para así abstraer en estos objetos el comportamiento y los datos en una entidad conceptual [314]. La OOP posee características como la herencia, la abstracción, el polimorfismo, el encapsulamiento, la ocultación y los recolectores de basura. C#, C++, Java, Ruby, Simula y Smalltalk son algunos de los lenguajes de este paradigma. A aquellos lenguajes que tratan todo como objetos, se les llama puros, siendo dos ejemplos de estos Smalltalk y Ruby.

Programación basada en prototipos: esta aproximación es similar a la OOP [312]. Una de las diferencias es que, en vez de crear los objetos como instancias de una clase, se hace mediante la clonación de otros objetos, es decir, los objetos existentes sirven de prototipos para los nuevos. Algunos lenguajes de la programación basada en prototipos son JavaScript y Self [315].

Programación declarativa: en este paradigma se describe el problema declarando sus propiedades y reglas en vez de darle las instrucciones que debe de realizar para cada estado del problema, como sucede en la programación imperativa, que es lo contrario. Por ello, los algoritmos que se crean en la programación declarativa contienen la descripción del problema diciendo exactamente lo que se quiere resolver y sin especificar el cómo. Ciertos lenguajes declarativos son Haskell, Lisp y Prolog.

Programación funcional: es un subparadigma de la **programación declarativa**. En la programación funcional, el elemento central son las funciones que intervienen en el problema a resolver. Cuando estas funciones solo permiten funciones matemáticas sin efectos laterales, se dice que el lenguaje es puramente funcional. Estos lenguajes poseen como característica las *funciones de orden superior*, es decir, pueden tener funciones como atributo y/o pueden devolver funciones. En este paradigma se encuentran lenguajes como Erlang, Haskell, ML [316], Lisp y Scheme [317].

Programación lógica: el pionero de este paradigma fue Kowalski [312], siendo el siguiente el artículo donde lo presentó [318]. El punto central de este paradigma es la descripción de las relaciones que intervienen en el problema, de manera que, en base a unas ciertas reglas se puedan deducir otros hechos [312]. Este paradigma es un subparadigma de la **programación declarativa**. Lenguajes de programación que pertenecen a esta categoría son: Gödel [319], Mercury [320]–[322] y Prolog.

En la Ilustración 22 se muestra un esquema que contiene los paradigmas explicados en esta sección junto con ejemplos de ellos.

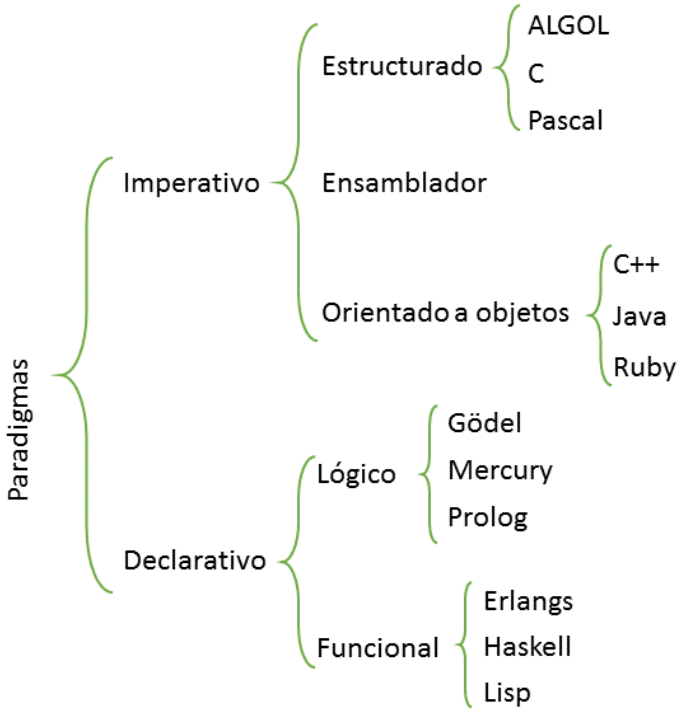


Ilustración 22 Esquema de los paradigmas de lenguajes de programación

Estos no son todos los paradigmas existentes, pues existen muchos más, pero si se pueden considerar los principales debido a su extenso uso en los diferentes lenguajes de programación o al extenso uso dado a los

lenguajes de programación pertenecientes a ellos. Otros ejemplos de paradigmas existentes son el de la programación dirigida por eventos o la programación visual.

6.2 *LENGUAJES DE DOMINIO ESPECÍFICO*

A veces, las soluciones genéricas no siempre son las más óptimas. Este es el problema de los GPLs, pues son genéricos y por esto surgieron lenguajes específicos para crear soluciones optimizadas y conseguir una mejor adaptación a un problema determinado [323]. Estos son los **Lenguajes de Dominio Específico**.

Un Lenguaje de Dominio Específico o DSL es un lenguaje, normalmente declarativo que hace llamadas a subprocesos y que son desarrollados para permite solucionar un determinado problema específico en un dominio determinado [324], además de caracterizarse por tener un gran poder expresivo [30], [232], [325].

Los DSLs abstraen el nivel de conocimientos y permiten reducir la dificultad encontrada en el uso de las APIs, lo que permite que los usuarios finales puedan trabajar sobre modelos sin necesidad de tener que tener conocimiento específico y complejo sobre el software o las APIs [171]. Como se vio en el capítulo anterior, los DSLs son uno de los pilares principales de la **Ingeniería Dirigida por Modelos** [232], y por ello, en estos casos, deben de estar basados en un metamodelo formal para así trabajar correctamente con MDE.

Los **Lenguajes de Dominio Específico** son muy importantes en la **Ingeniería Dirigida por Modelos** [232], pues forman parte de ella y de las soluciones que esta provee como se mostró en el capítulo 5. No obstante, los DSLs no son algo nuevo, pues uno de los primeros DSLs existentes fue *Automatically Programmed Tools* (APT), que se empezó a desarrollar en el año 1957 y se presentó a la prensa el 15 de febrero del año 1959 en el *Massachusetts Institute of Technology* (MIT) [326]. APT permitía programar máquinas de control numérico, y con el tiempo y un poco más de elaboración, APT se convirtió en un estándar internacional [326].

6.2.1 EJEMPLOS DE LENGUAJES DE DOMINIO ESPECÍFICO

Aunque no lo parezca, llevamos utilizando DSLs durante mucho tiempo [324], pues, como se ha mencionado, APT se presentó en el año 1959 [326]. A continuación, se muestra en la Tabla 1 varios ejemplos de DSLs existentes, pues, la construcción de una tabla con todos los DSLs existentes hasta la fecha sería imposible. Por este motivo, se muestran los más utilizados y conocidos.

Lenguajes de programación

Acrónimo	Nombre completo	Descripción
APT	*Automatically Programmed Tools*	Programar máquinas de control numérico [326]
BNF	*Backus–Naur Form* o *Backus-Normal Form*	Especificación de gramáticas libres de contexto [287]. También es conocida como *Backus-Normal Form* [286], aunque Naur modificó la original de Backus. El cambio de nombre fue sugerido Por Donald Knuth en [327]
Bitbloq		Creación de aplicaciones para el microcontrolador de BQ [328]
BPMN	*Business Process Management and Notation*	Creación de procesos de negocio [329]
CSS	*Cascading Style Sheets*	Presentación de las páginas web [309]
Excel		Hojas de cálculo [32]
EBNF	*Extended Backus–Naur Form*	Especificación de gramáticas libres de contexto [330]
HTML	*HyperText Markup Language*	Estructuración de páginas web [310]
LaTeX		Edición de texto [33]
Make		Procesos de «*build*» de software [331]
miniBloq		Creación de aplicaciones para Arduino [328]

[32] https://support.office.com/es-es/excel?ui=es-ES&rs=es-ES&ad=ES&fromAR=1
[33] https://www.latex-project.org/

Acrónimo	Nombre completo	Descripción
Plantillas de MediaWiki		Maquetación de texto [34]
SGML	*Standard Generalized Markup Language*	Definición de lenguajes de marcas [332]
SQL	*Structured Query Language*	Consultas a bases de datos [311]
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*	Diseño de Hardware [333]

Tabla 1 Ejemplos de DSLs

6.2.2 CLASIFICACIÓN DE LOS LENGUAJES DE DOMINIO ESPECÍFICO

Existen varias clasificaciones para organizar los DSLs pues estos pueden clasificarse según como se manipulen, según su lenguaje padre, acorde a como se construyan y según el tipo de cliente.

6.2.2.1 DESDE EL PUNTO DE VISTA DE LA MANIPULACIÓN DEL LENGUAJE

Una clasificación para los DSLs es aquella que se basa en según como el usuario final interactúe con ellos. Así, estos se pueden clasificar, a grandes rasgos, entre **textuales** y **gráficos** según como el usuario final interactúe con ellos. No obstante, también se pueden crear DSLs en forma de **árbol**, **formularios** o **tablas**.

Los **DSLs textuales** son aquellos que necesitan ser escritos y están formados por un conjunto ordenado de sentencias. Algunos de estos DSLs son CSS, HTML y SQL (Código fuente 8).

```
SELECT email FROM users WHERE email='iamanemail@internet.com';
INSERT INTO users(nombre_cuenta, clave) VALUES
('superhacker','yu0w0ntbr34kmyp44sw0rd');
DROP TABLE users;
UPDATE users SET password='alphanumeric with_special-characters!'
WHERE password='123456';
```

Código fuente 8 SQL es un DSL textual para consultar bases de datos

Estos DSLs se pueden construir de diversas formas. La primera posibilidad existente para construir un **DSL textual** es construyendo el lenguaje de programación. Es decir, primero se construye la gramática, por ejemplo, utilizando la notación *Backus-Naur Formalism* (BNF) [287], [327] y tras esto, construir el *parser*, por ejemplo, con la ayuda de Antlr [334], [335], Bison [336], [337] o Yacc [338], [339]. La segunda manera es utilizando XML. No obstante, en este caso estaríamos restringidos a su sintaxis, pero obtendríamos, como beneficio, el poder utilizar todas las herramientas que posee este estándar.

[34] https://www.mediawiki.org/wiki/Help:Templates/en

Los **DSLs gráficos** son aquellos que permiten a los usuarios trabajar con una notación gráfica, la cual consiste en conectores y figuras [220]. Ejemplo de ellos son Bitbloq [328] *Business Process Management and Notation* (BPMN) [329] y miniBloq [340]. La Ilustración 23 muestra un ejemplo de DSL gráfico utilizando BPMN2. Normalmente, en estos DSLs, las clases del metamodelo suelen corresponderse con un artefacto de tipo clase del modelo (ver apartado 5.1).

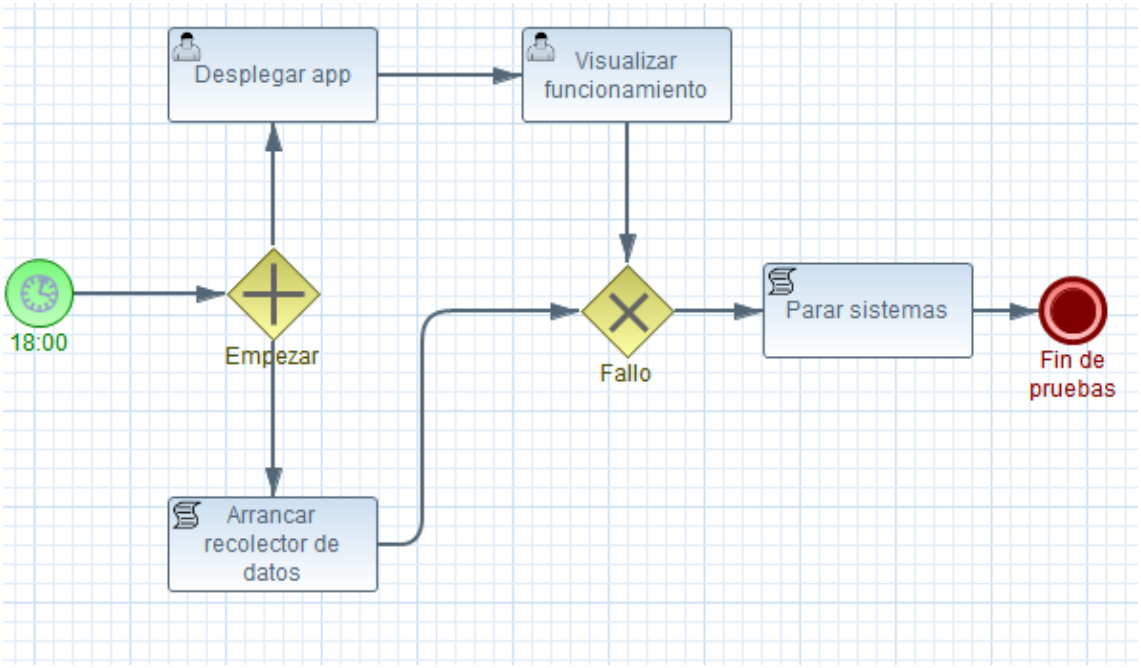


Ilustración 23 Proceso de negocio creado con BPMN2 en Eclipse usando el plugin v1.2.4

Por otro lado, los DSLs en forma de **árbol**, **formularios** o **tablas** utilizan, como su nombre indica, árboles para representar el modelo, formularios o bien tablas. La construcción de estos suele ser bastante parecida a la de los **DSLs gráficos**. Por ejemplo, *Eclipe Modelling Framework* posee la posibilidad de crear DSL en forma de árbol como se puede observar en Ilustración 24.

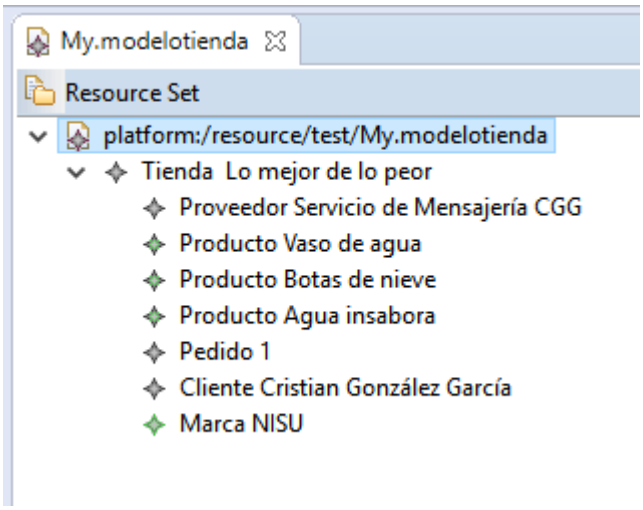


Ilustración 24 Un ejemplo de DSL en forma de árbol creado utilizando Eclipse Modelling *Framework*

6.2.2.2 *DESDE EL PUNTO DE VISTA DE SU LENGUAJE PADRE*

Normalmente, los DSLs son divididos en dos grandes grupos [324], [341], [342]. Pueden ser externos o internos.

Los **DSLs externos** son aquellos que son escritos sin estar atados al lenguaje de programación principal. Estos pueden ser desarrollados utilizando herramientas como es la de lenguaje textual *Xtext* para Java [343] o el lenguaje gráfico DSL Tools para .Net [344], o bien utilizando ficheros separados como puede ser XML [341] y cuyo ejemplo se muestra en el Código fuente 9. La ventaja de estas herramientas es que permiten crear DSLs desde cero, pero estos no tinen por qué parecerse al lenguaje de programación base, pues no hace falta utilizar las bases de estos. La ventaja de estos DSLs es que se puede crear la sintaxis que se crea conveniente sin estar atado a ninguna otra. No obstante, la desventaja de esto es que el desarrollador debe de crear el traductor.

```
<aplicacion name='Prueba'>
    <servicio id='c3b9f28c24f2be8b-1'>
        <if data1='c3b9f28c24f2be8b-1' condicion='mayor' data2='8'>
            <for desde='0' hasta='3' condicion='menor'>
                <accion orden='sms:ole!'></accion>
            </for>
            <accion orden =vibrar:5'></accion>
        </if>
    </servicio>
</aplicacion>
```

Código fuente 9 Ejemplo de DSL externo que utiliza XML

Los **DSLs internos** son escritos utilizando el lenguaje de programación principal de la aplicación, como C, C#, C++ o Java. Así, este se adapta para crear unas nuevas construcciones y semánticas basadas en el dominio de la aplicación. Por lo tanto, son lenguajes embebidos que no son creados desde cero y que comparten toda la infraestructura con la existente en el lenguaje de programación principal. Los DSLs internos cambian los pros y contras con los externos. Por ejemplo, en este caso, la sintaxis y la estructura está limitada por el lenguaje en el que se desarrolla, sin embargo, cuanto más dinámico sea el lenguaje de programación elegido, menos limitación existe. Además, cuanto más se parezca el DSL al lenguaje de programación, mayor dificultad habrá para expresar algo que no tenga representación en dicho lenguaje. Cabe resaltar que aquí el beneficio reside en que al utilizar el lenguaje de programación no hace falta realizar un traductor. Un ejemplo de ello está en el Código fuente 10, donde se utilizó Groovy [267], [268].

```
power on // Enciende robot
say hello like girl // El robot dice "hello" con voz de chica
move left, at: 20.km/h // Mueve el robot a la izquierda a 20 km/h
move right, at: 10.m/s // Mueve el robot a la izquierda a 10 m/h
prepare "mocca", with:milk+coffee+cacao+sugar // El robot prepara
una bebida con esos ingredientes
lookup drinks | coffee+milk // El robot busca una bebida con café y
leche
listen "gotoCris" >>> IoT // El robot escucha usando Twitter del
usuario @gotoCris que incluyan el hashtag #IoT
say goodbye like boy // El robot dice "goodbye" con voz de chico
power off // Apaga el robot
```

Código fuente 10 DSL interno creado con Groovy

No hay un consenso entre cuál tipo es mejor, pues cada uno tiene sus beneficios y sus contras. Los **DSLs externos** permiten al desarrollador crear DSL muy expresivos y fáciles de comprender para los usuarios a costa de un mayor esfuerzo en su desarrollo. Por otro lado, con los **DSLs internos** se facilita la labor del desarrollador, sobre todo con lenguajes como Python, Ruby y Groovy, que ofrecen cada uno ciertas ventajas, como son la legibilidad, el *Duck Typing* o la metaprogramación, respectivamente, y por ello son muy buenos lenguajes de programación para el desarrollo de **DSLs internos** [345], sobre todo en el caso de Groovy, que es el más utilizado para esta labor [346].

6.2.2.3 DESDE EL PUNTO DE VISTA DE SU CONSTRUCCIÓN

Krzysztof Czarnecki presentó en [347] una clasificación de los DSLs en base a como se construyen. Esta es parecida a la clasificación Desde el punto de vista de su lenguaje padre, pero menos genérica, pues, la clasificación entre DSLs externos o internos es poco específica.

Los **fixed** o **DSLs separados** son los DSLs que se implementan utilizando un procesador de lenguajes separado. No obstante, estos DSL son los más problemáticos debido a que se debe de desarrollar el traductor y el procesador para el lenguaje completo, lo que implica una larga labor de desarrollo y mantenimiento. Ejemplos de este tipo de DSL son LaTeX (Código fuente 11) [348] y SQL. Este tipo de DSLs son los que la gente suele imaginarse cuando se habla de DSLs [347].

```
\begin{frontmatter}

\title{Midgar & Bilrost}

\author[uniovi]{Cristian Gonz\'alez Garc\'ia\corref{cor1}}
\ead{gonzalezgarciacristian@hotmail.com}

\cortext[cor1]{Corresponding author}

\address[uniovi]{University of Oviedo, Department of Computer Science}

\begin{abstract}
In recent years many researches ...
\end{abstract}

\begin{keyword}
Internet of Things, Smart Objects, Model-Driven Engineering, Domain-Specific Language
\end{keyword}

\end{frontmatter}
```

Código fuente 11 Ejemplo de LaTeX, DSL para procesar texto

Los **DSLs embebidos** son aquellos que están embebidos en un GPL y se utilizan mediante el uso de clases y procedimientos. Algunos ejemplos de DSLs embebidos son la metaprogramación en C++ [347] y las APIs [324], como *Language Integrated Query* (LINQ) [349], mostrado en la Código fuente 12.

```
public static void run() {
    var personas = from persona in svcContext.WorkersSet
                   where persona.DNI > 9000000
                   orderby persona.Apellido ascending
                   select new Persona {Nombre = persona.Nombre,
                                       Apellido = persona.Apellido};

    foreach (var persona in personas)
        Console.WriteLine(persona);
}
```

Código fuente 12 Ejemplo de DSL embebido utilizando LINQ para realizar consultas

Los ***Modularly composable DSLs*** se basan en ver cada DSL como si fuera un componente que permita componer un sistema utilizando DSLs más grandes o pequeños utilizando diferentes combinaciones [347]. Estos tienen ciertas ventajas sobre los DSLs monolíticos. La primera es la *reusabilidad* debido a que permiten dividir un lenguaje en módulos reutilizables. También son *escalables*, pues permiten ir añadiendo los módulos necesarios según se vayan requiriendo permitiendo que sean fáciles de evolucionar. Finalmente, son *modulables* al permitir acoplar y desacoplar los diferentes lenguajes a un componente según este los necesite, pues, los monolíticos, no permiten deshacerse de opciones que pueden ser cuestionables una vez fueron incorporadas al lenguaje. De este tipo de DSLs hay dos subtipos:

- **Encapsulated DSLs**: no influencian en la semántica de otros DSLs que se encuentran en la composición, por lo que son fáciles de combinar. SQL incrustado pertenece a este subtipo.
- **Aspectual DSLs**: estos sí influencian en la semántica de los otros DSLs ya que su implementación en la composición debe de interactuar con el resto de DSLs pertenecientes a dicha composición. Este es el caso de los lenguajes de sincronización de gestión de errores y de aspectos como AspectJ.

Los **preprocesadores** son un mecanismo que permite extender un lenguaje sin necesidad de comprender el «lenguaje anfitrión». Debido a esto, no es necesario crear un compilador entero y basta con crear uno específico. Sin embargo, este método dificulta la traza de errores, ya que estos son reportados por la salida del compilador y no en la entrada del preprocesador, lo que dificulta también su depuración [347].

Por último, están los **lenguajes con soporte a metaprogramación** [347]. En este caso existen varios subtipos:

- El primero de ellos son las ***templates***, como las existentes en C++, que permiten implementar optimizaciones de dominio específico manteniendo la sintaxis y la semántica de C++.
- Mediante los **lenguajes reflexivos** como Common Lisp Object System (CLOS) [350], Groovy o Smalltalk, que permiten implementar casi cualquier extensión debido a que su definición es accesible y modificable por el programador. La desventaja es que esto hace que el código sea más difícil de entender, mantener y depurar.

6.2.2.4 DESDE EL PUNTO DE VISTA DEL DOMINIO DEL PROBLEMA

Esta clasificación los separa según el punto de vista del dominio del problema, pudiendo ser **horizontales** o **verticales** [220], [351], [352].

Los **DSLs horizontales** son aquellas orientados a usuarios que no pertenecer a un sector específico y que necesitan de un DSL para modelar «cualquier cosa», es decir, genérico y basado en la tecnología. Un ejemplo de **DSL horizontal** es *Windows Forms* de Visual Studio, que sirve para crear aplicaciones de escritorio, o bien el sistema de interfaces de Android, que sirve para crear interfaces para cualquier tipo de aplicación Android.

Los **DSLs verticales** son aquellos que sirven para modelar aplicaciones de un dominio específico al que pertenece el usuario. Es decir, un DSL para los desarrolles de videojuegos que sirva para crear videojuegos.

6.2.3 REQUISITOS DE UN DSL

De acuerdo con [353], los DSLs tiene ciertos requisitos, que tienen que ver con las **partes interesadas** en su creación, los **límites** del DSL y las **características** que deben de poseer.

6.2.3.1 PARTES INTERESADAS EN EL DESARROLLO

En el desarrollo de un DSL pueden coexistir tres tipos de personas [353]. Estos son los **ingenieros**, los **clientes** y los **desarrolladores**.

Los **ingenieros** son los responsables de elegir las herramientas apropiadas para desarrollar el DSL. Por este motivo, deben de tener una gran capacidad de abstracción.

Los **clientes** son las personas interesadas en el desarrollo del DSL y/o quienes deben de conocer el dominio del problema, el cual puede ser diferente del campo informático, y por ello deben de proveer de información que ellos conocen y que es necesaria para el desarrollo del DSL.

Los **desarrolladores** son las personas que utilizan el DSL durante la etapa de desarrollo.

Claramente, no siempre tienen por qué existir los tres roles ni estar limitados a estos. Por ejemplo, los ingenieros, los clientes y los desarrolladores podrían ser los mismos en el caso de que el DSL sea para facilitar algo que ellos necesiten. Otro caso sería en el que los clientes y los desarrolladores son los mismos o los ingenieros y los desarrolladores.

6.2.3.2 LÍMITES DE UN DSL

Cuando se crea un DSL, es muy importante definir y acotar correctamente el dominio del problema delimitando precisamente que parte del sistema se creará con el DSL y cuál con un GPL [220]. De esta manera, con un DSL deberá de ser más sencillo mostrar la correspondencia entre el DSL con los conceptos del dominio que con un GPL, claramente, a costa de la perdida de generalidad [353].

Esto se debe a que, depende de para que se desee utilizar y a quien esté dirigido, el DSL deberá de cumplir unas propiedades. Por ejemplo, si el DSL es para personas que no tengan conocimientos de programación,

entonces esto implicará que habrá que crear en el DSL todo lo necesario para permitir que estos no deban de tocar nada en un GPL. Por el contrario, si lo que se desea es facilitar o automatizar una cierta labor a los desarrolladores, entonces el DSL se concentrará en esa labor exactamente, dejando de lado el resto del trabajo que correrá a cargo de los desarrolladores, quien, presumiblemente, utilizarán un GPL.

6.2.3.3 CARACTERÍSTICAS QUE DEBEN POSEER

Los DSL, deben de tener una serie de características. Algunas de estas características son presentadas en [353] y [220].

- **Conformidad**: los elementos del lenguaje deben de corresponderse con los conceptos del dominio.
- **Ortogonalidad**: cada elemento del lenguaje es para representar un elemento del dominio.
- **Soporte**: deben de existir herramientas para trabajar con el DSL.
- **Integrabilidad**: el DSL y sus herramientas deben de poder interoperar con otras herramientas, así como deben de poder soportar la extensibilidad con otros elementos adicionales.
- **Longevidad**: debe de haber una justificación temporal de cara a la creación al DSL, pues si este periodo de uso es demasiado corto, puede no ser rentable realizar el DSL.
- **Simplicidad**: el DSL debe de poder representar de una manera simple los elementos del dominio.
- **Calidad**: se deben proveer mecanismos para poder crear sistemas de calidad, por ejemplo, mediante la inclusión de precondiciones y postcondiciones.
- **Escalabilidad**: este requisito no es necesario debido a que puede ser que se necesite un DSL para sistemas pequeños, pero a veces si es una característica deseable en algunos DSLs.
- **Usabilidad**: el lenguaje debe de ser usable, no obstante, no siempre esta característica es posible debido a que a veces el propio lenguaje que se crear debido al dominio utilizar impide que este sea usable. No obstante, la simplicidad puede ayudar a que sea usable.

6.2.4 PROPIEDADES DE LOS DSLS

Los DSLs tienen una serie de propiedades [299]. Suelen ser **pequeños**, **declarativos** y **realizados por el usuario final**.

Los DSLs suelen ser **pequeños**, ya que se centran en ofrecer una abstracción de un problema y solo características para poder solucionar este. No obstante, a veces, cuando el DSL es interno, este puede adoptar características generales, debido a la interacción que ocurre entre el DSL y el GPL con el que fue construido.

Estos lenguajes además suelen ser **declarativos**. Normalmente, suelen disponer de un compilador que *parsea* el modelo creado por el usuario utilizando el DSL para así generar la aplicación final. A este compilador se le suele llamar *generador de aplicaciones*, en el caso de que sean DSL específicos para crear aplicaciones. En cambio, otros DSLs, como ASDL [354] o Yacc [338], [339], no generan aplicaciones, sino que generan librerías o componentes. Por otro lado, están los DSLs que son como HTML y CSS que sirven para especificar la estructura y la presentación.

Una característica muy importante de los DSLs es que ofrecen una programación **realizada por el usuario final** debido a que son los que dicen cómo quieren el DSL ya que serán quienes lo usarán.

6.2.5 VENTAJAS Y DESVENTAJAS DEL USO DE DSLS

El uso de DSLs ofrece ciertas ventajas [344] como son la gran mejora en la **productividad**, la **comprensión**, la **portabilidad**, el **mantenimiento** [325] y la **reutilización** [30], [236]. Sin embargo, el uso de los DSLs también tiene ciertas desventajas [344] entre las que cabe destacar el **rendimiento**, el **tiempo**, el **coste**, la **dificultad**, la **documentación adicional** y la **preparación.**

6.2.5.1 VENTAJAS

La primera ventaja es la mejora en la **productividad** gracias a la reducción de fallos frente a un GPL debido a que el DSL nos abstrae de forma que se deba escribir menos para hacer lo mismo o más. Además, en el caso de que el DSL sea gráfico, este nos ofrece una **reducción de fallos** mayor que uno textual, pues en el gráfico se tiene una mayor abstracción y se debe de escribir mucho menos que en uno textual. Además, esta **productividad** se ve reflejada también en la **detección de errores** gracias al uso de modelos, pues estos, al ser expresados utilizando un DSL, permite que sean validados al mismo nivel de abstracción, lo que proporciona una detección de errores en ese mismo nivel de capa. Además, en algunos casos, los DSLs suelen proporcionar APIs que ayudan en la manipulación del modelo, lo que influye en la **productividad**.

Una ventaja muy importante es la **comprensión**, pues el trabajar con los términos relativos a un dominio concreto proporciona una mejor comprensión de los modelos a personas sin conocimientos de desarrollo de software.

La **simulación** es una de las ventajas proporcionadas al trabajar con modelos, pues estos podrían ser utilizados para simular las salidas que se obtendrían.

El utilizar **MDE** permite que la **portabilidad** entre tecnologías sea más sencilla gracias al uso de los metamodelos que son representados utilizando el DSL desarrollado, siendo esto una ventaja para el **mantenimiento** a largo plazo de las aplicaciones.

Por último, una ventaja del uso de DSLs es la **reutilización** que ofrecen al poder generar utilizando el mismo DSL diferentes aplicaciones para el mismo dominio, es decir, se reutiliza el conocimiento y las herramientas.

6.2.5.2 DESVENTAJAS

Como contras tiene la disminución de la eficiencia frente a la codificación nativa y la dificultad para crear el dominio y construir el DSL [30], [325].

El uso de DSLs tiene una disminución en el **rendimiento** frente a los GPLs debido a que, cuanta más abstracción se consigue, más se empeora el rendimiento [247].

La creación de un DSL requiere de **tiempo** y **coste** adicional debido a la investigación acerca de las posibles herramientas a utilizar, en que elementos del dominio se abstraerán, en testear todo el proceso, pues esto requiere más tiempo que en el desarrollo normal al introducir transformaciones internas.

Además, tienen una **dificultad** añadida debido al hecho de que se requieren pruebas diferentes y extras para poder probar las diferentes transformaciones.

Necesitan de **documentación adicional** que explique cómo funciona todo este sistema.

Se requiere una **preparación** del equipo de desarrollo para que sean capaces de trabajar con cada DSL creado para ellos, pues, es un lenguaje específico que deben de aprender a utilizar.

7. SEGURIDAD: CRIPTOLOGÍA

«Nunca confíes en un ordenador que no puedas lanzar por una ventana»
Steve Wozniak

«En cuestiones de espionaje, ya hace mucho que pasamos ese punto. Internet es una gigantesca máquina de espionaje al servicio del poder. Debemos luchar contra esta tendencia y convertirla en un motor de transparencia para el público, no solo para los poderosos»
Julian Assange [35]

En los últimos años se han visto muchos casos de **vulnerabilidades** en el que se han obtenido datos de los usuarios con información muy personal de ellos. Entre ellos cabe destacar el de gobiernos como Estados Unidos con el caso Snowden o WikiLeaks, de empresas como Adobe, el de fotos de actrices con el caso Celebgate, las puertas traseras existentes en el software para el espionaje de la Agencia de Seguridad Nacional de los Estados Unidos de América (NSA), la ruptura de cifrados de la NSA, las vulnerabilidades de la red Tor, el robo de Bitcoins, el secuestro de ordenadores mediante el cifrado de sus discos duros, la vulnerabilidad Heartbleed, los papeles de Panamá, la caída de Silk Road,… Muchos casos muy famosos y que pusieron en peligro o liberaron mucha información pública y privada de mucha gente y mucha información confidencial y privilegiada de muchos gobiernos.

Internet de las Cosas es el futuro, pero, si este no es lo suficientemente seguro, puede que provoque una intrusión en la privacidad de la gente o que se convierte en un «Gran Hermano» de la gente sin permiso de los propios usuarios de **IoT**.

Por ello, hay que tener muy en cuenta la seguridad clásica, la utilizada hasta ahora en la informática y que, siempre que fue bien aplicada, casi siempre funcionó correctamente. No obstante, hay que tener en cuenta que **IoT** trabaja sobre Internet, lo que conlleva a que esté expuesto a muchos más vectores de ataque que el software o hardware clásico.

Este capítulo explicará ciertos matices acerca de la seguridad, exactamente, sobre la criptología. Se verá una breve introducción de la criptografía, su historia, la terminología, las cualidades que deben poseer los mensajes para ser seguros, diferentes métodos criptográficos y el trabajo relacionado en el marco de **IoT**.

[35] http://web.archive.org/web/20101029005507/http://eldiariodigital.golbac.com/julian-assange-wikileaks-internet-es-una-gigantesca-maquina-de-espionaje-al-servicio-del-poder-eng/

*****

Seguridad: criptología

7.1 *INTRODUCCIÓN*

Debido a las pretensiones de IoT, que son interconectar todos los dispositivos, las **vulnerabilidades** pueden provocar una intrusión en la privacidad de la gente que, incluso, podría hacer que diversas compañías interesadas pudieran utilizarlo a modo de «**Gran Hermano**», con fines comerciales [161] o para realizar seguimiento de personas [114]. Una muestra de esto es el creciente grado de preocupación de los usuarios por su privacidad, en el que el 83.4% se preocupa mucho, en un rango de 7 a 10 sobre 10, por sus datos personales y un 77% por sus fotografías y videos [355]. Sin embargo, si toda la comunicación estuviera perfectamente protegida, IoT podría ser la fuente de una mejora sustancial en nuestra calidad de vida, gracias a la interconexión de aplicaciones y diferentes servicios que nos proporcionen ayuda en nuestra vida diaria.

El principal problema reside en que lo pretendido por IoT es seguir todos nuestros movimientos para conectar todos nuestros objetos entre ellos y con objetos de otra gente, lo que implica que mucha información personal pueda ser recolectada automáticamente [2]. Esta información puede ser nuestra geolocalización, datos bancarios o nuestro ritmo cardiaco. No obstante, el problema reside en que la privacidad en IoT es mucho más seria que la seguridad tradicional ya que los **vectores de ataque** son aparentemente mucho más grandes [356], [357], como ocurrió con una empresa de alimentos [358].

Por esto mismo, IoT necesita investigar en un mecanismo de seguridad que provea de cinco puntos importantes [19], todo de cara a crear un sistema seguro para los usuarios de este. El primer punto es **definir la seguridad y la privacidad** desde el punto de vista social, legal y cultural, de manera que se consiga saber que es cada cosa y conocer los límites de la libertad de lo que puede hacer la gente. El segundo es un **mecanismo de reputación y confianza** que permita valorar y dar más o menos permisos a los diferentes objetos o incluso hasta bloquearlos. Como tercer punto se habla de un sistema de comunicación **encriptado punto a punto**, con lo que se conseguiría que solo el emisor y el destinatario puedan leer el mensaje mientras que el servidor o los intermediarios nunca tendrían acceso a él. El cuarto punto marca la **privacidad** en la comunicación y sobre los datos del usuario, de acuerdo con que nadie pueda acceder a datos sin permiso. Por último, el sistema debe de ofrecer **seguridad** en los servicios y las aplicaciones, evitando así que estas tengan vulnerabilidades que permitan obtener datos a usuarios maliciosos.

Un ejemplo para todo esto, es la opinión de la gente involucrada en el transporte y la logística [19], quienes están preocupados por la fuga de información y por la invasión de su privacidad.

Una posible solución para mantener la privacidad de los mensajes es usar **criptografía** [359]. De esta forma, se podría cambiar el mensaje utilizando diferentes métodos y algoritmos para evitar que objetos y gente sin autorización puedan leer o modificar el mensaje, obteniendo así las cualidades que un mensaje seguro debería de cumplir.

7.2 *TERMINOLOGÍA*

En esta sección se muestran diferentes términos y palabras utilizadas en el campo de la seguridad, concretamente de la criptografía, y que son necesarias conocer para un mejor entendimiento de este capítulo.

La **criptografía** es el arte de transformar datos en otra secuencia de bits aparentemente aleatorios y sin significado, para impedir que se entiendan cuando lo lea un observador o atacante no deseado [360], [361]. Se considera que se consiguió romper la criptografía cuando el atacante pudo leer el mensaje original [359], [362]. Así, la **criptografía** sirve para proteger la información y dotar de seguridad a las comunicaciones.

El **criptoanálisis** es la ciencia que estudia las técnicas de desencriptado y detección de mensajes y que trata de desencriptar los mensajes [362]. Así, esta es la ingeniería inversa de la **criptografía**, que se encarga de buscar los puntos débiles de los algoritmos criptográficos y sus implementaciones para explotar estas debilidades [360]. Esto puede ser por motivos científicos, de seguridad o por ataques maliciosos.

La **criptología** abarca la **criptografía** y el **criptoanálisis** y mira a los problemas de matemáticas que subyacen en ellas [360]. Por ello, es la disciplina científica que se dedica al estudio de la escritura secreta, estudiando aquellos mensajes que se procesan para dificultar su lectura.

La **esteganografía** es el arte y la ciencia de comunicarse a través de la ocultación de mensajes en un fichero ordinario, evitando que se sepa que existe una comunicación [359], [362], [363]. La **esteganografía** incrusta el mensaje en un fichero impidiendo así que se vea que hay un mensaje en el documento y haciendo pensar que simplemente se transmite un documento normal. El tipo de fichero más famoso utilizado es la imagen, pero pueden ser también documentos de texto, videos o ficheros de sonido. Para incrustar este mensaje, hay que intercambiar los bits menos significativos por los datos del mensaje que deseamos ocultar [360]. Hasta hace poco, era considerada la versión «mala» de la criptografía, pero últimamente está ganando mucha popularidad en la industria, debido al uso de marcas de agua y huellas digitales en audio y video [362]. Se considera que un atacante rompió la esteganografía cuando este detectó que está ha sido usada y consigue leer el mensaje incrustado [362]. Así, el uso de la **esteganografía** puede deberse a la necesidad de utilizar un canal inseguro para enviar información sin que esta sea percibida mediante el uso de contenedores conocidos utilizando un algoritmo conocido solo por el receptor. En la Ilustración 25 se muestran dos imágenes, a la izquierda la imagen normal y a la derecha su versión al insertarle un fichero de texto con el contenido «El 13 del 11 a las 18h» con la contraseña «123456» por medio del uso de una técnica esteganográfica del programa «steghide».

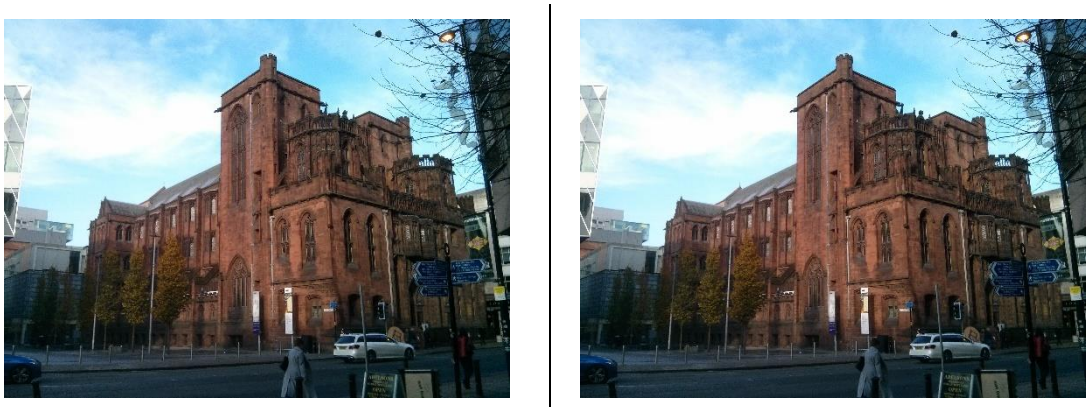


Ilustración 25 Imagen normal junto a su versión al aplicarle un método esteganográfico

El **estegoanálisis** es la ciencia que estudia las técnicas de desencriptado y detección de mensajes y que trata de descubrir los mensajes ocultos [362], luego, es la acción contraria a la esteganografía, tratando de romper sus métodos y los puntos débiles de sus algoritmos para vulnerar la seguridad.

El **cifrado por sustitución** se basa en el intercambio de una letra por otra letra o símbolo [362]. Ejemplos de este cifrado son ***Atbash*** [363], el **cifrado César** [363], [364] o **ROT13**, que es una variación del **cifrado César** en donde en vez de 3 saltos de letra da 13 [360]. No obstante, este es fácil de romper por los ordenadores actuales. Además, debido a diferentes estudios sobre los idiomas, se descubrió que estos tienen sus propias distribuciones de letras. Por ejemplo, en inglés la letra con mayor frecuencia de aparición es la «e», así pues, probablemente, se podría tomar como que la letra con más apariciones en un mensaje en inglés con cifrado por sustitución es la «e» [360], [365]. En español ocurre lo mismo, siendo la «e» la letra con más apariciones, seguida de cerca por la «a», como muestra este artículo que cogió los títulos de artículos científicos para ello [366].

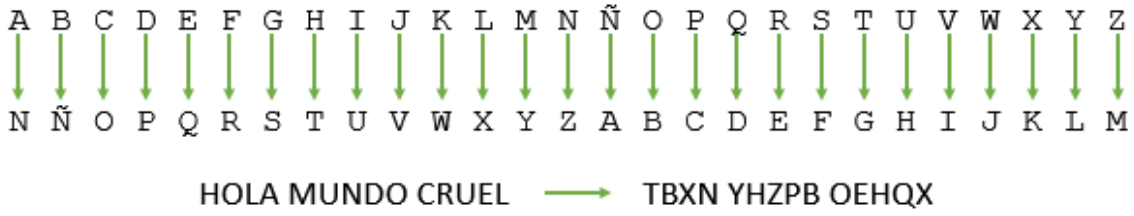


Ilustración 26 Cifrado ROT13 con el abecedario español

El **cifrado por transposición** o **permutación** consiste en cambiar el orden de las letras de un texto plano [365]. Esto puede ser mediante el intercambio de una letra, por pares o tríos de letras, por columnas, por filas, o bien, reflejando la palabra en un espejo.

Se llama **texto plano** al mensaje original que puede ser leído por cualquier persona, es decir, que no está encriptado [360].

El **texto encriptado** o **texto cifrado** es el mensaje original ya encriptado [362]. Este no puede ser leído a menos que se desencripte previamente.

Seguridad: criptología

La **encriptación** o **cifrado** es cuando se transforma el texto plano del mensaje original en un mensaje codificado para ocultar el contenido original [362]. Por otro lado, la definición de Symantec es [367]: «*La **encriptación** es un método de cifrado o codificación de datos para evitar que los usuarios no autorizados lean o manipulen los datos. Solo los individuos con acceso a una contraseña o clave pueden descifrar y utilizar los datos. A veces, el malware utiliza la encriptación para ocultarse del software de seguridad. Es decir, el malware cifrado resuelve el código del programa para que sea difícil detectarlo*».

La **desencriptación** o **descifrado** es cuando se transforma el mensaje codificado en el mensaje original para poder leerlo [362].

Así, según lo visto hasta aquí, se pueden resumir los diferentes métodos de escritura secreta según la Ilustración 27 [365].

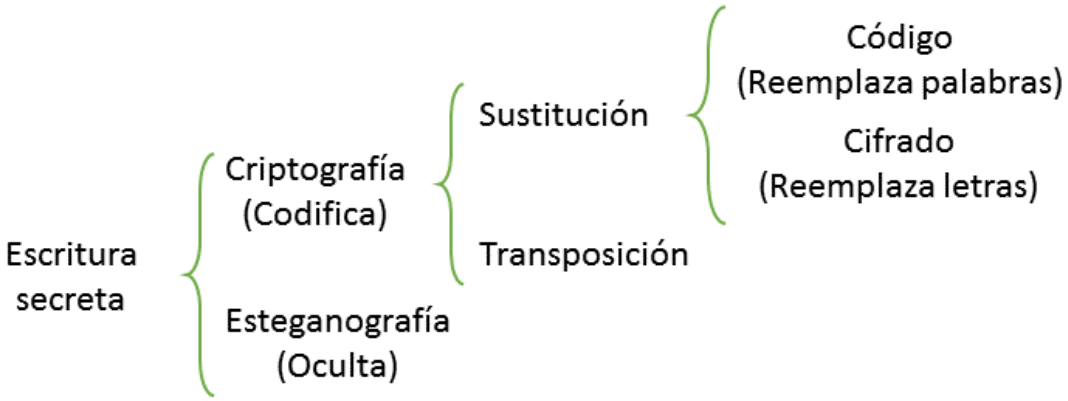


Ilustración 27 Ramas de la escritura secreta

La **filtración de datos** según la empresa de seguridad Symantec es: «Una **filtración de datos** sucede cuando se compromete un sistema, exponiendo la información a un entorno no confiable. Las filtraciones de datos a menudo son el resultado de ataques maliciosos, que tratan de adquirir información confidencial que puede utilizarse con fines delictivos o con otros fines malintencionados» [367].

La **clave**, **palabra clave** o **clave criptográfica**, o «***key***» en inglés, es la información necesaria por los métodos criptográficos para realizar la operación de transformación de texto plano a texto encriptado y viceversa. Normalmente, la **clave** suele ser una secuencia de números y letras, y a veces caracteres especiales. Así, un método criptográfico utilizado sobre un mismo **texto plano**, pero al que se le aplique una **clave** diferente, deberá dar una salida con un **texto encriptado** diferente del de la primera **clave**. Respecto a esta **clave**, cuanto más larga sea, más seguro será el mensaje, pero también será necesario más tiempo para realizar la operación requerida y más capacidad de cómputo para procesarla. La longitud de la **clave** se mide en bits, ya que se trabaja a nivel de bits en la actualidad, por ello lo típico es encontrar claves de longitud 56, 256, 512, 1024, 2048 y 4096 bits.

La **Criptografía Basada en Atributos (Attribute-Based Encryption [ABE])** es un tipo de criptografía de clave pública en la cuál la clave secreta de un usuario y el texto cifrado dependen de unos atributos, siendo así imposible de desencriptar si los atributos de la clave y del texto cifrado no coinciden [36].

Una **vulnerabilidad** es un estado existente en un sistema informático que afecta a la confidencialidad, integridad y disponibilidad del sistema y puede permitir ejecutar comandos, obtener datos, hacerse pasar por otra entidad o realizar una negación de servicio a un atacante [367]. Esta vulnerabilidad puede existir en la red, en el servidor, en la aplicación que se utiliza para conectarse o incluso en el propio mensaje.

Un **vector de ataque** es un método que utiliza alguien o algo para tratar de vulnerar un sistema [367].

Una **amenaza** informática es una circunstancia, evento o persona no deseada y con potencial suficiente para causar daño a un sistema, ya sea robo, destrucción, divulgación, modificación de datos o denegación de servicio [367], a través de vulnerabilidades que puedan ser aprovechadas. Las amenazas pueden ser *intencionales* o *no intencionales*.

Una **amenaza intencional** se da cuando se intenta producir daño deliberadamente.

La **amenaza no intencionada** es cuando se pueden producir fallos sin intención de explotar la **vulnerabilidad**, como puede ser un fallo de comprobación o fallos por fenómenos naturales.

Un **ataque** es cualquier intento de criptoanálisis, pues tratan de vulnerar el mensaje. No obstante, hay que tener en cuenta que un ataque de fuerza bruta no es considerado **criptoanálisis**, pero sigue siendo un **ataque** [360]. No obstante, esta palabra abarca mucho más, pues también incluye cualquier acción adoptada para utilizar una o más **vulnerabilidades** para materializar una **amenaza**.

La **disponibilidad** es la acción de garantizar el acceso oportuno y confiable al uso de los datos requeridos [368].

La **privacidad** es la recolección, procesamiento, comunicación, uso y disposición consistentes de datos asociados con la información de identificación personal (PII) a través del ciclo de vida de los datos [368]. O, dicho de otra manera y de acuerdo con la RAE, la **privacidad** engloba «el ámbito de la vida privada que se tiene derecho a proteger de cualquier intromisión» [154].

La **seguridad** es la protección de los datos, información y sistemas de accesos, usos, divulgación, alteración, modificación o destrucción no autorizados [368].

El ***Cipher Block Chaining*** (CBC) es uno de los cinco modos de cifrado por bloques, que son un conjunto de bits de longitud fija, siendo normalmente de 64 o 128 bits, y que se utiliza en los algoritmos de criptografía de cifrado por bloques, que son aquellos que no son de *streaming*, para proveer de confidencialidad y privacidad a los mensajes [37] [369]. Los mensajes a encriptar deben de tener una longitud de texto múltiplo del tamaño de bloque.

[36] http://crypto.stackexchange.com/questions/17893/what-is-attribute-based-encryption
[37] http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html

El ***padding*** es un mecanismo utilizado en criptografía para insertar información irrelevante para evitar que la predictibilidad de los mensajes puedan ayudar al criptoanálisis a encontrar el texto plano original rompiendo la encriptación [38]. Así, el *padding* permite encriptar mensajes que tengan una mayor cantidad de datos que los bloques mediante la adición de ceros al final.

7.3 *HISTORIA DE LA ESCRITURA SECRETA*

Desde hace miles de años, el hombre se ha preocupado de ocultar sus mensajes para que así solo el destinatario pudiese leerlos. El motivo, sencillo: que los enemigos o gente con cierto interés no pudiese acceder a información secreta, información que podían ser las tácticas de guerra o de pactos entre personas, generales, reyes, países o imperios. Es decir, lo que buscaban era asegurar las comunicaciones. Dos métodos muy populares para ofrecer esta seguridad requerida fueron y son la **criptografía** y la **esteganografía** [362].

Uno de los primeros casos fue entre los años 600 y 500 años antes de la Era Común (AEC), cuando los hebreos utilizaban **Atbash** para cifrar los mensajes. **Atbash** era un cifrado por sustitución, el cuál consistía en invertir el alfabeto [363].

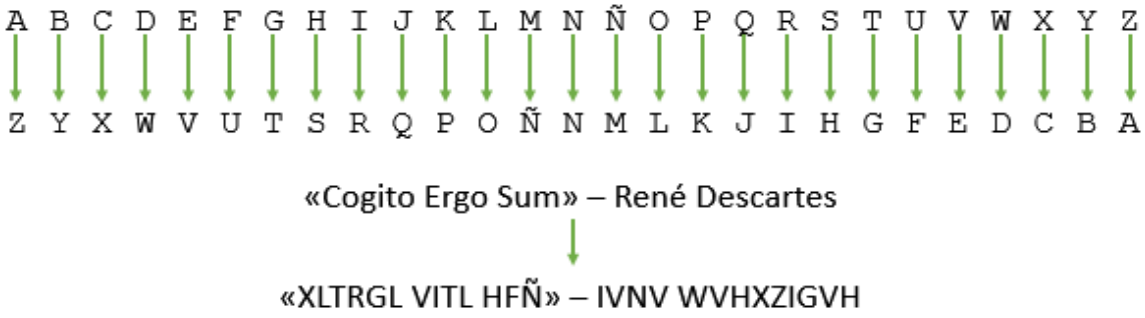


Ilustración 28 Cifrado Atbash con el abecedario español

Otro ejemplo, pero exactamente de **esteganografía**, que data del siglo V AEC, fue cuando los persas, con Jerjes I como rey de Persia, estaban en guerra con Grecia, tratando de anexionarla a su gran imperio tras la insolencia de Atenas y Esparta al no enviarle regalos tras construir la nueva capital del imperio persa, Persépolis. Cinco años pasaron para que, en el año 480 AEC, tras pensar y reunir un poderoso ejército, Jerjes I se decidiese a realizar el ataque sorpresa. Sin embargo, la acumulación militar fue observada por un griego, **Demarato**, un antiguo rey espartano que había sido expulsado de su tierra natal y que vivía en la ciudad persa de Susa. Demarato aún sentía apego por su tierra. Por ello, escribió un mensaje secreto avisando del plan de Jerjes I con el reto de que este mensaje no pudiera ser interceptado por los guardias persas. La solución fue escribir el mensaje en unas tablas de cera y tras esto, ocultar el mensaje con más cera. De esta manera, el mensaje llegó hasta los griegos, siendo la hija del rey espartano del momento, Cleomenes Gorgo, y esposa de Leónidas I, quien descubrió el mensaje y lo transmitió a todos los griegos [363], [365], [370].

[38] http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=64213

Unos años más tarde, en el 440 AEC, el griego **Histaeus** utilizó **esteganografía** mediante el afeitado de la cabeza de uno de sus esclavos para escribirle un mensaje en ella y que así, cuando le creciera el pelo, este tapase el mensaje. De esta manera, el esclavo podía ser enviado como contenedor llevando el mensaje a su destinatario [363].

Más tarde, entre los años 60-50 AEC, **Julio César**, emperador romano, utilizó el cifrado que lleva su nombre para encriptar las comunicaciones [363], [364]. El **cifrado César** sustituye una letra por la correspondiente tres posiciones más a la derecha en el alfabeto (Ilustración 29).

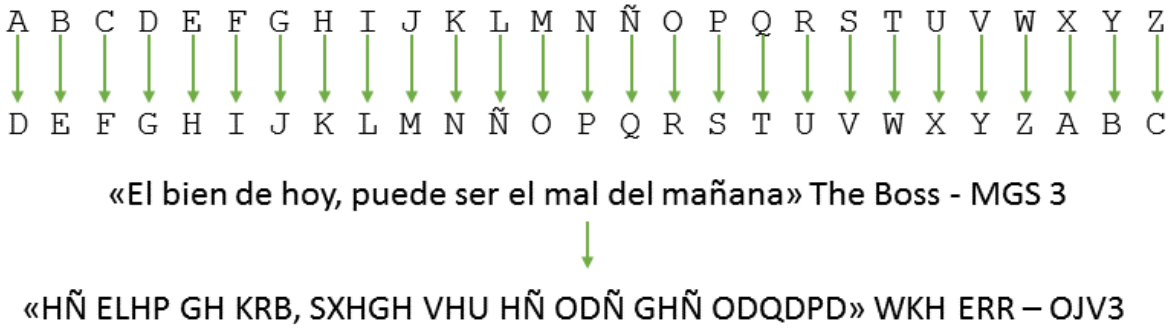


Ilustración 29 Cifrado César con el abecedario español

Otros ejemplos eran los mensajes que los **chinos** escondían mediante esteganografía. Estos escribían los mensajes en tela utilizando seda fina, después hacían una bola con ello y posteriormente la cubrían de cera para que, como último paso, los mensajeros se la tragaban. Más adelante, en el siglo I AEC, **Pliny «el viejo»** explicó como con la leche de ciertas plantas se podía hacer **tinta invisible** y como mediante el calentamiento podía hacerse visible [365]. Esta tinta podía ser leche, vinagre, zumo de fruta u orina y se hacía necesario calentarla o utilizar una luz especial para poder verla [363].

Mary, Queen of Scots, en el año 1586, estando presa, tenía que preparar ella misma su defensa tras ser acusada de matar a la reina Elizabeth para quedarse con su corona. No podía recibir ninguna ayuda, ni llamar a los testigos ni tener secretarias. Sin embargo, esto no fue un problema para ella, que mantuvo correspondencia cifrada con cierta gente que deseaba ayudarla, los llamados conspiradores. Mary cifraba las cartas y las enviaba a través de un emisario. Gracias a este cifrado, ni el emisario ni quien lo capturase podía saber el sentido de estas cartas, pues no contenían ninguna palabra con sentido a ojos de ellos [365].

Otros personajes ilustres que utilizaron criptografía fueron Leonardo Da Vinci [371] y Sir Francis Bacon, ambos en el renacimiento [363].

Sin embargo, el mayor ejemplo criptográfico es el de la **máquina Enigma** [365]. Esta máquina ,inventada por el ejército alemán, fue utilizada en la segunda guerra mundial para que el ejército aliado no pudiera entender sus comunicaciones a pesar de tenerlas monitorizadas. Esta máquina de cifrado rotatoria cambiaba la clave de cifrado todos los días, haciendo imposible utilizar la clave de un día para descifrar las de los siguientes días, en caso de que se descubriese, o, como ocurrió, hacer que fuese más difícil romper el cifrado debido al inútil uso de los mensajes de un día para otro. Este sistema de mensaje entre diferentes máquinas

Enigma funcionaba debido a que todas las máquinas tenían la misma configuración y entonces estaban sincronizadas para tener la misma clave el mismo día.

Aún con los avances realizados y el uso de ordenadores, hay ciertos documentos que se resisten. Este es el caso del **manuscrito Voynich** [372], escrito entre 1404 y 1438 y del que se desconoce el autor, aunque se presuponen varios. Este manuscrito es la perla de la criptografía debido a que, a pesar de los múltiples intentos por diferentes criptógrafos, empresas y universidades, todavía no se ha podido descifrar. De este manuscrito simplemente se conocen o se intuyen sus secciones debido a los dibujos que acompañan el texto, lo que deja entrever que contiene texto de muchas ramas, como la alquimia o la astrología. No obstante, hay diferentes hipótesis, entre las que destacan que puede ser un engaño o un lenguaje ficticio basado en una lengua natural, debido a que cumple la ley de Zipf [373], que dice que en una lengua, la frecuencia de aparición de las palabras sigue una determinada distribución.

Como se ve, estos ejemplos se basaban en la ocultación o encriptación de la información mediante diferentes métodos. Algunos «tapando» el mensaje original ocultándolo con otras cosas o bien, mediante sustitución o transposición de caracteres, es decir, usando **esteganografía** o **criptografía**.

Al principio, la **criptografía** era sencilla, simplemente utilizaban métodos de **transposición** o **sustitución** o bien utilizaban **esteganografía** para ocultar el texto tapándolo bajo algo. Tras esto, según la gente fue aprendiendo, los métodos fueron volviéndose más complejos, mediante el uso de la sustitución y transposición de caracteres, siendo esta, cada vez más trabajada para hacer casi irrompible su método, siendo su pináculo la máquina **Enigma**. No obstante, estos cifrados eran a nivel de caracteres. Con la llegada de la informática, estos mensajes eran más fáciles de descifrar debido a la potencia de los ordenadores, pues estos pueden hacer mucho más rápido, automáticamente y sin fallos, en el caso de que el programa este bien implementado, lo que hace una persona en más tiempo. Esto dio lugar a que la **criptografía** y la **esteganografía** se volviesen aún más complejas y que cada vez, con el aumento de procesamiento de los ordenadores estas fuesen aumento en complejidad. Además, los métodos pasaron de sustituir y transponer caracteres a hacerlo a nivel de bits, lo que implica una mayor seguridad y complejidad.

7.4 *CUALIDADES DE LOS MENSAJES*

La **criptografía** es necesaria cuando las comunicaciones se hacen para crear un mensaje seguro que pueda ser transmitido utilizando un medio de transporte no seguro. Para considerar que un sistema de mensajes es realmente seguro, el mensaje enviado debe de tener indispensablemente unas cualidades. Así, los mensajes deben de tener **autenticidad**, **integridad**, **no repudio** y **privacidad** [360].

La **privacidad** y la **confidencialidad**, son necesarias para garantizar que el mensaje será solo leído por los objetos y/o personas autorizadas por esa comunicación y no por otros objetos o personas [359], [368]. Así, la **privacidad** es necesaria para mantener a salvo los datos de cada objeto y usuario, para así evitar el acceso a estos datos por parte de otro objeto o usuario que no tengan los permisos adecuados para ello. Como podemos ver en [374], la gente no es consciente sobre toda la información que tiene expuesta, la privacidad que pueden perder o la posible violación de su privacidad debido a los sensores que están utilizando al utilizar

un entorno automatizado. Por eso, la privacidad es uno de los problemas principales en IoT [356], [374], haciendo que así, solo el personal autorizado pueda acceder a la información.

Sin embargo, para obtener ese permiso, la comunicación necesita asegurar la **autenticidad** de los objetos para identificar apropiadamente los objetos que pueden tener acceso a ese mensaje [359]. Para conseguir esto, una posible solución pasa por crear una identificación única para los diferentes objetos de forma que sean únicos en el sistema. De esta manera, se conseguiría solucionar definiendo un mecanismo de control de acceso y un proceso de autenticación [358], garantizando que solo el personal con una identidad y acreditación adecuada pueda acceder a la información.

Otra parte importante del mensaje es la **integridad**. La integridad es la verificación de que el mensaje recibido por el receptor no cambió durante la transmisión, así como que este mensaje todavía contiene exactamente el contenido original íntegro y sin modificar, lo que hace necesaria tener **integridad** para garantizar el **no repudio** y la **autenticidad** [359], [368]. Así, para poder proporcionar **integridad**, se deberá poder conocer si el mensaje es el original y que nadie lo haya modificó durante su comunicación. Esto se puede conseguir mediante la comprobación de que ese mensaje posea las mismas características, propiedades y valores con las que fue emitido. En caso contrario, el mensaje debería ser borrado y el objeto debería notificar la incidencia. Este problema ha sido estudiado previamente, pero únicamente sobre las comunicaciones tradicionales [375].

A pesar de estas cualidades, estas no son suficientes para certificar el emisor y el receptor. Para certificar que el emisor es realmente quien dice ser y que el receptor no pueda decir que no recibió el mensaje, se necesita implementar el **no repudio**, conocido a veces como **no rechazo**. Esto permite certificar la identidad de ambos para así evitar que tanto el emisor como el receptor puedan negar que han enviado o recibido dicho mensaje.

7.5 *MÉTODOS CRIPTOGRÁFICOS*

Las diferentes técnicas **criptográficas** modifican la estructura del mensaje. Según el tipo de método de encriptación utilizado, estos se pueden clasificar en tres tipos de criptografía: **clave secreta** o **criptografía de clave simétrica**, **clave pública** o **criptografía de clave asimétrica** y **criptografía híbrida**.

7.5.1 CLAVE SECRETA O CRIPTOGRAFÍA DE CLAVE SIMÉTRICA

La **clave secreta** o **criptografía de clave simétrica** usa la misma clave para encriptar y desencriptar el mensaje [359], [360]. Debido a esto, tanto el emisor como el receptor deben conocer la clave. Aquí el principal problema viene dado en que el emisor debe enviar la clave utilizada al receptor usando algún tipo de comunicación, a poder ser por seguridad, un sistema diferente al utilizado para enviar el mensaje encriptado. Por ello, ambos, emisor y receptor, han de conocer la clave. Este sistema conlleva a que exista el riesgo de que otra persona intercepte la clave cuando esta es enviada. Por eso, el problema se encuentra en la distribución de claves. Sin embargo, la **criptografía de clave simétrica** es el método más rápido para

encriptar y desencriptar mensajes. Por esto, este método es el más usual para encriptar grandes cantidades de datos.

La seguridad de este método se basa en la longitud de la clave haciendo que para romper una clave de 80 bits por fuerza bruta se necesiten probar 2^{80-1} combinaciones. Por eso se recomiendan claves de 128 bits.

Ejemplos de algoritmos de **criptografía de clave simétrica** son Rivest Cipher 4 (RC4) [376], RC5 [377], RC6 [378], *Advanced Encryption Standard* (AES) [379], [380], *Data Encryption Standard* (DES) [381], triple DES (3DES) [382], Blowfish [383], [384] e *International Data Encryption Algorithm* (IDEA) [383], [385].

El método comienza cuando el emisor encripta el mensaje utilizando una **clave secreta**. Después, el emisor envía el mensaje encriptado al receptor, quien desencripta el mensaje utilizando la misma clave secreta. Un ejemplo gráfico se puede ver en la Ilustración 30. En este ejemplo, Alice [39] encripta su nota utilizando el algoritmo Blowfish de 128 bits, 64 bits de bloque, sin *padding* [40] y en modo *Cipher Block Chaining* [41], usando la clave «clave_privada_sc», de tal manera que obtiene el mensaje encriptado y se lo envía a Bob. Bob desencripta este mensaje utilizando la misma clave, «clave_privada_sc», y así puede leer el mensaje original.

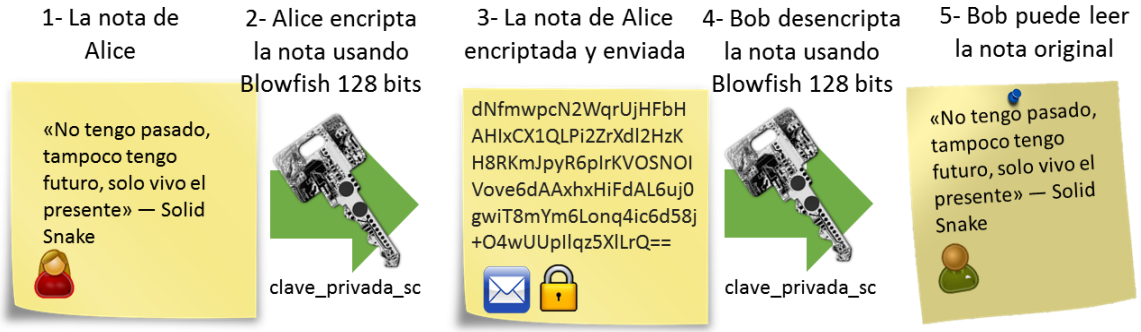


Ilustración 30 Funcionamiento de la criptografía de clave simétrica usando Blowfish

7.5.2 CLAVE PÚBLICA O CRIPTOGRAFÍA DE CLAVE ASIMÉTRICA

La **clave pública** o **criptografía de clave asimétrica** es otro método criptográfico. Este método utiliza dos claves: una pública y otra privada [359], [360]. Ambas son o pueden ser únicas, en el caso de la privada, siempre es única. La clave pública puede ser conocida por todo el mundo. Mientras tanto, la clave privada es secreta y solo el dueño debe conocerla. Con este tipo de clave, en el año 1976 se solucionó el problema existente de la *distribución de claves* [386], todo gracias a la investigación de Diffie y Hellman [387].

Una vez generadas las claves, el proceso de envío puede comenzar. Para ello, el emisor debe de encriptar el mensaje a enviar utilizando la clave pública del receptor. Así, con este método se consigue que solo el receptor pueda desencriptar ese mensaje, utilizando para ello su clave privada creada con RSA (Ilustración

[39] Glosario: **Alice y Bob**
[40] Terminología: ***padding***
[41] Terminología: ***El Cipher Block Chaining***

31). Esto provee a la **criptografía de clave asimétrica** de un muy buen sistema de seguridad y de un sistema de distribución de claves. No obstante, este método requiere de muchos más recursos para encriptar el mensaje, lo que significa que es más lento que la **criptografía de clave simétrica**.

Los algoritmos utilizados suelen ser de 125, 256 y 512 bits, aunque hay versiones de 1024 y de 2048 en algunos, haciendo que cuanto más larga sea la clave más segura sea el sistema, pero haciendo necesario emplear más tiempo y computación del sistema para procesarla, además de ocupar más espacio el mensaje encriptado. Ejemplos de algoritmos de **criptografía de clave asimétrica** son «*Rivest, Shamir, Adleman*» (RSA) [388] y ElGamal [389].

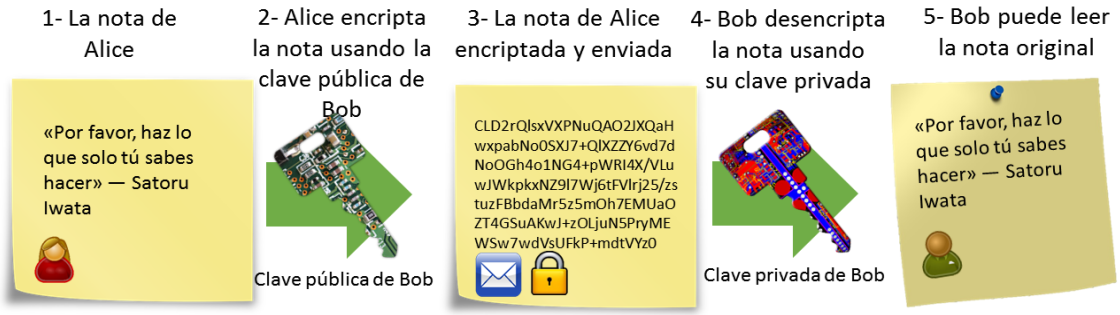


Ilustración 31 Ejemplo de encriptado utilizando el sistema de criptografía de clave pública usando RSA

7.5.3 CRIPTOGRAFÍA HÍBRIDA

La tercera opción es la **criptografía híbrida**. Este método criptográfico usa los tipos de criptografía vistos previamente: la **criptografía de clave simétrica** y la **criptografía de clave asimétrica**. En la **criptografía híbrida**, cuando el emisor quiere enviar un mensaje, el emisor tiene que encriptar el mensaje con la clave secreta. Después de esto, el emisor tiene que encriptar la clave secreta, utilizada previamente para encriptar el mensaje original, utilizando para ello la clave pública del receptor y añadir este resultado al mensaje para enviarle al receptor ambas: el mensaje encriptado con la clave secreta y la clave secreta encriptada con la clave pública del receptor. Así, el receptor obtiene la clave secreta para desencriptar el mensaje original utilizando su clave privada. Ejemplos de software de encriptación híbrida y que utilizan diferentes algoritmos criptográficos son *Pretty Good Privacy* (PGP) y OpenPGP [390], y *GNU Privacy Guard* (GnuPG) [391].

La **criptografía híbrida** es más lenta que la **criptografía de clave simétrica**. Sin embargo, es más rápida que la **criptografía de clave asimétrica**. En cuanto a seguridad, se podría decir que contiene lo mejor de ambos métodos, pues es casi o tan seguro como el sistema de **clave pública**, un poco más lento que el sistema de **clave secreta,** pero mucho más que el sistema de **clave pública**, y además soluciona el problema de distribución de claves, que es la solución aportada por el sistema de **clave pública**, pues esta es enviada encriptada junto al mensaje encriptado.

En la Ilustración 32 se muestra el ciclo de uso de la criptografía híbrida en un mensaje que le envía Alice a Bob. Alice tiene una nota que quiere enviarle y lo primero que hace es encriptarla con una clave secreta de criptografía simétrica como puede ser AES de 128 bits de bloque sin *padding* y en modo CBC, la clave es

«clave_privada_sc». Con esto obtiene el mensaje original encriptado. Tras esto, encripta la clave secreta con la clave pública de Bob y la añade al mensaje encriptado previamente y se lo envía a Bob. Una vez Bob recibe el mensaje, desencripta la clave secreta utilizando su clave privada. Con la clave secreta obtenida, que es «clave_privada_sc», desencripta el mensaje y obtiene la nota original.

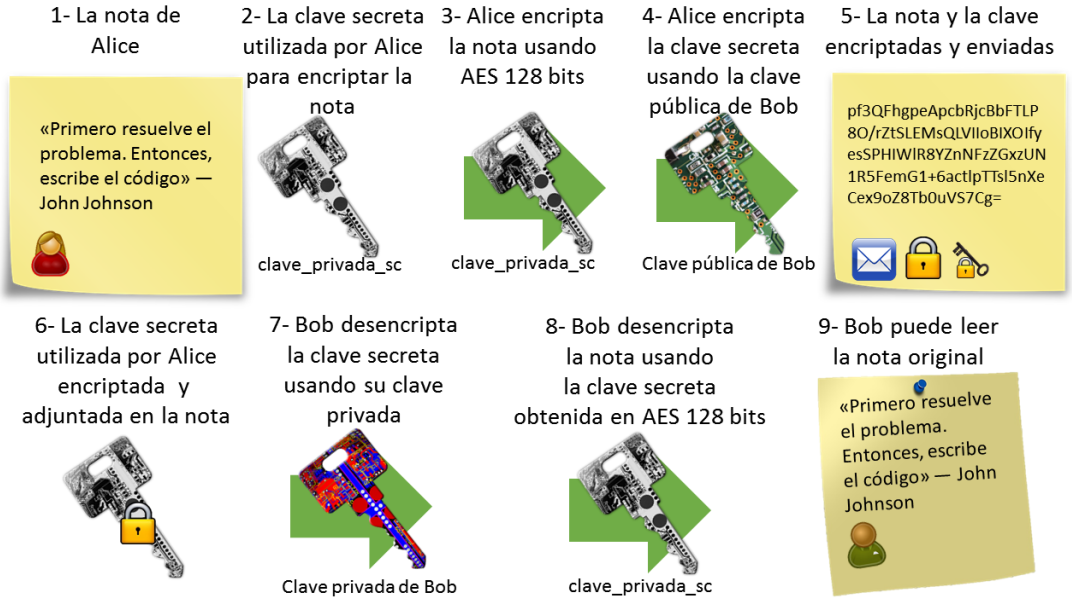


Ilustración 32 Ciclo de uso de la criptografía híbrida

7.5.4 FIRMA DIGITAL

Como bien se explicó, el sistema de **clave pública** posee dos claves, pero se utiliza la clave pública para distribuirla y que la use la gente para encriptar los mensajes y el destinatario guarda la privada para desencriptarlos. No obstante, en el caso de que el emisor encripte el mensaje utilizando su clave privada, cualquiera que tenga la clave pública del emisor puede desencriptarlo (Ilustración 33).

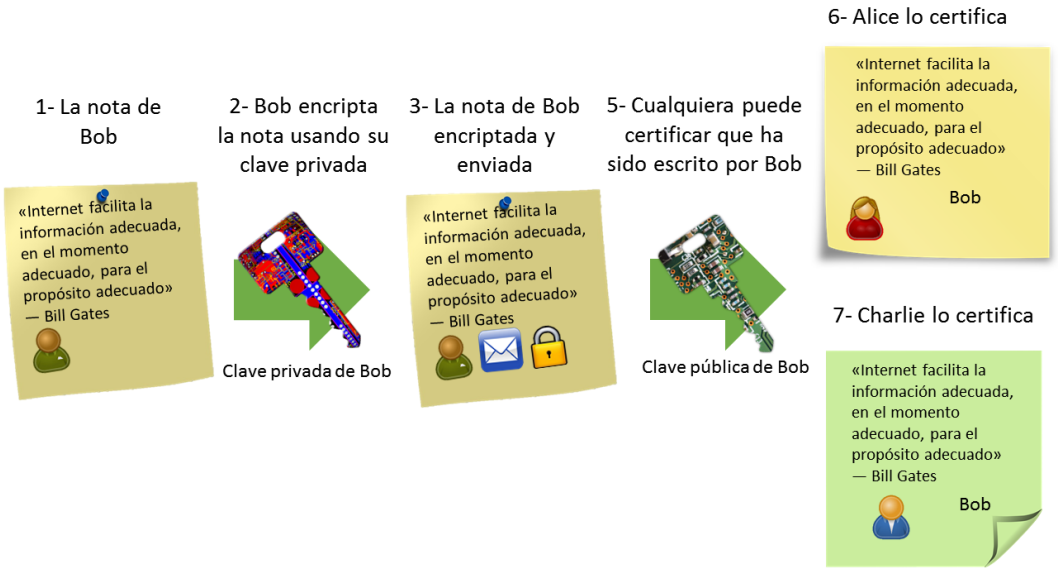


Ilustración 33 Uso del encriptado de la clave privada del encriptado asimétrico

Este método sirve para poder certificar que quien envió el mensaje es quien dice ser, debido a que, supuestamente, este es el único que debería de tener su clave privada, ya que son únicas. Este método surgió para asegurar la autenticación y se conoce como **firma digital** [359]. Así, cualquiera que tenga la clave pública del que encripto el texto, que puede ser cualquier persona, puede desencriptar ese texto, lo que sirve para validar que quien lo escribió es quien dice ser.

La **firma digital** debe de cumplir una serie de requisitos que son el ser *única* para cada persona, *infalsificable* gracias al algoritmo matemático, ser fácilmente *verificable* por el receptor, *no repudiable* y *viable* haciendo que sea fácilmente generada por el firmante. La **firma digital** a veces incorpora un resumen del mensaje para así dar **integridad** al mensaje por medio del resumen del mensaje original utilizando una **función Hash criptográfica**.

7.5.5 **FUNCIÓN HASH CRIPTOGRÁFICA**

Una **función Hash criptográfica** es un algoritmo que se aplica sobre unos texto y que devuelve un resumen del texto procesado, conocido como *hash* [359]. El uso de **funciones Hash criptográficas** sirve para verificar si el mensaje sufrió alguna modificación en el mensaje original y así comprobar su **integridad**.

Para considerarse una función como una **función Hash criptográfica**, el algoritmo de la función debe de cumplir una serie de características. La primera de ellas es que no se pueda obtener el mensaje original desde el mensaje resultante, así que debe de ser *unidireccional*. El *hash* obtenido, normalmente, debe tener una *longitud fija* de salida, independientemente de la longitud del mensaje de entrada. Como bien se comentó, debe de *comprimir* el texto. Además, estas funciones deben de ser *fáciles de calcular*. Por último, deben de tener una *colisión fuerte*, para así evitar que dos entradas distintas tengan la misma salida.

Así, el uso de **funciones Hash criptográficas** en la transmisión de los mensajes, permite al destinatario realizar un *hash* del mensaje recibido y comprobar si se corresponde con el *hash* adjuntado por el emisor y ver si el mensaje fue modificado o no por el camino. Por ello, muchas veces, el resumen realizado con **funciones Hash** acompaña a las firmas digitales, de manera que se hacen mucho más seguras. Algunos ejemplos de funciones hash bastante utilizadas son *Message-Digest Algorithm 4* (MD4) [392], MD5 [393], MD6 [394], *Secure Hash Algorithm 1* (SHA-1) [395]–[397], SHA-2 [396] y SHA-3 [398].

Un ejemplo de funcionamiento de una función hash es el mostrado en la Ilustración 34. En esta ilustración se ve como Alice tiene una nota que desea enviar a Bob. Para asegurar la integridad de la nota, Alice aplica una función Hash, exactamente la SHA3 de 256 bits a su texto, con lo que obtiene un resumen de este y lo adjunta junto a la nota para enviárselo a Bob. Una vez Bob ha recibido la nota, este crea un resumen, igual que hizo Alice, cogiendo el texto y pasándolo por la función Hash SHA3 de 512 bits y compara ambos resúmenes, el suyo y el adjuntado por Alice. Como estos son iguales, Bob sabe que el texto no ha sido modificado.

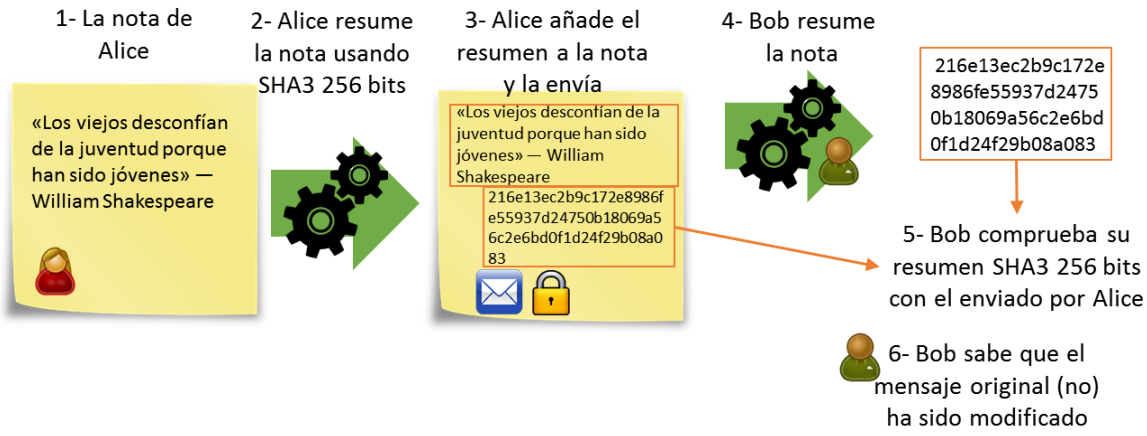


Ilustración 34 Uso del resumen de una función hash SHA3-512

7.5.6 COMPARACIÓN DE LOS MÉTODOS CRIPTOGRÁFICOS

En la Tabla 2 se puede ver una comparación entre las cualidades de seguridad de los diferentes métodos criptográficos. Como se puede ver, para obtener el mensaje más seguro y con mejor rendimiento, se debe de combinar la criptografía híbrida con la firma digital y añadirle el hash del mensaje. De esta manera se tiene solucionada la distribución de claves y se obtienen las cuatro cualidades que un mensaje seguro debe de cumplir.

Tipo de criptografía	**Distribución de claves**	**Confidencialidad y privacidad**	**Autenticación**	**No-repudio**	**Integridad**
Simétrica	No	Sí	No	No	No
Asimétrica	Sí	Sí	No	No	No
Híbrida	Sí	Sí	No	No	No
Firma digital	Sí	No	Sí	Sí	No
Hash	No	No	No	No	Sí

Tabla 2 Comparación entre las diferentes cualidades de los métodos criptográficos

7.6 *TRABAJO RELACIONADO EN EL MARCO DE IOT*

Hasta el momento, muchos investigadores han publicado diferentes métodos de seguridad para crear una red de IoT segura. Sin embargo, IoT está en su primera fase y todavía hay un largo camino por investigar [374]. Actualmente, IoT está en beta utilizando solo unos pocos objetos, mientras que la idea de IoT es conectar todo entre sí. Esto conllevará el procesamiento de muchos datos de objetos heterogéneos y ubicuos con todos los problemas que esto implica.

7.6.1 SOLUCIONES CRIPTOGRÁFICAS

Algunas investigaciones usan criptografía para mejorar la seguridad entre los diferentes objetos de una red de Internet de las Cosas. Por ejemplo, Mahalle et. al in [399] propuso usar criptografía en un protocolo de autenticación con un método de cifrado. Sin embargo, sus prototipos tienen el problema de que una tercera persona podría «esnifar» la red y copiar las claves para realizar un ataque de *replay*.

Otra solución para la distribución de claves es la propuesta de L. Eschenauer y V. D. Gligor [400]. En esta propuesta, los autores proponen como solución la predistribución de las claves en una red IoT. De acuerdo a conseguir esto, las claves tienen que estar en los nodos de sensores y para ello, usan un protocolo de descubrimiento compartido de clave simple para hacer la distribución de claves.

De acuerdo al uso de la criptografía de clave pública está la investigación [8]. En este artículo, los autores hablan sobre el uso de sensores y su limitación de capacidad limitada en una red inalámbrica. El problema es que los sensores no pueden asegurar la integridad, autenticación y la resistencia del mensaje a algunos tipos de ataque. Esto se debe a los recursos limitados que tienen los sensores.

Otras investigaciones sobre seguridad en IoT con criptografía son [401], [402], los cuáles proponen una autenticación basada en **criptografía de curva elíptica** (ECC). En esta investigación ellos eligen ECC para ahorrar energía y computación en los objetos. Sin embargo, esta investigación tiene el problema de que otra persona podría interceptar el mensaje, modificarlo y enviarlo modificado, debido a que no aseguran la integridad de este.

Como otra posible solución criptográfica, no enmarcada en IoT pero aplicable, es la fusión entre la criptografía y la esteganografía propuesta por [359]. De esta manera, se puede enviar un mensaje encriptado utilizando un fichero para ocultar la comunicación, lo que nos permitiría comunicarnos secretamente y, en caso de que fuera descubierta la comunicación, el mensaje no podría ser leído.

7.6.2 MEJORAS EN RFID UTILIZANDO CRIPTOGRAFÍA

Algunas propuestas para mejorar RFID son [403], [404]. En estas, los autores presentan un sistema criptográfico simétrico usando para encriptar el mensaje el algoritmo AES [379], [380].

Otra investigación añade pseudoruido a la comunicación entre el lector RFID y la tarjeta RFID, donde la tarjeta RFID es la que modula la señal. Así, ellos evitan la intercepción del mensaje por lectores RFID maliciosos [405].

7.6.3 OTRAS SOLUCIONES CRIPTOGRÁFICAS

Por el contrario, otros investigadores trataron de abstraer más las posibles soluciones. En [406], los autores proponen el uso de un proxy para filtrar las peticiones entre los usuarios y los servicios para trabajar solo con los objetos autorizados.

Acerca de los protocolos, [407] propone una implementación *Constrained Application Protocol* (COAP) sobre *Datagram Transport Layer Security* (DTLS). Los autores usaron el algoritmo AES, que es un algoritmo

de criptografía de clave simétrica debido a que es más rápido que usar la criptografía de clave asimétrica. Sin embargo, ellos necesitan un tercer nodo para comprobar la comunicación entre dos nodos y certificarlos. Además, ellos muestran que este sistema es vulnerable a ataques del tipo *Man-in-the-Middle* [42].

[42] Glosario: **Man-in-the-Middle**

8. REDES SOCIALES ONLINE

«Todos nosotros nos conectamos, como una red que no podemos ver»
Mickenberg y Dugan, Taxi Driver Wisdom, 1995

«La pregunta no es: ¿Qué queremos saber de la gente?, sino: ¿Qué quiere decir la gente de sí misma?»
Mark Zuckerberg

«La información es poder»
Bill Gates

«El ser humano es un ser social por naturaleza, y el insocial por naturaleza y no por azar o es mal humano o más que humano... La sociedad es por naturaleza y anterior al individuo... el que no puede vivir en sociedad, o no necesita nada por su propia suficiencia, no es miembro de la sociedad, sino una bestia o un dios»
Aristóteles, en «El animal social» de Elliot Aronson

Las redes sociales llevan existiendo desde que existe la humanidad, pues describen como se relaciona un ser humano con el resto de su especie o de las organizaciones con las que tiene una relación. En el siglo XX esto fue llevado a la red gracias a la combinación de Internet con los servicios web, siendo conocido como Redes Sociales Online. Esto hizo que la gente pudiera mantener contacto a través de Internet con la gente que había conocido en el mundo real. Posteriormente, estas se fueron especializando y permitiendo que no solo la gente se dedicase a mantener el contacto, sino que, gracias a la creación de un perfil virtual, pudieran conocer personas nuevas y afines a un tema, gusto o finalidad.

Esto permite que incluso los usuarios puedan mantener su conexión o conocerse, aunque estén en los sitios más remotos de la Tierra, desde la tundra Siberiana de Rusia hasta la selva del Amazonas en Brasil, desde Alert en la Isla de Ellesmere, en Canadá, hasta el Cabo de la Buena Esperanza en Sudáfrica, o desde la Isla de Pascua en el Océano Pacífico hasta Barneo en el Polo Norte. Es más, estamos conectados a cualquier persona por un máximo de seis grados de distancia.

Las Redes Sociales Online, por lo tanto, son contenedores de información humana, tanto pública como privada, constituyendo así un gran almacén de información creada por los humanos y que alberga sus hábitos, gustos y actividades, entre otras cosas. Esta información, en muchos casos, es recopilada por las empresas, pues es muy valiosa para ellas, y pueden hasta conseguirla gratis, o solo por dejarte jugar. Además, esta información es un buen complemento de la información recogida por sensores y por objetos inteligentes, sirviendo así para crear muchas aplicaciones que combinen esta información con la nube como son sistemas de recomendación de libros, juegos o sistemas de asistencia en tráfico o viajes.

Por esto, en este capítulo se hablará de las Redes Sociales Online, introduciéndolas y determinando que son, para así dar paso a su historia y sus posibles clasificaciones, hablar de las dos más importantes en el

panorama científico, Facebook y Twitter, comentar la teoría de los seis grados de separación y el trabajo relacionado existentes, desde el punto de vista de Internet de las Cosas y desde la influencia en las personas.

*****

8.1 *INTRODUCCIÓN*

Las redes sociales, también conocidas en la literatura como *Online Social Networks* (OSN) o *Social Network Sites* (SNS) [408], o *Social Networking Sites* (SNS) si se quiere hacer hincapié en la relación [408], están consideradas como una pieza importante para lograr la convergencia entre el mundo real y el mundo digital en la Web 2.0 [409].

Las redes sociales permiten mantener el contacto y compartir cosas con gente que conoces y en la que confías como son familiares, amigos, compañeros de trabajo u otras personas ya existentes en la propia red social de la vida real [408], [410]. Por otro lado, hay otras redes sociales que proporcionan la base para encontrar, conocer y mantener relaciones sociales con otros usuarios que tienen intereses parecidos o gustos similares [411], pues son OSNs muy específicas. Todo depende del tipo de OSN que sea. Además, las OSNs son consideradas como lugares para ponerse al día acerca de la información personal y las actividades actuales de las relaciones sociales [412].

Por ejemplo, dentro de estas, algunas, como Facebook [43], te permiten añadir aplicaciones y utilizarlas, mientras que otras te ofrecen servicios extendidos mediante pagos, como es LinkedIn [44]. Otras en cambio, solo muestran tu perfil a tus «amigos», como es MySpace [45], te muestran que usuarios visitaron tu perfil, como LinkedIn, o depende de la configuración de seguridad que se tenga seleccionada, como en Facebook. En otros casos las hay que solo son específicas de móviles, como Dodgeball, y otras son web, aunque ofrecen interacción móvil limitada, como Facebook, MySpace y Cyworld [46]. Hay muchos tipos de OSNs, y las hay que son muy diferentes.

De entre todas las OSNs que han existido Facebook es la red social más famosa [413], convirtiéndose rápidamente en la herramienta más popular para la comunicación social [414], pues Facebook es algo diferente, ya que aquí la mayoría de la gente que son amigos en la red han sido conocidos en la vida real y han sido añadidos más tarde. Estas relaciones que se crean en las OSNs entre las personas y todo el mapa global de todo el mundo y como se relacionan entre ellos es lo que Mark Zuckerberg denominó como **Grafo Social** [415]. Esto se debe a que las redes sociales suelen modelarse de forma natural como grafos, siendo así las entidades [o personas] los nodos del grafo y las relaciones entre las entidades las aristas [416].

Mucha de esta socialización entre personas a través de Internet se consigue debido a que las OSNs permiten enviar mensajes o chatear con los amigos, permiten la generación de contenidos creados por los propios usuarios, permiten compartir diferentes contenidos externos o de amigos y navegar por los perfiles de tus amigos o de la lista de amigos de estos. Todos los días millones de ciudadanos comparten sus observaciones, pensamientos, sentimientos y experiencias acerca de su ciudad a través de actualizaciones en las redes sociales [417].

[43] https://www.facebook.com/
[44] https://www.linkedin.com/
[45] https://myspace.com/
[46] http://www.cyworld.com/

Redes Sociales Online

Muchos investigadores han examinado las OSN para tratar de entender las prácticas, implicaciones, la cultura y el significado de los sitios y el compromiso de los usuarios con ellos, especialmente en la actualidad con Twitter [418]. Anteriormente, a fecha de 2011, Facebook era la red social sobre la que más artículos científicos se publicaban, seguida de MySpace [419]. Ambas, Twitter y Facebook, son usadas frecuentemente con fines de investigación [413], [418], pues son una muy buena fuente cuando se necesita recopilar datos sobre personas y/o eventos para así poder interpretar esta información como una forma de sabiduría colectiva, lo que permite que sean utilizadas para investigar la predicción de resultados en el mundo real [420]. Gracias a ellas se pueden obtener las relaciones entre personas, que están caracterizadas por los contenidos que comparten los diferentes individuos, quienes poseen una serie de atributos como son la edad, el sexo, las creencias y la actitud, entre muchas otras posibles [421].

Las Redes Sociales Online se usan mucho en el lado académico, en investigación, pues están siendo muy importantes en el día a día gracias a su facilidad de uso, velocidad y alcance [408], [420], pues están cambiando rápidamente el discurso público de la sociedad, estableciendo tendencias y agendas en temas muy importantes, como son el medio ambiente, la política, la tecnología y la industria del entretenimiento [420]. Actualmente están surgiendo investigaciones para darles una capacidad social a los objetos de Internet de las Cosas, pero aquí el principal problema encontrado es el permitir a la red mejorar el nivel de confianza entre los objetos y los «amigos» en la propia red [422]. Esto es algo que se consigue mediante el acceso de al perfil del usuario y sus mensajes, como bien se verá en el trabajo relacionado de las Redes Sociales en el marco de Internet de las Cosas. Hay varias propuestas sobre la combinación de OSNs e Internet de las Cosas basadas en una deducción hecha por científicos de Ericsson [422], pues estos científicos observaron que las personas se familiarizan mucho mejor con las tecnologías de Internet de las Cosas si existe una analogía entre estas y los hábitos diarios en las redes sociales como Twitter o Facebook. Muchas de las propuestas realizadas se basan en aportar a los objetos inteligentes habilidades de socialización con el fin de establecer relaciones entre los propios objetos. Este concepto se denomina Internet de las Cosas Social (SIoT), y fue presentado en [51], [422]. Sin embargo, para poder utilizar una OSN en el marco de IoT esta debe de cumplir ciertas cosas, como que tenga servicios de identificación y autorización, ofrezca una API que permita acceder y manipular el grafo de la OSN, publicar y leer y añadir lugares del mundo físico, y ofrecer una forma fácil de crear aplicaciones de terceros [409].

Pero no solo las OSNs se utilizan para estar en contacto con gente y para la investigación, pues la publicidad correcta de un producto en las Redes Sociales puede ser un buen indicador de rendimiento en la realidad [420]. Por ejemplo, según un estudio el 21% de los adultos de Internet estuvieron comprometidos con las campañas políticas a través de Facebook o MySpace y el 2% a través de Twitter en los meses previos a las elecciones de noviembre del 2010 de los Estados Unidos de América [423]. Es decir, se utiliza para política y publicidad también debido al gran impacto que tienen las noticias en las OSN gracias a la gran cantidad de gente que las utiliza y con la que están conectadas, gracias al Grafo Social y a que las personas compartan sus gustos e intereses con sus contactos, así como lo que ellos consideran interesante. Esto es gracias a que las redes sociales empezaron a ser un fenómeno global con un impacto social y económico enorme, pero solo desde hace unos pocos años [413].

Debido al volumen de datos las redes sociales sirven como fuente de investigación de mercados [413]. Esto es así ya que se demostró que sirven también para obtener una valiosa ayuda para interactuar con los consumidores y permitiendo disminuir los costes de desarrollo, como ha pasado con Fiat y Lego [424]. Fiat recibió más de 170.000 conceptos de diseños gratis mediante la integración de sus clientes en el proceso de desarrollo del modelo Fiat 500. En el caso de Lego la comunidad sugirió un nuevo modelo de Lego que fue evaluado y comentado por otros usuarios. En otros casos se pueden crear anuncios que fácilmente se pueden convertir en virales u obtener inteligencia de mercado mediante la recogida y análisis de los datos que generan los usuarios [419]. Todo esto dio lugar al concepto de la empresa 2.0, que describe la adopción de software social en el contexto de la empresa, siendo el software social aquel que es generado en Internet como puede ser en redes sociales, blogs, wikis u otros similares [419].

Las redes sociales ya son una parte de nuestras vidas [425], pues, desde su introducción, las redes sociales como MySpace, Facebook, Cyworld y Bebo [47] han atraído millones de usuarios, quienes las han integrado en su vida diaria [408]. Son tan importantes en nuestras vidas que incluso el uso de las OSNs puede ayudar a los usuarios a conocer y asentarse en un nuevo entorno social, según los hallazgos derivados de OSNs públicas [419]. Muchas de estas OSNs ayudan a mantener el contacto prexistente entre la gente, mientras que otras ayudan a conectarse con gente nueva basándose en los intereses, gustos o ideas de ambos [408]. Son tan importantes ya que incluso algunas redes sociales, como Facebook, ya no solo sirven para mantener la comunicación entre amigos, pues algunas compañías tienen aplicaciones integradas para obtener tus credenciales cuando tienen que pedírtelas, como es el caso del aeropuerto de Calgary en Canadá [425].

8.2 *¿QUÉ ES UNA RED SOCIAL ONLINE?*

En este aspecto para responder a la pregunta hay que tener mucho cuidado. Por red social se entiende cualquier estructura social en la vida real compuesta de actores, ya sean personas u organizaciones, que están relacionados entre ellos, ya sea por familiaridad, profesión, amistad, etc. Sin embargo, por otro lado, están las Redes Sociales Online, conocidas como OSNs, que son la misma idea, pero trasladada a Internet, y es lo que se estudia en esta tesis. Referente a las OSNs hay muchas definiciones, todo depende de cual se coja, aunque tienen muchos puntos en común.

La primera es la de danah m. boyd y Nicole B. Ellison dada en [408], y que data del año 2007:

> «Una OSN es un servicio web que permite a las personas construir un perfil público o semipúblico dentro de un sistema acotado, interactuar con otros usuarios con los que se comparte una conexión, y ver y recorrer la lista de conexiones y las acciones realizadas por otros dentro del sistema, las cuales dependen de la OSN, al igual que ocurre con las nomenclaturas».

Como se ve, esta primera definición explica primero de todo que debe de ser un servicio web y segundo las tres características que debe de tener para ser considerado una OSN. Estas tres cosas son: permitir crear

[47] https://bebo.com/

un perfil público o semipúblico, interactuar con otros usuarios, y ver la lista de amigos y de acciones de otras personas. Como se ve, esta definición se centra en la interacción con los demás y en el poder ser visible y ver al resto de gente, como si se saliese por una calle un día cualquiera.

La segunda definición que podemos coger es la dada por Fabian Schneider, Anja Feldmann, Balachander Krishnamurthy y Walter Willinger [426], en el año 2009:

> «Las OSNs forman comunidades online entre gente con intereses, actividades, trasfondos y/o amistades en común. Muchas OSNs están basadas en la web y permiten a los usuarios subir perfiles (texto, imágenes y videos) e interactuar con otros de numerosas formas».

Como se ve, lo primero que dice la definición es que deben de ser online y que permitan crear comunidades entre la gente. Además, recalca que las OSNs deben de permitir subir diferentes elementos a la red social para que así la gente interactúe con estos elementos. Sin embargo, esta definición no hace mención al tipo de perfil ni a las listas de amigos, luego, se puede interpretar que podrían ser privados, ofreciendo algo de privacidad a sus usuarios.

Otra posible definición es la dada por Jure Leskovec, Anand Rajaraman y Jeffrey D. Ullman [416], siendo mucho más actual pues es del año 2014:

> Para ellos las OSNs tienen que tener 3 características esenciales:
>
> 1. Tener una colección de entidades participativas en la red, las cuáles son normalmente la gente, pero que podrían ser otras cosas distintas.
> 2. Tener al menos una relación entre dos de estas entidades. Por ejemplo, en Facebook esta relación son los amigos, que es una relación de todo o nada, pues o son ambos o no existe relación. Mientras, en Google+ hay grados que los puede asignar el usuario por medio del uso de círculos. En cambio, en Twitter, un usuario puede seguir a otro, pero este otro no tiene por qué seguir al primero, es decir, las relaciones no tienen por qué ser recíprocas.
> 3. Las condiciones tienden a agruparse, es decir, si la entidad A se relaciona con las entidades B y C, entonces hay una probabilidad más alta de lo normal que B y C están relacionadas también.

En esta última definición se encuentran tres características que las OSNs deben de poseer, siendo la primera gente que pueda participar en la red creando contenido e interactuando, sin embargo, si que dice que podría ser algo diferente. Aquí encajaría, por ejemplo, Facebook, pues actualmente está investigando en Inteligencia Artificial para gestionar el contenido [48]. Como segundo punto aclara que estas entidades deben de tener relaciones, es decir, poder crear diferentes tipos de lazos entre la propia gente. Por último, la tercera

[48] https://research.facebook.com/ai/

condición aclara que en las OSNs suelen crearse grupos entre la gente que tiene conocidos en común. Como se ve, esta definición se centra en la socialización de la propia red.

Además de estas, hay otras pequeñas definiciones dadas por otros autores como son:

- Una red social puede ser considerada una red de ordenadores que conecta gente u organizaciones [421].
- La estructura de una red social es esencialmente una organización virtual dinámica con relaciones de confianza inherentes entre amigos [425].
- Una red social es una comunidad virtual y una plataforma de conexión donde la gente puede crear y compartir contenido, ideas y experiencias libremente. Son ambientes libres que ofrecen a los usuarios libertad de creación, manipulación y diseminación de la información [427].
- Las OSN permiten a sus usuarios actuar independientemente y construir su identidad virtual configurando su perfil, siendo la creación y el uso de las relaciones ya existentes o las nuevas que se creen el motivo central del uso de las OSN [413].

Como se ve, en estas últimas definiciones dan pequeños matices, una centrándose únicamente en el hardware que las mueve, pero la gran mayoría en la parte social, en que las redes sociales son sitios libres que permiten socializar con otra gente y contar tus cosas.

En base a estas definiciones y otros pequeños matices, se puede crear una definición que incluya y defina lo que es una red social online de una forma más detallada:

Una OSN es una red virtual online dinámica que permite **crear una identidad virtual**, normalmente basada en la del mundo real, para **socializar con otras entidades**, conocidas previamente o no, y que pueden ser personas, empresas u otros medios, con unos intereses, actividades, trasfondos y/o amistades en común, con el fin de **crear una Red Social Online navegable** e **interactuable mediante la creación, compartición y comentario de contenidos** (texto, imágenes, videos, …) libremente, y que permite **crear conexiones con las personas**, creando lazos, grupos y/o comunidades, siendo esta su principal finalidad.

8.3 *HISTORIA*

La historia de las Redes Sociales Online es larga y está llena de muchas OSNs que han triunfado, pero de pocas que hayan sobrevivido. Aquí se muestran las consideradas más importantes, en base a la definición previa, al estudio de las existentes, y a las ya estudiadas por [408], [413], [419], [424], presentando cada autor unas diferentes y coincidiendo en algunas otras. Lo mismo ocurre con los años de inicio, pues algunos autores no consideran el año de su creación como el año de inicio, y lo consideran a partir de cuándo implementó ciertas características. Por estos motivos, a continuación, se presentan las OSNs que, en mi opinión, son las

más relevantes, y dependiendo de la OSN, el año puede variar, coincidiendo con alguno de los autores o no. Además, este apartado actualizó algunos años con OSNs que han sobrevivido convirtiéndose en importantes y que los autores previos no consideraban, y se ha cubierto hasta el año 2015, siendo 2006, 2010 y 2011 los años límites cubiertos por los anteriores autores. En la Ilustración 35 se pueden ver los años de aparición de las OSNs surgidas entre los años 1995 y 2004.

1995	1997	1998	1999	2000	2001	2002	2003	2004
Classmates	SixDegrees.com	Xanga	LiveJournal	LunarStorm	Ryze	Fotolog	MySpace	Orkut
	AsianAvenue	Care2	BlackPlanet	MiGente	Jappy	Friendster	LinkedIn	Dogster
			Cyworld	Trombi	Kwick	StayFriends	Tribe.net	aSmallWorld
						Last.FM	Open BC/Xing	Flickr
						Reunion	Hi5	Mixi
						BeautifulPeople	Multiply	Facebook (Harvard)
						Draugiem.lv	WAYN	Catster
						Meetup	Nexopia	Hyves
							Zorpia	Tagged
							Visible Path	Vimeo
							Netlog	MSN Spaces
							Piczo	Grono
							Dodgeball	Couchsurfing

Ilustración 35 OSNs aparecidas entre los años 1995 y 2004

La primera OSN, según [419], apareció en 1995, llamada Classmates.com [49]. Classmates quería conectar a la gente que había estudiado en una misma guardería, colegio, instituto o universidad.

Sin embargo, según danah m. boyd y a Nicole B. Ellison [408], la primera red social online apareció en el año 1997 y fue SixDegrees.com. Esta permitía a los usuarios crearse perfiles y añadir amigos, para posteriormente en 1998 permitir navegar entre las listas de amigos. Cabe destacar que el perfil ya existía previamente en la mayoría de los sitios de citas, que AIM e ICQ permitían listas de amigos, aunque estas no eran visibles para los demás, y que Classmates.com sí que permitía navegar por las listas de amigos, pero fue SixDegrees.com la primera en combinar estas tres características. Posteriormente SixDegrees.com añadió la posibilidad de enviar mensajes entre gente, pero terminó cerrando en el año 2000 por culpa de que no consiguieron realizar un negocio sostenible [408].

Entre 1997 y 2001 surgieron varias OSNs que combinaban las tres características y que fueron añadiendo otras características, como permitir crear diferentes tipos de perfiles (personal, profesional, citas). Ejemplos de OSNs de esta época son AsianAvenue [50] (ahora AsianAve) en 1997 para conectar a asiáticos, BlackPlanet[51] en 1999 para conectar a gente de raza negra, MiGente en 2000 [52] para conectar a gente de Latinoamérica. Otras OSNs optaron por añadir el envío de mensajes instantáneos a los amigos como en LiveJournal [53] (1999),

[49] http://www.classmates.com/
[50] http://www.asianave.com/
[51] http://www.blackplanet.com/
[52] http://www.migente.com/
[53] http://www.livejournal.com/

o añadieron libros de visita y diarios como LunarStorm (2000) [408]. También en 1999 apareció Cyworld en Corea del Sur. Ya en 2001, Ryze apareció como una OSN de negocios mientras lo enlazaban con la parte personal, siguiendo estos pasos posteriormente otras redes sociales como LinkedIn.

Mientras, Friendster [54] surgió en el año 2002 como sitio de citas [428], en competencia de Match (1995) [413], pero restringiendo los contactos a 4 grados de distancia [429], y ayudando a conocer a amigos de amigos suponiendo que era más romántico conocer a amigos de amigos que a desconocidos. Friendster es considera la primera social que tuvo un éxito notable [419]. También en el 2002 hicieron su aparición Last.FM[55] para registrar los hábitos de música de la gente y conectarlos por gustos musicales, y Fotolog [56] con el fin de que sus usuarios pudieran compartir fotos. Una red de OSNs aparecida en el año 2002 fue Meetup[57], que permite crear grupos para reunir a los usuarios que tengan intereses comunes y ayudarles a reunirse en la vida real. En 2003 tuvo un crecimiento exponencial que hizo que esta OSN fuera la más grande hasta el año 2004, convirtiéndose en el servicio dominante en Singapur, Malasia y Filipinas. Sin embargo, en este mismo año comenzó a tener problemas técnicos, de caídas de la red, y sociales, pues la gente contactaba con sus compañeros de clase y sus jefes [408]. Esto haría más adelante que mucha gente migrase a MySpace [413], aparecida en el año 2003.

En 2003 hubo una explosión de OSNs que fueron apareciendo [408], significando este año y los posteriores el surgimiento de la mayoría de redes sociales que son populares a día de hoy [419]. Unas de ellas eran las profesionales como LinkedIn, Visible Path [58] y Xing [59], conocida previamente como Open BC. Otras eran las centradas en los perfiles de los usuarios y sus intereses, intentando imitar así la que había sido tan grande, Friendster, y siendo estas Dogster [60] (2004) para los dueños y amantes de perros, Care2 [61] (1998 y relanzada en el 2004) para ayudar a conocerse entre los activistas, Couchsurfing [62] (2004) para conectar viajeros con gente con sofás, MyChurch [63] (2006) para conectar a las iglesias cristianas y sus miembros, y Tribe.net [64] (2003) que se centraban en la recomendación [429]. En 2003 también apareció por fin MySpace con el propósito de hacer la competencia a Friendster, Xanga [65] (1998) y AsianAvenue, de acuerdo con las palabras de su cofundador Tom Anderson [408], recogiendo así a los usuarios frustrados de la OSN Friendster debido a los fallos que esta tenía [413]. Sin embargo, MySpace creció rápido gracias a la adopción de modelos de pago por parte de la OSN Friendster [408]. Tras esto, varias bandas de Rock Indie comenzaron a crear cuentas en MySpace para promocionarse y los promotores locales con el fin de informar. Esto hizo que MySpace contactase con los músicos locales para ver cómo podría ayudarles, pues en esta época MySpace

[54] http://www.friendster.com/
[55] http://www.last.fm/
[56] http://www.fotolog.com/
[57] https://www.meetup.com/
[58] https://www.cnet.com/news/social-networking-company-visible-path-to-be-acquired/
[59] https://www.xing.com
[60] http://www.dogster.com/
[61] http://www.care2.com/
[62] https://www.couchsurfing.com
[63] http://www.mychurch.org/
[64] http://www.tribe.net
[65] http://xanga.com/

no estaba centrada en la música, lo que supuso que muchos adolescentes entraran en masa para estar en contacto con sus grupos de música [408].

Ya en el año 2004 hubo un crecimiento del fenómeno de creación de contenido por el usuario y con ello las OSNs como Google Orkut [66], que se convirtió en la red social nacional de Brasil, y Windows Live Spaces de Microsoft (en diciembre 2004 como MSN Spaces y relanzada en el año 2006 bajo este nuevo nombre) que tuvo una tibia acogida en los Estados Unidos de América, pero fue extremadamente popular en otros lugares [408]. También, a principios de este año apareció Facebook dando soporte a redes de algunas universidades, como fue en Harvard, y requiriendo una cuenta de correo de estudiante de esta universidad. Poco a poco fue creciendo a otras universidades mediante el requisito de poseer una cuenta académica. También apareció Flickr [67] en el año 2004 de la mano de Ludicorp, una OSN para compartir fotos y videos y que fue adquirida por Yahoo! en el año 2005 [68]. Otra OSN que se mantuvo desde este año ha sido Vimeo [69], una OSN basada en videos.

La popularidad de una red social se puede demostrar por el número de gente que la utiliza [430], y en esta época el mundo estaba diseminado entre diferentes redes sociales muy populares [408]: MySpace en Estados Unidos Cyworld en Corea del Sur. En esta época también tuvieron éxito los blogs que tenían todas las características de las OSN, como sucedió en Estados Unidos de América con Xanga, LiveJournal y Box, Skyrock en Francia, y Windows Live Spaces en países del mundo, entre los que destacaron México, Italia y España. En la Ilustración 36 se muestra un mapa con las redes más populares entre los años 2003 y 2005, aproximadamente.

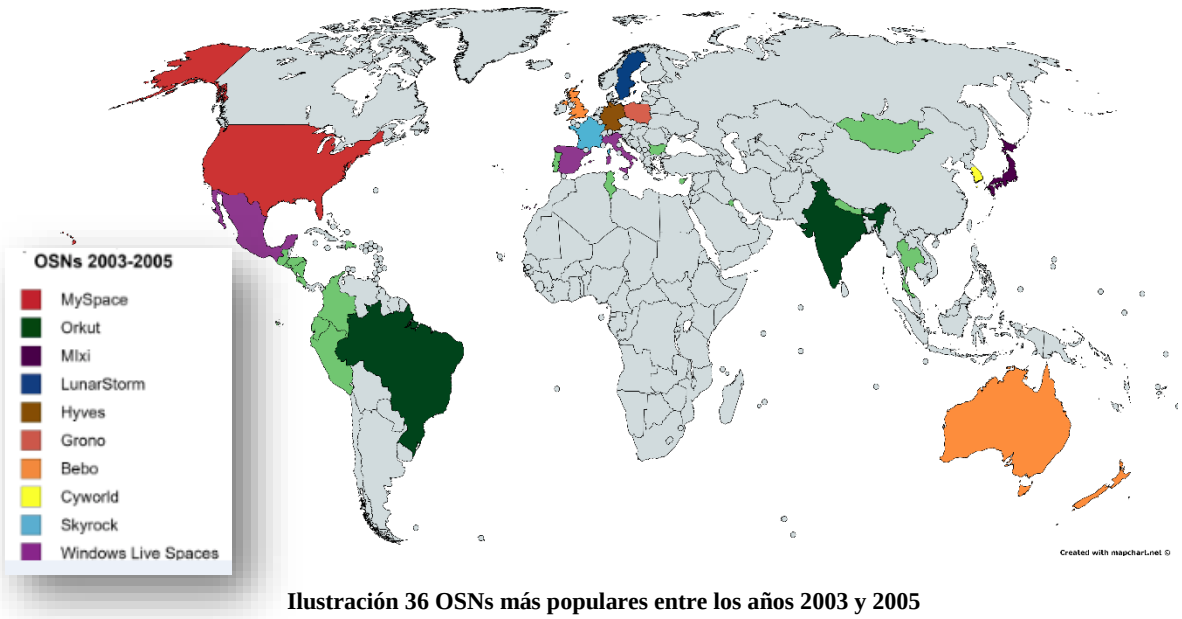


Ilustración 36 OSNs más populares entre los años 2003 y 2005

[66] http://orkut.com/
[67] https://www.flickr.com/
[68] https://www.theguardian.com/technology/2016/jul/25/yahoo-moves-next-for-flickr
[69] https://vimeo.com/

En el año 2005 hubo otra explosión de OSNs, la cual se puede ver en la Ilustración 37, y donde se muestran las OSNs aparecidas entre los años 2005 y 2015. En septiembre de 2005 Facebook se expandió a los institutos, a redes corporativas y al poco para todo el mundo [419]. Las características que diferenciaban a Facebook era el no poder hacer un perfil totalmente público a sus usuarios y el permitir a los desarrolladores crear aplicaciones para esta red social. Otra OSN del año 2005 fue la española Menéame [70], OSN de noticias en el que los usuarios registrados las pueden crear para cualquier persona las pueda ver, este registrada o no. En esta época también surgió el que sería una de las más grandes OSNs, exactamente la gran dominadora en China, Qzone [71], que permite tener blog, diarios, subir fotos, escuchar música y ver videos. En este año también apareció una de las que sería de las más grandes, YouTube [72], que de primeras solo permitía compartir videos pero que, con el tiempo, fue convirtiéndose en red social debido a que fueron añadiendo funcionalidades hasta convertirse en lo que se puede llamar una red social online de videos [408].

2005	2006	2007	2008	2009	2010	2011	2012	2014	2015
Facebook (Intitutos)	Windows Live Spaces (Relaunch)	Ning (144k OSNs)	Ning (700k OSNs)	Ning (1m OSNs)	Facebook (500m)	Google+	Vine	Swarm	Periscope
Blackplanet (Relaunch)	Facebook (Empresas)	Flixster	Facebook (100m)	Facebook (300m)	Audimated	Unthink			Meerkat
Asian Avenue (Relaunch)	Facebook (Todos)	Ravelry	The Sphere	DailyBooth	diaspora*	So.cl			
Youtube	Elgg	Platinnetz	MeinVZ	Foursquare	Folkdirect	Snapchat			
Bebo	Wer kennt wen	Sonico	Gays.com	SocialGo	Pinterest	Folksdirect			
Yahoo 360	CafeMom	Livemocha	ResearchGate	Sina Weibo	Instagram	MeetMe			
Ning	MyChurch	SchulerVZ	Plurk		Google Buzz				
Lokalisten	Tuenti	Wis.dm							
myYearBook	VK	Bahu							
Buzznet	Odnoklassniki	SkyRock							
Xiaonei/renren	Itsmy	Tumblr							
StudiVZ	SocialEngine	Geni.com							
Menéame	Badoo								
Qzone / QQ	Qype								
	Twitter								

Ilustración 37 Años de nacimiento de las OSNs surgidas entre 2005 y 2015

A partir de aquí había redes que se enfocaban en crecer exponencialmente mientras que otras se reducían [408]. Por ejemplo, aSmallWorld [73] (2004), BeautifulPeople [74] (2002 en Dinamarca, 2005 USA y UK, y 2009 globalmente) y Draugiem.lv [75] (2004) tenían el acceso restringido para parecer selectivas y de élite, otras estaban centradas en una actividad en concreto como Couchsurfing, dirigidas a la identidad como BlackPlanet, o enfocadas a la afiliación como MyChurch, lo que hacía que están tendiesen a ser pequeñas. Además, aquí surgió Ning [76] (2005), que era un servicio de hospedaje para crear redes sociales que permitía y ayudaba a

[70] https://www.meneame.net/
[71] http://qzone.qq.com/
[72] https://www.youtube.com/
[73] https://www.asmallworld.com/
[74] https://www.beautifulpeople.com
[75] https://www.draugiem.lv/
[76] https://www.ning.com

crear una red social a cualquier persona. Posteriormente surgieron otros servicios similares a Ning, como son Grou.ps [77], SocialGo [78] (2009), SocialEngine [79] (2006) y diaspora* [80] (2010), algunas de pago y otras gratis.

En cambio, otras OSNs surgían como sitios de citas, pero con funcionalidades de redes sociales, como fue el caso de Badoo [81] en el año 2006. Otra OSN que surgió, pero solo a nivel de España fue Tuenti [82], que sería la clara dominadora en España entre 2009 y 2012. También en este año aparecieron dos redes sociales rusas. Una de ellas es VK [83], conocida primeramente como Vkontakte y que significa «en contacto», y que al principio era solo para estudiantes rusos. La otra OSN rusa que apareció en este año fue Odnoklassniki [84], que significa «Compañeros de clase», y es la segunda OSN más utilizada en Rusia, por detrás de VK. Aquí también surgió la que es ahora una de las más populares OSNs, Twitter [85], siendo así un servicio de microblogging.

Un año más tarde, en el 2007, apareció Tumblr [86], una OSN muy parecida a Twitter. En este año también hubo un cambio importante en SkyRock [87], un blog lanzado en el año 2002 que se convertía en red social.

Además, en los años 2006 y 2007 sucedieron varias cosas relacionadas, como que ciertas investigaciones en marketing revelaron que las OSNs estaban creciendo en popularidad, lo que hizo que muchas empresas comenzaran a invertir en ellas tiempo y dinero para crear, comprar, promocionar y publicitar en OSNs. Sin embargo, también hubo problemas, pues se les baneó el acceso a MySpace a los militares de los Estados Unidos de América, el gobierno de Canadá prohibió utilizar las OSNs en el trabajo, y el congreso de los Estados Unidos de América propuso una legislación para banear a la juventud el uso de las OSNs en escuelas y librerías.

En el año 2008, una OSN que apareció y perduró desde entonces y siguió subiendo en usuarios ha sido ResearchGate [88], una OSN para compartir los artículos cualquier investigador del mundo. También en este año salió Plurk [89], siendo muy similar a Twitter al poseer envío de mensajes cortos de 140 caracteres, siendo muy popular en Taiwán, Filipinas, Indonesia y Estados Unidos.

Posteriormente, en el año 2009 apareció Sina Weibo [90], que es una OSN China que es una fusión entre el estilo de Twitter y el de Facebook y que cuenta con un 30% de los usuarios de China [91] [92]. Sin embargo, esta

[77] http://grou.ps/home
[78] https://www.socialgo.com/
[79] http://www.socialengine.com/
[80] https://diasporafoundation.org/
[81] https://badoo.com
[82] https://www.tuenti.com
[83] https://vk.com/
[84] https://ok.ru/
[85] https://twitter.com/
[86] https://www.tumblr.com/
[87] http://www.skyrock.com/
[88] https://www.researchgate.net
[89] http://www.plurk.com/
[90] http://weibo.com/
[91] https://www.techinasia.com/netease-weibo-260-million-users-numbers
[92] http://www.forbes.com/sites/kenrapoza/2011/05/17/chinas-weibos-vs-uss-twitter-and-the-winner-is

red social tiene censura debido al régimen del país [93] [94]. Otra OSN de este mismo año fue Foursquare [95], una OSN para smartphones que centrada en compartir las opiniones de los lugares que se visitan.

Como se ve, entre 2003 y 2009 explotó el gran fenómeno de las Redes Sociales Online, ya no solamente por la aparición de todas estas, sino por las inversiones realizadas en la compra de muchas de estas [413]. Una de estas fue la de MySpace por News Corporation en 2005 por 580 millones de dólares. Después, en 2007 Microsoft pagó 240 millones de dólares por un 1.6% de Facebook [96]. Un año más tarde AOL adquirió Bebo por 850 millones de dólares y la vendió a los dos años por 417 millones de dólares [97].

Posteriormente a esto, en el año 2010, ya el mundo estaba dominado solo por unas pocas OSNs [419]. La gran dominadora era Facebook, lo que hizo que el resto de Redes Sociales Online trataran de sobrevivir ante ella centrándose en los nichos, como es el caso de Ning y Audimated [98] (2010) [419]. En el año 2010 apareció Instagram [99], una red social centrada en compartir fotos y ayudando a retocarlas antes de su subida, aunque posteriormente permitieron la subida de videos. En el año 2012 fue comprada por Facebook, pero siguió operando independientemente. También en el año 2010 apareció Google Buzz, que fue una red social online que funcionaba como añadido de Gmail, pero que el 14 de octubre del siguiente año cerró, dando lugar posteriormente a la creación de Google+ [100] con el fin de centralizar así todos los servicios de la compañía. También en este año 2010 apareció Pinterest [101], una OSN para compartir imágenes que se considerasen interesantes mediante la organización en «tableros personales temáticos».

En 2011 salieron dos OSN que se centraban en la configuración fácil de seguridad, dos «antifacebook», que fueron Folksdirect y Unthink [431]. Ya en este año apareció Google+, que recibía toda la ayuda de los servicios de Google debido a que a los nuevos usuarios de cualquier servicio de Google le creaba automáticamente un perfil en Google+. Sin embargo, esta red social online no terminó de despegar a pesar de los intentos de Google de fusionarla con diferentes servicios y actualmente, tras varios años, Google está separando algunos de estos servicios de ella, haciendo de nuevo que no sea obligatorio crearse un perfil en Google+ para utilizar un servicio de Google. Hasta este año Microsoft no tenía OSN, cosa que cambió con el lanzamiento beta de So.cl [102], pronunciado «Social», muy similar a Google+ y con ideas basadas en Facebook, Twitter y Pinterest [103]. También en este año surgió Snapchat [104], que apareció como servicio de mensajería para los smartphones, pero que después añadió componentes sociales.

[93] http://www.jornada.unam.mx/2012/04/01/mundo/023n3mun
[94] http://www.abc.es/20120210/medios-redes/abci-twitter-china-anonimato-201202100806.html
[95] https://es.foursquare.com/
[96] http://www.nbcnews.com/id/21458486
[97] http://www.bbc.com/news/10341413
[98] http://www.audimated.com/
[99] https://www.instagram.com/
[100] https://plus.google.com/
[101] http://pinterest.com/
[102] http://www.so.cl/
[103] https://www.washingtonpost.com/business/technology/microsoft-silently-launches-socl-its-attempt-at-a-social-networking-site/2012/05/21/gIQADl80eU_story.html
[104] https://www.snapchat.com/

Desde el año 2010 también vino el declive de muchas redes sociales, como fue la venta de Bebo por AOL a un precio 10 millones inferior que por el que lo compró dos años antes, la caída de MySpace que desembocó en una venta de 35 millones de dólares seis años más tarde de ser comprada por 580 millones de dólares [413]. En estas fechas también hubo el cierre de algunas OSNs que fueron muy importantes en ciertos países. Una de ellas fue Orkut, que dejó de estar activa en junio de 2014 provocando que muchos brasileños se fueran a la red VK, pues según el portavoz de VK, Gueorgui Lóbushkin, en 48 horas el aumento de brasileños fue del 2.000% y seguía creciendo [105]. Por otro lado, Tuenti en el año 2016 cerró sus puertas como red social tras la pérdida continuada y masiva de usuarios en favor de Facebook, cambiando de negocio y convirtiéndose así en una operadora móvil virtual.

Sin embargo, también hubo grandes inversiones. Facebook compró Instagram en el año 2012 por 1 millardo de dólares, y el 20 de junio del año 2013 Yahoo! compró Tumblr por 1.100 millones de dólares.

Posteriormente siguieron surgiendo diferentes OSNs, como es el caso de Vine [106] en el año 2012, que permitía compartir videos de 6 segundos en *streaming*, y que fue comprada por Twitter en octubre del mismo año por 30 millones de dólares.

En el año 2014 Foursquare se dividió, creando así la filial Swarm [107], una OSN para smartphones que ayuda a hacer planes con los amigos utilizando la localización geográfica, pero manteniendo algunas ideas aplicadas en Foursquare, como los *chek-in* en los lugares visitados y añadiendo juegos y logros en ellos.

En el año 2015 apareció Periscope [108], aplicación para hacer *streaming* de video desde los smartphones y siendo adquirida al poco de su lanzamiento por Twitter por una cifra desconocida que está entre los 50 y 100 millones [109]. También en este año apareció Meerkat [110], que es muy similar a Periscope.

En la Ilustración **38** se muestra el estado actual de los países del mundo según la OSN más utilizada en ellos, que como bien muestra, pasó de estar divido el mundo en varias OSNs a estar claramente dominado por Facebook. Sin embargo, Alexa no incluye en este ranking de OSNs ciertas OSNs si consideradas aquí, como son Youtube, Instagram, entre otras, mientras que si incluye OSNs o sitios no considerados en este capítulo en base a la definición dada, como son HootSuite [111], el cual es un administrador de redes sociales, o Fiverr[112], que es un mercado online donde ofrecer servicios.

[105] http://www.interfax.by/news/world/1160572
[106] https://vine.co/
[107] https://www.swarmapp.com/
[108] https://www.periscope.tv/
[109] http://www.businessinsider.com/twitter-acquires-periscope-for-a-sizable-amount-2015-3
[110] http://meerkatapp.co/
[111] https://hootsuite.com/
[112] https://www.fiverr.com/

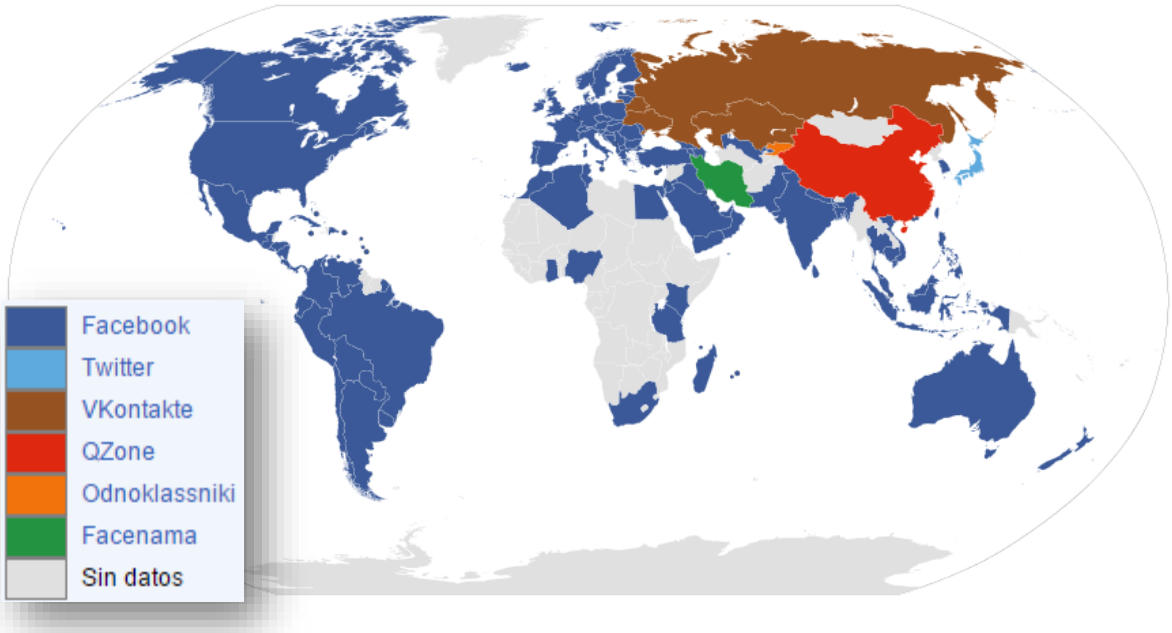


Ilustración 38 **OSNs más populares por país según Alexa a 8 de 11 de 2015** [113]

8.4 *REDES SOCIALES MÁS UTILIZADAS EN LA INVESTIGACIÓN*

En temas relacionados con la investigación se utilizan sobre todo dos redes sociales, como ha visto, previamente las más utilizadas eran Facebook, MySpace, LiveJournal, Friendster, CyWorld y Orkut, de más a menos. Actualmente la investigación se centra sobre todo en Twitter, y después en Facebook.

Estas redes sociales suelen ofrecer servicios a terceros desarrolladores a través de APIs para crear nuevas aplicaciones sociales, como es el caso de Facebook. Algunos de estos servicios ofrecidos a través de sus APIs son los servicios de autenticación, métodos para definir permisos de acceso, recolección de datos de la red social y nuevas funcionalidades para la propia red social integrando aplicaciones de terceros en ella.

Sin embargo, internamente su funcionamiento es muy diferente, Por eso mismo, en este apartado se explicarán las dos OSNs más utilizadas actualmente y sobre las que se recoge investigación en este capítulo.

8.4.1 FACEBOOK

Facebook fue lanzado en el año 2004 en la universidad de Harvard, restringido a sus estudiantes y posteriormente a otras universidades, pero siempre pidiendo cuenta de correo académica. Ya en septiembre de 2005 Facebook se expandió, dando cabida en él a gente de institutos, a las redes corporativas y poco después a todo el mundo. Actualmente es la red social más popular de todas [409], y sobre todo entre los universitarios [430].

En Facebook, el contenido relevante para el usuario es mostrado en lo que se llama el «muro». Este contiene toda la información de tus amigos o de las páginas que sigues. Ofrecía también la posibilidad de

[113] https://commons.wikimedia.org/wiki/File:Most_popular_social_networking_sites_by_country.svg

enviar mensajes privados, cambiado ahora a un chat en tiempo real y que posee aplicación móvil propia, Messenger. También posee desde hace poco los *hashtags*, al mismo estilo que Twitter, añadiendo la almohadilla (#) antes del *hashtag* a crear. Por otro lado, para mencionar a un amigo, solo hace falta empezar a escribir su nombre y seleccionarlo de la lista desplegable, lo que hace que se mencione al amigo y quede como hipervínculo hacia él.

Una de las características, y que más ha sido utilizada siempre, ha sido el que Facebook permita la creación de aplicaciones por terceros, ya sean juegos o aplicaciones para recopilar tus datos y darte un servicio, como es el caso del aeropuerto de Calgary en Canadá [425].

En Facebook, la relación entre los usuarios debe de ser recíproca, es decir, un usuario tiene que aceptar una invitación de otro usuario para establecer una relación. Esto hace que se envíe una invitación y, si el otro usuario acepta, se cree la arista del grafo indicando que son amigos. No obstante, hace tiempo que añadió la posibilidad de seguir a alguien sin necesidad de ser su amigo.

Algo que siempre ha sido muy popular y ha ayudado mucho a socializar ha sido el botón «Me Gusta», que permitía indicar que una publicación te gustaba, y, en el caso de que estuviera en una web externa, compartirla también a través de Facebook bajo una publicación tuya. Actualmente, este botón e ha transformado en 6 caras diferentes, mostradas en la Ilustración 39, las cuales tienen un estado de ánimo diferente y que son, respectivamente: «Me gusta», «Me encanta», «Me divierte», «Me asombra», «Me entristece», y «Me enfada». Esto incrementa la sociabilidad añadiendo una nueva característica que es el estado de ánimo sobre esa publicación.

Ilustración 39 Caras del «Me gusta» de Facebook

De cara a los desarrolladores y la investigación, Facebook ofrece el **Grafo Social** [415] y una API para navegar por él llamada *Open Graph* [114]. Sin embargo, a diferencia de Twitter, en esta API solo se puede consultar la información propia, la de tus amigos y la pública, que suele ser poca.

A fecha de junio de 2016, Facebook cuenta con 1,13 millardos de usuarios activos al día de media, 1,03 millardos usuarios activos desde un teléfono móvil de media, 1,71 millardos de usuarios activos en junio de 2016 y 1,57 desde el móvil. Del total, el 84,5% son de fuera de los Estados Unidos y de Canadá [115].

[114] https://developers.facebook.com/docs/graph-api
[115] http://newsroom.fb.com/company-info/

8.4.2 **TWITTER**

Twitter fue lanzado el 13 de julio de 2006, siendo la red social que más rápido en Internet, ofreciendo un servicio de microblog, teniendo a diez millones de usuarios activos participando en la creación en el año 2010 y propagación de contenido [420]. Todos estos mensajes son mostrados en el *timeline*, que es el equivalente al conocido muro en otras redes sociales, que es donde se muestra los mensajes de las personas que sigues. Twitter se basa en mensajes cortos de hasta 140 caracteres muy usada para fines de investigación por las características que tiene. Además, muchos usuarios de Twitter no protegen sus *tweets*, permitiendo así que estos aparezcan en el *timeline* público [432], lo que hace que se pueda acceder a ellos a través de la API de una forma sencilla.

Es considerada una red social directa en la que cada persona tiene un conjunto de seguidores, conocidos como *followers*, y en el que dicho sistema no tiene por qué ser reciproco [433], es decir, los usuarios son libres para seguir a cualquier otro usuario sin que este último siga al primero. Cada persona puede publicar un mensaje, llamado *tweet*, que posee un máximo de 140 caracteres, y que cada persona utiliza a su manera, pudiendo contar sus sentimientos, historias, chistes, juegos, noticias, gifs, etc. [418], [420].

Los *tweets* son mostrados en la página del usuario que lo publicó o en el resumen de tu página inicial, que contiene el resumen de todo lo que publicaron las personas que tus sigues. También tiene la posibilidad de enviar mensajes directos, conocidos en otras redes sociales como privados. Existe también la posibilidad de republicar un mensaje que hayas leído de otra persona, lo que se conoce como *retweet* (RT), y sirve para compartir tu opinión sobre ese mensaje o directamente hacer publicidad de él.

Twitter ha atraído la atención de las grandes empresas, pues se han dado cuenta de que tiene un potencial viral de marketing [420]. Debido a esto, muchas empresas cuentan con su propio *Community Manager*, quien es el encargado de hacer publicidad de los productos, responder a la gente, realizar el servicio técnico a través de la red social, contar anécdotas para hacer publicidad de la cuenta, gestionar los *followers* y seguir gente.

Esta red tiene un sistema para nombrar a una persona en un *tweet*, simplemente hay que poner el carácter «@» antes del nombre del usuario para así hacer una mención a este. Por otro lado, están los *hashtags* (#), palabras claves utilizadas libres de crear por cualquier y que sirven para agrupar los *tweets* públicos de un mismo tipo bajo ellos. Los *hashtags* más utilizados se incluyen en la lista llamada *trending topic* y que es mostrada en un lateral de la página web de Twitter.

Twitter cuenta, a fecha de 15 de septiembre de 2016, con 316 millones de usuarios activos al día, de los cuales el 77% son de fuera de los Estados Unidos, el 80% son usuarios de móvil, y se publican 500 millones de *tweets* [116].

[116] https://www.socialbakers.com/statistics/twitter/

8.5 *CLASIFICACIÓN DE LAS REDES SOCIALES ONLINE*

Existen varias clasificaciones para diferenciar las diferentes Redes Sociales Online existentes. Estos pueden ser según su especialidad, del sujeto de interés, de la localización geográfica, o en base al contenido compartido. Estas clasificaciones están muy extendidas por Internet y uno de los sitios donde se puede ver alguna es en [434].

8.5.1 **DESDE EL PUNTO DE VISTA DE SU ESPECIALDIAD**

Esta clasificación separa las Redes Sociales Online según el punto de vista de su especialidad, pudiendo ser **horizontal** o **verticales**.

Las **OSNs horizontales** son aquellas que no fueron creadas para un propósito o temática en concreto. Es decir, son OSNs que sirven para compartir y hablar de cualquier tipo de contenido libremente y sin restricciones, siendo así su principal función la relación entre personas. Ejemplos de este tipo de OSNs son Facebook, Twitter, Google+, Orkut, Tuenti y Bebo.

Las **OSNs verticales** son aquellas que están especializadas en un tema, por lo que están dirigidas a un público en concreto, lo que hace que la gente acuda a ellas debido a un interés en común. Estas pueden ser de muchos tipos:

- **Profesionales**: son las OSNs destinadas a establecer nexos entre diferentes profesionales, entre profesionales y empresas y entre empresas. Por ejemplo, de este tipo son LinkedIn para cualquier tipo de profesional, Xing, Viadeo [117], y Academia [118] y ResearchGate para investigadores.
- **Aficiones** / **Ocio** / **Problemas**: estas redes sociales se centran en compartir gustos relativos al ocio, como pueden ser OSNs dedicadas al deporte o un deporte concreto, a la música o los videojuegos. Ejemplos son Last.FM y MySpace dedicadas a la música, Wipley que es una OSN dedicada a los videojuegos, Dogster, una OSN dedicada a la gente que ama a los perros, Unitedcats [119] para la gente que ama a su gato, Funcook [120] para recetas de cocina, Moviehaku [121] de películas, Moterus [122] para los amantes de las motos, y PatientsLikeMe [123] para comunicar a personas que sufren una misma enfermedad.
- **Mixtas**: estas OSNs son una fusión entre las profesionales y las de aficiones, proporcionando un lugar donde hacer ambas actividades. Un ejemplo es Unience [124] para los amantes de la bolsa que

[117] www.viadeo.com
[118] https://www.academia.edu/
[119] www.unitedcats.com
[120] http://funcook.com/
[121] http://moviehaku.com/
[122] https://www.moterus.es/
[123] www.patientslikeme.com
[124] https://www.unience.com/

además pueden contactar con profesionales de ella, y ModelMayhem [125] para personas que trabajen en el mundo de la moda.

- **Identidad cultural**: el efecto de la globalización ha provocado que muchos grupos creasen su propia red social para mantener así su propia identidad. Este es el caso de AsianAve, una red social orientada a la gente asiático-americana.
- **Movimientos sociales**: estas OSNs se desarrollan alrededor a una o más preocupaciones sociales, como sucede con Care2 que sirve para las personas que están interesadas en la ecología y en el activismo social.
- **Viajes**: son aquellas OSNs que se centran en los viajes a diferentes lugares y que ayudan a preparar el viaje, comentarlos y recibir consejos, así como a conectar con viajeros que han estado en esos sitios o que quieran acompañarte a visitarlos. Algunos ejemplos son WAYN [126], TravellersPoint [127] y minube [128].
- **Citas**: estas OSNs están centradas en ayudar a la gente a tener citas o ligar por medio de amigos de otras OSNs que actúen como intermediarios, sistemas de búsqueda aleatoria o micropagos. Unos ejemplos de este tipo de OSNs son Badoo y BeautifulPeople.

8.5.2 DESDE EL PUNTO DE VISTA DEL SUJETO DE INTERÉS

Las OSNs pueden estar centradas en dos aspectos, en los humanos y sus relaciones, o en el contenido que se genera en la red.

Las **OSNs humanas** están centradas en los seres humanos, sus relaciones, en la interacción entre estos, y en sus gustos, intereses y actividades. OSNs de este tipo son Tuenti y Facebook.

Por otro lado, las **OSNs de contenido** son aquellas en el que el centro de interés está en el contenido que se publica, siendo así las relaciones importantes para poder acceder al contenido publicado, que es el que pondera la importancia en la relación. En este caso nos encontramos con OSNs como Flickr, Instagram, Twitter, Foursquare o Vimeo.

8.5.3 DESDE EL PUNTO DE VISTA DE LA LOCALIZACIÓN GEOGRÁFICA

Hay dos tipos de OSNs según su localización geográfica. Las primeras son las **OSNs sedentarias**. Estas OSNs son aquellas que se modifican en base a los contenidos que se publican en ellas, las relaciones existentes, los eventos que se crean, etc. Estas son las más típicas OSN existentes como son Facebook, Tuenti, Google+, Plurk, etc.

Por otro lado, están las **OSNs nómadas**, que son las que dependen de contenidos localizables mediante su localización geográfica. Esto puede ser bien debido a que sus usuarios se mueven, por los lugares que

[125] http://www.modelmayhem.com/
[126] www.wayn.com
[127] http://www.travellerspoint.com/
[128] http://www.minube.com/

visitan sus usuarios, por la ubicación de sus publicaciones, etc. Ejemplos de estas son Foursquare, que introduce la ubicación de los sitios visitados por la gente, y Swarm, que se centra en las visitas a los lugares, pero de una manera mucho más social que Foursquare mediante la inclusión de metas y juegos.

8.5.4 DESDE EL PUNTO DE VISTA DEL CONTENIDO COMPARTIDO

Las OSNs también se pueden clasificar según el tipo de contenido que se comparte, por lo cual hay muchos tipos diferentes y no se limitan a los expuestos a continuación:

- **Fotos**: estas OSNs permiten almacenar, ordenar, buscar y compartir fotografías, habiendo crecido en estos últimos años de manera exponencial gracias al uso de los smartphones. Algunos ejemplos son Flickr, Fotolog, Instagram, Pinterest y Snapchat.
- **Música**: las OSNs de este tipo permiten escuchar, clasificar, compartir y buscar música, como ocurre con Last.FM y MySpace.
- **Video**: las OSNs de video permiten subir, buscar y ver, como pasa en YouTube, Vimeo, dailymotion [129], Flickr, Vine, Periscope, Meerkat y Pinterest. Además, este tipo de OSNs se han popularizado enormemente.
- **Documentos**: estas OSNs permiten compartir, comentar y subir diferentes tipos de documentos, como ocurre en ResearchGate que permite compartir artículos de investigación.
- **Presentaciones**: estas Redes Sociales Online permiten compartir presentaciones de cualquier tipo y ofrecen facilidades sobre ellas, como ocurre con SlideShare [130].
- **Noticias**: estas OSNs se diferencian del resto por que su función es la de crear y compartir noticias, siendo un claro referente en España la OSN Menéame.
- **Lectura**: las OSNs de lectura de centran en permitir a los usuarios crear revisiones de los libros que han leído, como ocurre en anobii [131], LibraryThing [132] y EntreLectores [133], o bien, incluso crear sus propios libros y relatos y compartirlos, recibiendo así críticas y comentarios de ellos, como ocurre en Wattpad [134].

[129] http://www.dailymotion.com/
[130] http://www.slideshare.net/
[131] http://www.anobii.com/
[132] http://www.librarything.es/
[133] www.entrelectores.com
[134] www.wattpad.com/

8.6 *TEORÍA DE LOS SEIS GRADOS DE SEPARACIÓN*

Los seis grados de separación es una hipótesis que intenta probar que cualquier persona de la Tierra puede estar conectada a otra a través de una cadena de conocidos de no más cinco intermediarios.

El primero en proponer esta teoría fue el escritor Frigyes Karinthy [435], en el capítulo Chain-Links [436]. En este proponía que la población de la Tierra estaba muy cerca y que con simplemente seleccionar una persona de entro los 1.5 millardos de habitantes de la Tierra en 1929, se podría poner en contacto con cualquier persona a través de la primera, que es un conocido, sin utilizar a más de 5 personas. Todo esto gracias a las redes de contactos y su crecimiento exponencial.

Tras esto, en el año 1967, Stanley Milgram, psicólogo social, ideó una manera de probar esta teoría, por primera vez [435], y a cuyo experimento llamó *Small World Experiment* [435]. El experimento consistió en estudiar la probabilidad de que dos personas aleatorias de las ciudades Omaha, Wichita y Boston de los Estados Unidos de América se conocieran entre sí [437], y así contar el número de conexiones entre estas personas. Para esto utilizaban paquetes en el que la persona que lo recibía y los nexos intermedios debían de escribir su nombre y predirigir la carta a Harvard. Como resultado obtuvieron que las cadenas llevaban entre 5 y 7 intermediarios, y una media de 5,2 grados, mientras que ellos esperaban cientos. Sin embargo, este estudio tiene varias críticas por cómo se realizó el experimento

En base a esta teoría, surgieron investigaciones relacionadas. Una de ellas fue la realizada por la Universidad de Virginia [135], en la cual utilizaron la *Internet Movie Database* (IMDB) [136]. Esta prueba, llamada *The Oracle of Bacon* [137], ha utilizado al actor Kevin Bacon para calcular el número de personas en la cadena que están entere él y otro actor, siendo la conjetura de que no hay más de seis grados de diferencia entre Kevin Bacon y otro actor de películas [416]. Este número es conocido como el número de Bacon. Un caso similar es el del número de Erdős, realizado por la Universidad de Oakland [438], el cual se calcula en base a la distancia colaborativa a los trabajos entre un autor y el matemático Paul Erdős.

Posteriormente, en el año 2011, Facebook realizó un estudio denominado *Anatomy of Facebook* en el que, utilizando a todos los usuarios registrados en Facebook durante el estudio, sobre 721.000.000 personas, que se correspondía con más del 10% de la población mundial, para probar la teoría de los 6 grados [435], [439]. En este estudio, del que excluyeron personas famosas, y contabilizaron una media de amigos por persona de 100, la que consideraban baja. Como resultado obtuvieron que el 99.6% de las personas están conectadas por 5 grados de separación, lo que hace que sean 6 saltos, y que el 92% de las personas estas conectadas por 4 grados, lo que es lo mismo que 5 saltos. También obtuvieron que en el año 2008 tenían una media de 5,28 saltos y en el 2011, que fue cuando se realizó el estudio, esta había bajado a 4,74 saltos o 3,74 grados. En cambio, si limitaban el estudio a un simple país obtenían como resultado una media de separación entre la

[135] http://rt.cs.virginia.edu/misc/news-bacon.html
[136] http://www.imdb.com/
[137] http://oracleofbacon.org/

gente de 3 grados o 4 saltos. Lo que indica que Facebook es una red que está casi completamente conectada [440].

8.7 *TRABAJO RELACIONADO*

Las redes sociales se utilizan para muchas cosas como se ha visto, desde socializar, hasta marketing. Esto hace que sean un sitio de estudio muy popular para obtener datos de sus usuarios, pues, en muchos casos, estos son públicos. Por este motivo, muchas veces se utilizan las redes sociales para realizar diferentes estudios, e incluso son el propio equipo de las redes sociales quienes los hacen, como es el caso de Facebook, donde se realizó un estudio utilizando los gustos de los usuarios de su red social online para determinar como son la gente que le gustan los gatos y como son la gente que le gusta los perros, sacando así sus características, gustos de películas, libros y series, y sus sentimientos [441]. Sin embargo, para esto, a veces hace falta aplicar Big Data debido a la gran cantidad de datos que estas poseen, como ocurre con casos posteriormente explicados en el subcapítulo de Tendencias en Redes Sociales Online.

Sin embargo, a medida que la sociedad y la tecnología avanzan, es más frecuente que la comunicación no sea solo necesaria entre personas, sino también de estos con los objetos [442]. Debido a esto, han surgido muchas investigaciones que tratan de conseguir una comunicación H2M efectiva y fácil. Para ello, algunos investigadores han optado por utilizar las redes sociales como base de comunicación con los objetos. Las Redes Sociales ofrecen una serie de beneficios si se utilizan en Internet de las Cosas y esto se presenta en la primera subsección del trabajo relacionado.

Por otro lado, no hay que olvidarse del trabajo clásico de investigación sobre redes sociales. Este siempre ha sido el estudio de cómo estas afectan al comportamiento humano o como los humanos se comportan en ellas, todo desde un punto de vista más psicológico y antropológico. Algo muy importante, pues ayuda a explicar cómo interactuan las personas en las redes sociales y por qué, así como si lo que ven y hacen en ellas tienen algún efecto positivo o negativo en su vida. Por esto mismo, en la segunda subsección se presentan algunas investigaciones interesantes en este área.

8.7.1 **REDES SOCIALES EN EL MARCO DE INTERNET DE LAS COSAS**

Uno de estos es el trabajo presentado en [409], donde se exponen los beneficios de usar redes sociales en lo que se denomina la Web de las Cosas o *Web of Things* (WoT), que es como se llama al uso de servicios web para diversos objetivos dentro de Internet de las Cosas. Además, analizan el uso de contenedores de aplicaciones de terceros que algunas redes sociales ofrecen como servicio. Para ello, crean aplicaciones que integran los objetos inteligentes en una de las redes sociales más usadas y que ofrece el servicio de contenedor de aplicaciones de terceros, Facebook. Sin embargo, esta investigación no propone un sistema final para interconectar objetos inteligentes usando redes sociales como hacen otros trabajos.

Por ejemplo, uno de estos trabajos que permite a los objetos la capacidad de publicar en Twitter es la propuesta presentada en [443]. Sus autores hablan sobre las posibilidades que surgen de publicar eventos procedentes de objetos inteligentes, sensores y actuadores, en las redes sociales.

Por otro lado, hay investigaciones que permiten el acceso a los objetos en Twitter [410]. Esta propuesta la denominaron *Social Access Controller* (SAC). Los autores proponen el uso de redes sociales para compartir con los usuarios la conexión a los objetos inteligentes mediante un sistema que expone una API basada en servicios REST, la cual permite acceder y controlar objetos inteligentes registrados en el sistema. El uso que ellos hacen de las redes sociales no es con el fin de comunicar los objetos entre sí, sino que la usan para compartir como conectarse al objeto inteligente utilizando una arquitectura REST y haciendo uso de un servidor central o proxy.

En cambio, en [444], los autores usan Twitter para extraer información sobre eventos de tráfico, del transporte público, del suministro de agua, del tiempo, de las aguas residuales y de la seguridad pública mediante el procesado de los *tweets* mediante el uso de Procesamiento de Lenguaje Natural.

Mientras que en [417] presentan como los ciudadanos pueden servir como sensores humanos para proveer información suplementaria, alternativa y complementaria sobre las ciudades, utilizándolos así como otro sensor más dentro de *Smart Cities*. Siguiendo con ejemplos de personas como datos, tenemos la investigación de [418], donde han utilizado a personas como sensores, pero para lograr la detección de terremotos, exactamente su epicentro, y la trayectoria de los tifones. Otro ejemplo de uso de Twitter en el marco de Internet de las Cosas es el que se muestra en [445], donde los autores usan las opiniones de Twitter para ayudar a tomar decisiones inteligentes sobre el destino para hacer turismo. Para lograr esto, los autores crearon una plataforma que recogía los *tweets* durante el mundial de fútbol del año 2014 de forma que podían reconocer la nacionalidad y lenguaje de los *tweets* y los puntos de aglomeración y concentración de los visitantes, pudiendo así ser aplicables los *tweets* para gestionar los destinos turísticos.

Otro tipo de propuestas son aquellas en las que se utilizan mensajes instantáneos para comunicar objetos con personas en vez de servicios web o redes sociales. Un ejemplo es la de Aurrell [446], quien utiliza la Open Telecom Platform escrita en Erlang (Erlang/OTP) [138]. Otro ejemplo es [442], [447], donde los autores presentan un protocolo de comunicación basado en mensajes para H2H, H2M y M2M utilizando el protocolo XMPP, y una combinación de una arquitectura peer-to-peer (P2P) en Internet con una arquitectura cliente-servidor centralizado en casa. Este enfoque tiene ciertas desventajas como la necesidad de aplicaciones exclusivas para realizar la comunicación mientras que el uso de redes sociales permite usar las propias aplicaciones de las redes sociales o los navegadores web con el fin de acceder a las publicaciones de los sensores o para controlar los actuadores.

8.7.2 INFLUENCIA DE LAS REDES SOCIALES ONLINE EN EL SER HUMANO

En primer lugar, los autores de [448], investigaron si existe una relación en el rendimiento académico entre los estudiantes que usan Facebook y los que no. Siendo los resultados que los propios estudiantes no creen que afecte en el rendimiento el no usar Facebook pues priorizan los estudios antes que Facebook, pero sí que reportaron un impacto negativo los que sí lo usaban pues procrastinaban más.

[138] http://erlang.org/doc/

En otros, como es en [414], analizaron los cinco factores alternativos de la personalidad, que son la extraversión, la apertura al cambio, la conciencia o responsabilidad, la amabilidad y la inestabilidad emocional, en base al estudio de la red social Facebook. Sin embargo, los resultados ofrecieron pocos hallazgos significativos en relación con los factores de la personalidad. Otro de los resultados obtenidos fue que los usuarios no usan Facebook como una alternativa a las actividades sociales.

Otro estudio sobre Facebook utiliza el marco de la teoría del rango social de la depresión y conceptualiza la envidia en Facebook como un posible vínculo entre el uso de vigilancia en Facebook y la depresión entre los estudiantes universitarios [449]. Los resultados fueron que el efecto de vigilancia en Facebook es mediado por la envidia en Facebook. Sin embargo, cuando esta envidia es controlada la depresión disminuye. Por ejemplo, según este estudio [430], la presencia social es el factor más importante que determina a los estudiantes usar Facebook. Esto se debe a que algunas características de Facebook estimulan a los estudiantes a colaborar juntos. Esto es el caso del tablón de noticias, que permite a los usuarios sentir la presencia de sus amigos en la red social, o el chat que hace que los usuarios puedan descubrir quien está presenta y permite hablarle en tiempo real [430].

Las redes sociales han hecho que realizar preguntas a gran escala sea fácil y eficiente [412]. Por ejemplo, en [412], se estudia esta situación mediante un estudio a 624 personas que han preguntado en redes sociales y saber con qué frecuencia hacen preguntas, el tipo de preguntas que hacen, y ver la motivación de los encuestados para usar redes sociales en vez de usar motores de búsqueda web. En otro caso, han utilizado Facebook para ver cómo influye la personalidad de usar o no usar Facebook en gente de entre 18 y 44 años para saber cómo de tímidos, narcisistas y solitarios son cada grupo [450]. Por ejemplo, los usuarios de Facebook tienden a ser más extrovertidos y narcisistas, pero menos concienzudos y solitarios socialmente que los que no lo usan.

Otros estudios se basan en el estudio de la topología de la red para saber cómo funciona y ver en que afecta a cada característica de esta. En [433] realizaron este tipo de estudio para describir la topología de Twitter y ver cómo funciona internamente, viendo así los números de *followers* según el tipo de usuario, la relación entre el número de seguidores de una cuenta y el número de *tweets* por usuario, el grado de separación, que se encuentra en 4, hicieron diferentes estudios de rankings de usuarios, realizaron una comparación de tendencias y *retweets*, y estudiaron el impacto de los *retweets*. Hay otros estudios, como [411], que se centran en el crecimiento de la red por medio de la descripción de los tipos de grados y aristas que su estructura posee y la cercanía entre nodos y vértices.

9. LA NUBE (CLOUD COMPUTING)

«Tú no generas tu propia electricidad. ¿Por qué generar tu propia computación?»
Jeff Bezos, CEO de Amazon

«Si las computadores del tipo que yo he defendido se convierten en las computadores del futuro, entonces, algún día, la computación puede estar organizada como una utilidad pública como lo es el sistema de teléfono. La "Computer Utility" podría comenzar las bases de una nueva e importante industria»
John McCarthy, 1961

«Nunca asumas que todo el mundo tiene la misma comprensión, visión, expectación o incluso definición de "Cloud Computing"»
David Mitchell Smith, VP & Gartner Fellow

En los últimos años hay varias palabras que están de moda como son La Nube y *Cloud Computing*. Sin embargo, nadie sabe explicarlo. En cambio, cuando se encuentra a gente con la capacidad de saber explicarlo, te encuentras con que cada uno da una definición diferente, cada uno tiene su punto de vista y su opinión. Esto ocurre porque la nube, como ocurre cuando miramos al cielo, tiene muchas formas diferentes, cada nube tiene su forma, al igual que pasa cuando miras al cielo y a ti te parece ver una princesa y a otro un sapo. Para gustos los colores, y para formas las nubes.

La nube es el almacén permanente de la información, pues su escalabilidad y elasticidad permiten adaptarse a la gran cantidad de información generada por los objetos y por los humanos, mientras que su alta disponibilidad da confianza y fiabilidad permiten que los clientes crean en la nube a la vez que sus bajos costes permiten que sea contratada. *Cloud Computing* es un nuevo paradigma que se basa en mover todo lo que puede estar en un ordenador personal o en servidores individuales a ordenadores en la nube. Siendo la nube, a grandes rasgos, Internet.

Gracias a esto, una nueva era de la informática se abre, una era en la que los proveedores pueden ofrecer hardware y software como servicios y en la que los clientes pueden ahorrar costes, pagar por lo que usan como si fuese la factura del agua, y transferir los riesgos. Una era en la que tanto proveedor como cliente ganan. Esto ha hecho que se convierta en una de las piezas clave de la gestión del conocimiento [139] en la época actual.

Las nubes han llegado, el cielo se va encapotando, y no parece que vaya a despejar. Por esto, en este capítulo se hablará de ellas, de su historia, de sus ventajas e inconvenientes y de los diferentes tipos existentes.

[139] Glosario: **Gestión del conocimiento**

*****

9.1 *¿QUÉ ES CLOUD COMPUTING?*

¿Qué es la nube? La nube no es más que un centro de procesamiento de datos, a veces con memoria y almacenamiento compartido por varios usuarios, con un software específico y accesibles a través de Internet [451]–[453]. Estas nubes permiten realizar tu propia computación o subir tus datos o tus aplicaciones, así como ofrecer servicios centralizados a los clientes. Luego, la nube es el almacén permanente de la información.

A partir del uso de la nube surgió lo que se conoce como *Cloud Computing*. *Cloud Computing* ha sido un paradigma muy exitoso de computación orientada a servicios que ha revolucionado la infraestructura de computación tradicional mediante la abstracción, la flexibilidad y consumiendo solo lo que se utiliza [454], [455]. *Cloud Computing* ha sido un cambio en la mentalidad tradicional en informática al utilizar la nube, pues ha permitido realizar toda la computación en ella, sin necesidad de poseer servidores o mainframes. Esto ha abierto un nuevo modelo económico en la informática, pues ha permitido a las empresas ofrecer hardware, plataformas y software como servicios de suscripción y que se encuentran alojados en la nube [456]. Ahora y bajo este nuevo paradigma, las empresas que ofrecen *Cloud Computing* ofrecen diferentes servicios de hardware, como son computación, redes y almacenamiento [453], [457], y software, es decir, cualquier tipo de aplicación, todos estos empaquetados y bajo demanda, y normalmente a través de Internet [458]–[460].

De esta manera, se relega toda la computación a servidores que se encuentran en la nube y que ofrecen una serie de beneficios, como es que sean dinámica y automáticamente escalables, bastante elásticos, estén virtualizados y sean fáciles de utilizar [458]. Los servicios que se encuentran en la nube permiten ser gestionados, ya sea para aprovisionarlos, modificarlos o liberarlos con un mínimo esfuerzo gracias a las herramientas que proveen los proveedores de este tipo de servicios [461]. Además, los servicios ofrecidos deben de ser robustos, fiables y estar disponibles a cualquier hora y desde cualquier lugar [453], [457]. Por estos motivos los tres componentes básicos de *Cloud Computing* son la virtualización, la multitenencia y los servicios web [458].

Los servicios ofrecidos en *Cloud Computing* son muy diversos y dependen de la Capas de La Nube que utilice este servicio. Pueden ser máquinas virtuales para instalar dentro lo que se quiera, máquinas virtuales montadas con un software determinado para así poder utilizarlas para desarrollar, testear o desplegar una aplicación o servicio determinado, o pueden ser aplicaciones, como son videojuegos, almacenamiento de datos y fotografías, gestores de correo, etc.

Gracias a la nube no importa en qué ordenador estemos, si es nuestro o es de otro, o si estamos desde el portátil, el ordenador del trabajo, el ordenador de sobremesa de casa o el smartphone, como bien apuntó Sam Schillace de Google en [114], pues, en sus propias palabras, ahora, todos los servicios que se encuentren en la nube nos permiten acceder a ellos simplemente con un navegador web y sin necesidad de instalar nada, estemos donde estemos.

Cloud Computing ha sido un tema que ha crecido muy rápidamente hasta ser uno de los más utilizados, tanto en la empresa, que está invirtiendo una gran cantidad de recursos, como en la academia, habiendo una

gran investigación y cursos en este tema [462]. Por ejemplo, *Cloud Computing* es utilizado por muchas aplicaciones científicas para realizar experimentos extensos, lo que les permite ir ampliando la falta de computación o espacio dinámicamente, reducir costes al no tener servidores locales, e incrementar el volumen de datos consumidos y producidos en cada experimento, como fue demostrado en el evento *Cloud Computing and Scientific Applications* (CCSA) del año 2012 [457].

Además, los servidores de la nube, al igual que ocurría con Google File System [463], MapReduce [464] y Big Data, suelen utilizar *Commodity Hardware* y ofrecen accesibilidad desde casi cualquier dispositivo siempre y cuando este disponga de conexión a Internet, aunque para ello requieren redes de alto ancho de banda. Por esto, *Cloud Computing* elimina de las empresas la necesidad de mantener hardware caro para computación, espacio dedicado y su correspondiente software [465]. Esto es lo que supuso un cambio en la empresa tradicional, pues ahora no se necesita de una inversión inicial ni de un centro de computación propio, ahora se puede contratar y pagar únicamente por lo que se necesite.

Como se ve, responder la pregunta del título es difícil, pues para algunos la nube se parece a las aplicaciones basadas en la web, es decir, una nueva forma de cliente ligero. Para otros, la nube se ve como una utilidad de computación en la que se cobran precios según el tiempo de procesamiento. Ambos son válidos, pues hay muchas formas de nubes [459], todo depende del proveedor y de la Capas de La Nube en la que nos encontremos. Incluso, hay gente que cataloga la nube actual como la nube 1.0 y afirma que nos estamos moviendo ya a la nube 2.0, que es aquella que integra las mejoras y funcionalidades de la web 2.0 [462]. Esta tendencia es conocida como Social Cloud [425], que consiste en que los usuarios puedan descubrir y comerciar datos y servicios de computación a través de una Redes Sociales Online. No obstante, una cosa está clara, y es que todo esto ha hecho que *Cloud Computing* sea uno de los mayores avances en la historia de la computación de los últimos años [458]. La nube está aquí, y es para quedarse.

9.2 *TERMINOLOGÍA*

Esta sección contiene diferentes términos y palabras utilizadas en el ámbito de *Cloud Computing* que son necesarias explicar para tener un mejor conocimiento y entendimiento de este capítulo.

Virtualización: creación de una versión virtual de algo, como una plataforma de hardware, una red o un dispositivo de almacenamiento. Esta tecnología permite emular plataformas, sistemas operativos, incluidos los antiguos, independientemente del sistema operativo base, si es que se posee, para así poder ejecutar ciertas aplicaciones que se necesiten dentro de ellos [458]. Además, permite un mejor uso del hardware base utilizado para virtualizar debido a que la virtualización permite el reparto de recursos del ordenador de una mejor manera, así como otorga una mayor seguridad al tener todas las aplicaciones corriendo dentro de un entorno aislado [453], [465], lo que incrementa la eficiencia y la escalabilidad [465]. Esto hace que la virtualización sea la base de la nube al permitir ofrecer un ilusión de escalabilidad y de recursos infinitos [466].

Multitenencia o **tenencia múltiple**: esta es una de las características más relevantes de *Cloud Computing*. La multitenencia se da cuando una aplicación alojada en un servidor posee una sola instancia, pero permite el acceso simultáneo a varios clientes. Además, la multitenencia permite un mejor uso de los recursos del sistema

en temas de memoria y procesamiento [458]. La multitenencia permite a la aplicación particionar virtualmente sus datos y su configuración para que cada cliente tenga una instancia virtual adaptada a sus requerimientos. Tiene la desventaja de que este tipo de arquitectura es mucho más compleja y requiere ciertas medidas de seguridad para que los clientes no puedan acceder a la estancia de otro cliente.

Escalable: es la característica de un sistema que le permite evolucionar dinámicamente para satisfacer la demanda requerida en un momento dado. La escalabilidad puede ser de dos tipos:

- Vertical: cuando se cambia o mejora para tener una máquina más potente. Es interesante para cierto tipo de aplicaciones.
    - Más frecuencia de CPU, más memoria, añadir una GPGPU [140] más potente, …
- Horizontal: cuando se incluyen más máquinas. Esta es la básica que suelen ofrecer los servicios de Internet.
    - Se añaden más máquinas al clúster de ordenadores, por ejemplo, si estaba formado por 10 ordenadores, que ahora sean 15.

Elástico: es la característica de un sistema que le permite escalar según la demanda, creciendo y decreciendo de forma automática [461]. Al cliente le da la impresión de que esta capacidad es ilimitada y accesible en cualquier momento, siempre y cuando pague los costes [461]. Para que un sistema sea elástico debe de ser escalable, pero un sistema escalable no tiene por qué ser escalable si este no puede crecer y decrecer automáticamente.

Hipervisor [467]: es el software que permite aplicar diferentes técnicas de virtualización para utilizar diferentes sistemas operativos diferentes al mismo tiempo en una misma máquina mediante la virtualización de hardware de forma aislada mientras que permite acceder al hardware base. Lo que hace un hipervisor es traducir las llamadas al sistema capturadas, así como los accesos a memoria. Pueden ser de dos tipos (Ilustración 40):

- Tipo 1 Nativo – *Bare Metal*: el hipervisor se ejecuta directamente sobre el hardware, es decir sobre la máquina desnuda y sin sistema operativo, y puede albergar diferentes máquinas virtuales. Unos hipervisores de este tipo son VMWare ESXi, XEN, Citrix XenServer y Microsoft Hyper-V Server.
- Tipo 2 Hosted: el hipervisor se ejecuta sobre un sistema operativo anfitrión ya existente. Hipervisores pertenecientes a este grupo son Virtual Box, VMWare Workstation y Virtual PC.

[140] Glosario: **GPGPU**

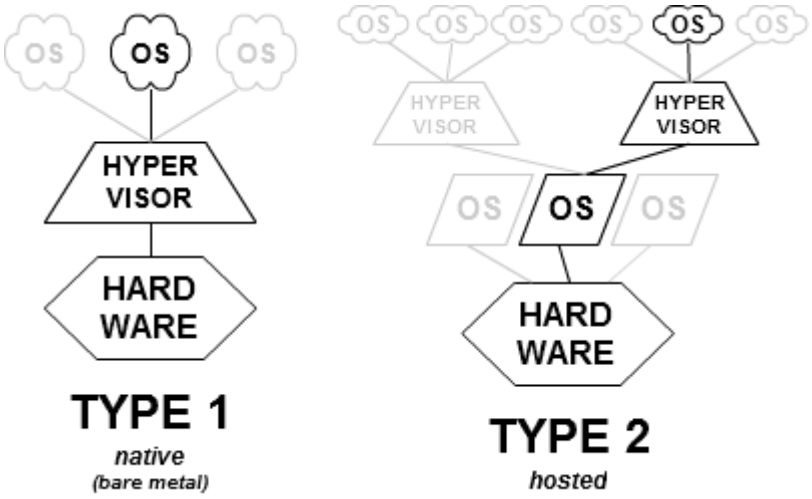


Ilustración 40 Hipervisores de tipo 1 y tipo 2 [141]

Cloud Middleware: software que permite orquestar todos los recursos necesarios para crear y gestionar una nube por medio de APIs. Algunos Cloud Middleware son, OpenStack que es utilizado por el CERN, Platform ISF, Eucalyptus y OpenNebula, utilizado por la NASA. Algunos ejemplos de lo que permite este tipo de software es controlar y desplegar de forma sencillas máquinas virtuales, incluso bajo demanda, tener contabilidad de lo utilizado, acceder a la API programáticamente o por medio de una interfaz gráfica, monitorización, crear balanceadores de carga, etc.

Commodity hardware [142]: es un dispositivo o un componente relativamente barato, ampliamente disponible y más o menos intercambiable con otro hardware de su mismo tipo. Para ser intercambiable este tipo de hardware suele ser y funcionar mediante *plug and play*. Este tipo de hardware es ampliamente utilizado en las nubes y en *Big Data*, como sucede y explicaron en el artículo de GFS [463].

9.3 *HISTORIA: MODELOS DE COMPUTACIÓN*

Año 1961, John McCarthy en un discurso para celebrar el centenario del MIT dijo: «*Algún día la computación puede estar organizada como una utilidad pública como lo es el sistema de teléfono. La* ***Computer Utility*** [143] *podría comenzar las bases de una nueva e importante industria*».

Al principio, en los años 70-80, los modelos de computación estaban en los *mainframes* [459]. Estos eran supercomputadores que compraban las empresas para su uso privado. Estos ordenadores podían virtualizar el tiempo de modo que pudieran ejecutar varias tareas de forma concurrente mientras compartían los recursos. Este tipo de computadora central grande, potente y muy costosa era la encargada de centralizar todos los servicios y se accedía a ella utilizando una terminal tonta. Un supercomputador para una o varias empresas, esta era la idea de aquella época como bien había expresado años atrás Thomas Watson, actual director

[141] https://commons.wikimedia.org/wiki/File:Hyperviseur.png
[142] http://whatis.techtarget.com/definition/commodity-hardware
[143] Glosario: **Utility Computing**

ejecutivo (CEO) de IBM: «*I think there's a world market for maybe five computers*». Tras esto, en los 70s, llegaron los primeros ordenadores personales.

Tras esto, en los 90 y principios del 2000, ya vino la época del WWW, inventado en el Consejo Europeo para la Investigación Nuclear (CERN) en el año 1989. Esto permitió la distribución en capas ofreciendo así una vista en el cliente y teniendo un controlador y un modelo en el servidor. Esto también permitió la virtualización de recursos. En esto años surgió también el boom de las punto-com, donde las empresas invirtieron millones en tener su propia página web y en diferentes servicios en Internet. Sin embargo, estos centros de datos eran similares a los de la era anterior, pero con un tamaño más reducido [459].

Una de las empresas surgidas en esta época fue Google, quien, con su innovación, presentó su arquitectura al mundo, GFS y MapReduce. Esto supuso una revolución significante [459]. La arquitectura de Google se ha basado en el uso de Commodity Hardware, que no es más que hardware barato, genérico, fácil de reemplazar y en grandes cantidades [463], [468].

Esto desembocó en la época actual, ya finalizando la década de los años 2000 apareció *Cloud Computing*. Aquí, gracias a la virtualización aplicada en arquitecturas similares a la de Google se consiguió disminuir el tiempo ocioso de un procesador [459]. Así, la nube representa un centro de computación de datos en una o más localizaciones que trabaja como si fuera un solo ordenador [459].

El primero de todos estos fue Amazon, en el año 2004, quien para administrar su tienda de libros fue creando diferentes servicios privados como el levantamiento de instancias para desarrollo y/o producción. Tras ver su potencial los publicaron al mundo a través de Amazon Web Services, siendo el primer servicio de estos *Simple Queue Service* (SQS) y posteriormente *Elastic Computer Cloud* (EC2) y *Simple Storage Service* (S3). Tras Amazon aparecieron otros proveedores pertenecientes a grandes empresas como Microsoft Azure y Google Cloud Platform.

9.4 *VENTAJAS E INCONVENIENTES*

El uso de *Cloud Computing* tiene ciertas ventajas e inconvenientes en general, pues después cada Capas de La Nube tiene otras ventajas e inconvenientes complementarios a los aquí presentados.

9.4.1 **VENTAJAS**

Las ventajas de los servicios en la nube para los usuarios son varias:

Escalabilidad [460], [465], [469]: en los entornos de *Cloud Computing* se puede decir que la escalabilidad es prácticamente infinita, tanto en términos de memoria RAM, de almacenamiento, de computación, procesadores y de ancho de banda, siempre que se seleccione y se esté dispuesto a pagarla.. Esto hace que desaparezcan los tiempos de espera cuando se necesita ampliar la capacidad de la máquina contratada en comparación con las máquinas tradicionales, al igual que tampoco se desaprovecha la capacidad que no se esté utilizando debido a que entonces se liberan los recursos sobrantes, lo que implica un ahorro de dinero.

Elasticidad [452], [454], [455], [461], [462]: las nubes permiten una gran elasticidad automática y rápida bajo demanda de manera que el cliente tiene la impresión de que la escalabilidad es infinita.

Baja inversión inicial [452], [454], [458], [460], [462]: no requiere comprar servidores ni invertir en otro hardware, contratar gente especializada que se encargue de ellos, ni mantener los servidores, pues los proveedor de servicios de *Cloud Computing* directamente te dan el hardware que necesites y te cobran por su uso, no por su compra, ni por su mantenimiento, actualización, renovación o reparaciones. Es decir, te lo alquilan a la vez que te dan tecnología genérica actual.

Pay-per-use [454], [469], ***pay-as-you-go*** [452], [455], [456], [462] o **tarificación** [453], [460], [469]: se paga únicamente por lo que se consume, siendo así la tarificación similar a la de los suministros públicos como son la luz y el gas, pero adaptada a este modelo donde se «compra» computación, espacio de almacenamiento, ancho de banda, etc. En caso de no utilizarlo, no se paga nada. Esto permite una gran flexibilidad en el caso de que se necesite una capacidad de cómputo de 1.000 ordenadores en una hora o simplemente 1 ordenador durante 1.000 horas, donde ambas pueden costar lo mismo. Los pagos son totalmente adaptables al uso que el usuario le quiera dar o que se requiera en el momento, si es que se necesita escalar. Muchos proveedores incluyen límites de lo que se está dispuesto a pagar en caso de que se permita la escalabilidad automática. Esta forma de tarificación permite a muchas empresas reducir el capital de la sección de informática y los gastos operativos, de forma que utilizan la nube para desplegar sus aplicaciones o aplicaciones en la nube [455].

Localización [453], [460], [461]: los servicios de *Cloud Computing* son independientes de la localización, lo que hace que sean accesibles desde cualquier lugar y dispositivo siempre y cuando se disponga de Internet en el dispositivo y la configuración de seguridad del protocolo de conexión y del propio servicio lo permitan [453]. Algunos ejemplos de dispositivos de acceso son los smartphones, ordenadores, clientes especializados, navegadores web, etc.

Acceso de red: permiten acceder prácticamente desde cualquier dispositivo que disponga de Internet a sus servicios mediante el uso de mecanismos estándares.

Disponibilidad [455], [469]: la disponibilidad de las nubes suele ser muy alta. Por ejemplo, Google App Engine y Amazon son muy agresivos, pues ofrecen una disponibilidad del 99,95% al año, lo que supone como mucho un tiempo inaccesible de 4,7 horas por año.

Seguridad física [460]: la seguridad física de los centros de datos corre de parte del proveedor, siendo él el encargado de protegerlos.

Fiabilidad [469] **y seguridad de la infraestructura** [460]: la infraestructura de los proveedores no tiene puntos de fallos debido a la redundancia de esta, como son varias redes de Internet contratadas, duplicación de los servicios eléctricos, duplicación del almacenamiento, servidores, etc. Además, los proveedores ofrecen duplicación de los datos a sus clientes en diferentes servidores, a veces incluido en el coste inicial y otras veces elegible, para evitar fallos, perdidas de datos y denegación del uso del servicio en caso de problemas físicos. También ofrecen otros servicios de seguridad como extras que incluyen la protección de los datos y la realización y recuperación de copias de seguridad.

Colaboración [460]: debido a que se accede por Internet y tan solo se necesita de un navegador web, el uso de la nube permite la colaboración entre equipos de desarrollo o desarrolladores individuales situados en diferentes lugares del mundo mientras colaboraran juntos en el desarrollo de la misma aplicación o en el mantenimiento del sistema.

Abstracción: los servicios en la nube ofrecen diferentes abstracciones, dependiendo del nivel de capa elegido (IaaS, PaaS y SaaS), el cual depende del objetivo del cliente.

Medida del servicio [453], [461]: los sistemas en la nube automáticamente miden y optimizan los diferentes recursos disponibles informando así a proveedor y al cliente de forma transparente del uso de cada servicio y su coste asociado.

Autoservicio bajo demanda [458], [461], [462]: estos sistemas permiten configurar a los usuarios el autoescalado de las capacidades según la demanda que vayan necesitando sin necesidad de ninguna interacción humana. Además, algunos ofrecen la opción de poner límite económico al autoescalado automático o no activarlo.

Transferencia de riesgos [451], [452], [454]: la compra, mantenimiento, reparación y actualización del hardware pasa al proveedor y es transparente al cliente. Además, si hay una caída prolongada ellos se hacen cargo, dependiendo de los términos de nivel del servicio (SLA), que es donde se fija la calidad del servicio.

Virtualización de los recursos [453], [465]: todos los recursos se encuentran virtualizados, lo que ofrece aislamiento de recursos, portabilidad entre máquinas, copias de seguridad y mayor facilidad de uso que tradicionalmente.

Creación de nuevos tipos de aplicaciones [458]: han permitido la aparición de nuevos tipos de aplicación como son por ejemplo computación de alto rendimiento gracias a la escalabilidad y elasticidad de la nube, aplicaciones sensibles al contexto que utilicen sensores o información humana de un determinado sitio, como puede ser la humedad dentro de un barco de carga, el uso de aplicaciones complicadas como Hadoop mediante la abstracción como sucede en HDInsight y Amazon Web Services (AWS) EMR, almacenaje y procesado e cantidades enormes de datos, aplicaciones que procesen todo en la nube y utilicen la aplicación del usuario solo como capa de presentación, etc.

9.4.2 **DESVENTAJAS**

Las desventajas de los servicios en la nube para los usuarios son las siguientes:

Migración:

- En el mejor caso, habrá que hacer «solo» pequeños cambios en la aplicación. En el peor caso, se necesitará la reescritura de la aplicación y hacer cambios en el *Back-end*.
- Automatizar despliegue.
- Automatizar escalado.

Seguridad, privacidad y copyright [462], [469]: el problema aquí reside en que los datos subidos a la nube pública no están en las manos de la empresa, sino en los servidores de otra empresa. Esto implica que

están en otras manos y que, según los términos firmados previamente, se puede perder cierta seguridad al no ser tu quien los controla, privacidad, pues podrían utilizarlos para cualquier cosa y, en caso de subir datos importantes y no existir unos buenos términos, tal vez hasta haya problemas de copyright.

Confidencialidad de los datos [452]: los datos se almacenan en servidores de terceros, siempre que sean nubes públicas o híbridas. Esto hace que el dueño de los datos pueda perder cierta confidencialidad sobre ellos o tener problemas con algunas leyes de protección de datos de ciertos países sobre sus residentes.

Durabilidad de los datos: se dieron ciertos casos en el que algunas nubes, por problemas de Copyright, tuvieron que borrar todos o algunos datos de los todos o algunos usuarios, haciendo que estos perdieran esos datos para siempre, inclusive de aquellos usuarios que no violaban ningún término o licencia, como fue en el caso de Megaupload [144] [145]. Otras veces se han perdido datos por fallos del proveedor, como sucedió con Dropbox [146], aunque finalmente fueron recuperados.

Vendor lock-in [452], [459] o **interoperabilidad** [469]: obligación de si se está con un proveedor, tener que utilizar todos sus servicios, aunque un tercero disponga de ese mismo servicio, debido a contratos con el proveedor o la dificultad de integrar servicios de terceros con los servicios del proveedor.

Pérdida de control: la nube está en manos del proveedor que se contrata y de los SLA que el proveedor tenga. Esto hace que se pierda flexibilidad y control en la nube, pues se depende de las actualizaciones del proveedor y de las mejoras que el incluya.

Disponibilidad del servicio [452], [469]: hay empresas que necesitan mucha disponibilidad y deben de ver si el servicio contratado puede soportar perfectamente los requisitos que el cliente necesita. Por eso, en caso de que se caiga la nube, si se está utilizando una pública, el cliente no tiene control sobre ello, y lo único que puede hacer es esperar a que el proveedor arregle el fallo. Sin embargo, si esto excede los términos de SLA firmados, el proveedor siempre deberá asumir su culpa.

Consumo eléctrico [459]: si se es el proveedor es algo a tener en cuenta. Aparte de la potencia necesaria para impulsar miles o cientos de miles de procesadores, unidades de disco duro, o periféricos, estos ordenadores generan una gran cantidad de calor, por lo que necesitan de ventiladores de enfriamiento. Se estima que el 50 por ciento de los costes de energía de un centro de datos se derivan solamente de las necesidades de refrigeración.

[144] http://www.zdnet.com/article/kim-dotcom-petabytes-of-megaupload-users-data-has-been-destroyed/
[145] http://www.zdnet.com/article/megaupload-data-to-be-destroyed-could-feds-swoop-on-file-sharers/
[146] http://www.zdnet.com/article/dropbox-sync-glitch-results-in-lost-data-for-some-subscribers/

9.5 *CLOUD COMPUTING VS MOBILE CLOUD COMPUTING VS CLUSTER COMPUTING VS GRID COMPUTING*

Ya se ha presentado que es *Cloud Computing*, pero existen otros tres términos parecidos y que pueden llevar a confusiones y que no hay que confundir. Estos son ***Mobile Cloud Computing***, ***Cluster Computing*** y ***Grid Computing***.

Mobile Cloud Computing es muy similar a *Cloud Computing*, aunque hay diferentes definiciones o usos de este término, pues abarca varios conceptos como bien explican en [466]. Uno de ellos sería el del uso de aplicaciones SaaS en el móvil, como pueden ser traductores, clientes de correo, etc. Es decir, de clientes ligeros, lo que implica que en este punto es igual que *Cloud Computing* en la capa de Software como un Servicio. Otro concepto es el de utilizar otros dispositivos móviles como proveedores recursos formando una nube local y así mejorar el uso de recursos al poder compartirlos entre dispositivos, por ejemplo, en el caso de que se necesite la ubicación, solo la pida un dispositivo y la comparta con el resto utilizando una nube local, obteniendo de esta manera un ahorro de diferentes recursos en los dispositivos. El tercer uso sería el de crear una nube intermedia entre el dispositivo y la nube general que hiciese de proxy y fuese la encargada de ejecutar la computación que debería de ejecutar el dispositivo móvil.

Cluster Computing es, de acuerdo con [470], [471], «un clúster es un tipo de sistema paralelo y distribuido, el cual consiste en una colección de ordenadores independientes interconectados trabajando juntos como un único recurso de computación integrado». En cambio, en ***Cloud Computing*** se utiliza la virtualización, en vez de trabajar sobre el ordenador directamente, para así poder permitir que sean aprovisionados dinámicamente [453].

Según Buyya [453], ***Grid Computing*** es «un tipo de sistema paralelo y distribuido que habilita el reparto, la selección y la colección dinámicamente de recursos autónomos distribuidos geográficamente en tiempo de ejecución dependiendo de su disponibilidad, capacidad, rendimiento, coste, y los requerimientos de la calidad del servicio (QoS) de los usuarios». Si miramos ***Cloud Computing***, los servidores están elegidos de antemano y pertenecen a una empresa o grupo de ellas que ya tienen todo configurado previamente para cuando se deba de escalar.

9.6 *CAPAS DE LA NUBE*

Cloud Computing está dividida en tres niveles [454], [462], [472], [473]: IaaS, PaaS y SaaS. Todos estos niveles, junto a otros más especializados, son conocidos como XaaS [469]. Estos se pueden ver en la Ilustración 41.

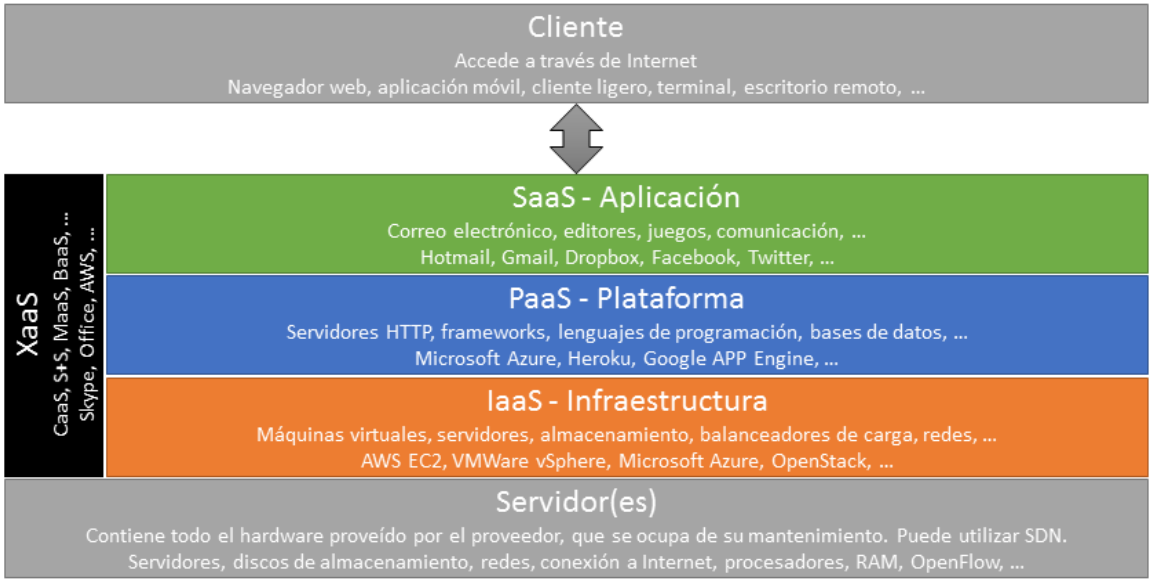


Ilustración 41 Capas de la nube

La ilustración anterior muestra cómo cuanto más hacía arriba más abstracción se consigue, pero también menos flexibilidad de cara al usuario final, que puede ser de diferente tipo. En IaaS y PaaS son los administradores y los desarrolladores de aplicaciones, respectivamente, mientras que en SaaS son los usuarios finales de las aplicaciones. Estas tres capas juntas son conocidas como XaaS, no obstante, hay muchos más servicios específicos que van montados dentro de estas capas y que poseen su propio nombre. También, como se muestra, las capas superiores pueden usar las capas inferiores, es decir, un SaaS puede estar montado sobre un PaaS y este a su vez sobre un IaaS, el cual ofrece la abstracción sobre el hardware base que puede utilizar a su vez SDN para controlar todo el hardware de una forma mucho más sencilla.

9.6.1 REDES DEFINIDAS POR SOFTWARE

Las redes definidas por software (SDN) son un conjunto de técnicas del área de redes computacionales que tratan de facilitar la implementación, elaboración y gestión de servicios de red de forma dinámica y escalable, evitando que el administrador deba de gestionar estos servicios a bajo nivel [474]. SDN han supuesto un avance significativo en la gestión de los recursos de las redes permitiendo la asignación de recursos de una manera más sencilla en comparación con los enfoques tradicionales [368].

9.6.1.1 CARACTERÍSTICAS DE LAS SDN

Las SDN han ido evolucionando a lo largo de los años, ofreciendo nuevas posibilidades a sus administradores:

- Las SDN son una abstracción de las redes permitiendo que el administrador no deba de gestionarlas a bajo nivel.
- Ofrece también separación de responsabilidades [474], de manera que se separa el plano de control, el cual es software de la topología y funcionamiento de la red, del plano de datos, que es hardware del router encargado de decidir qué hacer con los paquetes. Esto permite independizar la parte software de la parte hardware.
- Permite flexibilidad para adaptar el ancho de banda [474].
- Ofrece abstracciones sobre diferentes tecnologías y protocolos como VPN, GRE y OpenFlow [475].
- Permite virtualizar redes, crear túneles y optimizar túneles mediante el uso de OpenFlow.
- Permite migrar máquinas.

9.6.1.2 METADOMINIOS FUNCIONALES DE LAS SDN

Las técnicas de las SDN se pueden clasificar en varios metadominios funcionales según [474]:

- Técnicas para el descubrimiento dinámico de la topología de red, dispositivos y capacidades, con el fin de obtener los modelos de información y los datos relevantes para documentarlas.
- Técnicas para la exposición de servicios de red y de negociación dinámica del conjunto de parámetros del servicio para medir el nivel de calidad asociado a la prestación de un servicio determinado.
- Técnicas utilizadas por el servicio de requisito de asignación de recursos dinámicos y los esquemas de aplicación de políticas, permitiendo a las redes programarse en consecuencia. Ejemplo de estos casos son el estado de los recursos disponibles en la red en un momento dado, el número de solicitudes de suscripción a un servicio por parte de los clientes y que necesitan ser procesadas en un determinado período de tiempo, las previsiones de tráfico, la posible necesidad de desencadenar ciclos de aprovisionamiento de recursos adicionales de acuerdo con un plan maestro típica de varios años, etc.
- Mecanismos de retroalimentación dinámicos para evaluar la eficiencia de una política determinada.

9.6.2 **INFRAESTRUCTURA COMO UN SERVICIO**

Infraestructura como un Servicio, conocido por sus siglas en inglés *Infrastructure as a Service* (IaaS), proporciona entornos virtualizados, que son la base de *Cloud Computing* [465], computadores, almacenamiento, servidores, *switches*, balanceadores de carga y redes a través de una conexión de red, como es Internet [368], [460], [461], [472], [476].

En este caso, los usuarios, o clientes, son los encargados de mantener sus máquinas como ellos deseen, pero sin importar la infraestructura física, pues todo está virtualizado [368], [460], [472]. Es decir, es como tener un ordenador en remoto al que se accede a través de Internet desde cualquier lugar del mundo, siempre que la configuración de la nube lo permita, siendo los destinatarios de IaaS los administradores de servidores, o incluso los desarrolladores de aplicaciones. No obstante, dentro de estas máquinas virtualizadas el despliegue se hace mediante el método tradicional: instalar el sistema operativo, los programas, configurarlos, etc.

Luego, IaaS ofrece una abstracción al usuario sobre los detalles de la infraestructura como son es el hardware, la ubicación de las máquinas, la partición de los datos, los ajustes de seguridad (físicos y de red), las copias de seguridad, entre otros [460]. Por esto, en IaaS no es necesario comprar, configurar, ni mantener el hardware, además de poseer escalabilidad automática, siempre que se contrate. Además de relegar al proveedor el mantenimiento de todo el hardware. Esto se realiza mediante un hipervisor.

Además, IaaS permite el autoescalado sencillo si se utiliza un orquestador, pues este se encarga de monitorizar los recursos de las máquinas virtuales y los incrementa de forma automática, mientras que en caso contrario los reduce o incluso, en caso de necesitarse, puede iniciar nuevas réplicas de la máquina virtual.

Ejemplos de estos servicios son AWS EC2, AWS S3, AWS Glacier, las máquinas virtuales de Microsoft Azure, el IaaS de Red Hat Cloud Infrastructure, VMware vSphere, Citrix XEN Server, cualquier servidor privado virtual (VPS) y OpenStack.

9.6.3 PLATAFORMA COMO UN SERVICIO

Una Plataforma como un Servicio, conocida en inglés como *Platform as a Service* (PaaS), es un servicio en la nube que ofrece una abstracción sobre el entorno de ejecución de aplicaciones, como son el hardware, el sistema operativo, los *frameworks*, el almacenamiento y el autoescalado, que permite desplegar aplicaciones utilizando GPLs o herramientas del proveedor [368], [460], [461], [472], [476], tratando de juntar en un solo entorno todas las herramientas posibles y útiles para desarrollador, testear, desplegar y hospedar aplicaciones [469]. En PaaS no importan las máquinas, pues está centrado en el desarrollo de aplicaciones, haciéndolo menos flexible, pero más fácil de utilizar que IaaS, pues está orientado a los desarrolladores. Sin embargo, el usuario aquí no tiene control ni sobre el hardware, igual que pasaba en IaaS, ni sobre el sistema operativo, ni el almacenamiento [368].

PaaS es la base necesaria para el desarrollo de aplicaciones, pues además de poseer todos los beneficios de IaaS también permite instalar de una forma bastante rápida diferentes plataformas, es decir, diferentes bases de datos, lenguajes de programación, *frameworks*, librerías, etc. [460], [461], [472], [476], siempre que estén soportadas por el proveedor, quien además es el encargado de actualizar el software que ofrece y e ofrecer soporte a nuevas versiones. Además, también ofrece integración «fácil» con otros servicios web, pues basta con seleccionar otra configuración diferente a través de su interfaz [460]. Estas características hacen que PaaS sea bastante utilizado para entornos de desarrollos, producción y test, pues se puede montar una basa de datos, un servidor u otro componente software para desplegar aplicaciones de una forma rápida y fácil.

Para ofrecer este servicio, los proveedores suelen ofrecer una interfaz web desde la que se permite seleccionar que componentes se requieren en el sistema que se desee, para así de una manera rápida y automática la plataforma los instale. No obstante, a veces requiere cierta configuración por parte del usuario para seleccionar la versión, el puerto de despliegue y otros detalles relevantes.

Algunas ventajas extras de usar sistemas PaaS son [460]:

- **Gente no experta**: posible uso de gente no experta para realizar cierto tipo de desarrollos cuando se necesita instalar y configurar un tipo de programas gracias al uso de interfaces que facilitan este trabajo, como ocurre con la aplicación WordPress en servicios SaaS.
- **Flexibilidad**: permite instalar las herramientas específicas necesarias para adaptar la plataforma a las necesidades concretas simplemente mediante la elección de las que se estimen oportunas en la interfaz web.
- **Adaptabilidad:** las funcionalidades pueden modificarse si las circunstancias así lo aconsejan de una forma sencilla.
- **Actualizaciones son automáticas**: el proveedor es el encargado de añadir nuevas versiones del software que ofrece, así como de configurarlo de forma segura, hasta cierto punto. Esto hace que los clientes solo deban esperar a que el proveedor libere las nuevas actualizaciones del software que ofrece para seleccionar si se desea aplicar la actualización.

Algunos ejemplos de esto son AWS EC2 Container Service, contenedores de Microsoft Azure, el PaaS de Red Hat Cloud Infrastructure, Heroku y Google App Engine.

9.6.4 SOFTWARE COMO UN SERVICIO

SaaS, en español Software como un Servicio, también conocido como software bajo demanda, es la forma en que las compañías ofrecen servicios o aplicaciones alojadas en la nube a sus usuarios a través del acceso por Internet [460], [461], [476]. Esto hace que los usuarios no sepan donde se está ejecutando el software, ofreciéndole toda la abstracción máxima posible, sin necesidad de instalar nada en la mayoría de los casos y que incluye mantenimiento y actualizaciones, normalmente automáticas y gratuitas, y teniendo únicamente que preocuparse de cómo utilizar el software.

SaaS está ideado para que las compañías ofrezcan software online, simplificando así los procesos de distribución y ofreciendo una mejor calidad a los usuarios, pues las aplicaciones suelen ser multitenencia [469], lo que implica que el mantenimiento y las mejoras añadidas serán ofrecidos a todos los clientes. En este caso, SaaS ofrece todas las abstracciones presentadas en IaaS y PaaS [461], [472], [476] siendo la más sencilla de cara a los usuarios. Esto se debe a que SaaS está orientada a los usuarios, a ofrecerles un servicio el cuál solo deben de utilizar sin necesidad de instalarlo o preocuparse de su instalación, actualización, mantenimiento ni hardware necesario, es decir, sin necesidad de hacer nada. Además, los usuarios pagan por lo que usan [472], ya sea por tiempo, por ejecución u otra medida. Sin embargo, aquí el usuario dispone de menso flexibilidad pues simplemente puede utilizar la aplicación, ya que no puede ni configurar parámetros que no le permita el proveedor ni tiene acceso a todo lo que no se permitía tampoco en IaaS ni en PaaS [368].

El software tradicionalmente es adquirido mediante su compra física o digital y permite un número de claves ejecutándose en diferentes equipos, si permite más de una. Mientras, en SaaS se compra digitalmente, muchas veces es gratuito, y se utiliza directamente requiriendo normalmente la identificación del usuario a través de Internet, y guardando todos los datos en la nube [460]. Normalmente, el medio de acceso a este tipo de software es por medio del uso de navegadores web o aplicaciones específicas para ello, ya sea por una LAN o Internet vía HTTP(S), lo que ofrece un acceso universal al producto. Esto hace que los datos estén centralizados y sean accesibles desde cualquier sitio en el mismo estado en que se dejaron previamente.

Algunas ventajas extras de usar sistemas SaaS son [460]:

- **Actualizaciones automáticas**: cada vez que existe una actualización esta es liberada automáticamente y de forma transparente a todos los usuarios, aunque a veces se hace de forma escalable. Normalmente, las actualizaciones no tienen coste adicional.
- **Compatibilidad entre dispositivos**: normalmente, para acceder a las aplicaciones SaaS se puede utilizar cualquier dispositivo con conexión a Internet, siempre y cuando se utilice un navegador web, o en algunos casos, exista la aplicación nativa o el cliente ligero para múltiples plataformas.

Algunos ejemplos de SaaS son Google, Twitter, Facebook y Flickr [460], Hotmail, Gmail, Office 265, Amazon *Elastic MapReduce* (EMR), Evernote, Dropbox, Box, OneDrive, Drive, etc. Es decir, esto ofrece a los usuarios utilidades de todo tipo, siempre y cuando quieran y puedan utilizar aplicaciones en la nube, y que pueden ser desde redes sociales, servicios de correo, como vimos, hasta aplicaciones especializadas como EMR, de ofimática y de contabilidad y facturación.

9.6.5 X COMO UN SERVICIO

Tras la aparición de los tres anteriores ha habido la intención de subir todo a la nube, de hacerlo todo como servicios. Esto es llamado XaaS, EaaS o *aaS, siglas de X as a Service, Everything as a Service y * *as a Service*, respectivamente. La diferencia reside en que estos nuevos servicios son más especializados que los proveídos en SaaS, pero la idea originaría de XaaS surgió a partir de SaaS, sin embargo, algunos XaaS pueden ir montados directamente sobre las capas inferiores, pues pueden utilizar algo solo de esa capa, como es el caso de los *File as a Service* (FaaS) que utilizan solo el almacenamiento de IaaS o los *Backend as a Service* (BaaS), que se utilizan para desarrollo y serían parte de PaaS, pero siempre manteniendo al usuario final como objetivo. Es decir, según HP, EaaS trata de subir a la nube todo tipo de aplicaciones en forma de componentes separados[147].

Como se verá, prácticamente hay un XaaS diferente para cada tipo de aplicación, por eso, aquí se hablará de los que se creen más relevantes y utilizados en la actualidad, pues las tres capas de la nube se han comentado previamente, pues estas son IaaS, PaaS y SaaS.

[147] http://www.hp.com/hpinfo/initiatives/eaas/

Uno de ellos es la estrategia de Microsoft para servicios de conversación, ***Conversation as a Service*** (CaaS), presentado por Satya Nadella en la Build 2016 de marzo[148]. Esta estrategia trata de ofrecer a los seres humanos vías más fáciles de obtener información mediante la conversación con Inteligencia Artificial.

También existe la comunicación como un servicio, ***Communications as a Service*** (CaaS) [477][149], en el que se incluyen servicios de voz por IP (VoIP), mensajería instantánea y aplicaciones de videoconferencia. Los proveedores de CaaS se hacen cargo de la administración y mantenimiento del hardware y software necesario.

Microsoft creó más servicios, uno de ellos ha sido ***Software plus Services*** (S+S) [478], en donde fusionaron el enfoque tradicional con el enfoque de SaaS para hacer software de la manera tradicional que tenga partes en la nube, ofreciendo de esta manera más características o funcionalidades. Una de las mayores características es el poder trabajar sin conexión a Internet y en el momento en el que se disponga de ella, sincronizar todo el trabajo. Otra característica es el poder tener datos privados en local y datos no privados en la nube.

Otro de los términos existentes en Internet es el del monitoreo como un servicio, ***Monitoring as a Service*** (MaaS)[150]. Las aplicaciones de tipo MaaS son aquellas que se encargan de monitorear el funcionamiento de una plataforma que se encuentra en la nube y de notificar al usuario de diferentes eventos que surjan en ella, como errores, picos, etc. Servicios de este tipo serían CloudWatch[151] de AWS o Cloud Status para iPhone[152].

Otro caso es el de compañías que pasan a ofrecer sus servicios en la nube, creando así otro tipo de servicio, como es el caso de MuleSoft, empresa que ofrece un producto para facilitar la integración de servicios. Esto en la nube da lugar a ***Integration as a Service*** (IaaS)[153]. IaaS permite crear una forma fácil de compartir e integrar datos entre aplicaciones y con terceros.

Algo que está despuntando también actualmente con las aplicaciones multiplataforma son los *Back-end* como un servicio, es decir, ***Backend as a Service*** (BaaS)[154]. Estos proveen APIs y SDKs unificados en la nube, lo que permite a alguien que quiera realizar una aplicación multiplataforma crear únicamente la capa de presentación y enlazarla con un BaaS que le ofrezca la «información» necesaria para su aplicación, de forma que se ahorra todo el *Back-end*. Un ejemplo de esto es Parse[155], que ofrece una API para Xamarin, Android, iOSWindows Phone, Unity y React. Otro ejemplo es AWS Mobile[156].

148 http://blogs.microsoft.com/blog/2016/06/16/microsoft-acquires-wand-labs-to-accelerate-innovation-in-bing-intelligence-and-conversation-as-a-platform/
149 http://www.gartner.com/it-glossary/communications-as-a-service-caas/
150 https://www.techopedia.com/definition/29430/monitoring-as-a-service-maas
151 https://aws.amazon.com/es/cloudwatch/
152 https://itunes.apple.com/es/app/cloud-status/id1060625755
153 https://www.mulesoft.com/resources/cloudhub/integration-as-a-service
154 https://platzi.com/blog/mbaas/
155 http://parse.com/
156 https://aws.amazon.com/es/mobile/

Como ejemplo de almacenamiento están los ficheros como un servicio, en inglés ***File as a Service*** (FaaS)[157]. Estos son las aplicaciones que permiten almacenar a sus usuarios ficheros, sean del tipo que sean, en la nube. FaaS fue uno de los primeros servicios SaaS ofrecidos por medio de compañías como Dropbox. Otras aplicaciones FaaS son Box, OneDrive de Microsoft, Drive de Google, S3 de Amazon.

De almacenamiento también existe ***Data as a Service*** (DaaS o DBaaS)[158][479]. DaaS es un SaaS que permite almacenamiento de datos estructurados, ofreciendo algunas características de los sistemas de gestión de bases de datos relacionales (RDBMS), como son búsqueda, índices, asociaciones, etc. Sin embargo, actualmente hay mucha latencia frente a las bases de datos (BBDD) típicas. Algunos ejemplos de DaaS son AWS Simple DB[159], AWS Relational Database Service [160], SQL Azure Database de Microsoft[161] y Database.com de Salesforce[162].

Otro servicio muy de moda actualmente son las herramientas en la nube, conocido como ***Tools as a Service*** (TaaS)[163] [164] [480]. Los TaaS se dan cuando se ofrecen herramientas de desarrollo de software en la nube, como son los IDEs. Varios ejemplos son Python Fiddle [165], Codeanywhere [166], Visual Studio Online [167] y Cloud9 [168].

Otro servicio XaaS surgido no hace mucho es el que tiene que ver con los análisis de datos y con *Big Data*, conocido como ***Analytics as a Service*** (AaaS) o ***Big Data as a Service*** (BDaaS) [169] [455]. Estos son servicios que ofrecen análisis de datos o diferentes aplicaciones de *Big Data* como un servicio, ejemplos de esto son AWS EMR y HDInsight, que permiten ejecutar aplicaciones sobre Hadoop.

En cambio, si se utilizan los datos de los seres humanos para obtener información para solucionar una tarea determinada, se conoce como ***Human as a Service*** (HuaaS). Esto es muy aplicado sobre redes sociales para obtener los datos y poder predecir eventos.

[157] http://geekswithblogs.net/gotchas/archive/2010/08/09/la-tendencia-xaas-todo-como-servicio.aspx
[158] http://whatis.techtarget.com/definition/Database-as-a-Service-DBaaS
[159] https://aws.amazon.com/es/simpledb/
[160] https://aws.amazon.com/es/rds/
[161] https://azure.microsoft.com/es-es/services/sql-database/
[162] http://www.salesforce.com/platform/database/
[163] https://www.citrix.com/blogs/2011/06/30/citrix-support-tools-as-a-service-taas-webinar/
[164] http://www-01.ibm.com/software/globalization/terminology/t.html#x4627711
[165] http://pythonfiddle.com/
[166] https://codeanywhere.com/
[167] https://www.visualstudio.com/en-us/products/what-is-visual-studio-online-vs.aspx
[168] https://c9.io/
[169] http://www.forbes.com/sites/bernardmarr/2015/04/27/big-data-as-a-service-is-next-big-thing

9.7 *CLASIFICACIÓN DE LA NUBE*

Las nubes pueden ser de varios tipos, todo depende del proveedor o de si es la propia empresa quien la gestiona. En algunos casos también puede ser que la nube pertenezca a un consorcio o una comunidad. Por eso, a continuación, se explicarán las diferencias entre una nube pública, una privada, una comunitaria y una híbrida.

9.7.1 **NUBE PÚBLICA**

Una nube pública es cuando el proveedor no está vinculado a la organización que utiliza la nube y ofrece la nube a cualquiera que esté dispuesto a pagarla, es decir, es pública, y se paga por lo que se usa [452]. Estas nubes tienen multitenencia y son menos flexibles, pues se depende de lo que te ofrezca tu proveedor.

Los servicios son ofrecidos por medio de entornos virtualizados accesibles a través de Internet, ya sean IaaS, PaaS o SaaS. Además, ofrecen servicios a varios clientes utilizando la misma infraestructura compartida, es decir, los datos de diferentes clientes se almacenan, se computan y se transfieren utilizando el mismo hardware. Estas nubes están orientadas sobre todo a particulares que no requieran toda una infraestructura o a empresas o gobiernos que quieran externalizar esta parte [460], [461]. Este modelo es el más típico y conocido por los usuarios [460], pues ejemplos de estas nubes son AWS, que fue la primera nube pública [459], Microsoft Azure, Google Cloud Platform y Rackspace, entre otras, y siendo las dos primeras las más importantes y mejores de acuerdo al cuadrante mágico de Gartner [481].

Las nubes públicas ofrecen ciertas ventajas como son de escalabilidad, elasticidad, económicas, fiabilidad, flexibilidad, independencia de la localización [460]. Sin embargo, son menos flexibles en su configuración al depender del proveedor y encontrarse en las localizaciones de este [461], así como tener una seguridad más vulnerables y tener menos control sobre ellas [460]. Además, puede haber problemas legales o no ser usables según las leyes de protección de datos de cada país.

9.7.2 **NUBE PRIVADA**

Una nube privada se da cuando una empresa tiene su propio centro de datos de procesamiento, o bien lo tiene contratado con otra empresa, pero en donde por lo general hay un solo usuario autorizado, que es la propia empresa, es decir, no es de acceso público [452]. No obstante, hay empresas como Amazon, con su servicio AWS, o Microsoft con Azure, que tienen su propio centro de datos, pero ofrece servicios públicos a otras compañías. En cambio, Facebook tiene su propia nube privada y no ofrece servicios a terceros. Al igual que en el resto de tipos de nubes, la nube privada ofrece servicios de diferente tipo, IaaS, PaaS o SaaS, pero con la diferencia de que a estos servicios solo pueden acceder un solo tipo de usuario, como pueden ser el personal de esa empresa [460], [461]. Sin embargo, hay casos en que la nube privada es administrada por un tercero independiente de la empresa o por la combinación de la empresa y un tercero [461].

En este caso la flexibilidad es máxima, pues las compañías crean la nube como ellos desean y con los servicios que ellos necesiten. Un ejemplo de esto es Amazon, que los servicios que ofrece al público son

servicios que fueron creando previamente debido a la necesidad que ellos tenían para gestionar la tienda y crearon dichos servicios para facilitar su gestión.

Las ventajas de las nubes privadas son el acceso exclusivo a los recursos, supuestamente mayor seguridad y privacidad al haber menos usuarios y estar restringidos utilizando la infraestructura y muchas veces acceder mediante una LAN y no Internet y mayor flexibilidad sobre su configuración [460]. Además, ofrece un mayor control de los datos y posible mejor adecuación a las leyes de protección de datos de cada país. Sin embargo, puede que sean menos económicas debido al mantenimiento y actualización que necesitan [460].

9.7.3 **NUBE COMUNITARIA**

En este caso, según la NIST [461], una nube comunitaria es aquella nube que está preparada para el uso exclusivo de una comunidad específica de los consumidores, que pueden ser varias organizaciones o personas, que comparten ciertas preocupaciones, como son una misión, un proyecto, unos requisitos de seguridad, unas política y un cumplimiento. Esta nube puede estar en propiedad, ser administrada y estar operada por una o más de las organizaciones pertenecientes a esta comunidad, por un tercero, o por alguna combinación de ambos. Además, esta nube puede existir dentro o fuera de los locales de esta comunidad.

Un ejemplo de nube comunitaria es el gobierno de los Estados Unidos de América [458]. Han desarrollado una nube comunitaria que permite al gobierne desplegar rápidamente aplicaciones muy específicas de diferentes organizaciones como Forms.gov, Cars.gov y Flu.gov, y que pertenecen todas al mismo portal oficial del propio gobierno USA.gov.

9.7.4 **NUBE HÍBRIDA**

Una nube híbrida es una nube configurada de tal manera que utilice dos o más tipos diferentes de nubes, públicas, privadas y/o comunitarias, siendo transparentes y pareciendo una única nube con el fin de ofrecer un mejor servicio a una misma organización [460], [461], [476]. Estas nubes están unidas mediante tecnologías estandarizadas o propietarias que permiten la portabilidad de los datos y las aplicaciones [461]. El uso de nubes híbridas viene dado por el tema de que las nubes públicas ofrecen un mayor ahorro y escalabilidad que las nubes privadas, luego, esto permite a una organización que optimice sus servicios para recurrir a las nubes públicas cuando sea mejor que las privadas o se necesite [460].

Esto puede ser debido a que el hardware privado se encarecería demasiado para cubrir ciertos picos de usuarios, ciertas aplicaciones son mejores en la nube pública debido a la falta de recursos o poco uso de ellas en la privada, o a que se necesite la nube privada para guardar ciertos datos que no pueden alojarse en una nube pública, entre otras muchas posibilidades. Un ejemplo de un servicio que suelen utilizar las empresas en la parte de la nube pública es el del correo electrónico [170].

Las nubes híbridas pueden llevadas a cabo de tres maneras [460]:

- Proveedores que ofrezcan al contratante ambos tipos de nube.

[170] http://www.dell.com/learn/mx/es/mxbiz1/large-business/deploying-a-hybrid-cloud

- Una combinación de proveedores, donde uno provea de una nube pública y otro de una nube privada.
- Nube privada mantenida por la empresa, la cual solo contrata la nube pública a un proveedor.
- Aquí el cliente puede asegurarse la computación necesaria sin necesidad de pagar de más.

Los beneficios de usar una nube híbrida son de flexibilidad, economía, la seguridad, la escalabilidad y la elasticidad. Sin embargo, la desventaja es el cómo distribuir todo de una manera transparente y con buen rendimiento entre ambas nubes.

9.8 *TEOREMA CAP Y PACELC*

El teorema CAP fue presentado en el año 2000 por Eric Brewer, en el simposio de Principios de Computación Distribuida (PODC), enunciando que es imposible para un sistema de cómputo distribuido garantizar más de dos de las siguientes características de forma simultánea: consistencia, disponibilidad y tolerancia al particionado (CAP – *Consistency*, *Availability*, *Partition*) [482]. Posteriormente, en el año 2002, Seth Gilbert y Nancy Lynch, pertenecientes al MIT, publicaron la demostración formal de la conjetura que la convirtió en un teorema [483], [484]. Como nota, un servidor único [es decir, cuando solo existe un servidor, un ordenador] cumple consistencia y disponibilidad, pues no puede particionarse. En la Ilustración 42 se muestra el teorema CAP en base a diferentes bases de datos existentes.

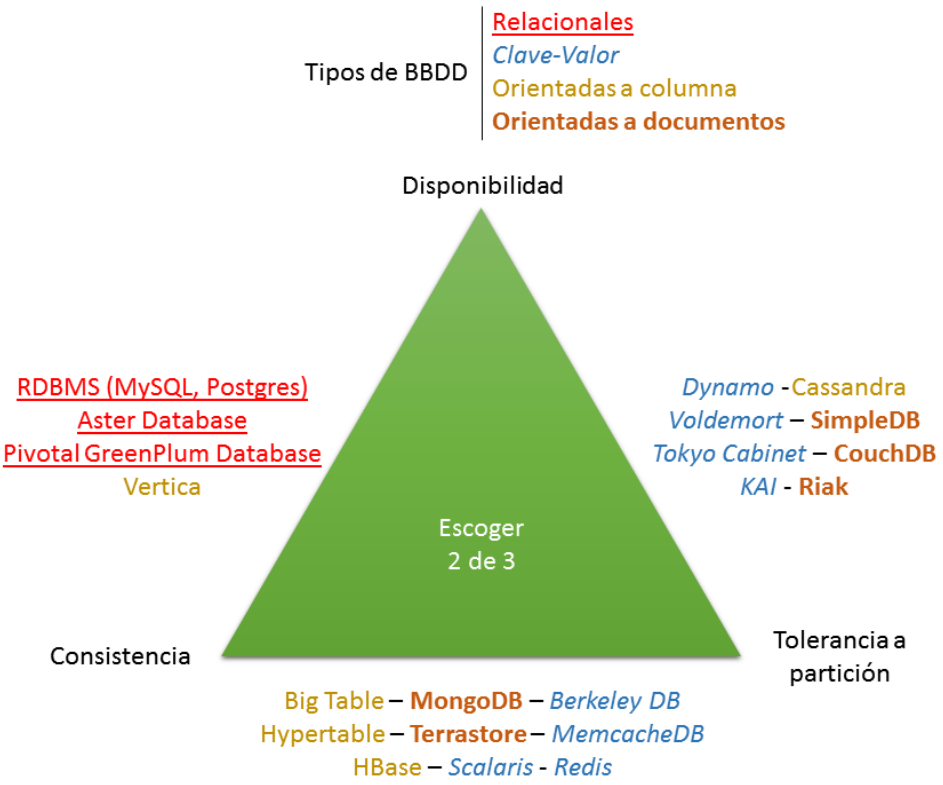


Ilustración 42 Teorema CAP con ejemplos de BBDD

La **consistencia** indica que todos los datos tienen que ser iguales en todas las localizaciones y réplicas, pero esto implica sacrificar la tolerancia a la partición. Por ejemplo, algunos portales famosos tienen inconsistencias para así tener o mejorar las otras características, estos son eBay y Amazon [485].

La **disponibilidad** es la garantía de que un nodo pueda notificar si ha recibido y resuelto o no la peticiones realizada. Esto hace que la disponibilidad sea muy difícil de sacrificar, pues en el caso de que no ofrezca disponibilidad, no se sabe si el nodo ha revivido o no el mensaje.

La **tolerancia al particionado** indica si el sistema puede seguir funcionando a pesar de que se haya partido por un fallo en la red, lo que puede significar perder la sincronización. Sin embargo, esto no implica que pueda haber corrupción de datos o inconsistencia. Algunos algoritmos de selección de líderes son Paxos y RAFT, lo que permite que unas particiones sea la que lidere todos los procesos para intentar evitar o disminuir los problemas que puedan surgir cuando se particiona. Para ganar en tolerancia, lo que hacen muchos sistemas es relajar la consistencia [484]. No obstante, con los avances surgidos en los centros de datos y la redundancia de redes que poseen es raro que exista un fallo de red, con lo que el compromiso entre la tolerancia y la consistencia sea menos relevante a día de hoy [484], [485].

Posteriormente, en el año 2012, Daniel Abadi presentó un teorema basado en el teorema CAP y que se llama teorema PACELC [486]. PACELC son las siglas de:

> «*If there is a partition (P), how does the system trade off availability and consistency (A and C) ?; else (E), when the system is running normally in the absence of partitions, how does the system trade off latency (L) and consistency (C)?*».
>
> «Si hay una partición (P), ¿cuál es el compromiso del sistema entre disponibilidad (A) y consistencia (C)? En (E) otro caso, cuando el sistema funciona normalmente sin particiones, ¿cuál es el compromiso del sistema entre latencia (L) y consistencia (C)?»

Esto permitió a los proveedores ampliar la interpretación de CAP, considerando así si una red es lenta entonces es considerada particionada [484], es decir, mirando la latencia de la red, pues el teorema CAP ignoraba la latencia a pesar de estar profundamente relacionada con las particiones [487].

Por ejemplo, las bases de datos Dynamo, Cassandra y Riak son PA/EL, lo que indica que, si ocurre una partición, eligen consistencia en vez de disponibilidad, mientras que en condiciones normales ofrecen consistencia con una latencia baja. Otros ejemplos de BBDD son los mostrados en la Tabla 3 en base a [486].

BBDD	P+A	P+C	E+L	E+C
Dynamo	X		X	
Cassandra	X		X	
Riak	X		X	
VoltDB/H-Store		X		X
Megastore		X		X
MongoDB	X			X
PNUTS		X	X	

Tabla 3 BBDD según el teorema PACELC

10. INTELIGENCIA ARTIFICIAL

«Una computadora puede ser llamada "inteligente" si logra engañar a una persona haciéndole creer que es un humano»
Alan Mathison Turing

«La tecnología conocerá cosas sobre los humanos, pero los humanos deberán saber sobre las máquinas»
Satya Nadella [488]

«Preguntarse cuándo los ordenadores podrán pensar es como preguntarse cuándo los submarinos podrán nadar»
«El esfuerzo de utilizar las máquina para emular el pensamiento humano siempre me ha parecido bastante estúpido. Preferiría usarlas para emular algo mejor»
Edsger W. Dijkstra

Los seres humanos tenemos la habilidad de **aprender** cosas nuevas de forma automática debido a las capacidades con las que nacemos. Simplemente necesitamos tener experiencias, leer, estudiar… vivir. Por estos procesos somos capaces de adquirir nuevas habilidades o modificar las que ya tenemos. Otra de las habilidades que poseemos es la de poder **pensar**, imaginar, formar nuestras propias ideas, soñar. Todo esto para nosotros resulta bastante sencillo y lo realizamos día a día sin darnos cuenta.

¿Pero qué ocurre cuando esto lo extrapolamos a las máquinas? **Las máquinas pueden aprender**. Les podemos enseñar. En los últimos años se hicieron notables avances y hemos visto como existen coches que reconocen peatones u otros coches, sistemas que diferencian animales, e incluso, hemos visto como algunas inteligencias artificiales han podido hasta soñar, pintar y componer música. Como se ve, en los últimos años se han realizado muchos avances tanto en el aprendizaje automático como en todos los algoritmos que se utilizan.

Pese a esto, la duda aquí es la siguiente: **¿Pueden las máquinas pensar?** O, mejor dicho, ¿Podría una máquina hacer creer a una persona con la que está hablando y que se encuentra en otro cuarto que piense que está hablando con un ser humano? Esta es una duda que ha estado presente desde que Alan Mathison Turing la realizó y que todavía no se ha contestado.

En este capítulo se verán los inicios de lo que se conoce como **Inteligencia Artificial** y las ramas de esta que fueron utilizadas en esta tesis: ***Machine Learning***, **Visión por Computador**, **Lógica Difusa** y **Procesamiento de Lenguaje Natural**. Se hablará de cada una de ellas, de sus conceptos y de cómo funcionan.

10.1 *¿PUEDEN LAS MÁQUINAS PENSAR?*

Esta fue la pregunta que se hizo Alan Mathison Turing y a la cual respondió con la propuesta del juego de la imitación [489]. ***The Imitation Game*** es jugado por tres personas, «A» que es un hombre, «B» que es una mujer, y «C» que es el interrogador y puede tener cualquiera de los sexos.

El interrogador se encuentra en una habitación diferente a «A» y «B». Durante el juego, «C» puede hacerles cualquier tipo de pregunta para así determinar quién es el hombre y quien es la mujer, por ejemplo, acerca de la longitud del pelo. Durante el juego, «C» conoce a los participantes bajo las etiquetas «X» e «Y». Cuando el juego termina, C debe de decir «X es A» e «Y es B» o viceversa. En este juego, el tono de voz no ayuda a «C», pues «A» y «B» deben de escribir las respuestas, preferiblemente utilizando una máquina, las cuales son enviadas por un teletipo o por medio de otro intermediario a «C».

A este problema se le puede meter una nueva variante que se basa en sustituir al hombre «A» por una máquina. En este caso, se podría pedir a «A» y a «B» que escribiesen poesía, pero la máquina podría negarse. Ante esto, se podría mandar una operación matemática y ver que la máquina la resolvió o bien preguntarles si juegan ajedrez. No obstante, para evitar comparaciones indebidas en el juego, se evita que el interrogador pueda pedir demostraciones, pues en algunos casos es mejor la máquina y en otros mejor el ser humano, además de poseer cada uno cualidades diferentes. Por estos motivos, este juego se trata de que la máquina trate de imitar el comportamiento del ser humano, asumiendo que podrá dar repuestas igual que un ser humano lo haría de forma natural. A partir de esta teoría surgió el conocido **Test de Turing**.

Sin embargo, la imitación de una máquina es muy difícil de darse, como bien comenta Turing desde diferentes puntos de vista, a saber: el teológico, el de la gente reacia a conocer el problema (*Head-in-the-sand*), el matemático, el de la consciencia, el de las diferentes discapacidades, desde el punto de vista de las memorias de Ada Lovelace [490], desde la continuidad del sistema nervioso, desde el argumento de la familiaridad del comportamiento y desde la percepción extrasensorial.

Por los motivos expresados desde los diferentes puntos de vista, Alan Turing se centró en lo comentado por Ada Lovelace cuando ella dice que un hombre puede «inyectar» una idea en una máquina. Así, en base a esto, planteó como hipótesis el poder inyectar el conocimiento si se tuviera un espacio lo suficientemente grande para ello, o al menos, para que la máquina pudiera jugar al ***The Imitation Game*** y dejando como problema el cómo programar las máquinas para que superaran el juego, pues según Turing, la capacidad hardware ya la tenían.

Para llevar a cabo esta idea, propuso hacer el cerebro de un niño en vez de tratar de hacer un cerebro adulto, y así educarlo para obtener el cerebro de un adulto. De esta manera, divide el problema en dos partes: el programa «niño» y el proceso de educación. Claramente, explica que no hay que esperar que la primera máquina «niño» salga en el primer intento y que habría que enseñarle para ver cómo evoluciona su aprendizaje. Así, tras varios intentos, se irían consiguiendo máquinas mejores, o peores, algo que comparó Turing con el proceso de la evolución y que diversos investigadores desarrollarían más adelante bajo el nombre de **algoritmos genéticos** [491].

En base a esto, Turing habló de diferentes experimentos que realizó y de necesidades de las máquinas para simular a un ser humano, como es el caso de la incorporación de elementos aleatorios para imitar la posibilidad de fallo que tenemos los seres humanos. Para finalizar, comentó el pensamiento de muchas personas acerca de que se podría empezar a hacer competir a las máquinas con los humanos en actividades abstractas, como jugar al ajedrez, pero remarcando que, bajo su punto de vista, era mejor montar en las máquinas los mejores «órganos» que el dinero pudiera comprar y enseñarles así a entender y hablar en inglés.

10.2 *¿QUÉ ES LA INTELIGENCIA ARTIFICIAL?*

Los ordenadores solo pueden procesar ceros y unos. Sin embargo, años atrás, la **Inteligencia Artificial** (**AI**) nació para ofrecer la posibilidad de crear programas que permitiesen a las computadoras aprender. Este fue el propósito de los primeros científicos informáticos, como Alan Mathison Turing, John von Neumann y Norbert Wiener, entre otros, que trataron de imbuir a los ordenadores con programas que contuvieran **inteligencia**, la capacidad de autocopiarse, de aprender y de controlar su entorno, siendo, el principio de esto, el tratar de modelar el cerebro humano, imitar el aprendizaje humano y simular la evolución biológica [492].

La investigación en **Inteligencia Artificial** comenzó después de la segunda guerra mundial, siendo, posiblemente, Alan Mathison Turing el primero en investigar en este campo en el año 1947 [493], y publicando un artículo acerca de si las máquinas podían pensar en 1950 [489]. De este artículo es de donde proviene el famoso ***The Imitation Game***, del que surgió el **Test de Turing**. Ambos sirven para saber si una máquina es lo suficientemente inteligente como para confundirla con un ser humano. En dicho artículo, Turing también explicó cómo se debería de crear un sistema de **AI**, empezando por la creación de una máquina niño que fuese aprendiendo hasta ser como un adulto. Posteriormente, en la década de los años 50s se sumaron muchos más investigadores a este campo [493].

Posteriormente, John McCarthy, Marvin L. Minsky, Nathaniel Rochester y Claude E. Shannon acuñaron el término **Inteligencia Artificial** en 1955 en el proyecto de investigación de verano de Dartmouth [494]–[496]. En este proyecto se propuso una estancia de dos meses para investigar los principios acerca del aprendizaje u otro tipo de inteligencia y ver si los ordenadores podían simularlo, o, mejor dicho, con el nombre que ellos acuñaron, investigar en **Inteligencia Artificial**.

¿Pero qué es exactamente la **Inteligencia Artificial**? Según John McCarthy [493], la **Inteligencia Artificial** es la ciencia e ingeniería que trata de hacer inteligentes a las máquinas, tratando de conseguir que entiendan el lenguaje humano y que lleguen a resolver problemas y metas tan bien como un ser humano. No obstante, hay algunos autores que afirman que el objetivo de la **AI** es el de poner la mente humana dentro de un computador, aunque posiblemente hablen metafóricamente. Una de las formas de saber si una máquina es inteligente es mediante el **Test de Turing**.

Para responder a la pregunta anterior, también hay que entender que se entiende por **Inteligencia**, para lo que se puede referenciar a [493]. La **inteligencia** es la parte computacional de la habilidad de lograr objetivos y que posee varios tipos y grados como ocurre en los humanos, muchos animales y algunas máquinas. No obstante, no hay que considerar como **inteligencia** el saber responder «sí» o «no» a una

pregunta, pues, a pesar de que esto sea a veces la meta, las máquinas pueden realizar mucho más, como, por ejemplo, resolver problemas.

Algunos de los campos donde la **Inteligencia Artificial** es aplicada son, por ejemplo, los videojuegos, la minería de datos, reconocimiento del discurso, entendimiento de lenguaje natural, Visión por Computador, la creación de sistemas expertos y en clasificación de datos [493].

La **Inteligencia Artificial** se puede dividir en varias ramas entre las que destacan el aprendizaje automático, la Visión por Computador, la Lógica Difusa, el procesamiento de lenguaje natural, la heurística y los agentes inteligentes.

10.3 *LAS SEIS REGLAS DE LA INTELIGENCIA ARTIFICIAL*

Al igual que para la robótica existen Las tres reglas de la robótica, creadas por Isaac Asimov en la primera literatura acerca de robots, Satya Nadella, CEO de Microsoft, ofreció en una entrevista a la revista online Slate [171] un esbozo de seis reglas que debían de ser observadas por los diseñadores de Inteligencia Artificial [488].

1. **La AI debe estar diseñada para asistir a la humanidad**, necesitando respetar la autonomía humana y utilizando la *Robótica* de robots colaborativos (subcapítulo 12.3.3 - Desde el punto de vista de los grupos de robots) para realizar trabajos peligrosos, como la minería, salvaguardando de esta manera a los trabajadores humanos.
2. **La AI debe ser transparente, siendo conscientes de cómo trabaja la tecnología y sus reglas**. O como dijo Satya Nadella: «La tecnología conocerá cosas sobre los humanos, pero los humanos deberán saber sobre las máquinas». Esto permitiría que los humanos entendiesen como la tecnología ve y analiza el mundo, pues la ética y el diseño deben de ir de la mano.
3. **La AI debe maximizar la eficacia sin destruir la dignidad de la gente**: deberá preservar los compromisos culturales, fortaleciendo la diversidad. Se necesita un compromiso más amplio, profundo y diverso con la población, pues la tecnología no debe de dictar ni los valores ni las virtudes del futuro.
4. **La AI debe ser diseñada para la privacidad inteligente**. Se necesitan sofisticados métodos de protección que aseguren la información personal en aras de ganarse la confianza.
5. **La AI debe tener responsabilidad algorítmica para que los humanos puedan deshacer el daño no intencionado**. Por esto se hace necesario diseñar la tecnología de AI para lo esperado y lo no esperado.
6. **La AI debe evitar el sesgo asegurando una investigación adecuada y representativa de modo que una heurística errónea no pueda ser utilizada para discriminar**.

[171] http://www.slate.com/

Sin embargo, Satya Nadella también aclaró que los humanos «deberán» priorizar y cultivar una serie de características para poder convivir con las AIs, como son:

1. La **empatía** para crear relaciones con otros y percibir sus sentimientos y pensamientos, pues esto es algo muy difícil de replicar en las máquinas.
2. La **educación** que será necesaria para creer y gestionar las innovaciones que no se entender a día de hoy. Para esto, se necesitará incrementar la inversión en educación que permita desarrollar el conocimiento y la habilidad necesarias para implementar las nuevas tecnologías y resolver estos problemas que necesitarán tanto tiempo.
3. La **creatividad**, pues esta es la habilidad humana más codiciada y que las máquinas continuarán enriqueciendo y aumentando.
4. El **juicio y la responsabilidad** para estar dispuestos a aceptar un diagnóstico o decisión legal tomada por una máquina, pero de la que esperaremos que todavía un humano siga siendo el último responsable de tomar la decisión.

10.4 *MACHINE LEARNING*

El **aprendizaje automático**, **aprendizaje máquina**, o más conocido por su nombre en inglés, ***Machine Learning***, es una de las ramas de la Inteligencia Artificial. Esta se basa en el desarrollo de técnicas de aprendizaje para permitir que las máquinas puedan aprender [497], basándose en el análisis de datos y aplicando unos algoritmos para este procesamiento y otros para la toma de decisiones a partir de los datos ya analizados.

Una de las metas del **aprendizaje automático** es la de facilitar el proceso de análisis de datos de forma inteligente, ejemplos de ellos son los filtros de spam [498], las técnicas de reconocimiento óptico de caracteres, del inglés ***Optical Character Recognition*** (OCR) [498], cuantificación de relaciones entre datos [498], diagnósticos médicos asistidos por computador [498], los motores de búsqueda [499], en **Visión por Computador** [500], [501] o ***Data Mining*** [416], [502], [503].

10.4.1 TAXONOMÍA DE LOS TIPOS SEGÚN EL TIPO DE APRENDIZAJE

Se pueden clasificar los algoritmos de ***Machine Learning*** en base al tipo de *feedback* recibido. En este subapartado se verán un total de seis categorías: **aprendizaje supervisado**, **aprendizaje no supervisado**, **aprendizaje semisupervisado**, **aprendizaje por refuerzo**, **aprendizaje por transducción** y **aprendizaje multitarea**.

La primera es la del **aprendizaje supervisado**. Estos algoritmos infieren una función por medio de datos de entrenamiento etiquetados que reciben como entrada [498], [504], [505]. Un ejemplo de estos algoritmos es el de los algoritmos utilizados para clasificar imágenes cuando se utilizan técnicas de Visión por Computador. Estos algoritmos deben de ser alimentados con imágenes etiquetadas, por ejemplo, como correctas o incorrectas. De esta manera, se alimenta el algoritmo para que aprenda a clasificar las imágenes según las imágenes previamente etiquetadas suministradas. Por ello, se puede decir que el algoritmo aprende

por si mismo [506]. Algunos de los algoritmos que se encuentran en este grupo son algunos de los pertenecientes a las **Redes Neuronales Artificiales**, las **Redes Bayesianas**, las **Máquinas de Vectores de Soporte** y los **Árboles de Decisión**.

Otra taxonomía es la del **aprendizaje no supervisado** [505], [506]. La diferencia con el anterior es que no se etiquetan los datos de entrada y es el propio sistema el que debe de inferir algoritmos o patrones para poder reconocer y etiquetar el mismo tipo de datos. En este grupo encontramos algunos tipos de **Redes Neuronales Artificiales**, las **Reglas de Asociación** y algunos **Algoritmos de Agrupamientos**.

En cambio, en el **aprendizaje semisupervisado** se realiza una combinación de los dos algoritmos anteriores [507]. De esta manera, el sistema ha de tener en cuenta tanto las imágenes etiquetadas como las imágenes sin etiquetar.

Otro tipo de aprendizaje es el **aprendizaje por refuerzo** [508]. Este se basa en que el algoritmo aprenda en base al mundo que lo rodea. En este caso, el algoritmo aprende a partir de las repuestas que le da el mundo exterior en base a sus acciones, criticándoselas si fueron buenas o malas. Por ello, este es un algoritmo basado en el clásico aprendizaje de ensayo-error.

Un algoritmo similar al del aprendizaje supervisado es el de **aprendizaje por transducción**, presentado en [509]. La diferencia reside en que, en este caso, no se crea la función de forma explícita, sino que se trata de predecir las categorías de los futuros ejemplos basándose en los ejemplos de entrada. De este modo, este aprendizaje trata de crear grupos mediante el etiquetado de los elementos de las entradas recibidas. Un algoritmo que se encuentra en esta categoría es el de ***Transductive Support Vector Machine*** (TSVM) [510], [511].

La última taxonomía es la de **aprendizaje multitarea**. Estos algoritmos utilizan conocimiento previamente aprendido en problemas parecidos ya experimentados para mejorar el aprendizaje del problema que se desea resolver [512].

10.4.2 TAXONOMÍA DE LOS ALGORITMOS UTILIZADOS EN MACHINE LEARNING

Existen diferentes tipos de algoritmos para realizar el aprendizaje que se pueden agrupar en función del tipo de salida de los mismos. En este subapartado se hablará de **árboles de decisión**, **reglas de asociación**, **algoritmos genéticos**, **Redes Neuronales Artificiales**, **Aprendizaje Profundo**, **Máquinas de Vectores de Soporte**, **algoritmos de agrupamiento** y **redes bayesianas**.

El primero es el uso de **árboles de decisión** [416], [503], de los cuales se puede encontrar información de su uso en la Inteligencia Artificial desde el año 1986 en [513]. Los **árboles de decisión** son un conjunto de nodos dispuestos como un árbol binario. En los nodos intermedios se encuentran las condiciones necesarias para llegar a las hojas, donde estas condiciones o predicados pueden implicar una o más características del elemento. Mientras, al final de cada rama, lo que se conoce como hoja, se encuentra la salida que debe de devolver el objeto tras analizar las diferentes posibilidades obtenidas en la entrada y recorrer el árbol, es decir, las hojas contienen la decisión a tomar. En base a esto, estos algoritmos contienen mapeadas las posibles entradas y las posibilidades de dichas entradas que puede contener el objeto. El funcionamiento de los **árboles**

de decisión para clasificar un elemento consiste en comenzar a analizar ese elemento por la raíz del árbol y, en base a si cumple o no el predicado del nodo en el que se encuentre, desplazar ese elemento al hijo izquierdo o derecho del nodo actual, repitiendo el proceso hasta llegar a una hoja que devuelva el resultado del árbol.

Las **reglas de asociación** contienen aquellos algoritmos que tratan de descubrir asociaciones entre variables de un mismo grupo [503]. De esta manera, estas permiten descubrir hechos que ocurren en un determinado conjunto de datos. Un posible ejemplo de estas es que, si alguien publica un *tweet* con una foto y el hashtag «*selfie*», entonces la foto la hizo con su móvil. Un caso real es el estudiado por [514], donde descubrieron que el 90% de las compras que contenían pan y mantequilla también tenían leche.

Otro tipo de algoritmos son los **algoritmos genéticos**. Estos no son más que un proceso de búsqueda heurística que intenta simular la selección natural siguiendo un proceso de evolución de individuos por medio de acciones aleatorias. Así, estos algoritmos contienen métodos como la mutación, el cruzamiento y la selección con el fin de encontrar la mejor solución a un problema dado. Algunos ejemplos de uso de estos algoritmos son en el campo de los diseños automatizados de equipos y maquinarias, en teoría de juegos, en aprendizaje de reglas de Lógica Difusa, procesamiento de lenguaje natural (NLP) y predicción, entre otros. La creación de los **algoritmos genéticos** se debe a John Holland y a sus estudiantes de la Universidad de Michigan, quienes los desarrollaron entre los años 60 y 70 y los presentaron de forma teórica en [491]. De esta manera, su meta no era buscar algoritmos para solucionar un problema concreto, sino importar en las computadoras el fenómeno que se da en la naturaleza por el cuál esta se adapta a lo que ocurre alrededor de ella [492].

Otro caso es el de las **Redes Neuronales Artificiales** (ANN) [505], [515]. Las ANN permiten el aprendizaje automático mediante la simulación del sistema de neuronas del sistema nervioso de los animales. Dentro de este sistema, las neuronas colaboran entre sí para producir la salida, teniendo en cuenta que cada conexión entre neuronas tiene un peso, el cual se va adaptando según la experiencia recogida a lo largo de su uso. Las ANN se componen de una capa de entrada, una o más capas ocultas y una capa final de salida, todas ellas compuestas por neuronas. Por ello, las redes neuronales son capaces de aprender, y se utilizan desde para la traducción de idiomas hasta para el aprendizaje de patrones.

El **Aprendizaje Profundo**, conocido en inglés como ***Deep Learning***, es un conjunto de algoritmos de ***Machine Learning*** que se basan en modelar abstracciones de alto nivel compuestas de transformaciones no-lineales múltiples [516], [517]. El uso de ***Deep Learning*** permite aprender representaciones de datos. Por ejemplo, una imagen puede representarse de muchas formas, pero no todas ellas facilitan la tarea de reconocimiento de caras, así, lo que hace es intentar definir cuál es el mejor método de representación para realizar esta tarea. Algunos ejemplos de su uso son en la **Visión por Computador** y en el reconocimiento del habla y de música [517]. Los algoritmos utilizados pueden ser tanto algoritmos de **aprendizaje supervisado** como de **aprendizaje no supervisado** [518]. Los algoritmos de **aprendizaje profundo** tienen diferentes transformaciones internas entre la entrada de los datos y la respuesta de salida, componiéndose cada transformación de diferentes pesos y umbrales que pueden ser entrenados [518]. No obstante, según [518], no existe una definición que diga exactamente los límites entre ***Deep Learning***, ***Shallow Learning*** y ***Very Deep***

Learning, pero si proponen que se considere ***Very Deep Learning*** cuando se exceda de 10 niveles de profundidad.

Las **Máquinas de Vectores de Soporte** [519], conocidos también por su nombre en inglés ***Support Vector Machine*** (SVM), y usadas en investigaciones similares como ***Support Vector Networks*** (SVN) [520] y ***Support Vector Regression*** (SVR) [521], son un conjunto de métodos supervisados que se utilizan para solucionar problemas de clasificación y regresión, produciendo límites no lineales mediante el uso de límites lineales, transformando la versión del espacio de características [502], [505]. El funcionamiento de este tipo de algoritmos se basa en la selección de un pequeño número de casos límite críticos, conocidos como vectores de soporte (*support vectors*), para construir la función discriminante que logre separar lo máximo posible los casos existentes [416], [502]. Por ejemplo, mediante el uso de SVM se puede clasificar un conjunto de imágenes a partir de un entrenamiento previo con imágenes etiquetadas para enseñarle al algoritmo cómo ha de clasificar. Un ejemplo de esto es el de reconocimiento óptico de caracteres presentado en el primer artículo que trató sobre ellas [519].

Los **algoritmos de agrupamiento** o ***clustering***, son aquellos que clasifican los datos similares en base a unos ciertos criterios en subgrupos, también llamados clústeres [416], [502], [522], [523]. El éxito del ***clustering*** se mide en base a lo útil que resulta esta agrupación para los seres humanos [502]. Existen muchos algoritmos diferentes para realizar esta tarea debido a que la noción de «clúster» no está definida muy precisamente [524]. Para ello busca la distancia o similitud de los datos según la función definida. Este tipo de algoritmos se suelen usar en la minería de datos [416], [502], [525], así como en otros casos de análisis estadísticos [526]. Estos algoritmos son algoritmos de **aprendizaje no supervisado**.

Otro tipo de algoritmo son las **redes bayesianas** [496], [527]. Las **redes bayesianas** son modelos probabilísticos que representan variables al azar y sus independencias incondicionales en un grafo acíclico dirigido. Esto sirve, por ejemplo, para representar relaciones probabilísticas entre enfermedades y síntomas. De esta forma, se puede calcular la probabilidad de que una enfermedad esté o no presente en el organismo.

10.5 *VISIÓN POR COMPUTADOR*

Dentro de la Inteligencia Artificial, uno de los campos es la **Visión por Computador**. La **Visión por Computador** es el campo que permite a los computadores «aprender» a reconocer una imagen y las características de esta imagen. La meta que busca la Visión por Computador es que las máquinas puedan entender el mundo [528]. Conseguir que las máquinas sean capaces de reconocer elementos por su imagen y de discernir que son, calcular a que distancia están y su lugar, y todo esto de una manera rápida. No obstante, realizar esta tarea presenta muchos retos.

El primer reto es el de cómo modelar los objetos para que la computadora pueda discernir los diferentes tipos de objetos para así diferenciar que es una persona o que es un coche, por ejemplo. No obstante, estos pueden ser vistos desde muchas posiciones diferentes. Aquí entra otro problema que es el de poder reconocer o diferenciar si una persona está sentada, de pie, detrás de un vehículo o agachada, así como si el coche esta de frente o de perfil. Además, como bien sabemos, a lo largo del día existen diferentes etapas lumínicas, no

solo el día y la noche, sino también puede estar nublado o soleado. Esto crea otro problema que es el de que un mismo objeto pueda estar en la misma posición que reconoce el computador pero que este no sea capaz de reconocerlo debido al cambio lumínico.

Otro reto es el que surge acerca de cómo procesar la imagen. Estos deben de ser rápidos y realizar búsquedas eficientes en la imagen, la cuál debe ser procesada de diferentes maneras para normalizarla y evitar algunos problemas anteriormente comentados.

Actualmente, uno de los campos más importantes en la Visión por Computador es la detección de personas [529]. Esto se debe al auge por crear sistemas inteligentes, en su gran mayoría para los coches, que detecten peatones para así intentar evitar o, al menos disminuir, la gravedad de un accidente. Incluso algunos tratan de reconocer el movimiento humano [530].

10.5.1 CONCEPTOS GENERALES

Toda parte de la imagen cuenta. Hay que trabajar con el color, el tamaño de la ventana que utilizaremos para analizar la imagen y la intensidad de la imagen. Por ello, una misma imagen se puede diferenciar en el color de la luz y la sensibilidad de la cámara. Cualquier variante de estos, hará la misma toma una imagen diferente, aunque se encuentren los mismos componentes en ella. También hay que tener en cuenta que el material de la superficie de los diferentes elementos afecta en la refracción de la luz, con lo que también afecta a las imágenes realizadas. Todo esto se debe a que las cámaras que utilizamos están basadas en el ojo humano pues poseen tres tipos de sensores, uno por cada cono [172] existente en el ojo humano. Los conos son fotoreceptores que se encuentran en la retina y que hay uno por cada longitud de onda: Conos-S (*Short*), Conos-M (*Middle*), Conos-L (*Long*) [529]. Así, los sensores equivalentes a estos conos son los sensores rojo, verde y azul (RGB).

A veces también se trabaja con otros tipos de imágenes como son las imágenes de infrarrojos cercanos (RGB-NIR), las cuales nos añaden un canal no visible en la longitud de onda más corta del espectro infrarrojo. En otros casos se pueden utilizar las imágenes térmicas, que reproducen la correlación entre la temperatura y la emisión infrarroja de los objetos. Otro tipo de imagen es el de las imágenes de profundidad (RGBD) que permite medir a los dispositivos, por ejemplo, un Kinect II, la profundidad de la escena respecto al sensor.

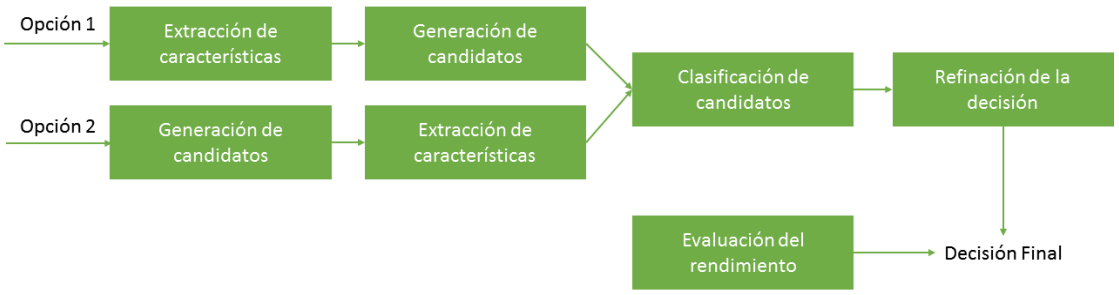


Ilustración 43 Diagrama de ciclo de creación de un módulo de Visión por Computador

[172] Glosario: **Cono**

El proceso de **Visión por Computador** se compone de varios pasos a realizar [500], [529], [531]. En la Ilustración 43 se muestra un posible diagrama correspondiente a estos procesos. El primer paso para crear un sistema de **Visión por Computador** es realizar la **extracción de características** para así construir los **descriptores** de los objetos de las imágenes. Un ejemplo de descriptor es el color del píxel, lo que permitiría definir el objeto como una región de píxeles conectados que tienen el mismo color. Sin embargo, los descriptores suelen contener decenas o cientos de características diferentes para ser más estables y mejores. Esto conlleva a problemas de rendimiento y lentitud en el escaneado de las imágenes.

Uno de los problemas que surgen es si hay cambios en la intensidad de las imágenes a procesar, pues, un mínimo cambio en ella, hará que el color sea diferente y no pueda reconocer ese objeto. Por ello, el primer paso debe de ser el de procesar las imágenes para eliminar los efectos de la intensidad y de cambios de color y crear **descriptores** utilizando un color invariante a la intensidad y a los efectos de color. Así se evita que una imagen tomada con diferente intensidad o con cambios de color debido a la luz o a la cámara sea diferente para el **clasificador** del sistema de **Visión por Computador**.

El segundo paso es el de la **generación de candidatos**. En este paso se definen las regiones de interés en la imagen para así resolver el problema de la localización del objeto dentro de ella. Para ello, se analizará la imagen en busca de los objetos definidos anteriormente y se recuadrarán con una ventana dentro del paso que se conoce como **generación de ventanas**. En esta generación existen múltiples formas acerca de cómo crear las ventanas y como aplicarlas, como por ejemplo la ventana deslizante (*sliding window*) y la pirámide con ventana deslizante (*pyramidal sliding window*). Esto ayuda a solventar uno de los problemas que se hay que solventar es el diferente tamaño que pueden tener los objetos y la profundidad a la que estos se pueden encontrar.

El siguiente paso es el de la **clasificación de candidatos**. En esta fase hay que estudiar y comprobar que el descriptor sea efectivo para las distintas instancias del objeto que deseamos detectar y ver que sea distinto de los objetos que no deseamos. Así, también hay que crear la frontera que definirá que es nuestro objeto y que no es el objeto que deseamos. Los problemas encontrados en esta fase van desde posibles varianzas en las tonalidades hasta la traslación, pues se puede dar que las ventanas generadas en la fase de **generación de ventanas** corten a los objetos, los trasladen o los corte y detecte la mitad de un objeto y la mitad de un segundo objeto como un objeto entero. Algunos de los algoritmos utilizados en esta fase de **clasificación de candidatos** son *Histogram of Oriented Gradients* (HOG) [531], *Local Binary Patterns* (LBP) [532]–[534], HOG-LBP [535], *Scale-Invariant Feature Transform* (SIFT) [536] y *Speeded-Up Robust Features* (SURF) [537] como descriptores, y la regresión logística [538], [539] y *Support Vector Machine* (SVM) [519] como métodos de clasificación. Además, uno de los métodos de aprendizaje utilizados es el de **aprendizaje supervisado**.

En la fase de **refinación de la decisión** hay que escoger las imágenes con mejor puntuación para así evitar duplicados debido al solapamiento de las ventanas creadas y quedarnos con la mejor identificación del objeto deseado.

En la fase de **evaluación de rendimiento** se ha de ver cómo funciona el clasificador y evaluar qué tan preciso es y los fallos que produce, es decir, si pueden obviarse o no. Una forma de hacerlo es utilizando una

matriz de confusión. Esta matriz contiene los reales positivos, falsos negativos, falsos positivos y reales negativos que identifico el módulo de **Visión por Computador**. Esto nos aporta la medición de la calidad del clasificador utilizando cuatro parámetros. El primero es su exactitud, que nos ayuda a saber la distancia que hay hasta la clasificación perfecta. El siguiente es su precisión, que nos indica la calidad de la respuesta del clasificador. El tercero es la sensibilidad que indica la eficiencia en la clasificación de todos los elementos que son de la clase que buscamos. Por último, la especificidad nos indica la eficiencia del clasificador en la clasificación de los elementos que no son de la clase para el cuál fue entrenado.

No obstante, la tarea de obtener un buen modelo para reconocer un objeto determinado es muy difícil. Se necesita utilizar muchas y muy buenas imágenes e intentar validar con otras imágenes diferentes que el modelo funciona correctamente. Además, se necesita crear un modelo para resolver un problema, pues si se crea un modelo genérico, este tal vez tenga una baja precisión, debido a la dificultad existente para reconocer imágenes. A todo esto, hay que sumarle los diferentes problemas existentes como son la calidad de las imágenes, las diferentes formas de entrenarlo y el tiempo que tarda en procesar una imagen.

10.5.2 **TRABAJO RELACIONADO AL RECONOCIMIENTO DE CARAS Y MOVIMIENTOS**

Durante la búsqueda de trabajo relacionado acerca de la combinación de **Internet de las Cosas** y **Visión por Computador** no se encontró nada. No obstante, sí que hay mucho trabajo relacionado que se enfoca en el reconocimiento de personas, caras y cabezas. Este trabajo está sobre todo orientado a los vehículos, para así prevenir atropellos o reducir el daño de estos, y a la salud, por medio de crear alarmas para avisar de forma rápida y eficaz cuando la gente mayor tenga una caída en casa.

Uno de los primeros trabajos acerca de los vehículos es el de [501]. Los autores de este artículo están investigando sistemas de conducción automático que puedan detectar peatones u otros vehículos y así poder evitar colisiones, ya sea, avisando al conductor o tomando control del vehículo antes de que ocurra el accidente. Este sistema se basa en el reconocimiento de objetos que se mueven y en el reconocimiento de peatones. Esto no es solo útil para **IIoT**, sino también en ***Smart Cities***, de cara a crear una ciudad más segura.

En cambio, en [500] presentan un *framework* para detectar caras de una forma extremadamente rápida mientras mantiene una buena tasa de acierto. Además, en este artículo presentan una forma nueva de representar imágenes que pueden ser computadas mucho más rápido, un nuevo clasificador basado en AdaBoost [540] y un nuevo método de combinación de clasificadores usando «cascada».

Otros autores se centran más a bajo nivel, como es el caso de la mejora del análisis de fotos RGB para ayudar a la clasificación de imágenes, estas mejoras son útiles en la clasificación de imágenes en general, a pesar de que ellos presentan su idea utilizando el reconocimiento de personas [529]. Exactamente, se centran en los colores que son opuestos, es decir, ellos tienen en cuenta las tuplas rojo-verde y amarillo-azul y la luminosidad.

Otras investigaciones utilizan Visión por Computador en la Robótica, de manera que permitan a los robots reconocer objetos o personas para poder realizar acciones en base a ello, como esquivar objetos, recoger el objeto deseado entre varios objetos reconocidos o reconocer a una persona. Por ejemplo, en [541] realizaron un *survey* del uso de Visión por Computador en robots cognitivos, que son aquellos que están diseñados

basándose en imitar la cognición humana, con el fin de que los robots puedan reconocer objetos, pero de tal manera que los robots no se saturen con la información de una imagen, pues aunque sea pequeña esta contiene mucha información. Por otro lado, en el «*survey*» de [542] tratan el estado del arte de la Visión por Computador en vehículos aéreos no tripulados (UAVs), pues con su uso se puede ampliar la autonomía mediante la mejora del control durante el vuelo y la percepción del ambiente que les rodea.

En lo relacionado a la salud tenemos el trabajo de los autores de [543]. Estos hicieron un estudio sobre diferentes sistemas utilizados para la detección de caídas para así ayudar a la gente anciana. Este artículo recoge diferentes sistemas de dispositivos «*wearables*», dispositivos que capten datos del ambiente y sistemas basados en visión. Según los autores de este «*survey*», los sistemas basados en visión tienen a ser mejores que las otras aproximaciones. Así mismo, los autores clasifican los sistemas de visión son de tres tipos: espacio-temporal, la inactividad o el cambio de forma y el movimiento y posición 3D de la cabeza. De entre todos estos está la propuesta de [544], que propone un sistema de detección de caídas basado en la combinación de imágenes en movimiento con su vector propio, de manera que tiene en cuenta datos espacio-temporales del movimiento y su ocurrencia. En cambio, los autores de [545] utilizaron un método de detección a partir del reconocimientos de los píxeles del fondo y por medio de diferentes algoritmos distinguir una caída de otros movimientos normales. Los autores de [546] presentaron un sistema por el que reconocían a la persona y la rodeaban con un óvalo para así basar su sistema de reconocimiento de caídas en base al cambio de forma del óvalo y comparando este con los óvalos previos por medio de los histogramas que estos crearon. Como ejemplo del tercer tipo de Visión por Computador aplicado a las caídas, este se corresponde con la posición 3D de la cabeza, que se describe en este trabajo [547], y en el que se presenta un sistema que combina las cámaras 3D con modelos de la cabeza de la persona obtenidos en momentos en los que está inactivo y así prever en conjunto con las diferentes características de los movimientos, como la localización y duración del movimiento, si es una caída o no.

10.6 *LÓGICA DIFUSA*

El término ***Fuzzy Logic***, **Lógica Difusa**, fue introducido en 1965 por Lotfi Asker Zadeh [548], profesor de la universidad de Berkeley, como una forma de lidiar con los problemas del sentido común. La **Lógica Difusa** surgió para resolver problemas que la lógica clásica no podía resolver, debido a que esta última solo puede tratar con valores binarios, es decir, ceros o unos, mientras que algunos problemas requieren tratar con más valores, pues estos problemas utilizan expresiones que no son totalmente verdaderas o falsas. Esto se puede ver ilustrado en la Ilustración 44. Así, al tener más de dos valores, se puede tener más estados, con lo que, se pueden tomar decisiones con más información y permitir que pueden ser representadas por un valor intermedio que se encuentre entre la verdad absoluta y la falsedad total. Normalmente se conoce a estos estados como **variables lingüísticas**, las cuáles pueden representar características de tipo «tamaño», la que en este caso podría tener tres posibles valores o incluso más del estilo «grande», «mediano» o «pequeño».

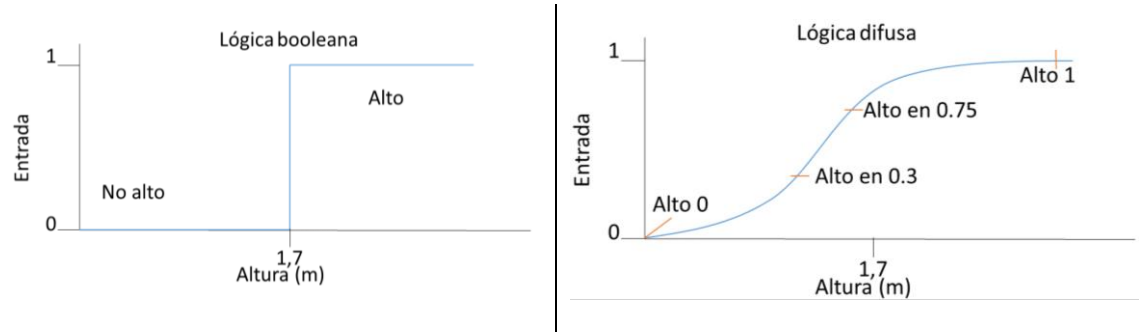


Ilustración 44 Comparación entre un sistema de lógica booleana y un sistema de Lógica Difusa

Así, este acercamiento trata de resolver preguntas del tipo: ¿cómo de grande es este edificio? La respuesta a este tipo de preguntas depende de la cognición individual, pues no todo el mundo respondería lo mismo, pues, no todos tenemos la misma opinión ni percepción y esto desemboca en que, para lo que unos algo es pequeño, para otros tal vez es mediano, o incluso grande. Un ejemplo de esta percepción podría ser el sitio del que proviene cada persona, pues, si una primera persona vive en una ciudad llena de rascacielos, como puede ser Nueva York, posiblemente opine que un edificio de 8 plantas sea pequeño, pero en el caso de que la segunda persona viva en un pueblo de casas, opinará que ese edificio es grande.

10.6.1 CONCEPTOS GENERALES

Lo primero que surgió en el campo de la **Lógica Difusa** fueron los **conjuntos difusos** en [549], presentados por Zadeh en 1965. Los **conjuntos difusos** son conjuntos con límites «no muy bien definidos» que tienen una escala entre 0 y 1 y sirven para medir el **grado de pertenencia**, conocido también como **grado de verdad** [496], de un determinado elemento en ese conjunto, pues, en la **Lógica Difusa**, un elemento puede pertenecer un determinado porcentaje a un conjunto y otro porcentaje a otro. Por ello, estos sirven para representar matemáticamente la imprecisión intrínseca de ciertas categorías de objetos y con ello modelar la representación humana de los conocimientos y mejorar así los sistemas de decisión y de la Inteligencia Artificial. La Ilustración 45 enseña un ejemplo de la definición de dos posibles conjuntos difusos, el de la izquierda de la temperatura y el de la derecha para la humedad.

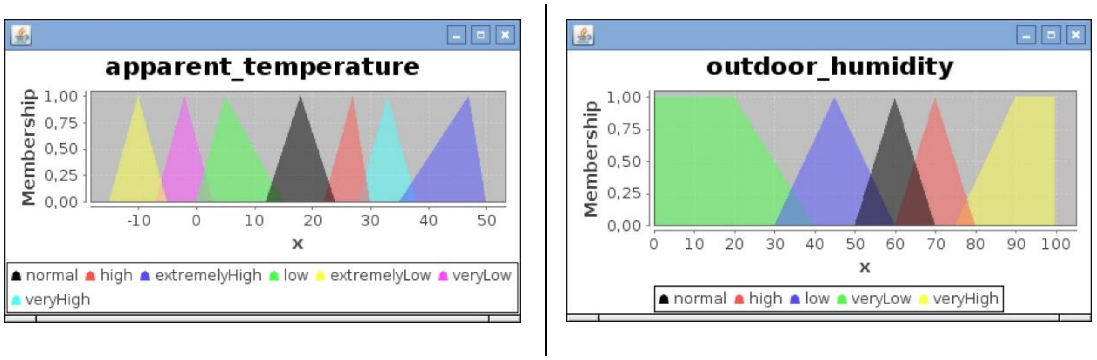


Ilustración 45 Conjuntos difusos definidos para representar la temperatura y la humedad

Más tarde ya se definió y se presentó la **Lógica Difusa** en [548]. Desde entonces, la **Lógica Difusa** ha estado ligada en muchas investigaciones. Un problema común donde la **Lógica Difusa** es aplicada es en el ahorro energético. No obstante, ejemplos de **Lógica Difusa** hay muchos más. La razón de esto se basa en que

todos estos necesitan más de dos valores para representar los estados. En el caso de Grant [550], este utilizó la **Lógica Difusa** para proponer un control diabético. De esta manera, conseguía permitir tratar con datos difusos y ambiguos y así tomar decisiones como una persona lo haría.

Para tomar las decisiones, la **Lógica Difusa** usa unos controladores llamados «***adaptative controllers***» [550] o «***expert systems***» [496]. Estos controladores se basan en reglas del tipo «Si X e Y entonces Z (*If X and Y then Z*)» para imitar el pensamiento difuso humano. Estas reglas representan el conocimiento que guía a estos controladores o sistemas para poder tomar la decisión óptima. La definición de estas reglas es compleja y requiere el uso de **variables lingüísticas** debido a que la representación humana del conocimiento es difusa. Al proceso de tomar una **variable lingüística**, por ejemplo, en el caso anterior de la entrada «tamaño», y procesarla por la función para decir que es la **variable lingüística** con el valor «pequeño», se conoce como ***fuzzification*** [551]. Así, una **variable lingüística** es el asignar a un valor numérico una palabra o sentencia del lenguaje natural.

En la Ilustración 46 se muestra el ciclo de vida de unos datos en un sistema de Lógica Difusa. En esta imagen se puede ver la entrada de datos, con dos valores, y 15 respectivamente. Estos datos se «*fuzzifican*» para transformarlos mediante las **funciones de pertenencia** [552], conocidas en inglés como ***membership functions*** [549]. La diferencia entre la lógica booleana y la Lógica Difusa en este punto es que, mientras la lógica booleana un valor solo puede ser 0 o 1 y pertenecer a un conjunto, en la Lógica Difusa este valor puede pertenecer a varios conjuntos ya que el valor puede ser cualquiera entre 0 y 1, siendo un caso que el valor puede pertenecer en 0,3 a un conjunto y en 0,7 a otro [549]. Las **funciones de pertenencia** se ejecutan tanto en la «*fuzzification*» como en la «*desfuzzification*». De esta manera, mediante las **funciones de pertenencia** se pueden establecer a que o cuáles **conjuntos difusos** pertenece un valor. Las **funciones de pertenencia** pueden tener diferentes formas, como son las triangulares, trapezoidales, funciones gamma, sigmoidales, gaussianas o pseudexponenciales.

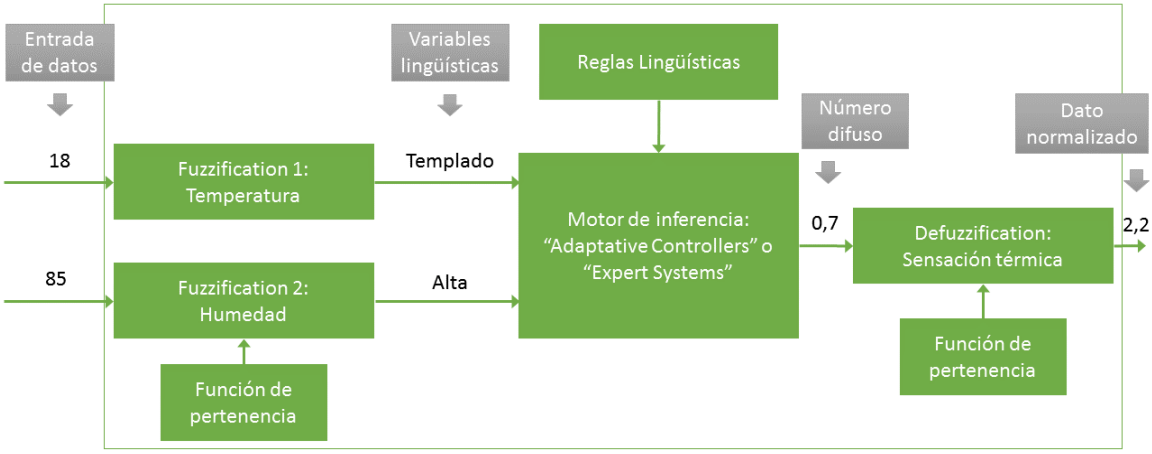


Ilustración 46 Diagrama de ciclo a través de un sistema de Lógica Difusa

Así, de los valores recibidos, que son 10 y 15, se obtienen las **variables lingüísticas** correspondientes, que en nuestro ejemplo son «pequeño» y «grande». El siguiente paso es aplicar las **reglas lingüísticas**, de tipo *IF-THEN*, las cuales se encuentran en el controlador llamado «**expert system**», a estas variables para

obtener el **número difuso** o **variable difusa**. Estas **reglas lingüísticas** son las encargadas de obtener el **número difuso** a partir de la combinación de las **variables lingüísticas**. Se definen a mano y aplicando un poco de sentido común [552], lo que implica que posiblemente haya que hacer un sistema de prueba-error hasta dar con las reglas que mejor se adapten a nuestro sistema. Por el contrario, si el sistema de **Lógica Difusa** tuviese un mecanismo de entrenamiento con el que ir aprendiendo y evolucionando en el tiempo, se diría que es un **sistema de Lógica Difusa adaptativo** [553].

Los **números difusos** tienen un valor entre 0 y 1 y son el resultado de juntar varios valores. El **número difuso** obtenido se pasa por el «*desfuzzificador*», fase en la que se elige uno de entre los muchos métodos existentes para realizar esta operación para así transformar el **número difuso** en el dato normalizado que utilizaremos para tomar la decisión.

10.6.2 TRABAJO RELACIONADO EN EL MARCO DE INTERNET DE LAS COSAS

A continuación, se muestran algunos ejemplos de uso de **Lógica Difusa**. Larios et al. en [554] localizan un dispositivo evitando muchos errores de localización y, por consiguiente, logran una mejor precisión. En este caso, los autores también consiguieron un ahorro en el consumo energético de diferentes elementos como el Sistema de Posicionamiento Global, más conocido por sus siglas del inglés *Global Positioning System* (GPS). Por otro lado, Chamodrakas y Martakos en [555] propusieron usar métodos de representación de **conjuntos difusos** para seleccionar una red y conectarse a ella de una manera eficiente, con un bajo consumo de energía, una buena calidad del servicio (*Quality of Service* [QoS]) y con un buen rendimiento.

En cambio, en la propuesta de Bagchi [556], se usó la **Lógica Difusa** para mantener la calidad de reproducción del *streaming* y lograr mejoras en el consumo de energía. No obstante, también hay propuestas para la mejora de los sistemas de los vehículos, como son los casos de [557], [558]. En la primera, los autores propusieron mejoras en el intercambio de información entre los vehículos y los servidores para así ahorrar energía. Mientras tanto que, en la segunda, los autores propusieron un sistema para crear aplicaciones a través del uso de voz. Todas estas investigaciones tienen en común el uso de **Lógica Difusa** para tomar decisiones difíciles sin tener opciones claras para considerarlas.

10.7 *PROCESAMIENTO DE LENGUAJE NATURAL*

El procesamiento de lenguaje natural o **NLP**, del inglés ***Natural Language Processing***, llamado a veces **lingüística computacional** [496], es una rama de la Inteligencia Artificial que trata de resolver el problema de entender el lenguaje natural utilizando máquinas, o dicho de otra manera, el permitir que los usuarios se comuniquen con las máquinas de una manera más rápida y efectiva utilizando lenguaje natural [559], [560], como si con otros humanos se estuvieran comunicando [561]. El **NLP** requiere entender la estructura de las sentencias, pero, además, este también requiere conocimientos de la materia sobre la que se habla y del contexto en el que se encuentra [496]. Con esto se busca realizar aplicaciones que puedan analizar, entender, alterar y/o generar lenguaje natural de acuerdo a ayudar con tareas que van desde la redacción utilizando computadoras hasta hacer estas más humanas por medio de los robots. Por esto, **NLP** es un campo que busca cerrar la brecha existente en la comunicación entre humanos y máquinas [562].

Sin embargo, **NLP** es un campo complicado, pues requiere de un corpus lingüístico, un modelo del dominio, conocimiento del dominio en que se quiere trabajar y conocimiento lingüístico del idioma en que se va a trabajar, además, claro está, de las herramientas software y hardware necesarias [563]. Esto implica que deban trabajar juntas diferentes disciplinas como son las ciencias de la computación para los métodos y la creación del software, la lingüística para crear los procesos y modelos lingüísticos, las matemáticas para identificar los modelos formales y los métodos, y la neurociencia para explorar los mecanismos del cerebro y otras actividades físicas [560].

El procesamiento de lenguaje natural existe desde los años 40 y desde entonces ha contribuido significativamente en los campos de interacción con el ordenador, tanto teórica como prácticamente [562]. Todo comenzó en el año 1946, cuando Warren Weaver y Andrew Donal Booth comentaron la viabilidad de la traducción máquina para romper los códigos enemigos en la segunda guerra mundial. No obstante, no fue hasta 1957 cuando se vio que resolver este problema podría ser mucho más complejo de lo que parecía [496]. Desde entonces, este campo mejoró desde usar traducciones basadas en diccionarios hasta lo que se conoce hoy como **NLP**. Un ejemplo es **NLP** en medicina, donde empezó a publicarse en el año 60 [563].

Cabe destacar que **NLP** es muy importante no solo en la rama de ciencias de la computación, sino para muchas otras. Por ejemplo, en la sanidad es muy importante debido a que hay mucha información relevante en texto que puede ser procesada por una máquina con el fin de ayudar a mejorar el cuidado del paciente y avanzar en la medicina [563]. Un ejemplo de esta importancia es la ocurrida en el año 2012, cuando la Biblioteca Nacional de Medicina de los Estados Unidos de América patrocinó un *workshop* los días 24 y 25 de abril para revisar el estado del arte actual, los desafíos, los obstáculos y el uso efectivo y las mejores prácticas de **NLP** centrado en textos en inglés, tanto en el lenguaje en general como en textos biomédicos [563].

10.7.1 ÁREAS DE APLICACIÓN

El **NLP** tiene muchas áreas de aplicación [559], [560], [562], [564]. Entre estas, algunas de las que se pueden desatacar son en el **reconocimiento y la generación del discurso**, **interfaces de lenguaje natural**, **gestión del discurso, entendimiento de la historia y generación de texto**, **traducción máquina** y los **asistentes de redacción inteligentes**.

En **el reconocimiento del discurso** la meta es conseguir convertir las palabras habladas por una persona a través de un micrófono para representarlas escritas en una máquina. Así, los sistemas de **entendimiento del discurso** tratan de mejorar el «entendimiento» de la máquina sobre lo que el usuario está diciendo mientras actúan y/o toman decisiones en base a ese contexto.

Por otro lado, **la generación del discurso** busca mejorar el proceso de habla de la máquina por medio de buscar la representación sonora ideal de las palabras escritas que debe de «decir» la máquina.

La investigación en **interfaces de lenguaje natural** intenta cerrar la brecha entre los diferentes lenguajes existentes de cara a internacionalizar correctamente las interfaces de usuario debido a las diferencias existentes entre los diferentes idiomas. Otro ejemplo dentro de esta categoría es la de facilitar el trabajo con las interfaces mediante comandos por voz.

Por otro lado, **NLP** también se utiliza para indexar y clasificar texto, realizar resúmenes, indexar en motores de búsqueda, *Data Mining*, realizar sistemas de diálogos pregunta-respuesta, extracción de contenido, entre otras aplicaciones similares. En estos casos, estamos hablando de **gestión del discurso**, **entendimiento de la historia** y **generación de texto**.

Un caso similar al anterior, donde el sistema debe de tener un mínimo conocimiento de lo que lee, es el de la **traducción máquina**, la aplicación más vieja de **NLP**. En este tipo de aplicaciones se trata de traducir un texto en un idioma dado a su equivalente en otro idioma requerido [496].

Otra área de **NLP** es la de los **asistentes de redacción inteligentes**. Estos incluyen varios tipos de aplicaciones como son los correctores ortográficos, agentes para dividir palabras, asistentes de formato/separación/selección de texto, tesauros automáticos y los entornos de creación y mantenimiento automatizado de documentos o clasificación de documentos [565], entre algunos ejemplos. Como se ve, todo este tipo de aplicaciones son aquellos que pretenden dar un apoyo inteligente cuando se trata con documentos.

10.7.2 MODELOS DE CONOCIMIENTO LINGÜÍSTICO

Para realizar **NLP**, se pueden aplicar diferentes **modelos de conocimiento lingüístico** [560], [562], [563], [565], [566]. Estos modelos pueden ser clasificados en cinco tipos diferentes: los **simbólicos o basados en el conocimiento**, los **estadísticos**, entre los que se encuentran los **estocásticos o probabilísticos** y los **conexionistas**, las **aproximaciones híbridas** e **Inteligencia Artificial**. Cada aproximación tiene sus ventajas e inconvenientes, lo que hace que cada uno de adapte mejor o peor según el tipo de problema de **NLP** que se desea resolver.

Los modelos **simbólicos o basados en el conocimiento** están basados en la representación explícita de los hechos sobre el lenguaje a través de esquemas de representación del conocimiento entendido, mediante el marcado del corpus lingüístico usado para entrenarlo, y los algoritmos asociados [562], [563]. Este tipo de técnicas son las más estudiados en **NLP**, sin embargo, tienen muchas deficientes en comparación con las técnicas estocásticas y conexionistas. Las técnicas simbólicas son especialmente usadas en casos en que el dominio lingüístico sea pequeño o esté muy bien definido [562].

Los modelos **estocásticos o probabilísticos** emplean varias técnicas probabilísticas para desarrollar modelos generalizados aproximados del fenómeno lingüístico, las cuales se basan en ejemplos reales de estos fenómenos. Este tipo de técnicas son muy efectivas en aquellos dominios donde fallan las técnicas simbólicas [562], [563], tratando de compensar la dificultas de creación de una gramática que se adecue a todo el idioma y que esté preparada para el razonamiento y la incertidumbre [563].

Los modelos **conexionistas** también utilizan ejemplos de fenómenos lingüísticos para desarrollar modelos generalizados. No obstante, al ser realizados desde arquitecturas conexionistas, estos modelos son menos restrictivos que los estocásticos, los cuales son más difíciles de desarrollar que los conexionistas. Al estar basados en los modelos estocásticos, las técnicas conexionistas se utilizan en los escenarios donde las técnicas simbólicas fallan [562].

Las **aproximaciones híbridas** utilizan la combinación de diferentes arquitecturas y de los anteriores modelos con el fin de encontrar la mejor aproximación. Un ejemplo de aproximación híbrida es la realizada por Miikkulainen en [567], donde combina técnicas simbólicas con técnicas conexionistas para procesar historias cortas y realizar resúmenes y posibles respuestas para preguntas. Las técnicas hibridas cogen las fortalezas de las técnicas simbólicas, estocásticas y conexionistas para tratar de minimizar el esfuerzo requerido para la construcción de modelos lingüísticos y maximizar su flexibilidad, efectividad y robustez [562], [563]. Este tipo de aproximaciones son las más utilizadas en los últimos años [563].

Los métodos basados en **Inteligencia Artificial** son muy variados. Estos pueden utilizar casi cualquier tipo de aprendizaje y de algoritmo visto en la sección 10.4.2 de esta tesis, como bien se ve en [565], donde pone ejemplos de trabajos que utilizaron aprendizaje supervisado o semisupervisado y algoritmos basados en árboles de decisión**,** *support vector machines*, redes neuronales artificiales, redes bayesianas o algoritmos genéticos, por nombrar algunos.

10.7.3 NIVELES DE CONOCIMIENTO INTERNOS DEL LENGUAJE NATURAL

NLP es un campo muy extenso y que necesita del análisis desde diferentes puntos de vista para llegar a procesar correctamente un discurso. Por ello, desde el punto de vista lingüístico, **NLP** tiene muchos aspectos que se deben de estudiar [568].

- **Fonética**: estudia el sonido del discurso humano desde el punto de vista de su producción y su percepción.
- **Fonología**: estudia la estructura del sonido y como este funciona.
- **Morfología**: estudia la estructura interna de las palabras para delimitarlas, definirlas y clasificarlas y la formación de nuevas palabras.
- **Sintaxis**: estudia la estructura de la sentencia por medio de las reglas y principios combinatorios de las palabras y las relaciones entre estas.
- **Semántica**: estudia el significado y la denotación de los símbolos, palabras, expresiones y representaciones formales.
- **Variación lingüística**: estudia los diferentes estilos de expresar el mismo significado y los dialectos de un idioma.
- **La evolución del lenguaje**: estudia el cómo emergió y evolucionó el lenguaje actual.
- **Pragmática**: estudia el uso y comunicación del lenguaje bajo un contexto determinado.
- **Psicolingüística**: es una rama de la psicología que estudia la producción y comprensión de lenguaje por los seres humanos para ver como lo adquieren y utilizan.
- **Adquisición lingüística**: estudia cómo es adquirido el lenguaje durante la infancia.
- **Neurolingüística**: estudia los mecanismos del cerebro que facilitan el conocimiento, la compresión y la adquisición del lenguaje, tanto escrito como hablado.

En base a esto, **NLP** se compone de diferentes niveles. Estos varían según los autores y la época, ya que algunos consideran cuatro [559] en 1993, otros seis [569] en 1995, otros autores ocho [560] en 1996, pero

que fueron actualizados a nueve posteriormente [562] en 1998, pero en general todos están de acuerdo con que se pueden subdividir principalmente en léxico, sintáctico, semántico y pragmático [562].

Cada uno de estos niveles ofrece un tipo de información de manera diferente sobre los mismos datos. Unos nos dicen el sonido de la palabra, otros nos dicen su significado, otros su género y número, otro nivel nos dice que significa dependiendo del contexto, entre otros muchos datos que podemos obtener dependiendo de la forma de analizar el texto. Así, las arquitecturas que añaden más niveles,consiguen obtener más datos y diferentes que las arquitecturas con menos niveles. Las arquitecturas con más niveles añaden, ente otras características, la fonética y el conocimiento del mundo, manteniendo los niveles de los anteriores.

A continuación, se explicarán todos los niveles encontrados en la literatura, enseñando entre paréntesis a qué tipo de clasificación pertenecen. Un ejemplo de los diferentes niveles se puede ver en Ilustración 47, que muestra un ejemplo de procesamiento del texto «El sabio dragón imita a un pato» que necesita de los niveles 4 a 6 debido a que se estudia su composición léxica, sintáctica y semántica. En el caso de que hubiera que reconocer la oración por medio de un micrófono cuando fuese dictada por alguien, entonces harían falta también los niveles 1 a 3. Si por el contrario se quisiera reconocer la situación en donde fue escrita esa oración con el fin de mantener una conversación, entonces harían falta también los niveles 7 a 9. Si, por el contrario, se quisiera mantener una conversación con alguien harían falta todos los niveles.

1. **Acústico o prosódico** (8, 9): estudia el ritmo y entonación del lenguaje y como formar sus fonemas.
2. **Fonético** (6, 8, 9): estudia la relación entre las palabras y el sonido que realizan y cómo funcionan, así como la formación de morfemas.
3. **Morfológico** (6, 8, 9): estudia como las palabras están construidas de sus morfemas, que son la unidad más pequeña de la lengua que tiene significado léxico o gramatical propio y no puede dividirse en unidades menores. Gracias a esto, se puede saber y crear el género y número de una palabra y así poder también formarlas. No obstante, algunos autores juntan el morfológico y el léxico [559].
4. **Léxico** (4, 8, 9): trata con las palabras, por medio de los lexemas y el significado de las palabras. En el caso de que no exista el nivel morfológico, se trata también en este nivel sus constituyentes como los morfemas y sus formas flexivas. Además, trabaja con la derivación de unidades de significado.
5. **Sintáctico** (4, 6, 8, 9): trata con la combinación de palabras que forman una sentencia. En este paso, cada palabra tiene una categoría diferente como son nombre, verbo y adjetivo, lo que hace que cada palabra tenga también unas reglas específicas de combinación en el lenguaje dado. También trabaja en como formar sentencias. En la Ilustración 47 se muestra un ejemplo de análisis sintáctico de una sentencia formada por un sujeto (primer sintagma nominal) y un predicado (VP en inglés), que contiene el verbo y el segundo sintagma nominal (NP en inglés). Dentro de ambos sintagmas nominales se ve el análisis de cada palabra que están clasificadas en determinante (det.), adjetivo (adj.), nombre y preposición (prep.).

6. **Semántico** (4, 6, 8, 9): trata el significado individual de las palabras, el significado de estas según la sentencia y el significado de toda la sentencia. Así, estudia el significado de la sentencia independiente del contexto. Por esto, trabaja en como derivar significado de las oraciones.
7. **Discurso** (8, 9): trata con los roles estructurales de la sentencia o colecciones de sentencias y en como formar diálogos.
8. **Pragmático** (4, 6, 8, 9): trata con la monitorización del contexto y la interpretación de las sentencias dentro de este contexto. Esto afecta a las sentencias posteriores pues influye en el significado de la interpretación de los pronombres.
9. **Conocimiento del mundo** (6, 9): este incluye la información acerca de la estructura del mundo de los usuarios del lenguaje, incluyendo así las creencias y metas del otro usuario de acuerdo a mantener, por ejemplo, una conversación.

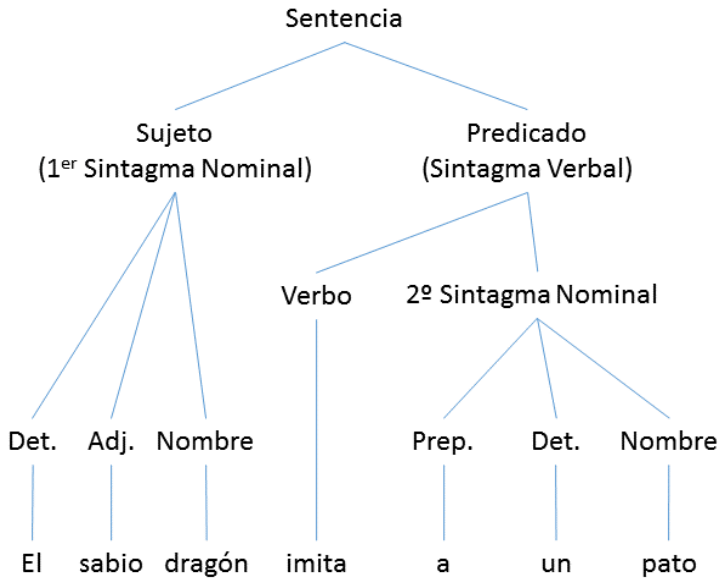


Ilustración 47 Análisis sintáctico de NLP

Hay que tener en cuenta que los diferentes niveles de abstracción necesarios de entre todos estos, o incluso alguno nuevo, dependerán totalmente del dominio de la aplicación que se quiera desarrollar o el problema que se quiera resolver [562], pues puede ser que no se necesiten todos ellos o solo se necesiten unos pocos. Por ejemplo, un traductor de texto necesitará, posiblemente, solo los niveles 3-7 o 3-8, para una aplicación de reconocimiento del discurso se usarán los niveles 1-5, para un asistente de redacción inteligente posiblemente se necesitarán los niveles 3-8 o 3-9 y un sistema de dictado necesitará los niveles 1-5.

10.7.4 TRABAJO RELACIONADO

En lo referente a trabajos relacionados en el campo de **procesamiento de lenguaje natural**, hay una variedad muy grande de estos. Como bien muestra el «*survey*» del congreso del año 2012 [563], ha existido mucha investigación con **NLP** en el campo de la medicina para así mejorar el servicio que se le da a los pacientes y ayudar a los doctores en su tarea.

Un primer ejemplo de **NLP** en medicina es el trabajo de [570], donde utilizan **NLP** para detectar problemas de malos usos de prescripciones de opioides con el fin de prevenir y controlar las medidas de estos, pues se detectaron problemas en clínicas tradicionales que contienen documentos o información incompleta o bien en casos en el que la información del paciente está oscurecida debido a la gran cantidad de documentos de las clínicas. Para solucionar esto, propusieron un método que combinase **NLP** y ayuda manual por computadora para revisar las notas y detectar problemas del mal uso de opiáceos.

Otros usos del **NLP** son aquellos aplicados a la lingüística para ayudar a traducir o crear herramientas para procesar idiomas determinados. En este primer artículo [571], los autores desarrollaron un *framework* para promover la colaboración entre lingüistas y no lingüistas en el uso de aplicaciones **NLP** para el lenguaje árabe. Por otro lado, en [561] presentaron una simulación del *parseador* para *Bahasa Indonesia*, que es la lengua de Indonesia, debido a la falta de investigación en esta área para este idioma.

NLP también se puede utilizar para dar órdenes, como hacen en este artículo [572] donde es utilizado para manejar robots. Los autores trataron de crear una interfaz humano-máquina con lenguaje natural debido a que estas tienen un mayor impacto en los usuarios, sobre todo cuando estos no tienen entrenamiento. Por ello, presentaron una interfaz que recibía órdenes en lenguaje natural y las transformaba en órdenes para el robot.

Un ejemplo similar al de los robots, es el de este artículo [573], donde los autores utilizaron **NLP** para crear sentencias SQL. Con esto, los autores buscaron acercar SQL a los usuarios no profesionales, de forma que ellos pudiesen escribir una sentencia en lenguaje natural que fuese transformada a la consulta correspondiente en SQL mediante el uso de un ***expert system*** realizado en Prolog que contiene la experiencia en ese dominio.

Actualmente, también hay investigaciones que tratan la recogida de información a partir de los requisitos textuales dados por un cliente. En el primer ejemplo tenemos la herramienta ***lips***, que integra **NLP** dentro del IDE Eclipse para traducir las especificaciones en texto plano a modelos formales de Ecore, UML o SysML, y que así pueden ser utilizados en el EMF para crear su implementación en Java [574]. Por otro lado y bastante parecido, está esta segunda investigación [575], en la que crearon un prototipo llamado ***Requirements Analysis and Class Diagram Extraction*** (RACE), el cual facilita el análisis de requisitos y la extracción de clases a partir de estos por medio de **NLP** y ontologías, utilizando la herramienta Apache OpenNLP, y así ayudar a los estudiantes a ver todos los pasos desde los requisitos hasta la creación de las clases. No obstante, RACE no soporta las relaciones entre clases. Como tercer ejemplo de este tipo de investigaciones está [576]. En esta investigación, los autores propusieron una forma de obtener el nombre de las clases y sus detalles a partir de los requisitos textuales entregados por un cliente, eliminando así las ambigüedades y posibles interpretaciones que estos documentos tienen.

11. **BIG DATA**

«Explotar los nuevos vastos flujos de información puede mejorar radicalmente el rendimiento de tu compañía. Pero primero tendrás que cambiar la cultura de la toma de decisiones»
«No puedes gestionar aquello que no mides»
Andrew McAfee y Erik Brynjolfsson [577]

«No tenemos mejores algoritmos [que antes]. Solo tenemos más datos»
Peter Norvig [577]

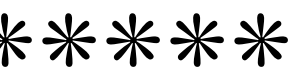
«Perlas en el polvo» [Hablando sobre encontrar los datos valiosos en los conjuntos de datos]
Tim Menzies y Ying Hu [503]

«Las frutas del conocimiento crecen en el árbol de los datos y no son fáciles de recoger. Para llegar a ella, necesitamos trepar una escalera de múltiples pasos»
William J. Frawley, Gregory Piatetsky-Shapiro y Christopher J. Matheus [578]

La historia hindú de los ciegos y el elefante relata como un grupo de hombres ciegos toca a un elefante por diferentes partes de su cuerpo. Estos, usando solo su sentido del tacto, no eran capaces de acertar, lo que desembocaba en que cada persona dijese que lo que estaba tocando era muy diferente de lo que tocaba otra. Esto se debía a la falta de información, de comunicación y de perspectiva, pues no podían ver ni hablar, solo sentir, y cada uno tocaba una parte diferente del elefante, lo que hacía que cada uno hablará de esa característica del elefante, cuando el elefante era la combinación de todas esas características.

Esto mismo ocurre con *Big Data*, pues existen muchas interpretaciones y desacuerdos sobre qué es y para qué sirve. Lo mismo ocurre con las herramientas existentes de este ecosistema, ya que proporcionan usos muy variados. Por esto mismo, se necesita una revisión de todo lo existente con el fin de recopilar y tratar de dar una posible solución a las discrepancias existentes.

Por estos motivos, en este capítulo se introducirá a qué es *Big Data* y por qué es importante y a las diferencias entre análisis de datos, minería de datos, KDD y *Big Data*. También se hablará de las características de *Big Data* tras la recopilación de diferentes artículos que las mencionan, pero que no se ponen de acuerdo, así como de diferentes arquitecturas y tecnologías que han sido relevantes. Para finalizar, se ha realizado un estudio acerca de los retos y obstáculos existentes en *Big Data* y el trabajo relacionado, las líneas de investigación y las aplicaciones existentes actualmente, tratando en todo momento de hablar de las más relevantes.



11.1 *INTRODUCCIÓN A BIG DATA*

En los últimos años ciertas cosas han estado cambiando en la **gestión del conocimiento**. Ahora más que nunca, existen más datos leíbles, obtenibles y útiles para la gente, para la empresa. Las empresas están recogiendo más datos de lo que saben hacer con ellos [577]. Por ello, se hace necesario nuevo conocimiento y nuevas tecnologías que sean capaces de recopilar, guardar, procesar y mostrar tan ingente cantidad de datos y sacar beneficio de ellos [577]. Dicho de otro modo, gracias a estas grandes cantidades de datos las empresas pueden medir y saber más acerca de sus negocios y así utilizar este conocimiento en la toma de decisiones y en mejorar el rendimiento de la empresa, su productividad, su competitividad, sus procesos y su innovación [577], [579]–[581]. No obstante, para lograr este aspecto, primero hace falta que las empresas hagan un cambio en su cultura de toma de decisiones [577]. Estas tienen que actualizar su infraestructura tecnológica de manera que puedan manejar estos nuevos datos y conseguir integrarlos a su infraestructura para analizarlos [579]. Estos datos se diferencian de los tradicionales en que tienen unas propiedades que se conocen como Las «6Vs» de Big Data, que son el volumen, velocidad, variedad, veracidad, variabilidad y valor. Las «6Vs» características de *Big Data* se explicarán en el subcapítulo 11.5, mientras que en el subcapítulo 11.6 se explicará el teorema HACE.

Uno de los problemas de la mayoría de los negocios modernos del siglo XXI es ignorar datos relevantes debido a que la gran mayoría de estos negocios pueden acceder electrónicamente a montañas de datos, como pueden ser las transacciones realizadas, el estado de la línea de ensamblado o datos sobre sus clientes [503]. Así, la accesibilidad y abundancia de esta información ha hecho que la **minería de datos** sea considerada un asunto de importancia y necesidad [582].

En 1992, W. J. Frawley, G. Piatetsky-Shapiro y C. J. Matheus estimaron que la cantidad de datos almacenados en las bases de datos existentes se doblaría cada veinte meses, o incluso más rápido, teniendo en cuenta que se incrementaría tanto el tamaño como en número de bases de datos [578]. Por otro lado, la actualidad ha sido testigo de la inundación de datos de diferentes medios y formatos, la cual consiguió superar la capacidad de procesamiento, análisis, almacenaje y entendimiento de estos conjuntos de datos [583]. Algo que se lleva usando desde hace mucho tiempo para facilitar el almacenaje y acceso de los datos son los *Data Warehouse* y los *Data Marts*, los cuáles se explicaran en el subcapítulo 11.3.

Por esto, una duda que surge es la siguiente: ¿es tan grande la cantidad de datos existente entre antes y ahora? Actualmente, el mundo está experimentando una revolución de datos, o dicho de otro modo, una inundación de datos, pues un gran montón de datos fluyen por la red pertenecientes a diferentes fuentes, utilizando diferentes canales, a cada minuto, debido a que estamos en la era digital [584]. Hay tantos datos en el mundo que es agobiante la situación, pues la cantidad de datos no deja de crecer continuamente y no hay un final a la vista [502]. Hablamos de millones de sensores repartidos por el mundo, de decenas de redes sociales en las que se publican millones de mensajes al día y de centenares de formatos diferentes que van desde texto plano hasta imágenes. La humanidad crea cada día cerca de 2.5 quintillones de bytes de datos, lo que supone que el 90% de los datos existentes se han creado en los últimos dos años [585]. Esto implica que sean necesarias nuevas herramientas en la gestión del conocimiento. Herramientas que sean capaces de trabajar con tantos y tan diversos datos de una manera rápida, sencilla y eficaz.

Pero, ¿qué está ocurriendo? Muy sencillo. Por un lado, las empresas están tratando de analizar todos estos datos en busca de tendencias que les ayuden a ver qué piensa sobre la empresa o que necesita o que quiere el cliente. De casos de éxito está Internet lleno. Ejemplos de esto son los mostrados por Hadoop en su página web oficial [586], pues en ella podemos encontrar a Facebook con su sistema de sugerencia de amigos y anuncios o LinkedIn con su sistema de sugerencia de contactos y empresas. Otras empresas, dedicadas a los anuncios, véase Google [587] o Double Click, quienes analizan todas las cookies recibidas para ver que sitios frecuenta el cliente y así mostrarle anuncios en función de sus hábitos de navegación. Casos similares son las empresas que se dedican a servicios de medición de audiencias como Nielsen y Gartner o los sistemas de recomendación de Strands y Amazon. Otras veces, como bien apunta el informe de McKinsey [588], ***Big Data*** fue utilizado por diferentes entidades para producir dinero, como es el caso de los servicios médicos de los Estados Unidos de América que obtuvieron 300 millardos de dólares, 250 millardos de dólares por la administración pública de la Unión Europea, incrementará un 60% el margen de operaciones de los minoristas y producirá un excedente de 600 millardos de dólares anuales a partir de la geolocalización de personas. Es decir, actualmente *Big Data* es algo muy importante, como se verá en el subcapítulo 11.2, en donde se explicará su importancia.

Por otro lado, está la **Inteligencia Artificial** (IA). ***Big Data*** necesita de técnicas de Machine Learning para tratar de forma correcta la inundación de datos existente y así analizar los datos para poder comprenderlos. Esto es algo que ya fue anunciado en el año 1990 [578] y que se ve hoy reflejado en el uso de las diferentes técnicas utilizadas [589]. Por otro lado, en los últimos años, muchas empresas como Yahoo! y Google están realizando grandes avances en IA gracias al uso de ***Big Data***. La IA requiere de grandes cantidades de datos para ser entrenada y que así pueda aprender, de forma que permitan crear modelos de buena calidad, ya sea para crear inteligencias artificiales como Cortana, Google Now o Siri, traductores de texto o modelos lingüísticos para mejorar los procesadores de lenguaje natural. ***Big Data*** necesita de técnicas de *Machine Learning* para tratar de forma correcta la inundación de datos existente y así analizarlos para poder comprenderlos, pero también provee de lo necesario para evolucionar y mejorar la Inteligencia Artificial actual. Se necesitan el uno al otro para seguir evolucionando.

¿De dónde provienen tantos datos? Esta cantidad ingente de datos proviene del boom de la mejora en las tecnologías y las investigaciones de algunas áreas de la ciencia que deben tratar con conjuntos de datos miles de veces más grandes en comparación con los que se generaban hace sólo una década, como son los procedentes de satélites, telescopios, instrumentos de alto rendimiento, redes de sensores, aceleradores y superordenadores [590]. Otras veces estas cantidades enormes de datos se deben a las nuevas tecnologías que han estado apareciendo, entre las cuales se presagia que las aplicaciones de Internet de las Cosas (**IoT**) supondrán un incremento sin precedentes en los datos existentes [583].

Internet de las Cosas engloba otras tecnologías como son los dispositivos inteligentes como los smartphones, La Nube (Cloud Computing), las Redes Sociales Online, los micro blogs, Open Data, los sensores, los microcontroladores como Arduino, *Internet Protocol* versión 6 (IPv6), RFID, NFC, Bluetooth y los códigos QR, sistemas incrustados, entre otras muchas. Además, en el caso de IoT, estos datos al final pasan o terminan en **la nube**, pues la nube es el centro de conexión intermedio entre los diferentes dispositivos heterogéneos y ubicuos que operan a través de Internet, es decir, de Internet de las Cosas [7], [9], [50]–[52].

Como bien se ha dicho, un asunto importante es mejorar la habilidad de gestionar, entender y actuar sobre esta cantidad ingente de datos de cara a incrementar la comprensión de las tecnologías necesarias para manipular y minar tales cantidades de información y así aplicar este conocimiento a otros campos como la sanidad, la educación, la energía o la defensa [591]. Hace décadas las compañías ya utilizaban la información de las bases de datos relacionales para realizar predicciones por medio del uso del **análisis de datos**, la **Minería de datos** y **Knowledge Discovery in Databases**, denominados muchas veces como las técnicas tradicionales, y que aportaron técnicas rigurosas para tomar decisiones [577]. Estas técnicas se explicarán y tratarán en el subcapítulo 11.4, para así explicar las diferencias y características de cada una.

Sin embargo, ahora se tienen en cuenta más sitios, como es Internet, los sensores, las aplicaciones, e incluso, las fotografías y los videos. Esto nos convierte a nosotros, los seres humanos, en generadores de datos andantes [577], pues somos quienes generamos de manera continua muchos de estos datos con el uso de los dispositivos móviles, entre otros. Debido a esta cantidad ingente de información creciente aparecida y las propiedades de estos conjuntos de datos, conocidas como Las «6Vs» de *Big Data*, la información ha dejado de ser manejable por las metodologías, técnicas y herramientas tradicionales [583], [592]. Esto dio lugar a lo que se conoce a día de hoy como ***Big Data***. ***Big Data*** es a la vez más simple, pero más potente que las técnicas tradicionales, pues como dijo el director de investigación de Google: «Nosotros no tenemos mejores algoritmos. Nosotros solo tenemos más datos» [577]. O, dicho de otro modo, las herramientas y técnicas ***Big Data*** permiten gestionar los diferentes conjuntos masivos de datos actuales y aplicar técnicas tradicionales a estos datos. Un requisito importante para conseguir esto es que el almacenamiento sea distribuido, escalable, elástico y tolerante a fallos [593]. Un ejemplo de ello es el sistema Google File System, de Google, y del que se habla en el subcapítulo 11.9.1 de este artículo.

Big Data es un término que está muy en boga. ***Big Data*** es una de las nuevas grandes fronteras en informática, incluso algunos la catalogan la cosa más grande que ha pasado en ciencias de la computación en años [591]. Los datos se están expandiendo tan rápido y la promesa de obtener una visión más profunda es tan convincente que los *IT Managers* están muy motivados para convertir ***Big Data*** en un activo que puedan gestionar y explotar para sus organizaciones. Sin embargo, hacen falta herramientas adecuadas para poder capturar y analizar este tipo de datos de manera fácil [579]. Algunas de estas herramientas y tecnologías emergentes son Hadoop y MapReduce, las cuales ofrecen una nueva forma de procesar y transformar la cantidad ingente de datos existente en algo con un significado relevante [594], entre otras muchas de las que se hablarán en el subcapítulo 11.11 y que ofrecen ayudas o formas alternativas a estos procesados.

El análisis de todos estos datos y el uso de estas herramientas se vieron beneficiadas gracias al decremento del coste del almacenamiento y el incremento en la potencia de computación. En el lado del hardware ha habido una mejora del almacenamiento de forma que ahora es más asequible que hace unas décadas, pues antes un terabyte podía costar millones, mientras que ahora cuesta 30 dólares. Por el lado del software, los avances en Inteligencia Artificial han hecho que se puedan utilizar algoritmos más inteligentes para procesar, comparar y visualizar los datos [595].

Estas mejoras han dado lugar a que nuevas compañías se hayan unido a la búsqueda de datos valiosos utilizando ***Big Data***. ***Big Data*** ha cambiado las ideas que se tienen acerca del valor, de la naturaleza de la

experiencia, pues ha significado una revolución en la gestión del conocimiento. Debido a esto, los encargados tienen que tomar decisiones en base a las evidencias, lo que obliga a las compañías a contratar científicos que puedan encontrar patrones entre estos datos y traducirlos a información empresarial útil, lo que implica que toda la empresa se redefina para basarse en este nuevo método [577]. Por ello, los líderes han visto obligados a hacer cambios importantes en sus negocios, pues ***Big Data*** requiere de una estructura diferente en la empresa que sea capaz de soportar el procesado distribuido y soportar una demanda en tiempo real o casi real de grandes cantidades de datos, que además requieren práctica en su manejo [577], [594]. Estos cambios son tantos debido al ciclo de vida de *Big Data*, que se muestra en el subcapítulo 11.7, como a las diferentes posibles arquitecturas que pueden existir, como se ve en el estudio del subcapítulo 11.8.

El reto más fundamental para las aplicaciones ***Big Data*** es explorar los grandes volúmenes de datos y extraer información que está «oculta» o conocimiento usable para acciones futuras [416], [596], lo que convierta a ***Big Data*** en uno de los campos con más futuro en los próximos años [583]. Sin embargo, el beneficio de analizar estos datos depende de su procedencia y de a quien pertenezcan, por lo que se necesita una gran capacidad para descubrir relaciones entre los datos y crear predicciones a partir de ellas [595].

Actualmente, ***Big Data*** se está expandiendo a todas las ciencias e ingenierías existentes [597] y a otras áreas de investigación como son la minería de datos, *Machine Learning*, la web semántica y las Redes Sociales Online [592]. Esto se debe a que ***Big Data*** puede ser aplicado a cualquier cosa: desde la meteorología con sus costes para ver sus tendencias e ineficiencias o comparar el GPS de las ambulancias con el registro de pacientes del hospital para determinar una correlación entre el tiempo de respuesta y de supervivencia, hasta ser usado para algo tan pequeño como puede ser la comparación de los movimiento, el sueño y las calorías gastadas de una persona a través de su dispositivo móvil [595]. Estos fueron algunos ejemplos de aplicaciones en las que se usa, pues a grandes rasgos, ***Big Data*** ha sido utilizado en comercio e inteligencia de mercado, en política y en gobierno, en tecnología y ciencia, en sanidad, en seguridad pública y contra el terrorismo, en educación, en crímenes y epidemias [581], [592]. Más trabajo relacionado, las líneas de investigación y las aplicaciones existentes más relevantes, será mostrado en el subcapítulo 11.13.

11.2 *¿POR QUÉ USAR BIG DATA? ¿ES TAN IMPORTANTE?*

Se puede leer que ***Big Data*** es importante, que es el futuro, que muchas empresas están utilizándolo y otras muchas van a comenzar a utilizarlo o que han montado un negocio en base al estudio de datos para venderle sus predicciones a terceros. La pregunta aquí es: ¿de verdad merece la pena?

Existe un amplio consenso entre los líderes comerciales, académicos, políticos y gubernamentales sobre el notable potencial que tiene ***Big Data*** para mejorar la innovación, mover el comercio, impulsar el progreso y mejorar la toma decisiones [596], [598]. Incluso fue considerada una de las diez tecnologías más de moda en 2013 por Gartner [71]. Posteriormente, en 2015, Gartner la incluyó dentro del Top 10 de las tecnologías y tendencias críticas con más impacto en las tecnologías de la información en los próximos 5 años [599], pues en 2015 se estimó que *Big Data* generaría un millón de empleos.

Big Data

Big Data tiene el potencial de transformar los negocios tradicionales obteniendo una ventaja competitiva siempre que entiendan sus datos [577], tanto en dinero como en supervivencia en el mercado, pues, sino se adaptan al mercado de forma que tomen buenas y rápidas decisiones, esto puede hacer que se queden atrás [595]. Esto es tal que se pueden dividir en dos las nuevas compañías que llegan a Silicon Valley: las que crean negocios utilizando ***Big Data*** y las que utilizan ***Big Data*** para obtener más ventas [577].

Algunos autores afirman que los análisis de datos con ***Big Data*** son mucho más poderosos que los análisis de datos utilizados en el pasado gracias a la medición y gestión de datos que se puede obtener actualmente y que nos otorga la posibilidad de predecir y tomar decisiones de una manera más inteligente [577]. Esto es algo que se puede ver reflejado en las compañías que lo utilizan, de entre las cuales algunas fueron comentadas previamente. Algo que permite ver el potencial de ***Big Data*** es su potencial para revolucionar la gestión, pues permite tomar decisiones en base a las evidencias en vez de por intuición [577]. Ejemplos de esto son compañías que nacieron digitalmente, como Google y Amazon, pues son especialistas en ***Big Data*** [577].

Sin embargo, nadie ha podido aún contestar de forma rigurosa si utilizar ***Big Data*** en una empresa mejora el rendimiento empresarial de esta [577]. Este tipo de uso de ***Big Data*** en aras de mejorar el rendimiento empresarial es conocido como dirigido por datos o *data-driven* [577], [600]. Hay estudios que todavía indican que los métodos tradicionales siguen siendo efectivos y que todavía hay un largo camino hasta que puedan superar a estos [601]. Un ejemplo de esto son ciertas herramientas, como *Google Flu Trends* (GFT), que se equivocó y preveía más del doble de casos que el Centro de Control y Enfermedades de los Estados Unidos, *Centers for Disease Control and Prevention* (CDC), cuando resulta que la meta de GFT era el de prever los informes del CDC [602].

En otras ocasiones, el uso de ***Big Data*** puede ser controvertido, como es el caso de patrones que utilicen características de la raza o la etnia. Este último caso se dio cuando una tabacalera hizo una campaña publicitaria para los hombres negros jóvenes debido a que, posiblemente, detectó que este grupo fumaba más que el resto [603].

Uno de los mejores ejemplos para decir o demostrar por qué es importante ***Big Data*** es haciendo referencia a la encuesta que Intel realizó a 200 profesionales en informática en el año 2012 [594]. Estos profesionales se encontraban en compañías estadounidenses con 1000 o más empleados, con al menos 100 servidores físicos que tuvieran 10 TB de datos almacenados, y debían trabajar en ***Big Data***. Esta encuesta preguntaba acerca de los cambios que ellos encaraban y las tecnologías que habían adoptado para capturar y analizar las nuevas fuentes de datos, así como su interés en el diseño de software e infraestructuras ***Big Data***. Como dato, la media de generación de datos de estas empresas semanalmente, rondaba por los 300 TB, incluso algunas superaban los 500 TB semanales. Muchos *IT Manager* consideraron los proyectos ***Big Data*** como los más importantes y fundamentales para su organización. De estos 200, el 25% consideró que ya habían implementado tecnologías ***Big Data***, mientras que un 20% se encontraban en el proceso de dicha implementación. En esta encuesta, la mitad de las personas encuestadas usaban procesos *batch*, mientras que la otra mitad usaban *real-time*, no obstante, se estimó que en el 2015 dos tercios de los encuestados utilizarían *real-time*. Sin embargo, los encuestados tenían miedo a utilizar y compartir sus datos entre su infraestructura Hadoop y sus socios externos debido a que estaban preocupados por la seguridad, afectando esto a servicios

de tipo analítico. La última premisa sacada de esta encuesta por Intel fue que los *IT Managers* dan mucho valor a la visualización de datos, los servicios que manejen datos de sensores y dispositivos, y al software construido junto a sus indicaciones de despliegue para *frameworks* como Hadoop.

Hablando de dinero, tenemos como primer ejemplo el del asesor científico del presidente de los Estados Unidos de América, John Paul Holdren, quien consideró en el año 2010 que su país estaba invirtiendo menos de lo que debería en ***Big Data*** [591].

Más datos interesantes de las inversiones realizadas en ***Big Data*** son las que se muestran en [591]. La Fundación Nacional de Ciencia (NSF) invirtió 35 millones de dólares. El Departamento de Energía (DOE) donó 5 millones de dólares al Laboratorio Nacional Lawrence Berkeley para avanzar esfuerzos en la gestión, análisis y visualización de datos. El Departamento de Defensa destinó 60 millones de dólares a premios enmarcados en ***Big Data***. El NSF y los Institutos Nacionales de la Salud (NIH) invirtieron en conjunto 25 millones de dólares en programas sobre ***Big Data***. El NSF destinó otros 10 millones de dólares para los premios de entre los años 2013 y 2018 de «Algoritmos, máquinas y gente de laboratorio» de la Universidad de Berkeley en California y que aborda los problemas relativos a ***Big Data***. La Agencia de Defensa de Proyectos de Investigación Avanzados de Estados Unidos de América anunció en el año 2012 que destinaba 24 millones de dólares al año en procesamiento y análisis en ***Big Data***. La Casa Blanca de los Estados Unidos de América lanzó en el año 2012 una iniciativa en el que anunciaba destinar más de 200 millones de dólares a mejorar las herramientas, técnicas, organización e investigación sobre grandes volúmenes de datos digitales [604].

No solo empresas y las inversiones de dinero pueden demostrar la importancia de ***Big Data***. Diferentes personas relacionadas con el mundo de la informática y de empresas relevantes han dado su visión personal sobre qué es y que aporta ***Big Data*** y que se espera de esta tecnología.

Una de ellas ha sido Susan Hauser, vicepresidenta corporativa y del grupo de socios de Microsoft [595]. Para ella, ***Big Data*** tiene el poder para impulsar una visión práctica que simplemente no eran posibles antes, tratando la gestión de todos los datos y proporcionando herramientas que permitan responder preguntas a cualquier persona que tal vez ni siquiera podría saber de antemano que querían preguntar, nos permitirá anunciar las tendencias antes de que estas ocurran, y nos ayudará en la toma de decisiones antes de que otros que no la utilicen estén deliberando acerca de si tomar o no esa decisión. Por esto mismo, ***Big Data*** tiene la capacidad de cambiar la forma de hacer negocios y descubrimientos de los gobiernos, organizaciones e instituciones académicas, así como la forma de vida de todo el mundo.

En palabras de Dan Vesset [595], vicepresidente del programa de investigación Business Analytics de la empresa especializada en investigación de mercados International Data Corporation (IDC): «Nuestras vidas diarias crean una colección de datos enormes». Ejemplos para corroborar esto son los datos que creamos al entrar en la web y navegar por ella, como los clics, las compras realizadas, el movimiento por la página, el contenido compartido, las interacciones con este, los sitios por los que nos movemos a diario, los mensajes y los emails que enviamos y los datos recogidos por otros dispositivos usados a diario como nuestros smartphones y los coches, entre otros muchos.

Por su parte, Eron Kelly [595], director general de marketing de Microsoft SQL Server, sostiene que cada vez el mundo generará más datos y estos datos servirán para que las personas los utilicen para tomar mejores decisiones de una forma más eficiente y así moverse más rápido hacia el futuro. Por esto, en sus palabras, ***Big Data*** es una gran oportunidad, pues es una herramienta que nos provee de recoger lo necesario de una forma fácil.

Como se ha visto, ha habido muchas subvenciones, sobre todo por parte de diferentes entidades gubernamentales estadounidenses entorno a ***Big Data***, empresas invirtiendo en esta nueva tecnología e importantes gestores de estas hablando de lo importante que es y será. ***Big Data*** ha sido catalogado como algo necesario para competir y obtener ventaja en cualquier campo conocido, pero requiere un cambio por parte de los gestores y las empresas, que se adapten. ***Big Data*** es el presente y cada vez se tendrá más en cuenta para el futuro, pues los datos nunca disminuirán debido a que cada vez seremos más personas generando datos al mismo tiempo que utilizaremos más dispositivos que generen más datos a su vez. La solución para poder gestionar todos estos datos y obtener conocimiento pasa y pasará por utilizar ***Big Data*.**

11.3 *DATA WAREHOUSE Y DATA MARTS*

Todos los datos han de ser recopilados y guardados en algún sitio para así poder analizarlos y consultarlos posteriormente cuando se necesite. Para esto hay diferentes sistemas, siendo el más conocido las bases de datos (BBDD). Sin embargo, muchas empresas han utilizado lo que se conoce como *Data Warehouse* y *Data Mart*. Ambos son sistemas de gestión de la información que ofrecen utilidades para recopilar, almacenar, analizar y consultar la información de forma más óptima. Así, la mayor diferencia con las BBDD es que estas solo almacenan información mientras que los *Data Warehouse* y *Data Mart* utilizan BBDD y proporcionan utilidades para obtener análisis de los datos poseídos.

Un almacén de datos [525], [526], [605], [606], conocido en inglés como ***Data Warehouse***, fue la solución general propuesta a principios de los años 90 para satisfacer el problema de gestión de la información. Un *Data Warehouse* es un repositorio apropiadamente diseñado e implementado, no volátil, para recolectar datos. Los *Data Warehouse* son actualizado de forma regular y permiten extraer y examinar la información perteneciente a una empresa, como sus productos, operaciones y los hábitos de compra de sus clientes. Estos almacenes de datos pueden reunir información en un formato determinado complementándolo con metadatos gracias al uso de diferentes mecanismos de entrada conocidas como herramientas de extracción, transformación y carga (ETL). Los ETLs permiten mover datos desde múltiples fuentes, reformatearlos, limpiarlos y cargarlos en otro sitio.

Un detalle de los almacenes de datos es que normalmente suelen estar separados de los sistemas de producción, siendo así añadidos los datos cada cierto tiempo o según el negocio los vaya necesitando [526]. Además, estos contienen metadatos, el modelo de datos, las reglas de agregación de datos, copias, manejo de excepciones, y otra información necesaria para mapear el almacén, como son sus entradas y salidas.

No obstante, el uso de almacenes de datos da lugar a que las empresas deban tener herramientas que les ayuden a extraer información útil de estos repositorios masivos de datos. Además, debido al incremento

constante de la cantidad de datos existentes, hace que se necesiten cada vez análisis más rápidos y potentes, así como una mejor infraestructura hardware [526].

Por otro lado, un ***Data Mart*** [606], [607], es una versión específica de un *Data Warehouse*, que contiene un subconjunto de datos, refinado y resumido, para tratar de ayudar en un área específica dentro del negocio, ya sea para poder tomar mejores decisiones u ofrecer mejores datos a los usuarios. Por esta razón, cuanto más grande son los *Data Warehouse* más atractivos son los *Data Mart*, a nivel organizativo, tecnológico y económico. Los *Data Mart* pueden ser reales o virtuales. Los *Data Mart* reales son sistemas que contienen su propia información traída del *Data Warehouse*, mientras que los virtuales son vistas definidas sobre el *Data Warehouse*.

Para preparar un almacén de datos para usarlo y desarrollar información a partir de él se necesitan tres componentes [526]. El primero son las **herramientas ETL**, que son usadas para obtener información de diversas fuentes y juntar toda esta información en una sola, con una estructura accesible, y cargarla en un *Data Mart* o almacén de datos (*Data Warehouse*). El segundo componente está formado por las herramientas de **Minería de datos**, las cuales proporcionan una gran diversidad de técnicas y estadísticas avanzadas para localizar patrones y desarrollar hipótesis. Por último, están las **herramientas analíticas** que se utilizan para analizar datos, determinar relaciones y testear hipótesis sobre los datos, permitiendo el desarrollo de procesos que puedan ser usados por personas ordinarias en sus puestos de trabajo y que no necesiten de conocimientos estadísticos avanzados y cumplan las doce reglas para herramientas de procesamiento analítico online (OLAP) propuestas por E. Codd, S. Codd y C. Salley en 1993 [608].

Como se ve, este proceso está basado o es similar al de las cadenas de suministro [607], pues primero se consiguen los datos, que son la materia prima y son almacenados en el *Data Warehouse*, después se envían a los *Data Mart* donde se procesan o se ofrecen a los clientes, que son los usuarios.

Un ejemplo de uso de los *Data Warehouse*, es el de utilizar toda su información para realizar los diferentes tipos de análisis necesarios y requeridos por la **minería de datos**, de tal manera que se puedan obtener patrones que pueden ser utilizados posteriormente [526]. O bien, el de permitir a los clientes acceder mediante herramientas especializadas y adaptadas para facilitar las consultar de datos [607].

Como bien se ha visto hasta ahora y teniendo en cuenta lo ya comentado previamente sobre por qué utilizar ***Big Data***, se ve que tanto los *Data Warehouse* como los *Data Mart* pueden ser una parte esencial e importante en la arquitectura de ***Big Data*** al igual que lo fueron en la minería de datos, haciendo mucho más fácil, rápido y óptimas las arquitecturas y el acceso a los datos.

11.4 *DATA MINING VERSUS KDD VERSUS BIG DATA*

Hay cierta controversia cuando se habla de **minería de datos**, conocido como *Data Mining*, y ***Big Data***. Además, si se mira atrás, hay otros términos que se incluyen cuando se habla de todo estos, como son el Descubrimiento de Conocimiento en Bases de Datos o ***Knowledge Discovery in Databases*** (**KDD**), y el de **análisis de datos**.

Por un lado tenemos el problema de confusión de la **minería de datos** con KDD, donde algunos autores lo consideran lo mismo [526], [562], [609], mientras que otros muestran claramente, en base a referencias de diferentes autores, que KDD es un proceso completo y la minería de datos un paso dentro del proceso de KDD [455], [525], [582], pero sin explicar las diferencias entre ambos y dando solo vagos matices.

Por otro lado tenemos a los autores para los que ***Big Data*** y la **minería de datos** son lo mismo o, incluso, algunos añaden también a esta confusión el término de **análisis de datos**. Sin embargo, sendos términos son diferentes y, a pesar de tener cosas en común y utilizarse a veces las mismas herramientas, su finalidad es diferente. Uno de los primeros ejemplos de confusión aquí viene dado por la existente entre el **análisis de datos** y la **minería de datos** [526].

Un ejemplo lo encontramos cuando se escucha decir a algunos empresarios que «***Big Data***» es simplemente otra forma de llamar a los análisis tradicionales de datos que se han realizado desde hace años. Esto es verdad a medias, pues ***Big Data*** trata de hacer «lo mismo», que es tratar de obtener inteligencia (patrones, tendencias, análisis) a partir del análisis de datos para conseguir una ventaja comercial. Empero, los conjuntos de datos con los que se trabaja en ***Big Data*** tienen una serie de características que se conocen como **Las «6Vs» de *Big Data*** [577].

Así pues, aunque digan que ambos son lo mismo o que la **minería de datos** actualmente se conoce como ***Big Data*** debido a la gran cantidad de datos que debe de manejar rápidamente [416], son verdades a medias. Otros, en cambio, dicen que ***Big Data*** es una minería de datos escalable [600], pues necesita cumplir las «Vs» [577]. Sin embargo, para esto hace falta el uso de herramientas específicas que soporten y den apoyo al tratamiento de grandes volúmenes de datos.

Como se ve, hay ciertas diferencias entre los términos, confusiones y equivocaciones. En los siguientes subapartados se definirán exactamente que son el **análisis de datos** y la **minería de datos**, así como su diferencia y se entrará un poco en detalle en los patrones de la minería de datos y se explicará la clasificación de sus métodos. En el siguiente apartado se explicará que es KDD y de que fases está compuesto. Para finalizar, en el último subapartado se discutirá acerca del término ***Big Data***, haciendo hincapié en si este nombre está bien escogido y se verán diferentes definiciones tanto de empresas como de académicos, de tal manera que se dará una definición exacta de qué es exactamente ***Big Data*.**

11.4.1 **MINERÍA DE DATOS**

La búsqueda de patrones dentro de datos es conocida por muy diferentes y diversos nombres, como son, por ejemplo, extracción de conocimiento, descubrimiento de información, recolección de información, procesamiento de patrones de datos y **minería de datos** [525], [610]. Este último término es el más utilizado por los estadísticos, los investigadores en bases de datos, los gestores de sistemas de la información y en los negocios [525], además de en informática, por lo que será el término que se utilice en esta tesis..

Este término fue acuñado por los estadísticos en los años 60 para referirse a la mala práctica, según su opinión y con cierta connotación negativa, de analizar datos sin una hipótesis inicial [610]. Esto coincide justo cuando se introdujeron los ordenadores para obtener patrones de datos a partir de una muestra de una población total [525], [605]. Ellos nombraron a este proceso como «*data dredging*» o «*data fishing*» [416], [525], [605], [610]. Mientras, el término utilizado actualmente, *Data Mining*, apareció en los años 90 en la comunidad de las bases de datos [610].

La **minería de datos**, conocida como *Data Mining*, es un subcampo de la ciencia de la computación que toca muchas partes de la informática, como son la gestión de bases de datos, la Inteligencia Artificial, el aprendizaje máquina, el reconocimiento de patrones y la visualización de datos [609].

Con la combinación de los conocimientos de todos estos campos que toca, la **minería de datos** trata de crear un análisis automatizado de grandes y complejos conjuntos de datos [609]. Estos conjuntos pueden ser desde bases de datos hasta mapas del cielo, información meteorológica, información de satélites, procesos de control industrial, y muchos más [609].

La **minería de datos** se basa en la extracción de datos previamente desconocidos, pero que pueden ser potencialmente útiles debido a la información que pueden aportar para tratar de obtener patrones, también conocidos como modelos [416], [525], y relaciones que tal vez se puedan generalizarse para hacer predicciones futuras o tomar decisiones importantes [502], [582], [609], [611]. Sin embargo, muchos de estos patrones obtenidos serán banales y no interesantes, otros falsos o poseer coincidencias accidentales dentro de un conjunto de datos utilizado. En el caso de que estos patrones tuvieran significado, pueden permitir hacer predicciones no triviales sobre nuevos datos [502]. Por ello, el fin de la **minería de datos** es buscar y analizar grandes cantidades de datos para para descubrir patrones o relaciones útiles que sirvan para predecir el futuro [526].

Las metas del uso de algoritmos de **minería de datos** son, según Lynn Greiner [526], las siguientes:

- Aplicar *clustering* para agrupar los datos en grupos de elementos similares.
- Búsqueda de un patrón explicativo o predictivo para un atributo destino en términos de otros atributos.
- Búsqueda de patrones y subpatrones frecuentes.
- Búsqueda de tendencias, desviaciones y correlaciones interesantes entre atributos.

Ejemplos de uso de **minería de datos** se encuentran por ejemplo en aplicaciones que intentan predecir cosas, como puede ser el tiempo o el cambio climático, que se basa en datos que describen hechos pasados

para crear un modelo que sirva para predecir posibles eventos futuros que vayan teniendo el mismo patrón. Otros ejemplos de su aplicación es en económicas [502], o para obtener información genética y adelantarse a las enfermedades y tratamientos existentes [114]. Todas las aplicaciones de **minería de datos** se centran en obtener interpretaciones de los datos, debido a que lo que buscan son tendencias y correlaciones en vez de probar una hipótesis [526].

11.4.1.1 ANÁLISIS DE DATOS VERSUS MINERÍA DE DATOS

A veces hay confusión entre los términos «**análisis de datos**» y «**minería de datos**» [526]. El **análisis de datos** analiza los datos existentes y aplica métodos estadísticos y de visualización para probar hipótesis acerca de los datos y así descubrir excepciones. Mientras, la **minería de datos** busca tendencias, dentro de los datos, que puedan ser utilizadas para el análisis posterior. Por esto, la **minería de datos**, es capaz de proporcionar nuevos conocimientos, que son los patrones, los cuales son totalmente independientes de las ideas preconcebidas, es decir, de las hipótesis.

Análisis de datos

El **análisis de datos** puede ser visto como la colección de métodos para dibujar inferencias a partir de los datos, extrayendo así información global que es generalmente la propiedad de los datos analizados [526].

Mientras, se puede definir la **minería de datos** de acuerdo con la definición dada por Lynn Greiner [526], y que fue utilizada por Microsoft cuando describió su arquitectura de ***Big Data*** para el *survey* de la *National Institute of Standards and Technology* (NIST) de los Estados Unidos de América [612]. Así, de acuerdo con estos la definición es:

Minería de datos

«La **minería de datos** es el proceso de extracción de datos y su correspondiente análisis posterior desde muchas dimensiones o perspectivas para producir un resumen de la información de una forma útil que identifique relaciones dentro de los datos. Hay dos tipos de minería de datos: el descriptivo, el cual nos da información sobre los datos existentes, y el predictivo, que hace pronósticos basándose en los datos».

11.4.1.2 PATRONES EN LA MINERÍA DE DATOS

Los patrones, también conocidos como modelos, que se obtienen al aplicar **minería de datos** se consiguen mediante el uso de diferentes técnicas de **aprendizaje máquina** [416], [502], [503], estadísticas y de Inteligencia Artificial que se aplican sobre los conjuntos de datos de ejemplos para reducir estos conjuntos a pequeños patrones entendibles [503].

Frecuentemente la base de la **minería de datos** son los árboles de decisión y las reglas de asociación [526]. No obstante, también se utilizan otros métodos de *Machine Learning*, como son, por ejemplo, las redes neuronales (ANN). Aún a pesar de que las ANN han sido aplicadas satisfactoriamente necesitan de mucho tiempo para la generación de patrones, los cuales a veces no son entendibles, razones por las que no suelen

ser utilizadas muy frecuentemente [526]. Otros tipos de algoritmos que se suelen utilizar y son muy importantes como técnicas de exploración en el **análisis de datos** son los algoritmos de *clustering*, debido a que de primeras no se tiene conocimiento acerca de las posibles distribuciones existentes en los datos [526].

Los patrones obtenidos pueden ser de dos tipos. Estos pueden ser bien cajas negras incomprensibles o bien cajas transparentes que muestren la estructura de dicho patrón, conocidos como patrones estructurales. No obstante, ambas posibilidades son igual de buenas. La diferencia de los patrones estructurales reside en que estos patrones extraídos están representados en una estructura que puede ser examinada, razonada y permiten utilizarlos para informar de decisiones futuras. De aquí viene su nombre, pues permiten explicar los datos en base a la estructura del patrón [502].

11.4.1.3 CLASIFICACIÓN DE LOS MÉTODOS DE LA MINERÍA DE DATOS

Según Lior Rokach y Oded Maimom, los métodos de la **minería de datos** se puede clasificar siguiendo la taxonomía mostrada en [582], que muestra que existen dos tipos posibles de uso de **minería de datos**. El primero es el de la *verificación*, que se utiliza para verificar una hipótesis. Debido a esta razón está menos asociado con la **minería de datos**, pues la **minería de datos** se centra en la selección de una hipótesis.

El segundo tipo es el del *descubrimiento*, que se utiliza para obtener nuevas reglas de patrones a partir de un conjunto de datos. Dentro de los métodos de descubrimiento existen dos ramas [526], [582], la de los métodos *descriptivos* que se centra en la compresión de cómo los datos operan para así crear reportes. La segunda rama es la de los métodos *predictivos*, que contiene aquellos métodos que tienen como objetivo construir un patrón de comportamiento para obtener nuevas muestras con las que predecir valores de una o más variables relacionadas en la muestra. A veces, los métodos *predictivos* también permiten ayudar a entender los datos [589]. En estos últimos se pueden diferenciar entre los métodos de *regresión* y los de *clasificación*.

En la *regresión* se aprende una función que clasifica un elemento en función de unas variables continuas para conseguir obtener una predicción o una previsión [605], [613]. Ejemplos de *regresión* son los estudios realizados para saber la tasa de mortalidad por tabaquismo, la cantidad de biomasa presente en un bosque, la probabilidad de supervivencia de un paciente tras realizar una serie de pruebas de diagnóstico o la predicción de demanda de un producto.

Mientras, los métodos de *clasificación* aprenden una función que clasifica un elemento en varias clases definidas [614]. Un uso muy relevante de este tipo de métodos es el utilizado en los últimos años en la clasificación de imágenes. En este método es donde se encuentran muchos tipos de algoritmos de Inteligencia Artificial como son las redes neuronales artificiales, las redes bayesianas, los árboles de decisión y los SVM. Todos estos se pueden ver más a fondo en el capítulo 10.4.2. En la Ilustración 48 se muestra el esquema propuesto por los mismo autores de esta clasificación.

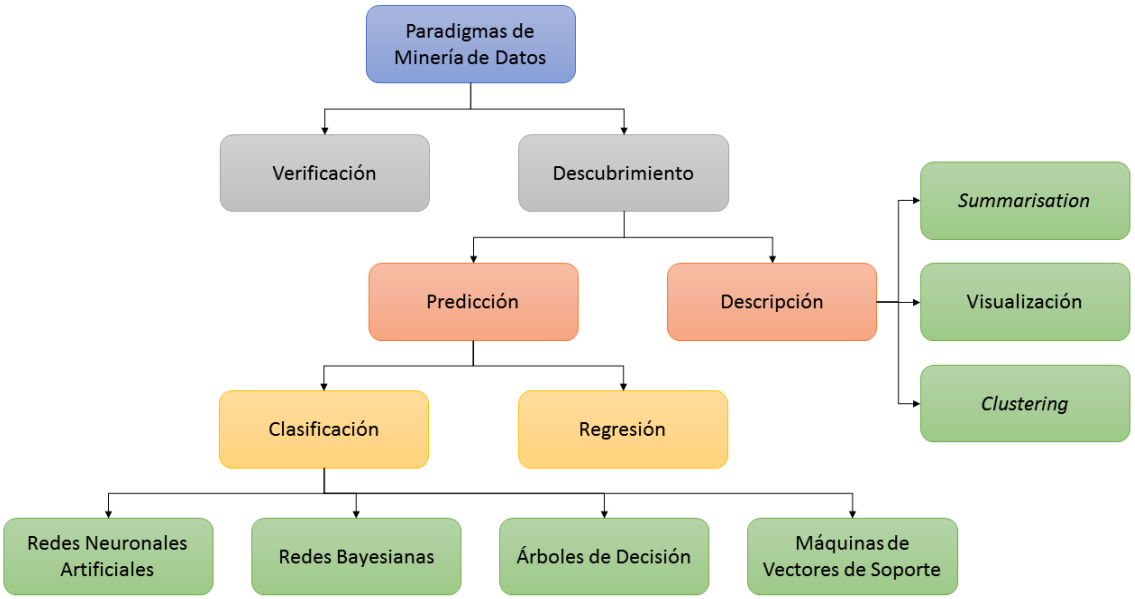


Ilustración 48 Taxonomía de los métodos utilizados en minería de datos basada en la propuesta de [582]

11.4.2 KNOWLEDGE DISCOVERY IN DATABASES

El término ***Knowledge Discovery in Databases*** (**KDD**) fue acuñado por Gregory Piatetsky-Shapiro en un *workshop* de 1989 [615], según explican en [605] y afirma el propio autor en [610]. Además, se confirmó tras haber realizado una comprobación realizando diversas búsquedas en los años anteriores en diferentes buscadores de revistas científicas y no ver la existencia de este término previamente. El fin de este término fue el de enfatizar que el conocimiento es el producto final del descubrimiento dirigido por datos [605].

El Descubrimiento de Conocimiento en Bases de Datos o **KDD** es el proceso automático de identificación válido, novedoso y potencialmente usable de entendimiento de patrones en grandes bases de datos [589], [605]. Este proceso debe de ser automático debido a la gran cantidad de datos que pueden ser recopilados y que los humanos ya apenas pueden digerir pero que las máquinas pueden facilitar. Esta es la finalidad de KDD: tratar de abordar el problema de la sobrecarga de datos [605].

Al igual que ocurre con la **minería de datos**, KDD es la interacción de diferentes campos, entre los cuales están incluidos las bases de datos, *Machine Learning*, el reconocimiento de patrones, estadística, Inteligencia Artificial, sistemas expertos, visualización de datos y la computación de alto rendimiento [525], [605].

KDD ha sido utilizado en muy diversas aplicaciones en diferentes tipos de bases de datos [578], [605], [616]. Estas son, por ejemplo, en medicina para el descubrimiento de efectos secundarios de las drogas o análisis de secuencias genéticas, en finanzas para la predicción de bancarrotas y de stock, en agricultura para la clasificación de enfermedades, en el ámbito social para la predicción de voto, en *marketing* para la identificación de subgrupos y los patrones de compra, en compañías aseguradoras para la detección de fraudes, en ingeniería para crear sistemas expertos de diagnósticos de automoción, para estimar trabajo, en el telescopio Hubble, para realizar una catalogación del cielo, en la búsqueda de volcanes en Venus, en bases de

datos con biosecuencias [173], para la geofísica de la Tierra en la inferencia de terremotos y en el análisis de datos atmosféricos, entre otros muchos ejemplos.

11.4.2.1 EL TÉRMINO KDD

Como bien se comentó previamente, hay confusión entre que es KDD y que es la **minería de datos**, pues hay personas que dicen que son lo mismo. Por eso, a continuación, se recogen diferentes definiciones para decir exactamente que es KDD y sus diferencias con la **minería de datos**.

Según Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth describen KDD como [525], [617]: «El proceso no trivial de identificación de patrones en datos válidos, nuevos, potencialmente útiles, y en última instancia, comprensibles». En otro artículo de estos mismos autores, estos especifican claramente que KDD se refiere al proceso completo de descubrir conocimiento a partir de los datos, mientras que la **minería de datos** es un paso en este proceso, exactamente es el paso en el que se especifican algoritmos para extraer patrones de los datos [605].

Por otro lado, W. J. Frawley, G. Piatetsky-Shapiro y C. J. Matheus agregan que KDD es la extracción no trivial de información potencialmente útil y previamente desconocida a partir de datos [578].

Por esto, hay que tener cuidado y no confundirlo con la **minería de datos**, pues la **minería de datos** es uno de los pasos central de **KDD**, exactamente es la aplicación de algoritmos para obtener patrones o modelos, mientras que **KDD** es todo el proceso completo de descubrimiento [525], [582], [618].

Knowledge Discovery in Databases

KDD es todo el proceso no trivial y automático de obtención de información y patrones de un conjunto de datos, que incluye todas las fases necesarias para ello, y que tiene como piedra central la **minería de datos** con el fin de encontrar patrones válidos, nuevos, útiles y, en última instancia, comprensibles. Las fases de KDD son el análisis del problema, el procesamiento de los datos, la aplicación de **minería de datos** para encontrar patrones, la evaluación del patrón y el despliegue del patrón.

11.4.2.2 FASES DE KDD

KDD se compone de varias fases o métodos, dependiendo del autor. Estas fases son las que diferencian KDD de la **minería de datos**, pues son fases previas y posteriores a la aplicación de la **minería de datos**. Estas fases sirven para asegurar que el conocimiento es derivado de los datos, pues a veces los patrones descubiertos de la **minería de datos** resultan ser banales por carecer de sentido, o bien por ser inválidos [605].

L. Rokach y O. Maimom [582] presentaron un método híbrido de KDD, en base a otros autores, compuesto de 8 fases. Básicamente, lo que hicieron los autores de este método fue fusionar la fase 3 y 4 de

[173] Glosario: **Biosecuencia**

otros métodos para hacer todo el preprocesado de datos en una sola fase a diferencia de los métodos de 9 fases [605]. Las ocho fases de KDD son las siguientes:

1. Desarrollo de la **comprensión del dominio de aplicación**, del conocimiento previo relevante y de los objetivos de los usuarios finales [582]. Este debe de ser relevante y aportar un beneficio financiero [578].
2. **Selección de un conjunto de datos**, o subconjunto de datos, sobre el que se va a realizar el estudio [582]. Este conjunto debe tener un número deseable de muestras y que el número de datos incompletos o con ruido sea muy pequeño [578].
3. **Preprocesamiento de datos**:
    a. **Reducción** de la dimensión por medio de la selección de funciones y la toma de muestras útiles para el fin perseguido, lo que ofrece una reducción del número de variables a considerar [605].
    b. **Limpieza** de los datos para eliminar el ruido que es generado por los diferentes tipos de datos, los valores extremos y los valores perdidos debido a valores por defecto o valores no obligatorios [578].
    c. **Transformación** de los datos para extraer sus atributos mediante discretización. La discretización se da cuando se requieren tipos de datos específicos para que un algoritmo funcione correctamente [619]. Lo que se hace es la división de un atributo de tipo continuo (numérico) en al menos dos subconjuntos [620].
4. **Elegir la tarea o el método de minería de datos** adecuada. Pueden ser de clasificación, regresión, *clustering* o *summarisation*.
5. **Elegir el algoritmo de minería de datos** seleccionando el método específico que se utilizará para la búsqueda de patrones. Un algoritmo de minería de datos no es más que un conjunto de cálculos y reglas heurísticas que permiten crear un modelo a partir de datos [613]. Por ejemplo, redes neuronales, máquinas de vectores de soporte, redes bayesianas, árboles de decisión o diferentes algoritmos de *clustering* o de regresión. Esta fase es difícil, pues se pueden utilizar diferentes algoritmos para realizar el mismo trabajo pero cada uno va a dar una salida diferente [613].
6. **Emplear el algoritmo elegido de minería de datos**.
7. **Evaluación e interpretación de los patrones extraídos**. Esto puede suponer tener que iterar de nuevo entre las fases anteriores. Además, este patrón puede involucrar la visualización de los patrones extraídos o de los datos [605].
8. **Despliegue del patrón encontrado en otro conjunto de datos** para utilizarlo y probarlo, y/o documentación del patrón.

11.4.3 **BIG DATA**

El término ***Big Data*** surgió por primera vez en el año 1997 en unas diapositivas, en el ámbito no académico y sin publicar, de Silicon Graphics (SGI), de John Mashey, tituladas «*Big Data* and the Next Wave of InfraStress» [621]. Sin embargo, previamente ya se había hablado de ***Big Data***, en el año 1984, pero no en el contexto como se le conoce hoy en día. En cambio, en el ámbito académico y publicable, el primer libro en el que se nombró este término fue un libro sobre **minería de datos** del año 1998 [622]. Todo esto de acuerdo con la historia del término investigada por Francis Diebold en [623]. A veces llaman a ***Big Data*** también como **Data-Intensive Computing** [587], [624], o bien se refieren a ***Big Data*** como el cuarto paradigma [596], [624].

Big Data es un término que cada vez es más usado para describir el proceso de aplicar seriamente el poder computacional a conjuntos de datos masivamente amplios, altamente complejos, heterogéneos y que se encuentran en crecimiento y poseen múltiples fuentes [583], [595], [597], siendo por esto no manejables con las metodologías y herramientas utilizadas para la **minería de datos** [583], las cuales ya en el año 2009 daban bastantes problemas para la captura de datos [624]. El análisis de estos datos con herramientas ***Big Data*** puede conducir a conclusiones mucho más fuertes para aplicaciones de minería de datos, aunque ha habido muchas dificultades en su uso [622]. Por esto, el fin de ***Big Data*** es contestar a diferentes preguntas para obtener respuestas tempranas o predicciones acerca del futuro [598].

Uno de los problemas de ***Big Data*** es la falta de consenso en su definición o las diferentes definiciones u opiniones que dan las personas [625]. Esto ha sido recogido un artículo que recoge la opinión de 40 personas relevantes [626], donde explica que muchas veces para definir *Big Data* se hace referencia a las «3Vs», mientras que otros hacen referencia a las herramientas utilizadas. Otras veces existe confusión con la **minería de datos**, pues hay gente, entre los que destacan empresas e investigadores, que dicen que es lo mismo.

Como se ve, el término ***Big Data*** está muy corrupto debido a la falta de una definición exacta y las diferentes definiciones ambiguas que diferentes grupos le han otorgado, entre los que destacan la academia y la industria [627]. Por eso, en los siguientes subcapítulos se definirá el significado de «grande/*big*» y se verán las diferentes definiciones dadas por las compañías y la comunidad científica sobre que es ***Big Data***.

11.4.3.1 DEFINICIÓN DE «GRANDE/BIG»

Una de las diferencias que se puede encontrar en el significado de la palabra «**Big**», es que esta palabra implica importancia, complejidad y desafío, pero también es utilizada para medir la magnitud, lo que da a lugar a la confusión en la definición [627]. Otros, como Dave Campbell, técnico de Microsoft, sostienen que ***Big Data*** no es el nombre apropiado, pues no tiene nada que ver con el tamaño de los datos [595], aunque ciertos investigadores se basen en la enorme cantidad de datos que se crea cada día para definir el término [583].

Esto último queda reflejado si se miran las definiciones en dos idiomas, español e inglés, recordando que los artículos publicados a lo largo de esta tesis son en inglés.

«Grande» según la Real Academia Española (RAE) [154] y *WordReference* [155] - Español

1. adj. Que supera en tamaño, importancia, dotes, intensidad, etc., a lo común y regular.

«Grande» según Google [156]

1. Que tiene un tamaño superior al que se considera normal o superior en comparación al de otra cosa de su misma naturaleza.

2. Que es especialmente intenso o perceptible.

3. Que es importante o destaca por alguna cualidad.

4. [persona] Que es muy bueno.

6. adjetivo/nombre masculino. [persona, organización, país] Que es uno de los de mayor importancia, peso económico o poder dentro de un ámbito.

«Big» según *WordReference* [155] - Ingles

adj.

1. large in size, height, width, or amount.

2. of major concern or importance; outstanding; influential.

«Big» según Oxford [157]

1. Of considerable size or extent.

1.1. [ATTRIBUTIVE] Larger than other items of the same kind.

1.4. informal On an ambitiously large scale.

1.5. [ATTRIBUTIVE] informal Doing a specified action very often or on a very large scale.

2. Of considerable importance or seriousness

2.1. informal Very popular or successful.

«Big» según Cambridge [158]

1. Large in size or amount.

2. Important or serious

Como se ve, en ambos idiomas se utiliza la palabra «grande/*big*» para indicar diferentes cosas, todo dependiendo del contexto en que se utilice. Esta palabra se usa para indicar que:

- Algo es grande en tamaño, peso, altura o anchura.
- Es grande en importancia.
- Es muy bueno.
- Es uno de los de mayor importancia y poder en su ámbito.
- Es muy influyente.
- Es de gran preocupación.
- Algo es excepcional.
- Algo es muy popular.

Por esto, como se indicó previamente, se puede ver que el haber utilizado la palabra «grande/*big*» para nombrar a este término dio juego a que pudiese haber ambigüedades y confusiones en su entendimiento, debido a su asociación con el tamaño cuando en verdad tiene más acepciones.

Big Data es mucho más que el tamaño de los datos, es también su importancia, algo muy bueno, aunque no tal vez para todas las personas, pero si para las empresas que lo utilizan, es la evolución del **análisis de datos** y de la **minería de datos**, tiene una gran influencia en las decisiones de las empresas y es de gran preocupación debido a que puede hacer de «Gran Hermano» y saber todo lo que hacemos si se utiliza mal, invadiendo así nuestra privacidad [585], [628], además de que ya ha comenzado a ser muy popular.

Por estas razones, a pesar de que algunos sostengan que el término está mal elegido, se puede ver claramente, como se verá en los siguientes apartados, que cumple con todo lo que refleja ***Big Data***.

11.4.3.2 DEFINICIONES DE EMPRESAS Y ACADÉMICOS

Muchas empresas e investigadores se metieron en ***Big Data***, y por ello, crearon diferentes artículos firmados por dicha empresa de acuerdo a establecer que es ***Big Data*** para ellos desde su punto de vista y hablar de ello.

Una de estas empresas fue Oracle, en el año 2013 [579]. Según Oracle, ***Big Data*** engloba los datos tradicionales de las bases de datos relacionales, los datos generados por máquinas como son los sensores y los logs, y los datos sociales que incluyen las redes sociales y los diferentes blogs existentes. Además, considera que existen «4Vs» que ***Big Data*** debe de cumplir, las cuales son el volumen, la velocidad, la variedad y el valor. Así, Oracle define ***Big Data*** como un conjunto de datos diversos que se obtienen y analizan gracias al uso de herramientas específicas y ayudan a mejorar diferentes aspectos de las compañías, que van desde procesos internos hasta la predicción de datos relevantes. Como se ve, aquí utilizan la palabra «Big» para definir el tamaño de los datos, la importancia, y la influencia de estos.

Big Data

Intel dio también su propia definición, y para ellos, ***Big Data*** es las cantidades de datos grandes, complejas o no estructuradas, las cuales pueden ser recopiladas y analizadas utilizando tecnologías emergentes como Hadoop y MapReduce, con el fin de obtener conocimiento significante [594]. En el caso de Intel, solo hace referencia al tamaño de los datos.

La empresa Gartner fue otra que hizo su propio artículo sobre ***Big Data*** [629], [630]. En este artículo explican que *Big Data* es un término utilizado para admitir el crecimiento exponencial, la disponibilidad y el uso de información de actualidad para liderar decisiones de la empresa. No obstante, estas decisiones pueden ser malas e interferir en la arquitectura de la empresa. Además, remarcan que algunos gestores se centran solo en el volumen de los datos, cuando resulta que deben de mirar también la velocidad y la variedad, es decir, las «3Vs». Para esto, según Gartner, «se necesitan formas innovadoras de procesamiento de la información que permitan tener una visión mejorada, tomar decisiones y la automatización de procesos». Por esto, para Gartner, el verdadero reto es obtener patrones que puedan ayudar a las empresas a tomar mejores decisiones utilizando tecnologías innovadoras. Como se ve, la definición de Gartner habla de «3Vs» y explica como el fin de ***Big Data*** es obtener patrones que ayuden a tomar decisiones a las empresas.

Un caso especial fue el de Microsoft, donde escribieron un artículo al respecto tomando la perspectiva de diferentes trabajadores importantes dentro de la empresa de Redmond. Según la perspectiva de los diferentes trabajadores o personal relacionado con Microsoft [595], se podría resumir en que ***Big Data*** es el proceso de gestionar y trabajar con cantidades masivas de datos digitalizados que requieren gran capacidad de cómputo para obtener información útil con el fin de tomar decisiones de una manera rápida y eficiente mediante el uso de algoritmos de Inteligencia Artificial. Aquí, con «Big» hacen referencia a la cantidad masiva de datos y a la importancia de estos.

IDC, empresa dedicada al análisis de mercados, dedicó un informe a ***Big Data*** [631]. En dicho informe resalta la importancia de analizar los datos para generar valor mediante la correcta extracción de la información del universo digital. El problema es que los datos están sufriendo un crecimiento exponencial y se encuentran encapsulados en muchos ficheros de diferente tipo. Para ello, sostienen que esto ahora es posible gracias a la convergencia de las tecnologías. Además, también destacan que *Big Data* no es algo nuevo y que no es una «cosa», sino una dinámica o actividad que cruza muchas fronteras informáticas. En base a esto, su definición de ***Big Data*** es: «*Las tecnologías Big Data describen una nueva generación de tecnologías y arquitecturas diseñadas para extraer económicamente valor desde unos volúmenes de datos muy grandes y ampliamente variados, permitiendo una alta capacidad de captura, descubrimiento y/o análisis*». Como se ve en esta definición y en el resto del artículo, tienen en cuenta «3Vs», el volumen, la variedad y la velocidad. Además, se centran sobre todo en las tecnologías nuevas que permiten cumplir estas «3Vs». Sin embargo, solo hace referencia al análisis de datos y no a lo que sería la minería, que es el descubrir tendencias, o bien a la predicción.

Por otro lado, tenemos al NIST, que dio varias definiciones, todas dependiendo del punto de vista desde la cual fue tomada. La primera de ellas tiene que ver con la arquitectura y la escalabilidad de estas para tratar con los datos: «***Big Data*** *consiste en grandes conjuntos de datos, con las características de volumen, variedad, velocidad y/o variabilidad que requieran una arquitectura escalable para obtener un almacenaje*,

manipulación y análisis eficiente» [598]. Como se ve, según esta definición, ***Big Data*** hace referencia a la incapacidad de las arquitecturas tradicionales de datos de trabajar con los nuevos conjuntos de datos que requieren cumplir las «4Vs» características, que según la NIST son el volumen, la variedad, la velocidad y la variabilidad [632]. Por otro lado, la otra definición se centra en la escalabilidad y sitúa ***Big Data*** como el corazón del escalado horizontal, donde todo se distribuye entre varios ordenadores y se amplía con la adición de nuevos equipos, en contra del vertical, donde se incrementa la capacidad interna de un ordenador para mejorar el rendimiento [598]: «*El paradigma **Big Data** consiste en la distribución de los sistemas de datos usando un escalado horizontal, independiente de los recursos para lograr la escalabilidad necesaria para el procesado eficiente de grandes conjuntos de datos*». Así, en estas definiciones, tratan la palabra «*Big*» como grande en volumen y en necesidad de computación.

Así, el NIST sostiene que ***Big Data*** es un término utilizado para describir los grandes volúmenes de datos en el mundo conectado, digitalizado, cargado de sensores y dirigido por la información [598]. Habiendo estos datos desbordado los enfoques de análisis tradicionales, conocidos como **minería de datos**, y manteniendo un crecimiento de los datos tan rápido que consiguen dejar atrás los avances científicos y tecnológicos en esta área. ***Big Data*** permite manejar y tratar estos volúmenes heterogéneos, los cuales no tienen por qué ser necesariamente grandes, o más grandes que antiguamente, sino que puede deberse a la velocidad de procesado necesaria o a la eficiencia requerida al procesar un volumen de datos y que no proporcionan las herramientas y métodos tradicionales. Sin embargo, hay que tener en cuenta, como bien sostienen con sus definiciones, que este término ha sido utilizado para describir muchos conceptos debido a que hay diferentes aspectos diferentes interactuando entre sí, y depende desde el punto de vista en que se mire y donde se utilice.

Otra definición, dada por Jonathan Stuart Ward y Adam Barker [627], donde hicieron un estudio de las diferentes definiciones existentes en la red y dadas por diferentes empresas, crearon su propia definición, siendo esta la siguiente: «***Big Data*** es un término que describe el almacenamiento y análisis de grandes o complejos conjuntos de datos usando una serie de técnicas que incluyen, pero no se limitan, a basas de datos *Not Only SQL* (NoSQL), MapReduce y *Machine Learning*».

La definición dada por Wei Fan y Albert Bifet acerca de ***Big Data*** se basa en la capacidad de extraer información útil de los grandes conjuntos de datos que debido a su volumen, variedad y velocidad, ya sean estáticos o en *streaming*, no era posible extraer antes [583]. Como se ve, consideran «3Vs» y no solo se aferran a la gran cantidad de datos, sino a la variedad y velocidad, es decir, eficiencia. Además, la consideran, como otros, la evolución de las técnicas antiguas, donde estaban KDD y minería de datos.

Por otro lado, la definición de Emmanuel Letouzé dice que ***Big Data*** son las herramientas y metodologías que pretenden transformar cantidades masivas de datos en bruto, tanto estructurados como no estructurados, que son difíciles de procesar con las técnicas software tradicionales en datos para fines analíticos [584].

Otros autores dicen que la **minería de datos** actualmente se conoce como ***Big Data*** debido a la gran cantidad de datos que debe de manejar rápidamente y que para poder tratar con aplicaciones de este tipo, se ha desarrollado un nuevo tipo de aplicaciones, las cuales ofrecen paralelismo para trabajar con estas grandes cantidades de datos en un clúster de ordenadores conectados por cables de Ethernet o *switches*, evitando así

el uso de supercomputadores [416]. Como se ve, en esta definición dejan claro que ***Big Data*** es la evolución, así como es la arquitectura utilizada.

Por otro lado, los autores de [625] estudiaron también esta falta de definición y contrastaron diferentes definiciones encontradas. Así, a grandes rasgos, ellos entienden por ***Big Data*** los conjuntos de datos que no se pueden percibir, adquirir, gestionar y procesar por las herramientas informáticas software y hardware tradicionales dentro de un tiempo aceptable. Como se ve, esta definición hace referencia a las herramientas tradicionales, algo que es verdad, pero no dice la función o la finalidad que tiene ***Big Data***, algo que es bastante importante.

En cambio, en este otro survey [455] dan como definición que ***Big Data*** es «la habilidad de obtener información privada de las preferencias de los consumidores y los productos cruzándola con información de *tweets*, blogs, evaluaciones de los productos y datos de las redes sociales abriendo un amplio rango de posibilidades para organizaciones para entender a sus clientes y predecir que quieren y demandan, y optimizar el uso de recursos». Como se ve, esta definición solo refleja la parte de las empresas en busca de información de sus clientes en base a unos pocos sitios, cuando resulta que ***Big Data*** es mucho más y permite no solo analizar información, sino tendencias de cualquier tipo, pues es aplicable a todo, incluyendo la astronomía, la física, la defensa y la seguridad, y a cualquier tipo de dato, como son las imágenes y el video.

En base a todo lo visto hasta ahora, una posible definición de ***Big Data*** en base a estas definiciones y acorde a lo que hace y el resultado obtenido de utilizar ***Big Data*** puede ser la siguiente:

Big Data

Big Data es la búsqueda útil de información en conjuntos de datos de cualquier tipo que es difícilmente realizable utilizando las metodologías, técnicas y herramientas tradicionales debido a las características de los datos a analizar y que son conocidas como las «Vs», que son el volumen, la velocidad, la variedad, la veracidad, la variabilidad y el valor, con el fin de conseguir obtener resultados que permitan analizar datos, sacar las tendencias actuales, optimizar el uso de los recursos y/o predecir el futuro, de una manera rápida y eficiente.

11.5 *LAS «6VS» DE BIG DATA*

La diferencia entre los análisis de datos que se hicieron «siempre» y el término ***Big Data*** radica en que este último debe de cumplir ciertas condiciones que antes no eran «tan importantes». ***Big Data*** debe de cumplir las «**3Vs**» [577], [629]. Estas «Vs» son el **volumen** y la cantidad de datos a gestionar, la **velocidad** de creación y de utilización requerida para tratar estos datos, la **variedad** de diferentes tipos de fuentes de datos. La primera vez que se definieron estas «3Vs» fue en [633], de acuerdo con [583].

Hay muchos autores que han defendido las «3Vs» [630], [631], sin embargo, a veces, algunos autores apuntan a las «4Vs» debido a los problemas que explica [634], siendo esta la **veracidad** de estos datos, mientras que otros abogan por las «4Vs», pero incluyendo en ellas el valor [579], y otros por las «4Vs» añadiendo la **variabilidad** [598], aunque algún autor habla también de la **veracidad** [635]. Por otro lado,

otros autores sostienen que hay que incorporar dos extras a las del primer autor, siendo esta diferente a los que abogan por las «4Vs» [627], [636]. Uno de los autores de las «5Vs» es [583], añadiendo así la **variabilidad** y el **valor**, mientras que otro autor que habla de «5Vs» añade **veracidad** y **valor** [455].

Big Data ha seguido evolucionando desde que se establecieron por primera vez las «3Vs» y se han ido dando nuevas características, que ya existían pero que no se han visto necesarias hasta que se investigó más en ***Big Data***. Por esa misma razón y bajo el estudio de los diferentes autores y en base a lo que ***Big Data*** necesita, surge la combinación de ellas que otorgan un total de «6Vs» que permiten definir tanto las características como las necesidades más importantes de ***Big Data***.

11.5.1 VOLUMEN

El **volumen** o la cantidad de datos es enorme, gigantesco. Esta característica describe exactamente eso, la gran cantidad de datos que están por venir, la gran cantidad que está diariamente trabajándose y que van desde los gigabytes, hasta los exabytes y más. La capacidad de almacenaje se ha ido duplicando aproximadamente cada 3 años, y aun así en muchos campos esto ha sido insuficiente, como ha ocurrido con los datos médicos y financieros [596].

Actualmente, muchas empresas tratan miles de terabytes (10^{12}) diariamente para obtener información útil para su negocio y acerca de sus usuarios. La cantidad de datos va creciendo, pues cada día es más la información y el número de usuarios existentes, lo que implica, como bien algunos estimaron, que tenga un crecimiento exponencial. Se estima que en el año 2020 haya 500 veces más cantidad de datos que en el año 2011 [598].

Como se ve en la Tabla 4, hay muchísimos datos, y más que se estiman para el futuro, siendo el año 2016 el año de redacción de esta tesis doctoral, y la manera de analizarlos es utilizando herramientas de ***Big Data*** que permitan gestionar este conocimiento en tiempo real o casi real. Esta tabla muestra los datos relevantes desde al año 1990 hasta los que se estiman en el año 2020. Estos datos demuestran el porqué del auge de las aplicaciones ***Big Data***, donde la recolección de datos ha crecido enormemente y está más allá de la capacidad de las herramientas de software utilizadas tradicionalmente para capturar, gestionar y procesar dentro de un «tiempo tolerable» estos datos [597].

Tipo de servicio	Cantidad de datos	Año de ocurrencia
Satélites de la Tierra en un día [578]	1 terabyte (10^{12})	1990
Páginas web indexadas por Google [583]	1 millón (10^{6})	1998
Páginas web indexadas por Google [583]	1.000 millones = 1 millardo (10^{9})	2000
Páginas web indexadas por Google [583]	1 trillón (10^{18})	2008
Estudio del genoma humano [637]	320 terabytes en 2 horas de computación	2008
Experimento astronómico o de partículas físicas [590]	1 petabyte (10^{15}) por año	2009
Fotos por segundo en Facebook [638]	1 millón	2010
Facebook [588]	30 millardos de contenido compartido cada mes	2010
Fotos subidas a la semana en Facebook [638]	1 millardo de fotos = 60 terabytes de espacio	2010
Fotos almacenadas en Facebook [638]	260 millardos de fotos = 20 petabytes de espacio	2010
Librería del Congreso de los Estados Unidos de América [588]	235 terabytes de datos recolectados	2011 (Abril)
Humanidad [585]	2,5 quintillones (10^{30}) de bytes de datos cada día	2012
Colisionador de Hadrones en el descubrimiento del Bosón de Higgs [623]	1 petabyte por segundo	2012
Información de los usuarios de Walmart cada día [577]	2,5 petabytes	2012

Tipo de servicio	Cantidad de datos	Año de ocurrencia
Datos electrónicos por año [591]	1,2 zettabytes (10^{21})	2012
Datos nuevos cada día a partir de 2012 y doblándose cada 40 meses [577]	2,5 exabytes (10^{18})	2012
Google [599]	2 millones de búsquedas en 1 minuto	2012
Flickr [599]	3 millones de subidas en 1 minuto	2012
Flickr [599]	20 millones de fotos vistas en 1 minuto	2012
Emails enviados [599]	204 millones en 1 minuto	2012
Servicio de Mensajería Multimedia (MMS) [639]	28.000 MMS por segundo	2012
Pandora: horas de música escuchadas [599]	61.000 horas en 1 minuto	2012
Twitter durante el debate presidencial de USA [640]	10.300.000 *tweets* en 1h30m	2012
Tweets [641]	143.199 por segundo	2013 (3 de agosto)
Actualizaciones en Facebook por día [583]	+ 800 millones	2013
Consultas a Google por día [583]	+ de 1.000 millones	2013
Tweets por día [583]	+ 250 millones	2013
Twitter [579]	+ 8 terabytes por día	2013
Vistas de YouTube por día [583]	+ 4.000 millones	2013
Páginas Web [642]	1,5 billón (10^{12})	2013
Internet [642]	20 exabytes de información	2013
Tweets [642]	20 millardos	2013
Blogs [642]	70 millones	2013
Un motor de un jet en 30 minutos de funcionamiento [vuelan 25.000 aviones por día] [579]	10 terabytes	2013

Tipo de servicio	Cantidad de datos	Año de ocurrencia
Twitter [642]	50 millones de usuarios	2013
Twitter [641]	310 millones de usuarios activos mensuales	2013
Creadores de contenido social [642]	600 Millones - 33% de los usuarios de Internet	2013
Twitter [641]	500 millones de *tweets* por día	2013 (Agosto)
Otras publicaciones periódicas [642] - Periódicos y otros	10.000	2013
Internet [643]	40,7% de la población lo utilizaba en 2014 = 2.954 millones	2014
Población del mundo [643]	7.259.691.769 personas en 2014	2014
YouTube [644]	+ Mil millones de usuarios = 1/3 de usuarios de Internet	2015
YouTube [644]	+ 100 Millones de horas de video vistas a diario	2015
Google Photos [645]	2 billones (10^{12}) de etiquetas en su primer año	2015
Google Photos [645]	1,6 millardos de animaciones en su primer año	2015
Google Photos [645]	24 millardos de *selfies* en su primer año	2015
Google Photos [645]	200 millones de usuarios en su primer año	2015
Flickr [646]	Casi 70 millones de fotos públicas subidas mensualmente	2015
Facebook [647]	1.650 millones de usuarios	2016 (31 de marzo)
Trafico anual de Internet [648]	1 zettabyte (10^{18})	2016
Trafico anual de Internet [648]	2,3 zettabytes	2020
Square Kilometer Array [649]	524 terabytes por segundo	2020
Volumen de datos por año [579]	Un 40% más de volumen de datos en 2020 que en 2009	2020

Tabla 4 Estadísticas de *Big Data*

Muchas veces este conocimiento debe de ser eficiente pues se necesita que sea en tiempo real debido a problemas con el espacio para almacenarlo [597]. Un ejemplo de esto es el del radiotelescopio *Square Kilometer Array* (SKA) [649], que se convertirá en el radiotelescopio más grande del mundo y con el que se pretende descubrir el principio y el final del universo. Los investigadores estiman que el SKA producirá aproximadamente 4.1 Pebibits (2^{50}) por segundo, o lo que es lo mismo, 562 terabytes (10^{12}) por segundo. Como se ve, habrá un futuro aún con más datos, con cantidades enormes de datos para analizar. Algunas estimaciones indican que la cantidad de nueva información se duplica cada tres años [526].

Otro ejemplo de cantidad de datos generados ocurrió el 4 de octubre de 2012, cuando el debate presidencial entre el presidente de los Estados Unidos de América, Barack Obama, y el Gobernador Mitt Romney hizo que los usuarios de Twitter posteasen más de diez millones de *tweets* en hora y media [640]. Estudiando estos datos de Twitter acerca del debate por la presidencia de EEUU [640], se puede observar que la gente *twitteó* más cuando se habló del seguro médico de las personas mayores. Así, en base a esto, se puede ver lo que importaba a la gente en ese momento.

Flickr, la red social de fotos también es muy utilizada en investigaciones debido a que las imágenes dan juego a poder obtener información de estas. Actualmente se suben cerca de 70 millones de fotos públicas nuevas que se suben mensualmente [646]. Así, gracias al análisis de estas fotos, se podrían utilizar para estudiar la sociedad humana, los eventos sociales o desastres [597].

Por otro lado, el asesor científico del presidente de los Estados Unidos de América, John Paul Holdren, dijo en el año 2011 que ***Big Data*** era un asunto importante debido a que cada año se crean cerca de 1.2 zettabytes (10^{21}) de datos electrónicos, cuya equivalencia en terabytes son 1.200.000.000, y que incluyen desde experimentos científicos hasta los datos de los telescopios y de los *tweets* [591]. Esto es certificado por otras estimaciones realizadas en el año 2012 [577], donde predijeron la creación de 2.5 exabytes (10^{18}) cada día, equivalentes a 2.500.000 terabytes, pero que esta capacidad de creación se duplicaría cada 40 meses, aproximadamente. Como se ve, esta predicción es bastante similar a las realizadas por John Paul Holdren.

No obstante, esta cantidad de datos no se está dando solo ahora, pues ya hace años había ciertas aplicaciones que generaban grandes cantidades de datos. En 1990, los satélites que orbitaban la Tierra generaban un terabyte (10^{15}) de información al día. Esto significaba que, si se tomaba una foto cada segundo y si se ponía a una persona a analizar todas estas fotos, suponiendo que trabajase por las noches y los fin de semana, le llevaría analizar muchos años todas las fotos de un día [578]. Esto sirve para recalcar que, ahora, 26 años más tarde y con unas tecnologías mejoradas, tanto hardware como software, lo que permite realizar análisis más rápidos y automáticos, pero también sobre imágenes con mejor resolución y más datos.

Otro proyecto relevante, tanto en importancia como en tamaño de datos, es el proyecto «*The Genome 1000 Project*», que trata acerca del genoma humano. En este proyecto, dos horas de ejecución de Solexa crean 320 terabytes de información. Esto hizo, que en el año 2008, fuese totalmente imposible de guardar y computar en aquella época [637].

En relación a esta evolución se puede ver la predicción de Eron Kelly, quien ha predicho que en los próximos cinco años generaremos más datos que todos los generados por la humanidad en los últimos 5.000 años [595]. Mientras, la NIST prevé que la generación de datos será el doble cada dos años, y estos alcanzarán los 40.000 exabytes en 2020, de los cuáles se espera que un tercio de estos datos podrán ser útiles si son analizados. Esto es algo que destaca la evolución de los datos que generamos los seres humanos y su crecimiento exponencial a lo largo de la historia y el que se espera que ocurra.

Para tratar toda esta información, hace años que ya se ha empezado a migrar los diferentes sistemas a la nube y realizar lo que se conoce como ***Cloud Computing***, ya sea en una privada, en una pública o con un sistema híbrido. Esto supuso un ahorro de dinero y una forma de procesar nueva los datos, lo que facilitó el

uso de ***Big Data*** en las empresas gracias a las diferentes tecnologías que proporcionan las compañías y el escalado de dichas herramientas [625].

Sin embargo, hay que tener en cuenta que no siempre por tener conjuntos de datos más grandes se van a obtener mejores predicciones, esto puede ser clasificado de arrogancia, pues en verdad ***Big Data*** no es el sustituto de la recopilación de datos y la **minería de datos** [602]. Por eso, a pesar de tener estos grandes conjuntos, no hay que olvidarse de las técnicas básicas de medición y de construcción de datos fiables y válidos y las dependencias entre estos datos [628].

11.5.2 VELOCIDAD

La **velocidad** describe la rapidez con que los datos son procesados, pues, en ***Big Data***, la creación de estos datos es continua, nunca cesa. En Internet, ya sea por medio de una red social como Twitter, o en Facebook, o por diferentes servicios como son los blogs o servicios de video, hay gente en todo momento escribiendo información, subiendo un video, enviando correos o accediendo a las páginas web. Esto ocurre continuamente, miles de datos cada minuto. Esto hace que haya una gran **velocidad de creación** de contenido o de lectura de estos.

La gran mayoría de estos servicios necesitan datos de sus usuarios, de cómo utilizan su servicio o bien de sus preferencias para así adaptar su contenido o sus anuncios a sus usuarios. Esto hace que existe una gran necesidad de **velocidad de procesamiento** de los datos. Esta velocidad puede ser de cuatro tipos: en *batch* o intervalos, *near time* o *nearly time* que es casi a tiempo real, *real time* o a tiempo real, y *streaming*.

Como se ve, esta «V» tiene dos tipos de velocidades, la de lectura o creación de contenido y la de procesado, que pueden ser independientes o dependientes, esto según los requisitos de la aplicación.

Además, hay aplicaciones que necesitan más velocidad de procesamiento que volumen de datos, y así permitir ser a una compañía mucho más ágil que sus competidoras mediante el ofrecimiento de aplicación en tiempo real o casi real, conocidas como aplicaciones *real-time* o *nearly real-time* [577], [598]. Es importante, en este punto, el tener en cuenta que esto no se refiere al ancho de banda ni a las cuestiones de los protocolos, sino a la velocidad de creación de los contenidos y la capacidad de gestión que se puede dar de estos, que se divide en su almacenaje, análisis y visualización, esperando que con el uso de ***Big Data*** este tiempo sea mínimo [598], [633].

Un dato que sirve para ejemplificar esta «V» es el concurso que se realizó en un congreso del año 2008, en el SC08 International Conference for High Performance Computing [650]. En dicho concurso los participantes tenían que realizar una consulta en el Sloan Digital Sky Survey (SDSS). El SDSS es un proyecto de investigación del espacio en el cual se toman imágenes en tres dimensiones de este con el fin de mapearlo [651]. El ganador tardó 12 minutos en realizar una consulta en un clúster en paralelo mientras que la misma consulta sin utilizar paralelismo tardaba 13 días [590]. Esto destaca la importancia de la velocidad de procesado cuando se computan grandes cantidades de datos.

11.5.3 **VARIEDAD**

La **variedad** describe la organización de los datos, si estos están estructurados, semiestructurados, no estructurados o varios de estos mezclados. Actualmente hay un sinfín de posibles fuentes de datos que vienen dadas por la gran variedad de fuentes existentes para recabar datos, donde en muchos casos estos datos no están estructurados, los cuales son difíciles de manejar y que poseen mucho ruido [577].

Existen páginas web donde los usuarios escriben sus opiniones, los *tweets*, Facebook, LinkedIn, Instagram, YouTube, Tumblr, Flickr y otras redes sociales, las cuales son consideradas los recursos más valiosos [592]. Otra variedad de datos y muy importante es la ofrecida por los diferentes gobiernos cuando proporcionan diferentes datos abiertos, pues estos datos los ofrecen en diferentes formatos como Excel, CSV, Word, PDF, JSON, RDF-N3, RDF-Turtle, RDF-XML, XML, XHTML, texto plano, HTML, RSS, KML y GeoRSS. Otros tipos de datos que son interesantes son los videos, en diferentes formatos como FLV, WMV, AVI, MKV o MOV, que además muchas veces vienen acompañados de comentarios, en este caso nos encontramos con servicios como YouTube y Vimeo, entre otros. Un caso similar es el de los servicios de audio o radios, que tienen diferentes formatos como MP3, OGG, Flacc, MIDI o WAV.

Las últimas tecnologías como son los diferentes dispositivos móviles, ya sean televisiones, coches, tablets, smartphones o cualquier otro *Smart Object*, pues ofrecen muchos datos por medio de los diferentes medios que tienen, sea a través de sus sensores o de sus sistemas GPS. Hablando se sensores, tenemos todos los disponibles en el mercado y que se están abriendo cada día más hueco en el mercado gracias a su facilidad de uso mediante un Arduino o una Raspberry Pi. A esto hay que añadirle otros dispositivos provenientes de IoT [583], el cual es uno de los pilares fundamentales como fuente de información, tanto estructurada como semiestructurada o no estructurada, de ***Big Data*** [625].

No hay que olvidarse tampoco de otros datos muy importantes a medir y que se utilizan para ver la interacción del usuario como son los clics de ratón, pulsaciones de teclado, desplazamiento de página, el tiempo de lectura en un artículo, el contenido compartido, o las interacciones con el contenido, como hacen muchas redes sociales, como bien son Facebook, Twitter y Weibo. Todo esto muchas veces va acompañado de fotos, lo que aún amplía más si cabe la multitud de formatos y de tratamientos con JPG, PNG, TIFF o BMP.

Además, por si fuera poco, a veces hay que tratar con datos legados en diferentes bases de datos, ya sean SQL o NoSQL, documentos, emails, conversaciones telefónicas o documentos escaneados [594]. Otras veces, puede ser información de páginas web que están en HTML o XMLs que están bien estructuradas o PDFs que no tienen una forma estructurada de mostrar los datos. Otras veces igual hay diferentes datos agrupados de diferente tipo en ficheros comprimidos en RAR, RAR4, ZIP o TAR, los cuales primero hay que tratar para descomprimir y analizar su contenido.

Como se ve, hay multitud de formatos, y eso que aquí solo se pusieron pocos ejemplos. Cada uno de estos ficheros, además, necesita un tratamiento especial, aunque sean del mismo tipo, por ejemplo, las imágenes, pues no todos los formatos tienen las mismas propiedades y poseen pequeñas diferencias. Además, cada año tenemos nuevos dispositivos o servicios que dan nuevos datos útiles o los mismos datos, pero en

otros formatos, o modifican el cómo mostraban esa información, lo que implica modificar ciertas partes del software de lectura de datos. Hay que tener también en cuenta, que además de los formatos, todo esto depende también de la aplicación, pues no es lo mismo analizar imágenes de telescopios o de satélites, que imágenes de redes sociales, o analizar datos de usuarios con datos del tiempo.

Así, esta heterogeneidad de datos, formatos incompatibles y de datos inconsistentes es una de las mayores barreras para la gestión eficaz de los datos [633]. Además, estos datos cambian continuamente y se añaden nuevos sistemas que deben, o se modifica la forma en que se ofrecen la información existente al usuario final, en contra de como ocurría antiguamente, donde solo se tenía una base de datos estructurada con un esquema bien definido y que cambiaba muy lentamente [579].

11.5.4 **VERACIDAD**

La **veracidad** viene dada debido a que no todo lo que existe es verdad y puede que se estén recopilando datos falsos o erróneos. Por ello, cuando se trabaja, se ha de tener cuidado con las fuentes de datos y comprobar que sean datos verídicos, tratando así de tener unos datos exactos e íntegros. Esta a su vez se puede dividir en tres subapartados, a saber: la **procedencia**, la **veracidad** y la **validez**.

La **procedencia** de los datos es importante para mantener una calidad, proteger la seguridad y mantener las políticas de privacidad. Hay que tener en cuenta que los datos en ***Big Data*** se mueven desde los límites individuales hasta los de grupos o comunidades de interés y que estos van desde un límite comarcal, regional o nacional, hasta el internacional. Por esto, la procedencia ayuda a saber de dónde vienen estos datos y cuál es su fuente original, por ejemplo, mediante el uso de metadatos. Esto es muy importante, pues, sabiendo su procedencia, se puede mantener la privacidad de estos datos y así poder tomar algunas decisiones importantes, como puede ser el derecho al olvido. Otros datos a insertarse podrían ser los relacionados con la cadena de suministro, como son la calibración, los errores, o los datos faltantes (marcas de tiempo, ubicación, número de serie del equipo, número de transacción, y la autoridad).

La **veracidad** abarca la garantía de la información de los medios utilizados para recopilar la información. Un ejemplo claro es en el caso de que se utilizasen sensores, pues se debería de incluir la trazabilidad, su calibración, la versión, el tiempo de muestreo y la configuración del dispositivo.

Otro ejemplo de falta o problemas con la veracidad es el comentado en [652]. En este artículo se habla de que, como estudio previo para comprobar cierta suposición en el año 2006, el autor había utilizado Google Trends para saber si el presidente que había salido elegido para el Real Madrid Club de Fútbol fue el presidente con más consultas realizadas por la gente en Google. Como bien apunta el autor, el apellido de este presidente era Calderón. El problema, como bien comenta, es que, en el mismo día, salió elegido en México un presidente con el mismo apellido. Así, el problema estaba en que Google Trends no diferenciaba a que Calderón se refería cada consulta, luego, estaba fusionándolas. Es decir, había una falta de datos complementarios, una falta de veracidad.

La **validez** se refiere a la precisión y exactitud de los datos, o, mejor dicho, la calidad de los datos. Poniendo ejemplos se puede citar si, en unos datos continuos y discretos sobre el género de las personas y

siendo hombre=1 y mujer=2, se recibe un 1,5, pues esto no significa un nuevo género, sino un fallo. Otro tipo de error es el referente al fraude de clics en páginas, clics realizados por robots, anuncios ocultos, etc.

Un ejemplo de veracidad es lo comentado por el autor de este artículo [634], en el que pone como ejemplo varios problemas que han ocurrido en los Estados Unidos de América. El primero de ellos es que los políticos siempre les gusta hablar sobre más datos, pero finalmente estos nunca se encuentran entre sus prioridades, pues esto tiene que competir con otras cosas que ofrecen un impacto más inmediato, lo que hace que el dinero sea insuficiente para mantener los datos accesibles y digeribles. El segundo problema reside en que los datos están institucionalizados, cuando deberían de estar aislados de los políticos, por ejemplo, con la que iba a ser la supuesta creación de la Agencia de Estadísticas del Medio Ambiente (*Bureau of Environmental Statistics* - BES), que además de recolectar los datos los analizaría. Como tercer y último problema se presenta el ocurrido cuando las compañías químicas se niegan a presentar sus datos en aras de ver lo que contaminan en base a qué sino revelarían muchos datos útiles para sus competidores. Como expone en el artículo David Goldton, se puede deducir que estos datos pueden no ser de fácil acceso, estar desactualizados, no ser correctos, estar sesgados o ser difíciles de entender. Claramente, si se tienen un gran conjunto de datos, por ejemplo, para ver una tendencia y hay pocos datos «malos», estos se perderán y obviarán automáticamente gracias a que se ocultarán entre tanto dato «bueno». No obstante, si se trata de algo casual, puede que estos datos «malos» si estropeen el experimento, lo cual hace que sea extremadamente importante la veracidad de estos [598].

11.5.5 **VARIABILIDAD**

La **variabilidad** se debe a los cambios que puede haber en la estructura de los datos y como los usuarios quieren interpretar esos datos. Los datos pueden cambiar con el tiempo, modificando su estructura, que modificaría el cómo se recorre el árbol de datos, tal vez por la modificación del modelo en XML, su flujo de velocidad, por ejemplo, de velocidad de creación o de actualización, el formato en que son ofrecidos migrando de un formato, como puede ser XML a otro, como podría ser JSON, o la composición de estos, añadiendo o quitando elementos.

Debido a esto, hay que estar pendientes de estos cambios, siempre que queramos tenerlos actualizados, pero también hay que mantener los datos originales sin modificar, es decir, saber cómo fueron cambiando los datos en el tiempo. Esto tiene como inconveniente la redundancia de datos y el espacio de almacenamiento necesario. Además, claro está, del sistema necesario de supervisión de cambios y comparación con los datos existentes ya almacenados en nuestro sistema.

11.5.6 **VALOR**

El **valor** de los datos reside en lo que estos pueden aportar a la empresa y si pueden aportar una ventaja debido a que ayudan a la toma decisiones en base a preguntas que ahora pueden ser respondidas gracias al análisis de estos datos, o bien en el descubrimiento de tendencias. No obstante, este valor es variable, pues depende de los datos que se tengan y de la información que contengan oculta y si esta es encontrada y extraída. Por ello, el reto está en identificar este valor y conseguir extraerlo para realizar un análisis de los datos que

nos aporten y den este valor. Según la encuesta analizada en [580], que se corresponde con la realizada por el MIT e IBM a 3000 ejecutivos, analistas y gestores de más de 30 empresas en 100 países, uno de cada cinco dijeron estar preocupados por la calidad de los datos o ineficacia de los datos del gobierno como principal preocupación.

11.6 *TEOREMA HACE: CARACTERÍSTICAS DE BIG DATA*

Big Data posee una serie de características, conocidas por algunos autores como teorema HACE [597]. Estas siglas se corresponden con la **h**eterogeneidad de los datos, la **a**utonomía de las fuentes con control distribuido y descentralizado y la **c**omplejidad y **e**volución de las relaciones.

Big Data está compuesta de una gran cantidad de datos **heterogéneos**. La **heterogeneidad** se debe a que cada recolector de información utiliza su propio esquema o protocolo para guardar los datos y las diferentes aplicaciones utilizan diversas formas de representación. Como se ve, esta característica está ligada a la V de variedad.

Otra de las características son las fuentes **autónomas** con control distribuido y descentralizado, que debido a su **autonomía** se consigue obtener fuentes que permitan generar y recolectar información sin depender de un control centralizado. Esto es similar a los servidores web pertenecientes a la *World Wide Web* (WWW), donde cada servidor provee un poco de información y cada usuario puede acceder a esa información sin necesidad de depender de otros servidores.

Las otras dos características de ***Big Data***, la **complejidad** y la **evolución**, se deben al constante crecimiento de los datos existentes y las relaciones entre ellos, así como a su evolución. Por ejemplo, estas relaciones evolucionan en el cibermundo debido al uso de redes sociales y los círculos de amigos que tienen las personas, pues esto presenta una relación entre personas, compartan o no más datos del dominio sobre el que se trabaja.

11.7 *CICLO DE VIDA DE BIG DATA*

Para explicar el ciclo de vida de ***Big Data***, primero se hace necesario explicar el ciclo de vida general de los datos. El **ciclo de vida de los datos** se compone de cuatro pasos básicos [598]. Estos son muy similares a las fases de KDD vistas en el punto 11.4.2.2.:

1. *Recolección* y *almacenaje*: primero de todo hay que recolectar los datos y almacenarlos en su forma original.
2. El segundo paso consta de la *preparación* de los datos, los cuales hay que procesar para así limpiarlos de cualquier problema que pudieran tener, como son posibles datos inservibles como enlaces web o impurezas en forma caracteres extraños. Tras esto se organizan, sobrescribiendo los datos originales de forma que se dejen totalmente listos para su uso.
3. Se hace el *análisis* de los datos ya preparados, el cual involucra técnicas que producen conocimiento sintetizado a partir de la información preparada.

4. Por último, en la *acción* se ejecutan procesos que usan la información sintetizada para generar un valor para la compañía, ya sea en forma de estadísticas, tendencias o predicciones.

Sin embargo, el ciclo de vida de ***Big Data*** tiene cinco diferencias respecto al ciclo de vida clásico de las Fases de KDD [598]:

1. Se almacenan los datos en crudo (*raw*), es decir, se guardan las formas originales de estos, antes de la *preparación*. Esto permite tener siempre los datos originales para hacer diferente análisis, de forma que se podrán utilizar diferentes métodos sobre estos que nos permitan después aplicar diferentes técnicas. Además, esto hace que tengamos siempre toda la semántica original.
2. Los datos obtenidos son almacenados después de la *preparación*.
3. Estos almacenajes dan lugar a que siempre se tengan los datos originales guardados y se tenga por otro lado los datos procesados, mientras que, en el ciclo de vida tradicional, los datos originales se perdían tras la *preparación*.
4. La preparación tiene lugar en el proceso de lectura, lo que se conoce como «*schema-on-read*».
5. La recolección, la preparación y el análisis se hace al vuelo, y posiblemente incluye algún resumen o agregación antes de su almacenaje.

Así, una posible definición de los pasos en el ciclo de vida en ***Big Data*** es la basada en [598], [624], [653] y expuesta a continuación:

1. **Recolección** y **almacenaje** de datos en su forma original. Este paso incluye los siguientes mecanismos: protocolo de transporte, seguridad, el formato de los datos y los metadatos.
2. Se hace la **recolección**, **preparación** y **análisis** de los datos al vuelo, es decir, todo de una vez. Así es como trabaja Hadoop, pues al igual que se divide el procesado entre los diferentes nodos del clúster, también se divide la transformación entre ellos, todo debido a que los datos son tan grandes que sería muy costoso moverlo entero.
    a. En la **recolección** se recogen los datos a partir de los datos originales almacenados.
    b. **Preparación** de los datos en donde se procesan los datos almacenados, los cuales se limpian y organizan. La diferencia respecto a la forma tradicional reside en que, aquí, los datos están siendo procesados a partir de los datos en crudo, que nunca se modifican, mediante el uso de un programa, lo que se conoce como «*schema-on-read*». Esto se realiza para cuando los datos son tan grandes que hacen prohibitiva su copia, así, al aplicar el *schema-on-read*, se hace la transformación, la limpieza y la integración de los datos en la etapa de preparación [598], todo esto con los datos al vuelo a partir de los datos originales. De esta manera, se mantienen los datos originales siempre para realizar nuevas consultas.
        i. La fase de **limpieza** hace validaciones de hashes y de formatos, limpieza de duplicados y de campos inválidos, estandarización de los datos e indexado, entre otros [653].
        ii. La fase de **resumen** o ***summarisation***, que es muy importante debido a que cuando se trabaja con muchos datos resulta muy difícil dar una visualización

buena de estos. Por ello, esto fase realiza un resumen para crear subconjuntos que sean característicos de los datos analizados.

c. Se hace el **análisis**, el cual involucra técnicas que producen conocimiento sintetizado a partir de la información preparada, no de la original. Sin embargo, estos métodos no han cambiado, pero si se ha modificado su implementación para que funcionen en entornos de computación paralela. Para ello, se han divido las tareas en subtareas independientes que puedan ser asignadas a cada nodo del clúster por separado y que permitan su recopilación para así juntarlas al final del proceso.
    i. Algunas consideraciones que hay que tener en cuenta en el análisis son:
        1. Procesos de *matching* de metadatos.
        2. Análisis de complejidades: computacional, *Machine Learning*, localización de datos, …
        3. Análisis de latencia: *real-time* o *streaming*, *near real-time* o interactivo, por lotes (*batch*) u *offline*.
        4. Las necesidades humanas, como son el descubrimiento, la hipótesis y el testeo de la hipótesis.
    ii. La **visualización** se realiza en la fase análisis. La visualización tiene tres categorías generales, donde las dos últimas son las más importantes actualmente para ayudar al entendimiento de los grandes volúmenes de datos, siendo la última la más importante actualmente:
        1. Visualización de datos para el entendimiento: navegación, detección de valores atípicos y condiciones límite.
        2. Visualización explicativa para resultados analíticos: confirmación, presentación de datos *near real-time*, interpretación de los resultados.
        3. Visualización aclaratoria para «contar la historia»: reportes, inteligencia de negocio, resúmenes.

11.8 *LA ARQUITECTURA DE BIG DATA*

Como se puede ver en el *survey* de la NIST acerca de las arquitecturas en ***Big Data*** [612], existen infinidad de ellas y con diferentes suplementos, todo depende de la compañía, del tipo de aplicación y de los requisitos, es decir, no hay una arquitectura clara. Por eso, no se puede decir que existe «la arquitectura ideal» pero si se puede hablar de una posible arquitectura genérica, pues esto fue lo que propuso la NIST tras su estudio de las diferentes arquitecturas de las empresas que enviaron la suya.

No obstante, el que pueda existir una arquitectura genérica o con componentes bastante similares no implica que se haya seguido una buena construcción de esta o que este bien construida. Empero si sirve de referencia o de ayuda para montar una arquitectura propia en base al estudio e lo que utilizan las grandes empresas.

11.8.1 ARQUITECTURA GENÉRICA

En el *survey* de la NIST se puede ver las arquitecturas propuestas por diferentes compañías como ET Strategies, Microsoft, la Universidad de Ámsterdam, IBM, Oracle, Pivotal, SAP, 9Sight y LexisNexis. Cada uno propuso una arquitectura para la realización del artículo, siendo todas ellas totalmente diferente, tanto en el concepto, hardware y software utilizado y capas, como en la forma de presentarlas, pero dando las definiciones y explicaciones de cada capa y componente, tanto software, como hardware y conceptual. Así, como se puede ver, este aspecto es bastante ambiguo según el punto de vista y las necesidades de cada empresa. No obstante, como bien apunta la NIST, se puede observar que, a grandes rasgos, todas estas arquitecturas poseen tres capas principales, la gestión y almacenamiento, los análisis, y las interfaces. de las aplicaciones, la infraestructura, y otros elementos.

La **capa de la gestión y almacenamiento** contiene los componentes que proveen capacidades para trabajar con datos estructurados y no estructurados, la gestión de SQL o NoSQL y el sistema de ficheros distribuidos (DFS).

La **capa de análisis** incluye el procesamiento de datos que engloba la Extracción, Transformación y Carga (ETL), bases de datos, análisis de datos, visualización de datos, reportes y la gestión de los datos.

La **capa de la infraestructura** da soporte a la infraestructura y los sistemas con capacidad de integrar componentes internos y externos y la forma de interactuar con los usuarios de la plataforma.

El resto de elementos son los que son diferentes en cada arquitectura y no se engloban en las tres capas anteriores, como son la seguridad, otras herramientas, la generación de grafos, mapeados, traducciones y las estadísticas, entre otros.

En base a este artículo, la NIST propuso su propia arquitectura de referencia abierta con el fin de proveer de un lenguaje común y una referencia a los interesados (*stakeholder*) en ella, estimular el uso de estándares, especificaciones y patrones comunes, proveer métodos consistentes para la implementación de tecnología que resuelven problemas similares, ilustrar y entender diferentes componentes, procesos y sistemas de ***Big Data***, y facilitar un análisis de candidatos estándares para la interoperabilidad, portabilidad, reusabilidad y extensibilidad [368]. Claramente, esta propuesta se basa como se comentó, en el estudio que realizaron en el volumen 5 acerca de las diferentes propuestas enviadas por empresas y de las cuales sacaron puntos en común.

11.8.2 OTRAS ARQUITECTURAS

Además de cómo construir la arquitectura generalmente, también hay que tener en cuenta diferentes principios acerca de cómo construirla correctamente. Acerca de esto, hay diferentes artículos que hablan y dan consejos sobre cómo construir o que cosas tener en cuenta a la hora de construir una arquitectura ***Big Data***.

Estos pueden ser los siete principios propuestos recogidos y explicados de [596], con los que tratan de guiar en el desarrollo y nueva forma de pensar de una arquitectura *Big Data*:

1. **Buenas arquitecturas y *frameworks* como prioridad en el diseño.**
2. **Tener diversidad de métodos analíticos** con el fin de poder analizar y resolver cualquier tarea compleja.
3. **El tamaño no cabe en cualquier solución**: esto implica que habrá que elegir las herramientas adecuadas para la tarea que se requiera y mediante la combinación de diferentes herramientas se podrá conseguir el cometido, pues no siempre una herramienta sirve para cualquier tipo de tarea y hay que centrarse en los beneficios que estas nos otorgan.
4. **Llevar el análisis a los datos**: puede ser que los datos sean tan grandes que sean imposibles de mover y no se pueden dividir para llevarlos al centro de datos. En casos como este, lo aconsejable es llevar el análisis a donde se encuentran los datos.
5. **Realizar el análisis en memoria de acceso aleatorio (RAM)**: los análisis en memoria RAM en vez de en disco, ya sean HDD o SSD, son mucho más rápidos [654], lo que permite acercar las aplicaciones a que sean *real time*.
6. **El almacenamiento de los datos debe de ser distribuible**: los datos almacenados bien en el *Data Warehouse* o en la nube deben de ser distribuibles para así permitir el trabajo con estos datos en cualquier ordenador.
7. **La coordinación es necesaria entre las unidades de procesamiento y de datos**: sin embargo, esta parte ya la traen muchas herramientas por defecto de manera que ofrecen tolerancia a fallos, eficiencia y escalabilidad.

No obstante, estos principios están basados en trabajar sobre memoria, y no sobre disco, pues pretender dar soporte a aplicaciones *real time*. En este caso, hay que tener cuidado pues, a veces puede resultar innecesario al igual que excesivamente costoso, pues puede ser que nuestras aplicaciones no requieran de procesamiento en memoria y baste con un procesamiento en disco. Esto puede ser en el caso de que, aunque nuestra aplicación deba ser lo suficientemente rápida, los datos no sean tan grandes como para requerir este tipo de procesados o bien, no sea una aplicación *real time*. Además, algunos de estos principios ya son otorgados por ciertas herramientas, como pueden ser Apache Hadoop, MapReduce o Apache Storm.

Otros autores, como es el caso de Nathan Marz y James Warren [655], presentaron una arquitectura en tres capas conocida como Lambda: *batch layer*, *serving layer* y *speed layer*. En la primera capa ellos almacenan la copia maestra del conjunto de datos y la dividen en diferentes vistas. La segunda capa es una especia de base de datos distribuida especializada que se carga en estas vistas creadas anteriormente y hace que sea posible las lecturas aleatorias en ellas. La última capa actualiza la primera capa cuando la vista terminó de procesarse. Así, esta última capa es la encargada de asegurar que los nuevos datos son representados tan rápido como son requeridos por la aplicación.

En [656], los autores crearon una arquitectura para crear un sistema escalable y distribuido de procesado de lenguaje natural, pues el procesamiento de grandes cantidades de datos requieren de grandes programas distribuidos en grandes clústeres. Para desarrollar esta arquitectura utilizaron Apache Storm y diferentes

máquinas virtuales que contenían el mismo sistema completo de NLP. Con esta arquitectura buscaron configurar la fuente lingüística y así mejorar la extracción y el descubrimiento de conocimiento automático de enormes cantidades de texto utilizando ***Big Data***. Los experimentos demostraron que hay mucho margen de mejora en este campo cuando se utilizan arquitecturas paralelas.

11.9 *EL COMIENZO: GOOGLE FILE SYSTEM Y MAPREDUCE*

Todo comenzó con Google. Google para poder dar soporte a su buscador y otras herramientas que poseen desarrolló dos tecnologías que son la base de las herramientas utilizadas actualmente en Big Data. Google no tenía en mente crear un nuevo sistema de ficheros, pero en los primeros años de uso, cuando estaban en sus primeras versiones rastreando e indexando las primeras webs, los ingenieros de Google vieron que no tenían otra alternativa que crear un sistema de ficheros [657]. Esto dio lugar al nacimiento de Google File System (GFS) [463], que aparecido en el año 2003. Tras el sistema de ficheros apareció en el año 2004 MapReduce [468]. Google utiliza ambos sistemas en conjunto: el GFS para distribuir los ficheros y MapReduce para procesarlos.

Hay que decir que GFS y MapReduce no fueron los primeros de todos en lo referente a su campo, no obstante, marcaron un punto de inflexión, pues, a partir de ellos, se crearon y basaron diferentes herramientas *open source* de *Big Data* ampliamente utilizados por las compañías y los académicos [454], como han sido Apache Hadoop [586], junto todo el ecosistema que se ha creado alrededor de él, y Phoenix [658].

GFS incluyo mejoras que otros sistemas de ficheros distribuidos no poseían, o bien modificaciones sobre esto de tal manera que GFS tuviera tolerancia a fallos y una rápida recuperación en caso de caída, así como diferentes optimizaciones. Claramente, en algunas cosas no es el mejor, como en las copias frente a un conjunto redundante de discos independientes, más conocido como RAID [174] (*Redundant Array of Inexpensive Disks*).

En lo que respecta a MapReduce ya existían modelos de programación restringida donde se utilizaban esas restricciones para programar en paralelo. MapReduce, según Google, puede ser considerado una simplificación y destilación de lo que existía hasta su fecha, además de haber incorporado cosas nuevas, como la tolerancia a fallos.

Google ha presentado varios artículos de investigación en conferencias acerca de cómo trabaja el GFS y el MapReduce, los cuales han ayudado mucho a la comunidad científica y en el desarrollo de herramientas ***Big Data***. Empero, al menos hasta el año 2008, nunca habló ni permitió entrar a su infraestructura a prensa ni revistas especializadas como *Nature*, lo que imposibilitó ver la infraestructura necesaria que se necesita para correr o copiar su modelo. Es decir, todo lo contrario a otros como el Consejo Europeo para la Investigación Nuclear, *Sanger Institute* o el proveedor de servicios de Internet (ISP) holandés XS4ALL [637].

[174] Glosario: **RAID**

11.9.1 GOOGLE FILE SYSTEM

Google presentó en el año 2003 un sistema distribuido y escalable de ficheros para aplicaciones con un gran uso de datos llamado Google File System (GFS) [463]. Este sistema es tolerante a fallos en sistemas hardware de bajo coste mientras ofrece un alto rendimiento a un gran número de clientes. Este sistema ha servido para cubrir las necesidades de computación del buscador de Google en lo referente a almacenamiento y procesamiento de grandes conjuntos de datos, estando constituido el más grande, en 2003, de cientos de terabytes distribuidos en miles de discos en más de mil máquinas y accedido por cientos de clientes.

11.9.1.1 LAS BASES DE DISEÑO DEL GOOGLE FILE SYSTEM

GFS comparte las mismas metas que los sistemas distribuidos previos a él: el rendimiento, la escalabilidad, la fiabilidad y la disponibilidad. Sin embargo, basaron su diseño en mejorar este tipo de sistemas para así conseguir que el GFS pudiera manejar la carga, tanto actual como la prevista que Google iba a tener. Para diseñar GFS correctamente, Google estudió como debía de ser el sistema en base a cuatro puntos diferentes y ciertas suposiciones. Estos cuatro puntos son el **fallo de los componentes**, los **ficheros de gran tamaño**, la **mutación de los datos** y la **simplificación del sistema de ficheros**.

El **fallo de los componentes** es algo natural debido a que trabajan con cientos o miles de máquinas de almacenamiento montadas con elementos de bajo coste. Así, debido a la cantidad de máquinas, es más fácil que se rompan más, pues a mayor número, más probabilidades existen de que alguna se estropee. Si a esto se le suma que utilizan componentes de baja calidad, el porcentaje de fallo se incremente respecto a si utilizasen componentes de mejor calidad, teóricamente.

Una máquina puede estar no disponible ya sea porque alguno componente no está disponible o bien porque haya tenido un fallo y la máquina no se pudiera recuperar. Estos fallos pueden ser debidos a errores del sistema, de la aplicación, humanos, de memoria, de los conectores, de las redes o de la energía. Esto implica que la supervisión constante para la detección de errores, la tolerancia a fallos y la recuperación automática deba de ser una parte importante del sistema.

El segundo punto es el de los **ficheros enormes**, pues estos son enormes para los estándares tradicionales, siendo lo más común trabajar con gigabytes y terabytes. No obstante, también reconocieron que puede haber millones de pequeños ficheros y por esto deben de ser soportados, pero no optimizados, pues no son la prioridad del sistema.

La **mutación de los datos** se debe a que los ficheros van añadiendo nuevos datos con el tiempo en vez de sobrescribir los datos existentes. Esto se debe a que una vez los ficheros son escritos, se convierten en archivos de solo lectura.

Otra conjetura a la hora de construirlo fue que las grandes cargas de trabajo conllevan grandes escrituras secuenciales que añaden datos a los ficheros, basándose en que los tamaños de las operaciones son similares a sus lecturas. Esto es algo que consideraron de gran importancia y esencial en un sistema de ficheros

distribuidos [175]. No obstante, una vez que se escriben es muy raro que se modifiquen, GFS soporta las escrituras pequeñas, pero sin asegurar su eficiencia.

Por último, la **simplificación del sistema de ficheros** se debe a que se ha relajado la consistencia del modelo con lo que consiguieron que las aplicaciones no sufrieran una carga extra.

Además, el sistema de ficheros debía de soporta la carga concurrente de adición y de lectura de datos de varios clientes sobre el mismo fichero, garantizando la atomicidad de cada operación de cada cliente. En base a esto, el sistema tiene operaciones atómicas que permiten añadir y leer información concurrentemente a un fichero por múltiples clientes sin necesitar sincronización extra entre ello. Esto es llamado operaciones *record append*.

La carga de los trabajos puede ser de dos tipos, de lectura de ficheros grandes de alrededor 1 megabyte en *streaming* o de lectura aleatoria de ficheros pequeños y de pocos kilobytes.

Un detalle importante del sistema, a pesar del pensamiento normal, es que es más importante una red con gran ancho de banda que una latencia baja debido a que las cargas de datos suelen ser muy grandes y rara vez se requiere una lectura o escritura pequeña rápida, algo muy común en ***Big Data***.

11.9.1.2 ARQUITECTURA

Un clúster GFS se compone de seis partes importantes y que se muestran en la Ilustración 49: e los **clientes, e**l **servidor maestro**, el **maestro** «**shadow**», los ***chunkservers***, los **trozos** del fichero, conocidos en inglés como *chunks*, los **metadatos** y los ***Name Spaces***.

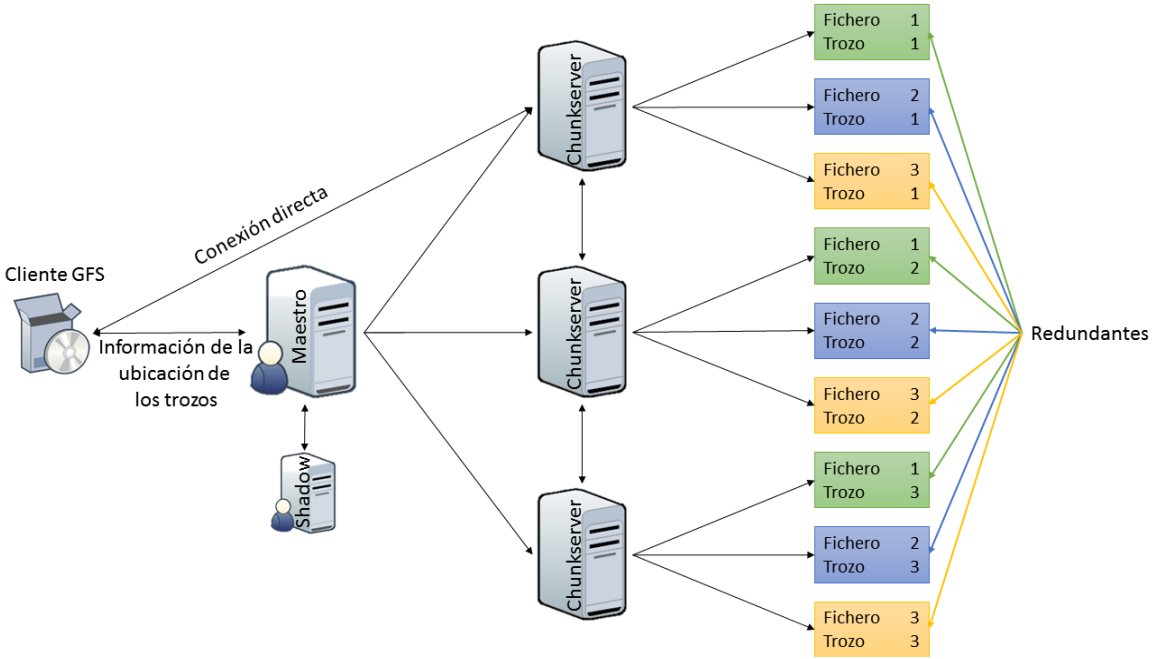


Ilustración 49 Redundancia de los ficheros en la arquitectura del GFS

[175] Glosario: **Sistema de ficheros distribuidos**

El **cliente** es una aplicación, o, mejor dicho, una petición de una aplicación, pues esta podría contener múltiples hilos llamando al sistema GFS, luego, serían múltiples clientes. En el GFS puede haber más de un cliente ejecutándose a la vez. También puede haber una aplicación en un solo **servidor maestro** o bien en varios [657].

Un clúster GFS posee un **servidor maestro** y múltiples ***chunkservers*** que son accedidos por múltiples **clientes**. Normalmente los servidores tienen un sistema operativo Linux. Una opción de montaje y que es posible, es tener en un mismo servidor un ***chunkserver*** y un **cliente**, aunque por rendimiento no es recomendable, según el caso.

La idea inicial era tener un **servidor maestro** por clúster y por centro de datos. No obstante, debido a los requisitos, Google creó lo que llaman «multi-cell». Esto significa que los ***chunkservers*** pueden tener varios **servidores maestros** asignados [657].

El **servidor «shadow»** es una copia del **servidor maestro** que se mantiene para aumentar la fiabilidad del sistema y que posee los logs y *checkpoints* del **servidor maestro**. De esta manera, en caso de que ocurra un error en el **servidor maestro**, el **servidor «shadow»** estará listo para reemplazarlo inmediatamente [657].

Los ficheros que se introducen en el GFS son divididos en múltiples **trozos**, a los cuales se les asigna un ID único y que se almacenan en los ***chunkservers***. No obstante, por motivos de fiabilidad y balanceado, estos trozos son copiados en diferentes ***chunkservers***, por defecto en 3 aunque esto es modificable. Los **trozos**, por defecto, son de 64 megabytes y se guardan en ficheros planos de Linux. Esto se puede ver claramente en la Ilustración 49, donde se ve que hay tres ficheros y cada fichero está copiado tres veces en tres *chunkservers* distintos.

El sistema de ficheros ofrece una interfaz que permite las operaciones de crear, borrar, abrir, cerrar, leer y grabar ficheros. Estos ficheros están organizados jerárquicamente en directorios e identificados por nombres de ruta.

El **servidor maestro** contiene todos los **metadatos** del sistema en memoria. Esto no se cachea por motivos de rendimiento, coherencia y de sincronización, en su lugar, utiliza un registro de operaciones almacenado en los ***chunkservers***. Esto permite tener un servidor maestro más simple, fiable y sin riesgo de inconsistencias. Los **metadatos del servidor maestro** contienen el *namespace* del fichero y de los **trozos** del fichero, la información de control de acceso, el mapeo de los ficheros a **trozos** y la dirección de todas las réplicas de los **trozos**. Cabe destacar que, estos trozos, ocupen lo que ocupen, siempre ocuparán lo mismo en el fichero de metadatos, es decir, un fichero de 64 MB ocupará en disco 64 MB, pero en los metadatos 3 bits, mientras, un fichero de 1MB ocupará en disco 1 MB, pero en los metadatos ocupará también 2 bits. Así, aquí el problema viene dado cuando se tienen muchos ficheros pequeños, lo que hace que el sistema tenga menos cabida para manejar muchos ficheros por esta limitación o sistema de funcionamiento de los **metadatos** [657].

Por otro lado, los ***chunkservers*** no necesitan cachear sus **metadatos** porque tienen los ficheros en local y ya se encarga automáticamente el buffer de su sistema operativo de cachear los datos que sean necesarios. En cambio, los **clientes** si cachean sus **metadatos**, siendo así la única parte de la arquitectura que los cachea.

Otro detalle que simplifica el sistema de ficheros es el poseer únicamente un **servidor maestro** que centralice todo y tome decisiones en base al conocimiento global del sistema. Como contra, esto implicó el minimizar las lecturas y escrituras para no crear un cuello de botella. Por esto, los ***chunkservers*** conocen con quienes se deben de comunicar y quienes contienen las réplicas de sus **trozos**, evitando utilizar el **servidor maestro** para ello, y enviando las escrituras y lecturas directamente a los otros ***chunkservers***. Toda esta información está cacheada en los **clientes**.

Los ***Name Spaces*** permiten ocultar los datos de una aplicación y, así, en caso de que ocupe todo el espacio disponible en una célula de servidores, esta se puede distribuir por un clúster de servidores ocultando la partición exacta de la aplicación. Así, el ***Name Space*** contiene como la aplicación esta particionada a lo largo de los diferentes clústeres [657].

11.9.1.3 IMPLEMENTACIÓN

La Ilustración 50 muestra el ciclo de vida de una petición dentro de la arquitectura del GFS.

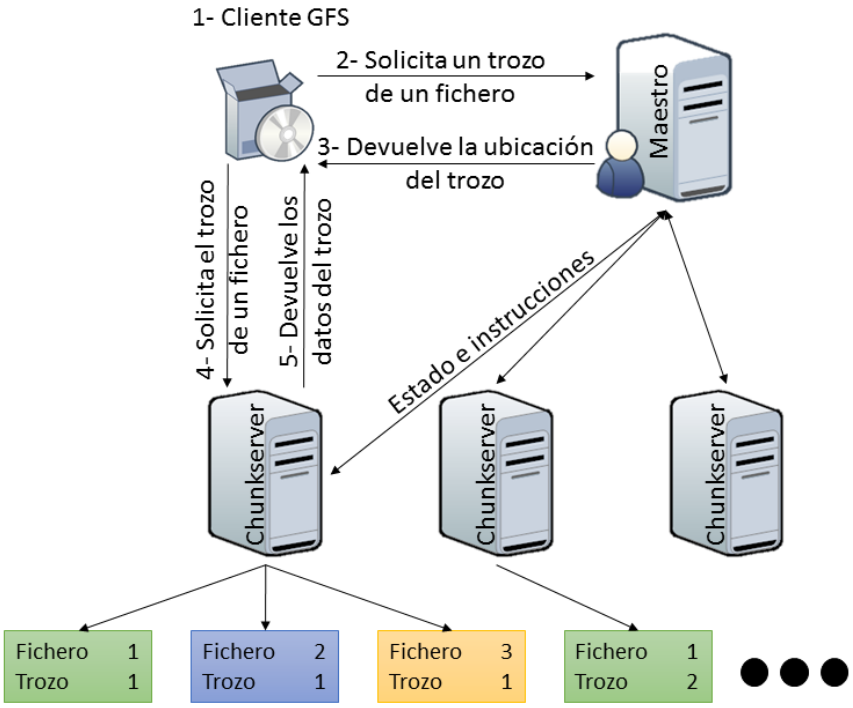


Ilustración 50 Funcionamiento de la arquitectura del GFS

El primer paso de todos es cuando un **cliente** solicita un trozo de un fichero al **servidor maestro**. El **cliente** se comunica con el servidor **maestro** diciéndole el nombre del fichero y el número del trozo de ese fichero con el fin de obtener la dirección del trozo del fichero necesario y así saber a qué ***chunckserver*** consultar.

Tras recibir esta solicitud, el **servidor maestro** responde con el identificador único de ese trozo, conocido en inglés como *chunk handle*, y con las ubicaciones de ese trozo solicitado, pues esta replicado en varios *chunkservers*.

Una vez recibidos los datos, el **cliente** consulta un ***chunkserver*** enviándole el *chunk handle* y el intervalo de bytes. Una vez el ***chunkserver*** recibe estos datos responde con los datos del trozo solicitado.

El **servidor maestro** es quien controla todo el sistema, entre lo que destaca la gestión de todos los **trozos**, desde su movimiento hasta su eliminación. Este **servidor maestro** se comunica continuamente con los ***chunkservers*** utilizando mensajes *HeartBeat* para enviarles las instrucciones, saber su estado y los **trozos** de ficheros que tienen cuando el sistema se inicia o es añadido un nuevo nodo al clúster.

11.9.1.4 ALTA DISPONIBILIDAD

Para conseguir la mayor disponibilidad posible, Google ha diseñado el GFS en base a que tuviera una rápida recuperación y tuviera copias. Así, tanto el maestro como los *chunkserver* han sido diseñados para que se inicien en cuestión de pocos segundos sin importar como hayan terminado.

En lo referente a las copias, por defecto, cada trozo se copia otras dos veces, además del original luego, existen tres copias en total. Estas copias se envían a diferentes servidores en diferentes *racks*. Estas copias ayudan en caso de que un *chunkserver* se caiga o se corrompan los datos de una de las copias.

El servidor maestro también tiene una copia propia por fiabilidad del sistema. Esta copia incluye sus logs y sus *checkpoints*. Cuando se produce un error, ya sea en la máquina o en el disco, el servidor maestro se reinicia instantáneamente y la infraestructura de monitorización del GFS inicia un nuevo servidor maestro en base al registro de operaciones copiado. Como el sistema utiliza un nombre para nombrar al servidor maestro, esto provoca que el sistema actúe de manera similar a un DNS, lo que implica que los clientes no necesiten ningún cambio al utilizar siempre ese nombre para encontrar al servidor maestro. Cabe destacar que este nuevo servidor maestro es una «sombra», luego solo permite acceso de lectura y no de modificación. Esto implica que este servidor «sombra» provea de archivos no actualizados, a diferencia de lo que haría un servidor «espejo». Su nombre dentro de la arquitectura es maestro «shadow».

11.9.2 **MAPREDUCE**

MapReduce fue presentado originalmente en un congreso en el año 2004 de la mano de Google [468]. Posteriormente fue mejorado ligeramente para una revista de *Association for Computing Machinery* (ACM) en el año 2008 [464]. MapReduce es un modelo de programación y una implementación para el procesamiento y la generación de grandes conjuntos de datos. Este modelo lleva siendo utilizado por Google desde el año 1999 para procesar datos en bruto y computar diferentes datos derivados de estos distribuyéndolos entre cientos o miles de máquinas para terminar de procesarlo en un tiempo razonable. Actualmente, es uno de los más importantes los modelos de programación más utilizados para esta tarea [455], [598].

Este modelo de programación permite la paralelización del software automáticamente, lo que permite ejecutar este tipo de programas en clústeres. El sistema se encarga en tiempo de ejecución todo lo necesario para la partición de los datos de entrada, la ejecución del programa en un clúster, la gestión de fallos y la gestión de la comunicación entre las máquinas. Es por esto que permite ejecutar programas en sistemas distribuidos de forma paralela de una forma fácil a programadores sin experiencia en este tipo de sistemas. Además, MapReduce es un sistema escalable que permite la computación de muchos terabytes de datos en

miles de máquinas. La prueba de esto es la propia Google, quien lo utiliza para muchos y muy diferentes propósitos dentro de la empresa.

También cabe destacar que MapReduce es muy flexible, debido a que puede utilizarse en muchos tipos de memoria diferente, desde una máquina con una pequeña memoria compartida hasta un gran sistema multiprocesador con Acceso a Memoria No Uniforme, conocidas en inglés como *Non-uniform memory access* (NUMA).

Algunos casos en los que se puede utilizar MapReduce son, por ejemplo, en el uso distribuido de la utilidad Grep, contar la frecuencia de acceso de una URL (*Uniform Resource Locator*) procesando su log, hacer el opuesto de un grafo de la web, resumir las palabras más importantes de un documento, crear un *Inverted Index* [176] para los algoritmos de indexado de los motores de búsqueda, una ordenación de elementos distribuida, en problemas de *Machine Learning* a gran escala, problemas de *clustering*, computación de grafos a gran escala, extracción de propiedades de páginas web, extraer datos utilizados para crear reportes de consultas populares, o Minería de datos.

11.9.2.1 MOTIVOS

Google necesitaba computar grandes volúmenes de datos de una forma rápida. Ejemplos de esto son los datos en bruto que conseguían, como podían ser los documentos o los logs, con el fin de obtener datos importantes derivados de estos, como puede ser la representación de la estructura de los documentos, resúmenes de las páginas inspeccionadas, o las búsquedas realizadas en el buscador.

Con el fin de lidiar con esta complejidad, los ingenieros de Google diseñaron un sistema que permitiese la abstracción de programas que trabajasen en paralelo, así como los fallos de tolerancia, la distribución y el balanceo de carga de este tipo de aplicaciones. Para conseguir esto, diseñaron una librería inspirada en las primitivas *map* y *reduce* de Lisp y otros lenguajes de programación funcionales.

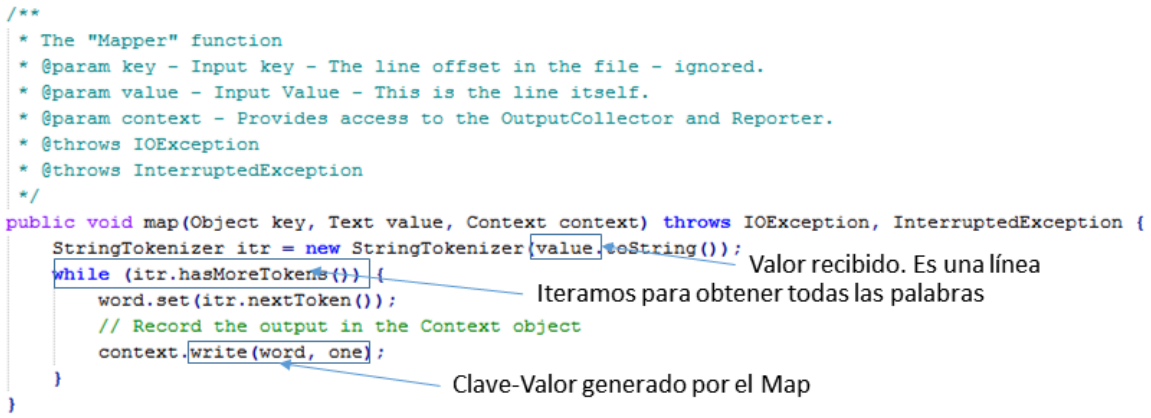
El motivo de inspirarse en el *map* y *reduce* fue debido a que se dieron cuenta que la mayor parte de la computación que ellos requerían se podía resolver aplicando un *map* a cada registro de entrada para crear un conjunto intermedio de datos clave/valor y después aplicar un *reduce* para combinar todos los valores que compartiesen la misma clave.

11.9.2.2 FUNCIONAMIENTO

El funcionamiento de MapReduce se basa en dos funciones que forman su nombre, *map* y *reduce*, y de las cuáles es encargado el usuario de definir que desea que están hagan. La función *map* se utiliza para procesar la entrada de datos y dividirlos en un conjunto de datos intermedio basado en pares clave/valor. La función *reduce* junta todos estos datos intermedios agrupándolos en base a su clave, es decir, todos los que tengan la misma clave son fusionados. Básicamente se puede decir que MapReduce se basa en el método de divide y vencerás [596].

[176] Glosario: ***Inverted Index***

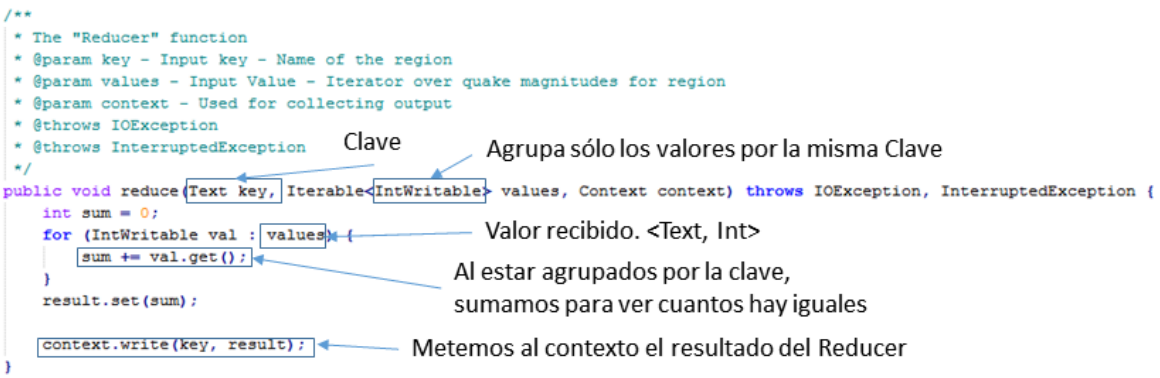
A continuación, se muestran dos ejemplos, uno de la función *map* y otro de la función *reduce*, implementados utilizando el *framework* Hadoop y su implementación de MapReduce. En el ejemplo se cuenta el número de repeticiones de las palabras en un texto dado. Este es el clásico ejemplo utilizado para aprender a utilizar MapReduce, comúnmente llamado el «Hola Mundo». Este ejemplo fue el dado por Google en su artículo original.




```
/**
 * The "Mapper" function
 * @param key - Input key - The line offset in the file - ignored.
 * @param value - Input Value - This is the line itself.
 * @param context - Provides access to the OutputCollector and Reporter.
 * @throws IOException
 * @throws InterruptedException
 */
public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        // Record the output in the Context object
        context.write(word, one);
    }
}
```

Ilustración 51 Función Map

En la Ilustración 51 se muestra un ejemplo de la implementación de la función *map* en el lenguaje de programación Java. Aquí, la función *map* recibe una línea completa e itera sobre cada palabra de la línea. En cada iteración la función *map* graba en el contexto de la aplicación una 2-tupla, o dupla, compuesta de la clave otorgada al objeto, que es la palabra en sí, y un valor, que en este caso es el valor IntWritable «one», equivalente al valor entero 1. Se utiliza como clave la palabra debido a que es lo que se quiere contar y por eso se requiere que sea un identificador único.




```
/**
 * The "Reducer" function
 * @param key - Input key - Name of the region
 * @param values - Input Value - Iterator over quake magnitudes for region
 * @param context - Used for collecting output
 * @throws IOException
 * @throws InterruptedException
 */
public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
        sum += val.get();
    }
    result.set(sum);

    context.write(key, result);
}
```

Ilustración 52 Función Reduce

La Ilustración 52 muestra un ejemplo de la función *reduce* en el lenguaje de programación Java, finalizando del ejemplo anterior. La función *reduce* es la encargada de combinar todos los elementos que tengan la misma clave, en este caso, las mismas palabras. En el caso que ocupa, la función *reduce* suma uno por cada clave existente, pues el valor que se metió en la función *map* era 1, es decir, suma uno por cada palabra que sea la misma, o, dicho de otro modo, devuelve el número total de palabras iguales.

En MapReduce, las claves y valores de la entrada son de diferente contexto que las claves y valores intermedios y de la salida, haciendo que los datos de entrada puedan ser procesados en paralelo por diferentes máquinas.

Además de estas dos funciones, también se puede utilizar la función *combiner* en caso de que se quiera hacer una mezcla parcial de los datos antes de que estos lleguen al *reduce*, aunque habitualmente suelen usar el mismo código. El *combiner* se ejecuta en cada máquina que lleva a cabo una tarea *map*. La diferencia entre un *combine* y un *reduce* está en cómo el MapReduce trata la salida de ambas funciones, pues en el *combiner* la salida se almacena en el sistema de ficheros intermedios y en el *reduce* en el sistema de ficheros global. Además, esta combinación parcial acelera ciertas operaciones en MapReduce y permite ordenarlos antes de que sean enviados al *reduce*.

11.9.2.3 IMPLEMENTACIÓN

Un clúster que ejecute MapReduce se compone de varias partes, a saber: el programa realizado por el usuario, el nodo maestro, los ficheros de entrada, los *map worker*, los ficheros intermedios, los *reduce worker* y los ficheros de salida. Los servidores *map* y *reduce* son conocidos como *worker*.

El nodo maestro es el encargado mantener todos los datos del estado de cada tarea *map* y *reduce*. Los estados son *idle*, *in-progress* y *completed*. Además, también es el conducto por el que se propaga la ubicación de los archivos intermedios de las tareas *map* a las tareas *reduce*. Así pues, almacena la ubicación y el tamaño de los ficheros intermedios. Esta información es enviada cuando la función *map* termina y se va enviando de forma incremental a los servidores que tienen en proceso la función *reduce*.

Las invocaciones *map* están distribuidas por diferentes máquinas para que el conjunto de datos de entrada sea dividido automáticamente en conjuntos clave-valor. Las invocaciones *reduce* están distribuidas para hacer particiones según la clave asignada a los datos en la función *map*.

En la Ilustración 53 se muestra cómo funciona el sistema MapReduce. El proceso MapReduce se compone de 6 fases.

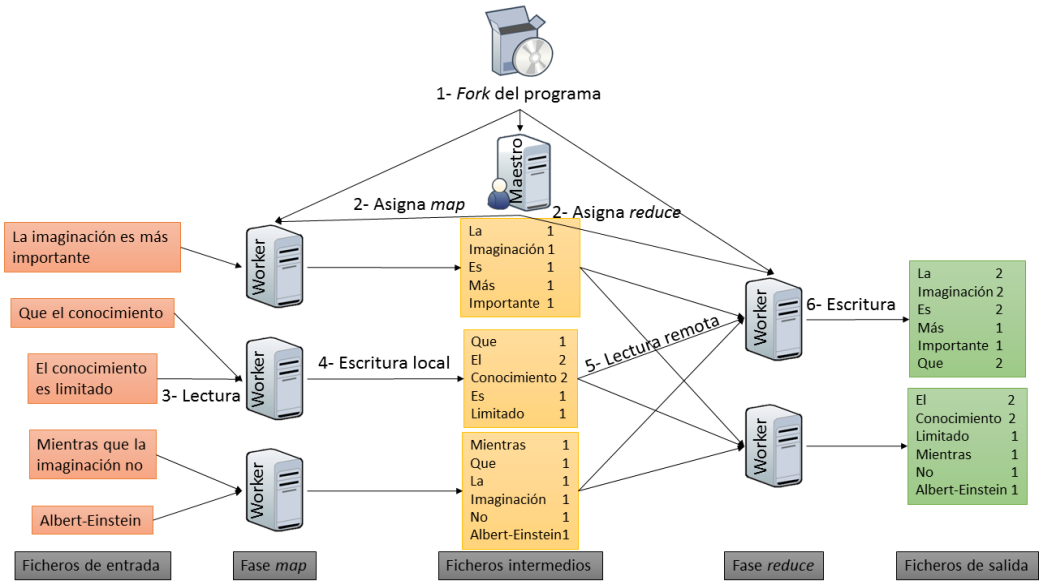


Ilustración 53 Ciclo de vida de MapReduce

En la primera fase se distribuyen copias del programa entre los diferentes elementos de procesamiento, es decir, entre el nodo maestro, los *map worker* y los *reduce worker*. También se dividen los ficheros de entrada en paquetes de entre 16 y 64 megabytes, aunque esto es configurable.

En la segunda fase entra en juego el servidor maestro. La copia del programa del usuario del servidor maestro es especial, pues la copia del resto de *workers* es asignada por el servidor maestro, que escoge entre los *workers* en estado *idle*, es decir, los que no están haciendo nada, y les asigna una tarea *map* o *reduce*.

En la tercera fase los *map workers* leen los datos de entrada correspondientes. Analizan los ficheros de entrada pasan los datos de estos por la función *map* para obtener las clave-valor en base a la función *map* programada por el usuario y almacenar la salida de esa función en los ficheros intermedios.

En la cuarta fase está la escritura al disco local de cada *map worker*. Esta escritura se hace periódicamente. La ubicación de los pares clave-valor se le pasa al servidor maestro debido a que este es el responsable de enviar esa ubicación a los *reduce workers*.

La quinta fase contiene la lectura remota de los *reduce workers*. Cuando un *reduce worker* recibe una notificación del servidor maestro con la ubicación temporal de los ficheros intermedios y así leerlos. Una vez leídos, ordena los datos según su clave para agrupar todos los que tengan la misma clave juntos. En el caso de que los datos fueran demasiado grandes para caber en memoria, habría que utilizar una ordenación externa.

Tras esto, en la sexta fase, los *reduce worker*s iteran sobre las claves intermedias ordenadas, que son únicas. Esta clave junto todo el conjunto de datos que posee son enviadas a un *reducer worker* para que los pase por la función *reducer* definida por el usuario. Los resultados de esta función se adjuntan al archivo de salida. Existe un archivo de salida por *reduce worker*. Aun así, los usuarios no necesitan combinar estos ficheros. Lo que normalmente se suele hacer es pasar estos ficheros por otro MapReduce o utilizarlos en alguna otra aplicación distribuida que pueda trabajar con ficheros divididos en varios archivos.

Cuando todas las fases *map* y *reduce* han terminado, el servidor maestro despierta el programa del usuario. Esto significa que el MapReduce devuelve el control al programa del usuario que puede seguir realizando acciones si así lo requiriese.

Cabe destacar que MapReduce también tiene un modo en el que permite ejecutar todo el proceso secuencialmente en el mismo nodo con el fin de encontrar posibles errores en la implementación. Este modo se conoce como *standalone*.

11.9.2.4 TOLERANCIA A FALLOS

El nodo maestro también es el encargado de gestionar el buen funcionamiento de los nodos y llevar a cabo la tolerancia de fallos. El nodo maestro se comunica continuamente con los *workers* para saber si siguen vivos o no. En caso de que no reciba la contestación de estos, da el trabajo asignado a ese servidor como fallido, resetea el estado a *idle* de ese paquete y le manda el trabajo a otro *worker*.

La única diferencia entre las operaciones *map* y *reduce* completadas en casos de fallo es que las tareas *map* deben reejecutarse al completo porque estas se almacenan en los discos locales del *worker* que ejecutaba

esa tarea, lo que implica que sean inaccesibles en caso de fallo. Mientras, las tareas *reduce* no hace falta reejecutarlas porque se almacenan en el sistema de ficheros global.

Google comprobó que tiene mucha tolerancia a fallos cuando una vez tuvieron 80 máquinas sin acceso durante muchos minutos y MapReduce reasigno todo el trabajo a las que seguían vivas.

En caso de fallo del maestro se puede iniciar una nueva copia de este gracias a que se van guardando *checkpoints* continuamente. No obstante, al menos en la primera implementación, el funcionamiento en caso de fallo del maestro era el de abortar el trabajo MapReduce en caso de fallo.

11.10 *HADOOP*

Apache™ Hadoop® [586] es un *framework* software multiplataforma desarrollado en Java *Open Source* con licencia Apache License 2.0. Actualmente Hadoop está mantenido por la Apache Foundation, aunque su mayor contribuyente ha sido Yahoo!, pues fue su creador, basándose en el GFS para ello.

Ilustración 54 Imagotipo de Hadoop

Este *framework* da soporte para crear un sistema de computación distribuida que sirve para gestionar volúmenes grandes de datos siendo escalable, teniendo detección de fallos y ofreciendo una alta disponibilidad, cumpliendo así lo mismo que Google File System, además de implementar el modelo de programación MapReduce. Debido a que Hadoop permite la ejecución de computación distribuida, este puede ejecutar cualquier tarea por lotes ofreciendo así una gran capacidad de cómputo. Hadoop es capaz de soportar clústeres pequeños como incluso un clúster con más de 1100 nodos con 8800 núcleos en total y más de 12 petabytes de datos crudos, que es el caso de Facebook [586].

Hadoop fue creado por Doug Cutting y Mike Cafarella cuando coincidieron en Yahoo!, y fue nombrado bajo este nombre debido al elefante de juguete del hijo de Doug Cutting [577]. Una de las primeras versiones, la 0.1.0, data del 1 de abril de 2006 [659]. Para la creación de Hadoop se basaron en los artículos de Google sobre su Google File System y MapReduce. La intención principal de la creación de Hadoop era que diese soporte al motor de búsqueda Nutch. Para ello, Hadoop es capaz de coordinar un grupo de servidores de bajo coste [577], al igual que ocurre con GFS.

Hadoop es el *framework* de MapReduce más utilizado de entre los *frameworks* existentes de archivos y computación distribuida [455], [596]. Esto se debe a que se puede utilizar con hardware normal y barato combinándolo con y otros *frameworks* que trabajan con él, siendo todos de código abierto [577]. Empero hay muchas otras compañías que ofrecen la suya propia en la nube como son Amazon con Elastic MapReduce (EMR) [660], Microsoft Azure con HDInsinght [661], Infochimps [662], IBM BigInsights [663] o Hadoop Loader de Oracle [579].

Ha tenido una adaptación muy buena tanto en la industria empresas, como por el mundo académico [454], [625]. Actualmente es utilizado por muchas empresas, entre las cuales destacan algunas muy importantes. Algunas de estas empresas son Adobe, Alibaba, eBay, Facebook, Last.fm, LinkedIn, Twitter, Yahoo!,

Amazon, Microsoft, Spotify, Telefónica y diversas universidades, entre muchas más que pueden ser consultadas en su página web [586].

Hadoop se puede utilizar para casi cualquier cosa. Las anteriores empresas lo utilizan para almacenar y procesar datos, dar soporte a motores de búsqueda, optimizar búsquedas, almacenar logs, realizar análisis de datos, *Machine Learning*, calcular gráficas, realizar informes, buscar relaciones entre datos, filtro de spam, análisis de clics, sistemas de recomendación, y un sinfín de cosas más. Incluso, en algunos casos se está trabajando en su integración con bases de datos relacionales para garantizar el poder combinar todos los tipos de datos, los estructurados y los no estructurados, como hace Microsoft [595].

11.10.1 ARQUITECTURA

La arquitectura de Apache™ Hadoop® es bastante similar a la del GFS [586], pues es en quien está basado. No obstante, no se ha mostrado cómo funciona internamente el GFS, pero si se conoce como es el interior de Hadoop. Hadoop se compone de un nodo maestro y varios nodos esclavos. Requiere Java 1.6 o mayor para funcionar, no obstante, a partir de la versión 2.7.1 requiere una versión de Java 1.7 o mayor. Además, requiere Secure SHell (SSH) para conectar los nodos entre sí.

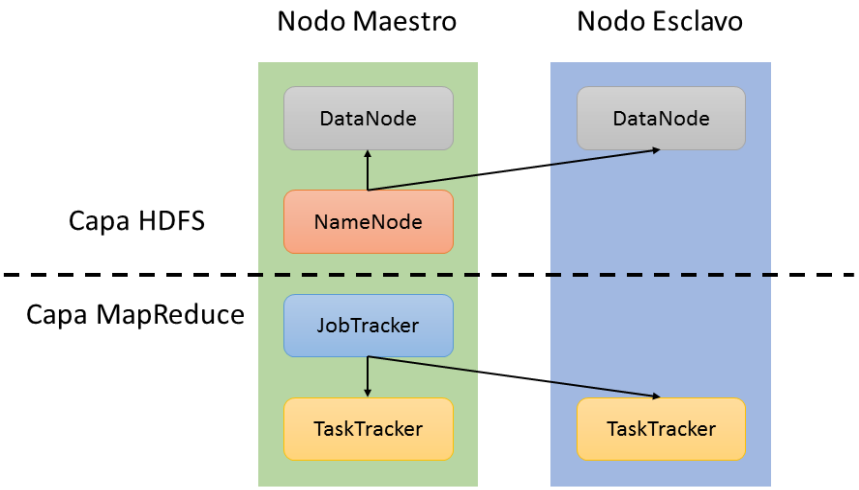


Ilustración 55 Arquitectura de Hadoop

Como muestra la Ilustración 55 cada nodo tiene dos capas, la de almacenamiento, conocido como Hadoop Distributed File System (HDFS), y la de procesamiento, que es el MapReduce.

El nodo maestro contiene en capa dos procesos. En la capa de almacenamiento tiene el **NameNode**, que es el encargado de gestionar los datos, y el **DataNode**, que es el encargado de almacenar los datos. En la capa de procesamiento se encuentran el proceso encargado de gestionar los componentes y enviar las tareas MapReduce, conocido como **JobTracker**, y el **TaskTracker**, que obedece al JobTracker y ejecuta las tareas MapReduce que le encarga.

Cada nodo esclavo se compone de un **DataNode** para gestionar la parte del HDFS y de un **TaskTracker** para manejar la parte del MapReduce.

11.10.1.1 JOBTRACKER

El **JobTracker** es el proceso encargado de gestionar los trabajos MapReduce y los recursos del clúster. El funcionamiento de este proceso es el siguiente:

1. El JobTracker recibe el trabajo de la aplicación cliente.
2. Este proceso se comunica con el NameNode para localizar los datos.
3. Localiza nodos TaskTracker con ranuras disponibles o cercanas a los datos.
4. Envía el trabajo a los nodos TaskTracker elegidos.
5. Monitoriza los nodos TaskTracker.
    a. Si no recibe «noticias» de un nodo TaskTracker lo dará por fallido y reprogramará ese trabajo para otro nodo TaskTracker.
6. Actualiza el estado cuando se completó el trabajo.
7. La aplicación cliente puede preguntar al JobTracker por el estado del proceso.

El principal problema de este proceso es que es si se cae, todos los trabajos en ejecución se detienen.

11.10.1.2 TASKTRACKER

Los TaskTracker son los nodos que aceptan tareas del nodo JobTracker. Las operaciones que pueden realizar estos nodos son *Map*, *Reduce* y *Shuffle*.

Cada nodo de estos tiene configurado un número de slots que se corresponden con el número de trabajos que pueden aceptar. No obstante, tienen preferencia los nodos con un DataNode que contenga datos de ese trabajo. En caso contrario busca un slot en un nodo del mismo rack.

Para realizar el trabajo crea un proceso separado de forma que evita que se caiga el TaskTracker si el proceso falla, manteniendo así una alta tolerancia a fallos. Cuando termina de realizar el proceso, el TaskTracker notifica el JobTracker de que lo terminó. Este mensaje es diferente, pues los TaskTracker deben de enviar cada pocos minutos un mensaje del JobTracker para notificarle que sigue vivo y así evitar que este piense que murió y le dé su trabajo a otro nodo. En estos mensajes de notificación se adjuntan el número de slots libres.

11.10.1.3 DATANODE

El DataNode es el encargado de almacenar los datos en el sistema HDFS. Este proceso se conecta y le responde peticiones al NameNode. Normalmente un sistema funcional suele tener más de un DataNode con varias copias de los datos distribuidos por diferentes DataNodes.

Las aplicaciones cliente y los trabajos MapReduce hablan directamente con los DataNode una vez estas recibieron la localización del NameNode.

Se recomienda por cuestiones de rendimiento que los DataNode compartan nodo con los TaskTracker, pues de esta manera ambos procesos están cerca y se consigue que el almacenamiento este cerca de las operaciones MapReduce. La configuración ideal según la página de Hadoop es tener 1 disco físico y un

TaskTracker por CPU, de tal manera que usarían así el 100% de la CPU. Además, recomiendan no utilizar el protocolo Sistema de Archivos de Red, conocido originalmente como Network File System (NFS).

Los DataNodes pueden hablar entre ellos para mantener las copias de los datos. Por defecto el número de copias es de 3. Con esto, al igual que pasa en el GFS, se evita el tener que utilizar discos RAID ya que proporciona tolerancia a fallos.

11.10.2 **MÓDULOS**

El proyecto Apache™ Hadoop® se encuentra dividido en cuatro grandes módulos [586]. Estos módulos son mantenidos y actualizados por separado según se mejoren o requieran en cada nueva actualización del proyecto. Esto permite que en cada actualización del proyecto solo se toque aquello que sea necesario permitiendo así que pueda mejorar cada módulo independientemente sin tocar los restantes. Estos cuatro módulos son Hadoop Common, HDFS, Hadoop YARN y Hadoop MapReduce.

HDFS y MapReduce al principio eran el mismo proyecto, pero estos se separaron en dos proyectos diferentes en julio del año 2009.

Además de estos cuatro módulos, hay muchos otros proyectos relacionados con Hadoop como Ambari™, Avro™, Cassandra™, Chukwa™, HBase™, Hive™, Mahout™, Pig™, Spark™, Tez™ y Zookeeper™.

11.10.2.1 HADOOP COMMON

Previamente era conocido como Hadoop Core hasta el julio del año 2009. Contiene las librerías y utilidades necesitadas por otros módulos de Hadoop. Por ejemplo, este contiene todos los Java ARchive (JAR) y scripts necesarios para arrancar Hadoop.

11.10.2.2 HADOOP DISTRIBUTED FILE SYSTEM

Hadoop utiliza un sistema de ficheros distribuido y escalable llamado Hadoop Distributed File System (HDFS) que está basado en el GFS y está escrito en Java. Este sistema es el encargado de distribuir los ficheros y replicarlos por los diferentes módulos. También ofrece la opción de distribuir los ficheros vía URL, siendo un ejemplo de esto el sistema de almacenamiento Amazon S3 [664].

Este sistema se configura mediante los diferentes ficheros XML que posee para esta labor y que permiten que sea altamente configurable.

El HDFS de Hadoop soporta diferentes comandos Shell, permitiendo así utilizar comandos de entornos UNIX® para realizar búsquedas, crear, borrar, mover y editar carpetas y ficheros, y muchas más cosas.

11.10.2.3 HADOOP YARN

Hadoop YARN (*Yet-Another-Resource-Negotiator*), conocido también como NextGen MapReduce, es un *framework* para planificar tareas y gestionar los recursos del clúster. Ofrece un interfaz web formado por APIs REST que dan acceso al clúster, a los nodos, a las aplicaciones y al histórico de las tareas realizadas.

Fue introducido en la versión 2.0.0 el 23 de mayo de 2012, aunque ya había estado previamente en la 0.23.0 del 11 de noviembre de 2011, para separar las funciones del JobTracker en dos demonios [177] independientes. El primero para gestionar los recursos y el segundo para la planificación y monitorización de las tareas, los cuales se conocen respectivamente como ResourceManager y ApplicationMaster.

11.10.2.4 HADOOP MAPREDUCE

Hadoop está basado en la implementación MapReduce presentada por Google. Para trabajar con MapReduce, Hadoop proporciona una API que permite trabajar con el de una forma fácil [455]. No obstante, hay muchas otras aplicaciones que permiten trabajar a más alto nivel con MapReduce, como son Hive y Pig.

Este módulo soporta la programación en diferentes lenguajes de programación como Java, Python, Ruby o C++.

11.10.3 **MODOS DE FUNCIONAMIENTO**

Apache™ Hadoop® posee tres posibles modos de funcionamiento [586]: *Standalone*, pseudodistribuido y totalmente distribuido.

El modo ***standalone*** es el modo por defecto de Hadoop. Este modo no está distribuido y se ejecuta todo el proceso sobre un mismo ordenador bajo un mismo proceso Java. Este es el modo más útil y el recomendado para depurar las aplicaciones en busca de problemas y para comprobar su correcto funcionamiento.

El modo **pseudodistribuido**, *pseudo-ditributed* en inglés, es un modo distribuido, pero bajo una misma máquina. En este proceso se ejecuta cada demonio de Hadoop en procesos Java separados.

El modo **totalmente distribuido**, *fully-ditributed* en inglés, es el modo normal utilizado para ejecutar la aplicación en producción. En este modo la aplicación se distribuye en varios procesos Java por todos los nodos del clúster.

11.11 *OTRAS TECNOLOGÍAS BIG DATA*

Las herramientas necesarias para trabajar con ***Big Data*** han mejorado notablemente en los últimos años. Además, estas, no son caras y suelen ser *Open Source*. Sin embargo, necesitan nuevas habilidades y una integración relevante con los flujos de datos internos y externos de las empresas [577], siempre y cuando se adapten a lo que necesiten o ayuden en esa función.

En esta subsección se hablará de los diferentes tipos de aplicaciones que existen, primeramente, de tres grandes grupos, como son los sistemas de computación distribuida, de las distribuciones de Hadoop más utilizadas e importantes, y de las Bases de Datos NoSQL. Tras esto, se hablará de otras tecnologías del ecosistema *Big Data* muy relevantes y utilizadas.

[177] Glosario: **Demonio**

11.11.1 SISTEMAS DE COMPUTACIÓN DISTRIBUIDA

GFS y Hadoop no son los únicos sistemas distribuidos que han existido. Como bien han anotado en el artículo de GFS, este se basó en muchos otros previos intentando mejorar u ofrecer lo que se supone que debía ser un sistema distribuido de archivos. Sin embargo, Hadoop no fue el único, pues otras compañías crearon otros sistemas de computación distribuida, aunque algunos al final han cambiado el suyo propio por Hadoop.

Dryad fue un motor de ejecución distribuida de propósito general para ejecutar aplicaciones en paralelo creado por Microsoft en el año 2007 [665]. Fue utilizado por Microsoft hasta que se cambiaron a Hadoop. Dryad fue diseñado de forma que pudiese ser utilizado tanto en clústeres pequeños como en clústeres de miles de computadores de forma fácil, pues se encarga también de cualquier problema que surgiese, como es la planificación, los fallos y el transporte de los datos.

DryadLINQ es otro sistema de computación distribuida que incorporada extensiones para facilitar la creación de programas de este tipo de modelo de programación y que fue desarrollado por Microsoft [666]. Este combinaba **Dryad** con el lenguaje de .NET Language-Integrated Query (LINQ) [667]. De esta manera, añadieron a Dryad la opción de poder utilizar programación declarativa y un modelo de datos fuertemente tipado.

Microsoft desarrolló y presentó **Cosmos** en el año 2008 en [668], un sistema de computación distribuido. Con Cosmos pretendieron dar soporte a su negocio de búsquedas y publicidad. Cosmos fue diseñado para correr en clústeres de cientos de servidores básicos. Los objetivos de su creación fueron crear un sistema con alta disponibilidad, fiabilidad y escalabilidad, buen rendimiento y de bajo coste. Además, proveían de un lenguaje de alto nivel llamado SCOPE para escribir los trabajos.

Por otro lado, Facebook creó **Haystack** [638], un sistema de almacenamiento de grandes volúmenes compuestos de imágenes de pequeño tamaño. Cuando lo presentaron, en el año 2010, el sistema tenía almacenados 260 millardos de imágenes, o lo que es lo mismo, 20 petabytes de datos. El sistema aguantaba 60 terabytes de subidas a la semana y servía 1 millón de fotos por segundo en su momento álgido. Haystack tuvo como objetivos proporcionar un alto rendimiento, una baja latencia, tolerancia a fallos, ser económico y ser simple. Al contrario que Hadoop, Haystack utiliza el protocolo Network File System (NFS).

11.11.2 DISTRIBUCIONES CON HADOOP

Como bien se ha visto, Hadoop es un sistema con muchos elementos a su alrededor. No obstante, algunas compañías han creado su propio ecosistema basándose en Hadoop como piedra angular. Algunas de estas distribuciones que cuentan con su propia versión de Hadoop son Amazon con Elastic MapReduce (EMR) [660], Microsoft Azure con HDInsinght [661], Cloudera [669], Infochimps de CSC [662], IBM BigInsights de IBM [663] o Hadoop Loader de Oracle [579].

Amazon Elastic MapReduce o EMR es el servicio web que proporciona Amazon Web Services (AWS) para procesar grandes cantidades de datos. Para esto, utiliza Apache Hadoop, del cual ofrece varias opciones, el original o uno modificado por ellos y que supuestamente funciona mejor con los

Ilustración 56 Imagotipo de EMR

servicios de AWS. Para esto, ejecuta una instancia del EC2 automáticamente y la borra tras terminar el trabajo. Para almacenar los datos utiliza el servicio S3, o bien se puede utilizar la base de datos DynamoDB. Para ejecutar el trabajo de Hadoop ofrece una serie de formularios junto con una monitorización gráfica de todo el proceso. En este proceso permite incluir otros *frameworks* como Pig, Hive, Mahout, Hue, Phoenix, Zookeeper, Spark, Presto, HBase, Ganglia, Tez, Sqoop, Oozie, Zeppelin y HCatalog. También incorpora la posibilidad de bajarse el Kit de Desarrollo de Software (SDK) y programáticamente hacer aplicaciones en Java, Node, PHP, Python, Ruby o .Net. Además, permite el uso de otros servicios de AWS y solo cobran por lo computado según el clúster elegido.

Microsoft ofrece un servicio similar pero llamado **HDInsight** en su nube Microsoft Azure. Este sirve para procesar grandes volúmenes de datos utilizando únicamente la versión oficial Apache Hadoop. Para el almacenamiento utiliza clústeres que crea en la configuración mediante formularios del entorno y que siguen activos hasta que se mandan parar, a diferencia de EMR. HDInsight ofrece también Hive preinstalado y listo para utilizar, interfaz gráfica o por escritorio para cada instancia, un SKD para Java y .Net e integración con herramientas de Microsoft como son Excel y PowerShell, así como con otras herramientas del Microsoft Azure. También da soporte a otros *frameworks* del ecosistema Apache Hadoop como son Pig, Hive, HBase, Oozie, Zookeeper, Mahout, Storm y Spark, además de R.

Cloudera es una compañía que ha desarrollado su propia distribución de código abierto basada en Hadoop. Resta distribución es ofrecida para ser descargada mediante una máquina virtual o un Docker. Esta distribución viene con Hadoop y otras herramientas instaladas como son Apache Impala. La distribución además incluye herramientas de monitorización del clúster, gestión y control, permitiendo agregar, modificar y eliminar servicios y nodos desde ellas.

11.11.3 BASES DE DATOS NOSQL

Al principio, la mayoría de los datos estaban estructurados, lo que hacía que pudiesen ser guardados de forma consistente utilizando modelos relacionales y realizar búsquedas en base a un campo común, como puede ser el código postal, gracias a que todo seguía una cierta estructura. Con el tiempo, los datos no estructurados, como imágenes, videos, datos relacionados y otros textos, aparecieron, ganando en volumen e importancia, siendo esto un problema debido a la falta de esta estructura [595], [598].

Debido a este tipo de datos, las bases de datos modernas comenzaron a soportarlos para así permitir que se pudieran analizar, indexar y procesar de una forma directa, por medio del uso de extensiones de SQL no estándar. Estas bases de datos se conocen como bases de datos **NoSQL**, también conocido como **modelos no relacionales** (***Non-relational models***), debido a que hacen referencia a los modelos de datos lógicos que no siguen el esquema de álgebra relacional [311], tanto para el almacenaje como para la manipulación de datos [598]. Por este motivo, las bases de datos NoSQL tienen diferentes aplicaciones que las SQL, además de tener sus pros y sus contras. A pesar de esto, en algunos casos todavía se usa el concepto de tablas [598]. Por ejemplo, las BBDD NoSQL se suelen utilizar para almacenar datos sin categorizarlos ni analizarlos, dando lugar así a una amplia variedad de datos, mientras que las BBDD SQL necesitan una estructura muy bien definida para asegurar la consistencia y la validación de tipos [579].

Las primeras BBDD no relacionales llevan desde 1960 entre nosotros [670], exactamente las de grafos y las orientadas a objetos. Dentro de las propias NoSQL también existen diferentes tipos BBDD según su forma de funcionamiento interno o hacia lo que están dirigidas [598], [625], [670]–[672]: las documentales, las clave-valor, las de almacenamiento en columnas y las de grafos.

En las **bases de datos documentales** las filas no tienen que tener el mismo número de columnas, siendo esta una de las diferencias entre los modelos relacional y no relacional. Además, para realizar las búsquedas se basa en el clave-valor o por el contenido del documento, pues este importa y por ello se suele almacenar en JSON o XML. La devolución de la consulta es un documento con estructura, por ejemplo, un JSON. A este grupo pertenecen CouchBD, IBM Lotus Dominom MongoDB y RavenDB.

Las **bases de datos clave-valor** recuperan los valores almacenados a través de su clave, luego, es como una tabla hash. Estas BBDD ofrecen una alta concurrencia y tienen una estructura muy simple. Algunos ejemplos son BerkleyDB, DynamoDB [673], Redis, Riak y Voldemort.

Las **BBDD de almacenamiento en columnas** son parecidas a las clave-valor, pero con la diferencia de que tienen dos niveles de profundidad, luego, bajo la primera clave se almacenan unas segundas claves. Estas se inspiraron en la BigTable de Google. La ventaja de estas es que son más adecuadas para la agregación de datos y así expresar relaciones entre los datos. Además, este tipo de BBDD utilizan tablas a pesar de que no soportan la asociación de estas. Estas suelen ser consideradas dentro de las BBDD de clave-valor. Ejemplos de este tipo de BBDD son Accumulo, BigTable [674], Cassandra [675] y HBase.

Las **bases de datos en grafo** representan la relación entre los elementos mediante el uso de nodos y su relación correspondiente con los otros nodos, llamada link o arco, que pueden ser unidireccionales o bidireccionales. Estas bases de datos representan los datos como una serie de sujeto, predicado y tripletas. Este tipo de BBDD son algo diferentes a los otros tipos de BBDD NoSQL debido a que se centran en la relación entre los nodos. Frecuentemente se suelen usar ontologías para describir las relaciones y los objetos. Algunas BBDD de este tipo son AllegroGraph, InfiniteGraph, Neo4j y OrientDB.

Como se ha visto, hay diferentes tipos de bases de datos, relacionales y NoSQL. Estas últimas a su vez se diferencian en otros cuatro tipos con diferentes características. Cada una depende de qué tipo de datos se vayan a almacenar, como documentos en las BBDD documentales, o qué tipo de aplicación se necesita, como es el caso de las BBDD en grafo para aplicaciones semánticas.

11.11.4 OTRAS TECNOLOGÍAS DEL ECOSISTEMA BIG DATA

Big Data no solamente es Hadoop, GFS y MapReduce, a pesar de ser las tecnologías más conocidas y de las que más se habla. Hay muchas otras que han sido creadas para facilitar ciertas labores cuando se trabaja con datos de esta índole. Por ejemplo, tecnologías para recolectar datos, herramientas para realizar consultas con lenguajes tipo SQL, lenguajes abstraídos para crear aplicaciones de una manera más fácil, cargadores de datos, gestores de herramientas, librerías de Inteligencia Artificial, motores de computación, o los GPLs y las herramientas para el análisis de datos.

Big Data

Debido a esta gran cantidad de tipos de herramientas para trabajar con *Big Data*, en esta sección se hará una visión de las herramientas más importantes y más utilizadas, pues es casi imposible abarcar todas, pues hay cientos. Sin embargo, Andrea Mostosi en [676] ha realizado una lista incompleta que contiene más de 600 proyectos de *Big Data*. Esta lista incluye herramientas, *frameworks* y tecnologías clasificadas en 50 temáticas, junto su enlace web y los artículos relacionados.

Apache Hive [677]: fue desarrollado por Facebook y ahora es mantenido por la Fundación Apache. Este proyecto ofrece un lenguaje similar a SQL llamado HiveQL. Mediante el uso de HiveQL se pueden crea trabajos MapReduce que se ejecuten sobre Hadoop, de manera que escribiendo una consulta estilo SQL se obtenga el resultado directamente. Esto es similar a si se trabajase contra una base de datos con SQL. Hive escrito en Java y proporciona interfaces Java Database Connectivity (JDBC) y Open DataBase Connectivity (ODBC).

Apache Pig [678], [679]: Pig es un proyecto de la Apache Foundation que fue en sus inicios desarrollado por Yahoo!. Este último tiene el 40% de sus trabajos realizados con Pig. Pig, haciendo clara referencia a su nombre, come de todo, como los cerdos, pues esta herramienta está pensada para trabajar con cualquier tipo de datos. Pig ofrece una forma de analizar grandes volúmenes de datos de forma fácil utilizando MapReduce gracias al uso de un lenguaje de alto nivel, Pig Latin, que posee una sintaxis similar a SQL.

Apache Impala [680]: es un motor de consultas SQL de código abierto que permite realizar procesado paralelo masivo sobre que se encuentren en el HDFS o HBase sin necesidad de transformar los datos. Utiliza los mismos metadatos, sintaxis SQL, driver ODBC e interfaz de usuario que Apache Hive. No obstante, necesita de Parquet.

Sawzall [681]: es un lenguaje de dominio específico desarrollado por Google y escrito en C++. Sawzall trabaja sobre MapReduce y trata de facilitar la creación de aplicaciones de este tipo, facilitando así su paralelización sin necesidad de escribir la aplicación utilizando un lenguaje de propósito general (GPL).

Apache Sqoop [682]: es un proyecto de la Apache Foundation que ayuda en la transferencia bidireccional entre Hadoop y las BBDD relacionales por medio de la interfaz de comandos. Su nombre proviene de Sql-to-Hadoop.

Apache Flume [683]: esta herramienta es un cargador en tiempo real para transmitir datos a Hadoop de forma que permita almacenar los datos en HDFS y HBase. Entre las opciones que ofrece están la captura, agregación y movimiento de datos de forma eficiente desde su origen, como son los logs, hasta un repositorio. Posee una arquitectura flexible y simple basada en la transmisión de datos y proporciona tolerancia a fallos.

Chukwa [684], [685]: es un sistema de código abierto para la recolección de datos que ayudan a monitorizar grandes sistemas distribuidos ofreciendo herramientas para la monitorización y el análisis de los resultados, siendo de gran ayuda para las recuperaciones después de un fallo. Chukwa se encuentra en la cima del HDFS y de Map/Reduce y hereda la robustez y escalabilidad de Hadoop.

Apache Oozie [686]: es una aplicación REST realizada en Java que sirve para orquestar procesos por medio de la creación de flujos de trabajo creados de una forma similar a Business Process Management (BPM). Además, permite acumular y coordinar procesos y definir y monitorizar aplicaciones. Los procesos posibles a orquestar pueden ser de casi cualquier tipo, como por ejemplo MapReduce, Hive, Pig, Spark, comandos de consola, emails, SSH o Sqoop. Este proyecto también pertenece a la Apache Foundation.

HUE [687]: es un proyecto de código abierto que proporciona una interfaz gráfica para desarrollar bajo juntar bajo misma interfaz diferentes utilidades de Hadoop, como son Hive, Pig, Sqoop, Oozie. Además, Hue proporciona gráficas, monitorización y gestión, entre otras. HUE está escrito en Python.

Apache Ambari [688]: es una herramienta web para aprovisionar, gestionar y monitorizar los clústeres de Apache Hadoop. Incluye soporta para HDFS, MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig y Sqoop, así como herramientas para ver el estado del clúster, de las aplicaciones MapReduce y de los trabajos de Hive y Pig.

Apache Zookeeper [689], [690]: provee de una estructura centralizada para la sincronización y coordinación de servicios de las aplicaciones distribuidas y soportando alta disponibilidad al tener redundancia de servicios. Zookeeper se encarga de almacenar toda la información necesaria para distribuir las aplicaciones, como su configuración y jerarquía de nombres, y coordinar los servicios para aplicaciones distribuidas.

Apache Mahout [691]: es una librería de aprendizaje máquina y de minería de datos que utiliza Hadoop y su MapReduce para lograrlo. Mahout se utiliza para realizar análisis predictivos y avanzados de manera que se pueden descubrir patrones. Posee algoritmos de recomendación, de *clustering* y de clasificación.

Pregel [692]: esta propuesta de Google es un *framework que* sirve para programar algoritmos de una forma fácil para trabajar con grafos de manera distribuida. Pregel sirve para mantener en memoria distribuida a través de los diferentes nodos un grafo determinado, de forma que se evita la latencia de acceso a disco y se consigue una aplicación más rápida. Propuestas de código abierto y con un propósito similar a Pregel son GoldenOrb [693] y Apache Giraph [694], ambas implementaciones en Java, y Phoebus [695] en Erlangs.

Apache Avro [696]: es un sistema de serialización de datos que provee de un sistema de compactación rápido en formato binario, de ficheros contenedores de datos, de comunicación de procesos por *Remote Procedure Call* (RPC), integración de lenguajes dinámicos para optimizar los procesos.

Apache Spark [697], [698]: es un motor de computación general para los datos de Hadoop. Spark provee de un modelo de programación simple y expresivo. La diferencia entre Spark y HDFS es que Spark monta todos los datos en memoria RAM en vez de en disco, lo que hace que este sea mucho hasta 10 veces más rápido que HDFS en disco, que se corresponde con cuando se configura para que guarde datos temporales en disco por si el sistema fallase. Además, provee de otro sistema de ejecución similar a MapReduce, pero siendo el de Spark hasta 100 veces más rápido que MapReduce cuando Spark se ejecuta íntegramente. No obstante, Spark requiere de muchos más recursos para la ejecución de los trabajos que MapReduce. Permite escribir las aplicaciones en Java, Scala, Python y R, pues ofrece APIs para estas 4 GPLs, siendo recomendable Scala debido a que es el GPL sobre el que Spark está desarrollado.

Tez [699]: es un *framework* construido sobre Hadoop Yarn que provee de un motor para ejecutar tareas para procesar datos en lotes o interactivamente. Para lograr esto, Tez ofrece una API, un modelo de datos flexible, datos agnósticos, simplificación del despliegue y diferentes mejoras, como la optimización de recursos y la reconfiguración de planes en tiempo de ejecución, entre otros. Tez proporciona la posibilidad de fusionar varios trabajar Hive y Pig bajo un mismo trabajo Tez. Actualmente se encuentra en la cima de Hadoop Yarn y está siendo adoptado por otros *frameworks* de Hadoop, como Hive y Pig, con el fin de reemplazar Hadoop MapReduce.

Apache Storm [700]: es un *framework* de código abierto y gratuito para sistemas de computación distribuida en tiempo real, a diferencia de Hadoop. Storm realiza el procesado en memoria, lo que da un alto rendimiento a costa de un alto consumo. Como ocurre con otros sistemas, Storm es escalable, tiene tolerancia a fallos y está diseñado para poder soportar cualquier lenguaje de programación que se desee para programarlo, no obstante, no está integrado con Hadoop. Actualmente lo mantiene la Apache Foundation pero fue creado por Nathan Marz de Twitter [583].

Apache Phoenix [701]: es un motor de bases de datos relacionales para Hadoop y que utiliza HBase como almacenamiento de datos. Provee de un driver JDBC para poder realizar consultas SQL y ocultar o abstraer al usuario de los procesos NoSQL. Además, provee Atomicidad, Consistencia, Aislamiento y Durabilidad, más conocido en inglés por transacciones ACID, pero manteniendo la flexibilidad de las bases de datos NoSQL. Apache Phoenix está integrado con Spark, Hive, Pig, Flume y MapReduce.

Apache S4 [702]: es una plataforma escalable, distribuida, parcialmente tolerante a fallos y de propósito general que permite desarrollar más fácilmente aplicaciones para procesar flujos continuos de datos. El propósito de S4 es solventar el vacío existente entre los sistemas propietarias complejos y las plataformas de computación

orientadas a lotes de código abierto. Está escrita en Java siguiendo una implementación modular y de plugins, lo que permite que las aplicaciones realizadas para S4 pueden combinarse dinámicamente para formar otras más complejas de una manera fácil.

R [703]: es un lenguaje de programación de código abierto para la computación estadística y gráfica, soportando grandes conjuntos de datos. Este sirve para generar modelos de distribución. R está compuesto de un GPL multiparadigma y de un entorno de desarrollo integrado (IDE). Además, es multiplataforma, pues es soportado en sistemas Microsoft Windows y Unix. R fue diseñado por Ross Ihaka y Robert Gentleman y presentado en el año 1993. Otras «herramientas» muy utilizadas para realizar el análisis de datos, de acuerdo con la encuesta realizada por KDnuggets en el año 2016 a 2.895 personas [704], son Python, SQL y Microsoft Excel.

11.12 *RETOS Y OBSTÁCULOS*

Los mayores retos a los que se enfrenta ***Big Data*** son sus propias características, es decir, sus «6Vs». A todo esto, hay que sumarle los diferentes riesgos de seguridad software que se comentarán a continuación, pues, el trabajar en sistemas nuevos, distribuidos y heterogéneos hacen necesaria la investigación en nuevos sistemas de seguridad debido a que los sistemas tradicionales eran sistemas cerrados que no requerían medidas en estos aspectos. Además, estos nuevos sistemas fueron creados con un propósito inicial muy diferente al de la seguridad, que fue relegada a un segundo plano.

Sin embargo, la seguridad no es el único reto, pues también está la falta de estandarización, algo que quedó demostrado en el artículo de la NIST [632], donde hacen un estudio de todos los estándares existentes. Por otro lado, también hay una infinidad de otros retos a afrontar como son los relacionados con los sistemas de recuperación, el anonimizar correctamente los conjuntos de datos, la veracidad, los relacionados con los metadatos y diferentes retos y obstáculos recogidos por Intel, así como los recogidos por el MIT e IBM, entre otros muchos.

11.12.1 LAS «6VS»

Las «6Vs» de Big Data son tanto sus características como sus principales retos y obstáculos que se deben superar:

- El volumen tiene el inconveniente del tema de tratar con tan enorme conjunto de datos como son Internet y todos los nuevos datos que se generan cada año. Esto es lo que marcó la evolución de las herramientas tradicionales a *Big Data* como necesaria, pues el tamaño de las bases de datos ha estado creciendo continuamente hasta tal punto que las técnicas tradicionales para analizar y visualizar datos han quedado obsoletas [32]. Esto hace que uno de los retos de *Big Data* sea poder gestionar su gran volumen, pues hay que buscarlo, almacenarlo, analizarlo, compararlo, refinarlo, combinarlo y visualizarlo [27].

- La velocidad como segundo reto, pues estos datos se necesitan procesar rápidamente, de nada sirve obtener valor para el negocio una vez ya no existe negocio, y eso sin decir si se necesita que sea en tiempo real las respuestas.
- Ficheros de muy y diferente tipo, con información que se complementar, analizadores diferentes para cada tipo de fichero e incluso dependiendo de quien lo haya creado.
- Recoger datos verídicos, de fuentes confiables o bien que esos datos estén correctos y sirvan para el propósito requerido.
- Cada día se generan nuevos datos sobre lo que ya existía, luego, se hace falta actualizarlos. Esta variabilidad crea un gran problema pues, puede que necesitemos actualizar nuestro *Data Warehouse* a partir de nuestra fuente de información cada poco, o cada mucho. Por esto, se hace indispensable encontrar técnicas que se adapten para la detección de este cambio [583]. No obstante, ambos procesos requieren búsquedas, actualizaciones y nuevos procesados.
- Por si no fuera poco, el valor que consigamos de todo esto marcará el éxito o el rotundo fracaso de la empresa. Luego, hay que planear muy bien la estrategia a seguir para que esta sea rentable y aporte un valor real a la empresa.

Las «6Vs» definen ***Big Data***, pero también marcan sus 6 principales retos y obstáculos a superar. A pesar de esto, no son los únicos. Otro de los problema de ***Big Data*** reside en que muchos conjuntos de datos están empezando a hacer más fácil el acceso a datos con carácter personal [635]. Estos conjuntos pueden ofrecer datos a los que se acceda sin autorización y que contengan información personal que permita identificar a una persona. Aquí aparece la cierta polémica existente como bien explican en [578], [615], donde comentan que puede que KDD abriese la caja de Pandora, pues, el descubrimiento de datos puede llegar a ser ilegal según ciertos países o reglas, como puede ser la investigación de personas o la predicción de ganadores en algunos juegos. Esto hace que ***Big Data*** puede acabar o destruir la privacidad de la gente sin que estos lo sepan o lo permitan.

11.12.2 SEGURIDAD Y PRIVACIDAD

La seguridad es un campo de rápido movimiento con múltiples vectores de ataque y contramedidas [635]. Por eso mismo, tanto la seguridad y la privacidad se han visto afectadas por ***Big Data*** [598], [635], debido a que esta tecnología se basa en recopilar una cantidad ingente de datos públicos para utilizarlos [635], y así obtener tendencias de estos. Esto es tal que el estudio de la NIST dijo que en el año 2010 menos de un tercio de los datos necesitó protección mientras que en el año 2020 lo necesitarán más del 40% de los datos.

El gran problema de seguridad en ***Big Data*** es que se trabaja con grandes cantidades de muy diversos datos. Algunos ejemplos de datos incluyen dominios como la sanidad, la investigación en medicinas, seguros, finanzas, comercios, y muchos otros sectores, tanto públicos como privados [635]. En estos dominios se puede encontrar información muy personal, como la frecuencia cardiaca, las horas de sueño, la localización, enfermedades y análisis u otros datos biométricos, como la huella dactilar, el iris del ojo, la forma de caminar, la palma de la mano o los rasgos del rostro. Por otro lado, también se pueden encontrar datos de empresas privadas, que puede ir desde datos de empresas, a datos bancarios o datos de los empleados. En lo referente a

este tipo de datos hay que destacar que cada día la gente comparte más datos personales de esta índole, permitiendo que pueden ser minados, sin ser conscientes de ello en muchas ocasiones [631].

Algunos ejemplos de sistemas que podrían verse involucrados en fallos de seguridad y privacidad son los presentados por la NIST en su volumen número 4 [635]. Pueden ser sistemas de entretenimiento como Netflix, subsidiaria que trabajen en proyectos importantes, como es el caso de Nielsen Homescan en el proyecto Apollo, los análisis de tráfico web con todos los datos de personas que estos contienen, el campo de la sanidad con todos los datos de pacientes, medicinas y enfermedades que mueven continuamente, datos genéticos, datos de los vehículos no tripulados, educación, aviación y los barcos mercantes, entre otros muchos.

11.12.2.1 SEGURIDAD Y PRIVACIDAD RESPECTO A LAS «VS»

Como se vio previamente, ***Big Data*** se compone de diferentes «Vs» que ha de cumplir y que la describen. De estas, hay cuatro que poseen o pueden dar problemas de seguridad [598], [635], magnificando así los problemas de seguridad y privacidad debido a que los mecanismos tradicionales son inadecuados para ***Big Data***, pues esta utiliza grandes infraestructuras en la nube, para analizar muy diversas fuentes de datos y formatos, y con una naturaleza basada en el *streaming* de datos [585]. A continuación, se comentarán los problemas de cuatro de estas «Vs».

El **volumen** de ***Big Data*** necesita almacenamiento en diferentes capas, desde su origen hasta su destino [585], [635]. Este movimiento de datos ha dado lugar a la necesidad de realizar un análisis sistemático de los modelos de amenazas [705], pues al ser volúmenes de datos tan grandes se utilizan obliga a utilizar procesos automatizados que impiden al responsable saber qué datos se están moviendo y cuando, lo que implica que se deba hacer una trazabilidad de todo el proceso con nuevos métodos de seguridad [585]. Estas amenazas son, según *Big Data* Working Group las siguientes [585]: la confidencialidad y la integridad, la procedencia, la disponibilidad, la consistencia, las conspiraciones, ataques de *roll-back*, y las disputas de registros. Además, estas estructuras montadas en la nube no han sido probadas correctamente y no se sabe cómo la seguridad actual afectará y sentará en estas estructuras tan grandes [585]. No obstante, ***Big Data*** también ha dado lugar a la investigación y desarrollo de técnicas novedosas, pues, uno de sus usos es también la detección de brechas en la seguridad [635].

La velocidad, al igual que ocurre con las bases de datos no relacionales, de las cuales no se ha probado bien la inyección NoSQL, y los *frameworks* distribuidos y que utilicen Map/Reduce, como Hadoop, no han sido desarrollados con seguridad como primer objetivo [585]. Por esto, un mal funcionamiento de alguno de los nodos o un ataque parcial a la infraestructura pueden suponer una fuga de datos [635]. Además, debido a que muchos datos son en *streaming*, esto requiere de una seguridad y privacidad que sean ultra rápidas [585].

Respecto a **la variedad**, hay que destacar que estos sistemas no fueron diseñados pensando en ser seguros, siendo, siempre relegada la seguridad a la capa del *middleware*, como ocurre en el caso de las bases de datos NoSQL [585]. En los últimos años ha habido un cambio de seguridad entre las bases de datos relacionales a las no relacionales que supuso todo un reto [585]. Este nuevo reto fue debido a la gran variedad de datos surgidos y que permitió inferir la identidad a partir de datos anonimizados por medio de la relación de datos provenientes de diferentes conjuntos de datos públicos aparentemente inocuos.

El problema de seguridad que viene dado por la variabilidad de debe a que los requisitos de seguridad y privacidad pueden cambiar con el tiempo según el proceso del Ciclo de vida de Big Data que se esté desarrollando, por ejemplo, en la recolección, en el análisis y en el almacenamiento.

11.12.2.2 DIFERENCIAS RESPECTO A LAS APLICACIONES TRADICIONALES

La forma de trabajar de ***Big Data*** provoca que existan diferencias con la seguridad y privacidad tradicionales, pues antes se tenían sistemas de pequeños escalados con datos estáticos, detrás de un cortafuegos y aislados [585], [635], [706]:

1. Los componentes utilizados en los proyectos ***Big Data*** no suelen poseer una seguridad diseñada desde el principio.
2. La mayoría de los métodos de seguridad y privacidad han sido diseñados para procesos de procesamiento por lotes u online, mientras que los proyectos ***Big Data*** requieren juntar datos en *streaming* con datos almacenados, lo que crea escenarios únicos o nuevos debido a la gran cantidad de objetos existentes, que probablemente además estén distribuidos.
3. Muchas de las fuentes de datos no fueron diseñadas originalmente para usar en conjunto con otras, lo que puede desembocar en un compromiso de la seguridad, la privacidad o ambas. Por esto, los métodos tradicionales de anonimizar la información personal de identificación (PII) pueden no ser efectivos en ***Big Data***, aunque si sirvan para las fuentes individuales.
4. El incremento de la dependencia de datos de los sensores, proveniente del ámbito de **Internet de las Cosas**, puede crear vulnerabilidad antes de introducirlo en ***Big Data***.
5. Ciertos tipos de datos que fueron pensados que al ser tan grandes para los análisis no fueron provistos de seguridad ni de privacidad, como es el caso de las imágenes geoespaciales y los videos.
6. Toda la información se magnifica al utilizar ***Big Data***, lo que implica que con el tiempo cada vez será más la información que la gente encuentre sobre sí misma en diferentes análisis.
7. La volatilidad es significante debido a que los escenarios de ***Big Data*** por defecto guardan los datos originales sin modificar. Por esto, estos datos son vulnerables, ya que serán conservados más allá de la vida útil de la seguridad que se diseñó para protegerlos.
8. Los diversos medios existentes: actualmente no solo hay proveedores y consumidores de datos tradicionales, también hay dispositivos móviles y redes sociales como primeros consumidores. Esto sumado a la gran escala de usuarios existentes y que no tiene precedentes hace que necesarios nuevos mecanismos de protección y de vectores de ataque para mitigar las posibles vulnerabilidades.
9. Las soluciones de seguridad tradicionales necesitan ser rediseñadas de acuerdo con los cambios necesarios para las aplicaciones ***Big Data***, pues la infraestructura es diferente, lo que implica la necesidad de cambio en la autenticación, el control de acceso y la autorización.
10. ***Big Data*** hace que los objetivos sean mucho más jugosos para los atacantes debido a toda la cantidad de información y de análisis que poseen. Esto provoca que, como cada vez se está extendiendo más el uso de ***Big Data***, existan más posibles objetivos.

11. Volatilidad: los datos cambian continuamente debido a que los propios sistemas que los crean pueden almacenarlos indefinidamente. Por esto, se necesitan unos roles de seguridad acordes con esta característica, pues, si hay una fusión o desaparición de las empresas, la gobernabilidad de estos datos puede verse afectada.
12. La monitorización en tiempo real siempre ha sido un reto en la seguridad, y ahora, con ***Big Data***, este problema se ha visto incrementado debido al volumen con el que se trabaja y la velocidad de los datos. Sin embargo, ***Big Data*** también provee de la oportunidad de utilizarlo para la propia seguridad gracias al rápido procesamiento que estas tecnologías ofrecen sobre los grandes volúmenes de datos [585]. Es decir, ***Big Data*** tiene dos caras, la que dificulta su monitorización, pero también la que ofrece las herramientas para hacerla.
13. Mejorar el cifrado basado en atributos (ABE) para que sea más rico, eficiente y escalable, de tal manera que permita asegurar los datos de extremo a extremo de manera que solo sean accesibles por las entidades autorizadas.
14. Aplicación de un acceso de control granular para así conseguir compartir datos tanto como sea posible, pero sin comprometer la seguridad impidiendo su acceso a aquella gente que no debería poder.
15. La procedencia de los metadatos crecerá en complejidad debido a los grandes grafos que se generarán sobre estos metadatos. Además, el análisis de estos grandes grafos para detectar la procedencia de los metadatos para aplicaciones de seguridad y confidencialidad será computacionalmente alto.

11.12.3 **ESTÁNDARES**

Debido al interés en ***Big Data***, diversas compañías, organizaciones y consorcios se han unido con el fin de colaborar en la creación de estándares. Entre estas, se encuentran el *International Committee for Information Technology Standards* (INCITS), *International Organization for Standardization* (ISO), *Institute of Electrical and Electronics Engineers* (IEEE), *International Electrotechnical Commission* (IEC), *Internet Engineering Task Force* (IETF), *World Wide Web Consortium* (W3C), *Open Geospatial Consortium* (OGC®), *Organization for the Advancement of Structured Information Standards* (OASIS) y el *Open Grid Forum* (OGF) [632]. Esto se hace necesario para poder equilibrar la eficiencia de los sistemas de computación ***Big Data*** de manera matemática [625].

Algunas de sus metas son crear subcomités y grupos de trabajo en American National Standards Institute (ANSI), crear organizaciones de desarrollo de estándares acreditados, crear consorcios industriales, crear implementaciones de referencia y crear implementaciones de código abierto [632].

No obstante, actualmente, como se ha recogido en [632], existen más de 120 estándares aplicables a ***Big Data*** y que tienen cabida en el *framework* presentado por la NIST. Estos engloban entre otras cosas, SQL, XML, Xquery, XMI, metadatos, guías de documentación y código, compresión y tipos imágenes, datos geográficos, protocolos (MQTT, REST, SOAP [*Simple Object Access Protocol*]), seguridad y privacidad, requisitos, JSON (*JavaScript Object Notation*), DOM (*Document Object Model*), XPath, XSLT (*eXtensible Stylesheet Language Transformations*), RDF (*Resource Description Framework*), SPARQL (*SPARQL*

Protocol and RDF Query Language), y arquitecturas. No obstante, según ellos, aún tienen muchos huecos que cubrir y estandarizar, como son las definiciones, vocabulario, DSLs, lenguajes, transformaciones de datos, seguridad y privacidad, intercambio, almacenamiento, visualización y consumo de datos, medición de energía, y calidad.

11.12.4 OTROS RETOS Y OBSTÁCULOS EN BIG DATA

Además de los anteriores retos y obstáculos presentados, hay muchos más. Entre estos se puede destacar los sistemas de recuperación, el anonimizar los datos correctamente para que no puedan sacar en ningún caso datos personales de la gente, la veracidad de los datos, los metadatos, los diferentes retos y obstáculos según los profesionales de *Big Data* consultados por Intel, los retos y obstáculos según la encuesta del MIT e IBM, la redundancia de los datos, la gestión de la energía, la integridad, la abstracción, la algoritmia, diversos retos arquitectónicos en el hardware, en la visualización y en la minería de datos distribuida. Estos solo son algunos de los muchos y más importantes retos y obstáculos existen en *Big Data*.

Sistemas de recuperación: debido a la escalabilidad que tienen las aplicaciones ***Big Data***, están requieren de sistemas de recuperación de desastres y de aseguramiento de la información nuevos, únicos y que están emergiendo todavía, pues los métodos tradicionales pueden ser poco prácticos al igual que ocurre con los métodos de análisis, almacenamiento y visualización [635]. Hay que tener en cuenta que también los métodos de copia, verificación y testeo de datos también son diferentes y se necesita comprobar que puedan completar en el tiempo requerido [635].

Anonimizar correctamente los conjuntos de datos: otro riesgo respecto a la PII es que, a pesar de que se anonimicen los datos, hay expertos que afirman que es posible realizar el proceso contrario gracias a la cantidad de conjuntos de datos existentes, entre los cuáles los hay que no estaban previstos que se usasen. Así, por medio de estos conjuntos, se podría identificar datos que fueron anonimizados previamente. Esto se puede dar debido a que esos datos no estaban previstos para ser analizados a gran escala, o faltase consentimiento para su recogida. Por eso, se hace indispensable aumentar los controles técnicos y legales [635].

Un ejemplo de esto fue lo que pasó con Netflix, pues se dio el caso de que pudieron identificar usuarios de Netflix a partir de un conjunto de datos anonimizado gracias a correlaciones encontradas en los datos de IMDB [585]. Otro ejemplo fue cuando AOL anonimizó los *logs* de búsquedas para que pudieran ser usados para propósitos académicos, pero pudieron relacionar los usuarios fácilmente a través de sus búsquedas [585].

Por este motivo se hace indispensable el anonimizar todos los conjuntos de datos públicos de forma correcta y que lo usuarios puedan elegir su derecho al olvido, de tal manera que, aunque tengan datos públicos en un sitio, estos no puedan servir para identificar otros datos de ellos pertenecientes a otro conjunto de datos.

Veracidad: puede haber problemas debido a la mala gestión de los datos que den análisis de mala veracidad debido al uso de datos derivados o que no pertenezcan a la fuente primaria [635]. Por eso, hay que tener muy en cuenta siempre las fuentes de donde se toman los datos para saber si se están utilizando datos fiables o si, por el contrario, los utilizados pueden ser falsos y perjudicar todo el análisis. Por esta razón se hace indispensable cumplir los tres subapartados de la Veracidad: procedencia, veracidad y validez.

Metadatos: algunos expertos consideran que el reto de definir y mantener los metadatos, indicando así toda la cadena de custodia de los datos, en lugar de solo su procedencia [635]. Esto se debe a que se obligaría a todos los niveles que trabajen con el conjunto de datos a introducir todos aquellos metadatos relevantes o faltantes, como pueden ser: marcas de tiempo, equipo de trabajo, modificaciones, datos del sistema de recogida de datos, como su precisión y versión, etc.

Retos según la encuesta de Intel [594]: según la encuesta que Intel realizó a 200 gestores de empresas que trabajan con *Big Data*, las preocupaciones de estos son en base a los datos, que son por orden: el crecimiento de los datos, la infraestructura de los datos, el gobierno, la integración, la velocidad, la variedad, la regulación y la visualización.

Obstáculos según la encuesta de Intel [594]: por otro lado, en la encuesta de Intel también se analizaron los grandes retos que preveían los managers de estas empresas en *Big Data*. Estos obstáculos, por orden de importancia son la seguridad, el gasto, el aumento de los cuellos de botella en la red, la escasez de profesionales con las habilidades requeridas, relación de datos inmanejable, la capacidad de copia de datos, la falta de comprensión de la capacidad, una mayor latencia en la red y el insuficiente poder computacional.

Retos y obstáculos según la encuesta del MIT e IBM: esta encuesta analizada en [580], fue realizada por el MIT Sloan Management Re*View* e IBM a 3000 ejecutivos, analistas y gestores de más de 30 empresas en 108 países. En ella, 1 de cada 5 estaban preocupados por la calidad de los datos y la ineficacia del gobierno, 4 de cada 10 por la falta de entendimiento de cómo utilizar los análisis para mejorar la empresa, y más de 1 de cada 3 en la falta de gestión del ancho de banda debido al conflicto de prioridades. Otros retos u obstáculos con los que estuvieron de acuerdo los encuestados fueron la falta de habilidades dentro de la línea del negocio, el talento para conseguir los datos, la cultura actual que no fomenta el compartir datos, la propiedad de los datos, la falta de patrocinio, sopesar que los costes percibidos con los beneficios previstos, no tener un caso para cambiar a utilizar ***Big Data***, y el no saber por dónde empezar.

Redundancia de los datos [625]: debido a los conjuntos de datos utilizados, muchas veces hay mucha redundancia en ellos. Eliminar esta redundancia podría ahorrar espacio en disco, como bien comenta el autor. Además, esto podría facilitar los análisis que se realicen, así como ahorrar coste de computación al estar ya preprocesados. El problema aquí reside en guardar los datos originales intactos y que estos no se vean afectados por esta característica.

Gestión de la energía [625]: en caso de que se tenga un servidor propio, esto puede ser un gran problema, pues es un aspecto importante en la economía de la empresa y del medioambiente. En este aspecto, cuanto más espacio se requiriese, más computación, y más demanda hubiese, más energía consumiría, teóricamente.

Integridad [587]: un requisito es que se mantenga la integridad de los datos para verificar que estos no han sido modificados. En caso de que no se mantuviera la integridad, se estaría conseguiría una falta de veracidad, pues los datos podrían estar incumpliendo la Veracidad. La integridad se puede lograr mediante el uso correcto del control de acceso, protocolos criptográficos y la redundancia.

Abstracción [587]: esta abstracción puede ser mediante el uso de nuevos tipos de visualizaciones de datos o abstracción a la hora de crear los algoritmos. Esto es necesario para ofrecer una mejor comunicación

a los usuarios para así obtener información de una manera más fácil. No obstante, hay que tener cuidado pues, esta abstracción puede llevar a engaños al usuario. Esto podría ser debido a la eliminación o sesgo de datos que podrían no parecer importantes.

Algoritmia [587]: ***Big Data*** se basa en análisis estadísticos, búsqueda de patrones y en la gestión de la información. Sendos campos necesitan de diferentes algoritmos para realizar correctamente su trabajo. Además, cada vez existen más datos, que se suman a los que todavía no han sido analizados. Por estos motivos, se hace indispensable la mejora de los diferentes algoritmos con el fin de trabajar mejor con los datos, realizar mejores análisis y detectar mejores patrones. Estos campos requieren mejoras en diferentes campos como son las matemáticas y la informática, o precisando, la estadística, la Inteligencia Artificial y la teoría de la información.

Retos arquitectónicos en el hardware [587], [590], [593], [596]: ***Big Data*** requiere de una gran computación debido al gran volumen de datos, lo que implica que se requieran mejorar diferentes aspectos hardware en pro de la mejora de la comunicación. Algo que se hace indispensable es mejorar la velocidad de lectura, tanto de memoria como de disco, así como la transferencia de las redes y en los procesadores.

Por ejemplo, en lo que refiere a los discos duros, hay dos opciones [707], elegir entre alta capacidad y baja lectura, con los Hard Disk Drives (HDDs) o baja capacidad y alta velocidad de lectura con los *Solid-State Drive* (SSD). No obstante, se están dando sistemas híbridos entre ambos tipos de almacenamiento.

En lo que se refiere a la transferencia de las redes, hay que tener en cuenta que al manejar datos grandes esto puede provocar que donde se puede formar un cuello de botella tanto antes de «cortar» los datos para enviarlos a los diferentes servidores, como al agregarlos todos de nuevo bajo un mismo servidor.

Por otro lado, tenemos los procesadores, pues hay que tener en cuenta la localidad temporal y espacial de los procesadores, ya que estos no fueron creados pensando en el tipo de análisis de ***Big Data***, pues sus aplicaciones tienen una pobre localidad. Es decir, hay cuellos de botella tanto a lo largo de toda la arquitectura, como es en la transmisión por red, en los procesadores y en el almacenaje en disco duro. También, otro problema con los procesadores es el correspondiente a la energía que consumen.

Visualización de datos [455], [583], [596]: debido al gran tamaño de los datos con los que se trabaja, y a veces, a la velocidad, se hace muy difícil el crear la visualización, pues, las herramientas actuales tienen rendimientos pobres en lo referente a funcionalidades, escalabilidad y tiempo de respuesta. Esto se complica sobre todo si se desea hacer aplicaciones *real time*. Otras veces, al haber tantos datos puede pasar que la visualización sea difícil de comprender

Minería de datos distribuida [583]: muchas técnicas de minería de datos usan técnicas que no son fáciles de paralizar. Esto implica que se necesite investigación nueva con el fin de adaptar estas técnicas tradicionales al uso distribuido para poder mejorar su rendimiento en *Big Data*.

11.13 *TRABAJO RELACIONADO, LÍNEAS DE INVESTIGACIÓN Y APLICACIONES*

Como se ha visto, ***Big Data*** es aplicable a casi cualquier campo existente. Estos se pueden ver beneficiados en muchos aspectos, pero básicamente, en el económico. Puede ayudar a encontrar tendencias, a distribuir datos, hacer predicciones, todo esto con el fin de mejorar un servicio dado o propio para poder competir u obtener un beneficio.

Muchas veces para los análisis de datos sobre la población en general los investigadores suelen utilizar Twitter. Esto les sirve para trabajar con una amplia muestra pública de todo el mundo en base a lo que ellos escriben para así sacar patrones y tendencias. Otras veces lo utilizaron para contrastar diferentes datos de salud. Por otro lado, las empresas trabajan suelen utilizar sus propios datos privados para así obtener un beneficio determinado.

A continuación, se presentan diferentes trabajos relaciones y aplicaciones dadas a ***Big Data*** que se corresponden con las líneas de investigación más utilizadas actualmente: tiendas, empresas y fábricas, transporte, gobierno, tendencias en redes sociales, publicidad, salud e investigación. Aquí solo se recogen algunos de los muchos trabajos que se han investigado y realizado utilizando *Big Data*, que han sido muchos y se dividieron en base a unas líneas de investigación, que no implica que estas sean todas, pues existen muchas más.

Otros ejemplos además de los recogidos en este apartado son los casos de uso de ***Big Data*** documentados por el NIST en [708], en donde presenta 51 casos de uso en 9 áreas diferentes. En este artículo se encuentran casos de uso de propuestas para su uso gubernamental (4), comercial (8), en defensa (3), para la asistencia médica y las ciencias biológicas (10), el aprendizaje profundo y los medios de comunicación sociales (6), el ecosistema de investigación (4), astronomía y física (5), la Tierra, el medioambiente y la ciencia polar (10), y la energía (1). Otros casos de uso documentados como posibles aplicaciones de ***Big Data*** son en la Tierra y el medioambiente, exactamente centrados en el océano y en el cielo, en la salud y el bienestar, en la ciencia, y en la educación, como bien ha documentado el libro «*The Fourth Paradigm: Data-Intensive Scientific Discovery*» [624].

11.13.1 **TIENDAS**

Un ámbito en el que es muy utilizado ***Big Data*** es en las tiendas, tanto para descubrir patrones de los consumidores, como para ofrecer un mejor servicio de recomendaciones y maximizar sus beneficios y su eficiencia.

Un ejemplo del uso de ***Big Data*** es el de las tiendas de libros online [577], como Amazon. Estas pueden hacer un seguimiento de los libros que compra cada usuario, las páginas de los libros que visitan, los resúmenes de libros que leen, las diferentes partes del sitio que vigilan, el tiempo que pasan en cada página, las revisiones que leen, compararlos con otras personas con características similares, y más. Amazon hace esto con el único fin de ofrecerles un mejor servicio por medio de mejorar el sistema de recomendación de libros que se les recomienda leer y que posiblemente les gusten, de tal manera que mejoran su servicio de descubrimiento a los usuarios y este comprará más.

Un grupo de investigación del MIT utilizó la geolocalización de los dispositivos móviles de la gente para inferir cuanta gente estuvo en el aparcamiento de los grandes almacenes *Macy's* el día del *Black Friday*, que es cuando empiezan las compras en los Estados Unidos de América. Con esto buscaron poder estimar las ventas de los pequeños comercios en ese día crítico antes de que la propia *Macy's* reportase sus ventas ofreciendo de esta manera una ventaja evidente a los analistas de Wall Street [577].

Hace unos años, la compañía estadounidense Sears Holdings se dio cuenta que debía generar valor a través de la recolección de datos de sus clientes, tiendas y promociones. De esta manera podrían poder crear promociones a medida, ofertas para los clientes y tomar ventajas según las condiciones locales. Antes de hacer estos cambios, Sears Holdings necesitaba ocho semanas para crear promociones personalizadas debido al gran volumen de datos fragmentados a analizar, tras estas ocho semanas desde trabajo, muchas veces ya no les servían las promociones creadas. De cara a solucionar esto y conseguir algo más rápido y barato, comenzaron a utilizar ***Big Data***, comenzando con un clúster **Hadoop**, exactamente contratando a Cloudera, al que subían todos los datos recopilados por sus marcas y les permitía hacer promociones más rápidas, precisas y baratas. Según Phil Shelley, director de tecnología de la empresa, puesto conocido normalmente en inglés como *Chief Technology Officer* (CTO), consiguieron disminuir de ocho a una semana el tiempo de trabajo, a pesar de que todavía había margen de mejora [577].

Otro ejemplo es el de Gilbert y Karahalios [709], quienes demostraron que las emociones como la ansiedad, la preocupación y el miedo, recogidas de un conjunto de datos con más de 20 millones de *posts* realizados en LiveJournal, proveían información acerca del precio futuro de los stocks, pudiendo así anticiparse a los cambios necesarios y sugiriendo así una nueva forma de medir la opinión pública.

Por otro lado, los minoristas pueden utilizar las Redes Sociales para saber quiénes compran en su tienda y crear campañas personalizadas y mejorar su cadena de suministro [579].

11.13.2 **EMPRESAS Y FÁBRICAS**

Otra de las líneas de investigación en ***Big Data*** son las empresas. La aplicación en este campo varía, pues es utilizado para mejorar la gestión interna, mejorar las cadenas de suministro, reducción de costes o la inversión en bolsa.

Un estudio realizado sobre 330 compañías de los Estados Unidos de América, dirigido por el *MIT Center for Digital Business*, la consultora McKinsey y varios profesores y un doctorando del MIT, intentó demostrar si las compañías dirigidas por datos serían mejores. Este estudio abarcaba las prácticas de gestión de la organización y de la tecnología para recopilar datos anuales de sus informes anuales y de otras fuentes. Con este estudio demostraron que las empresas que utilizaron *data-driven* obtuvieron mejores resultados financieros y operativos, exactamente tuvieron de promedio un 5% más de productividad y un 6% más de rentabilidad que los competidores que no utilizaron ***Big Data*** para tomar decisiones [577].

Big Data también se puede utilizar en las fábricas para poder tener diferentes medidas con ayuda de sensores, como ocurre con General Motors' OnStar ® y Renault's R-Link ®, de cara a mejorar su producción pudiendo reducir los costes de desarrollo y de ensamblado [579].

Más casos similares que afectan al plano económico y donde hay investigaciones que trataron de predecir las tendencias del mercado a través del análisis de sentimientos en Twitter es el investigado en [710]. Para realizar esto, los autores clasificaban los *Tweets* entre estado de ánimo positivos y negativos y entre calma, alerta, seguro, vital, amable y feliz. Los autores detectaron que podrían mejorar el índice bursátil Dow Jones Industrial Average por medio de la inclusión del sentimiento de los *tweets*.

11.13.3 **TRANSPORTE**

Una parte muy importante de la humanidad es el medio de transporte, sea la época que sea. Actualmente, este se está saturando y diversas investigaciones se han llevado a cabo con el fin de intentar arreglar este embotellamiento de gente y ahorrar o no desperdiciar tanto dinero. Un buen ejemplo de esto es el de los aeropuertos.

En el año 2001 la empresa *PASSUR Aerospace* propuso solucionar el problema de la «Hora de Llegada Estimada», pues hacía subir los costes de los aeropuertos al tener a los pasajeros, la tripulación y el personal atrapado y sin hacer nada [577]. Exactamente, el 10% de los vuelos de las aerolíneas estadounidenses tenían una diferencia de al menos 10 minutos entre la hora estimada de llegada y la hora de llegada real, mientras que el 30% tenía una diferencia de 5 minutos. La forma de obtener este tiempo era por medio de los pilotos, quienes, al acercarse al aeropuerto, tenían que calcular esta hora mientras se encargaban de otros quehaceres del vuelo.

Para solucionar esto, hablaron con *PASSUR Aerospace*, quienes comenzaron en el año 2001 a ofrecer su hora de llegada estimada por medio de un servicio llamado *RightETA*, donde ETA significa *Estimated Time of Arrival*. Este servicio calculaba la hora estimada de llegada combinando datos públicos del tiempo y de los horarios de vuelo con otros factores propietarios que la compañía recolectaba como datos de radares que había instalado cerca de los aeropuertos con el fin de obtener más datos del cielo.

En el año 2012 la compañía ya tenía más de 155 instalaciones de este tipo, que cada 4,6 segundos recogían datos de todos los vuelos que «veían». Todos estos datos han sido conservados por la compañía desde que comenzó el sistema, lo que le permite un análisis en busca de patrones para buscar que pudo ocurrir en situaciones similares que se diesen en el pasado. Gracias a este sistema, las aerolíneas eliminaron la diferencia existente entre el tiempo de llegada estimado y el tiempo real de llegada. Según *Passur Aerolines*, esto supuso un de varios millones de dólares a cada aeropuerto por año gracias al uso de ***Big Data*** para obtener mejores predicciones.

11.13.4 **GOBIERNO**

Algo muy relevante en el funcionamiento de nuestras ciudades y países es el gobierno. Por este motivo, diferentes organismos y gobiernos han realizado mejoras en el sistema con el fin de facilitar el gobierno de una región, los trámites burocráticos, e incluso la propia defensa del país.

En el año 2009, las Naciones Unidas lanzaron la iniciativa Global Pulse para realizar un estudio utilizando ***Big Data*** de cara a mejorar la vida de los países en desarrollo mediante la obtención de información a partir de datos no estructurados, imperfectos y complejos [584]. Han investigado nuevos métodos y técnicas con las

que poder analizar datos en tiempo real con el fin de detectar vulnerabilidades emergentes, utilizando un montaje de herramientas *open source*, además de establecer una red de laboratorios. Con esto han esperado poder conseguir un sistema de alertas temprano para detectar anomalías en tiempos de crisis, diseñar programas y políticas más exactas y representativas de la realidad al tiempo en el que transcurren, y verificar las políticas y programas en tiempo real utilizando la retroalimentación para lograr este objetivo.

Otro caso de ejemplo del uso de ***Big Data*** en el gobierno fue en [711], donde los autores realizaron la primera validación de voto en 50 estados utilizando las nuevas tecnologías. Esto les ayudó a sacar ciertas conclusiones según las respuestas de los votantes y no votantes. De esta manera, utilizando tecnologías *Big Data* consiguieron hacer una validación menos costosa entre las respuestas de las encuestas y los registros públicos. Esta validación fue la primera realizada sobre una encuesta a nivel nacional desde que se descontinuase este proceso en el año 1990, que es cuando lo hacía el *National Election Study* (NES) de los Estados Unidos de América. Este estudio les permitió tanto validar el voto, como si están registrados, el partido donde están registrados y el método por el que votaron.

En lo referente al gobierno también tiene su cabida en materia de defensa [587]. El *US Department of Defense* obtiene conocimiento mediante el uso de sensores y la computación de diferentes datos. Mientras, el *US Department of Energy* lo utiliza para extraer significado de los datos.

11.13.5 TENDENCIAS EN REDES SOCIALES ONLINE

Mientras tanto, otros investigadores se dedican a usar las Redes Sociales Online para obtener tendencias o sacar patrones, conclusiones o estudios en base a lo que publica la gente. En este campo la red social más utilizada es Twitter, aunque a veces utilizan también información proporcionada por los buscadores como Yahoo!. Mediante el uso de redes sociales tratan obtener diferentes tipos de predicciones cuantitativas a partir de modelos obtenidos de las opiniones de los usuarios, prediciendo así posibles tendencias futuras [420].

Un ejemplo de esto es el de Asur y Huberman [420], quienes trataron de predecir la probabilidad de éxito de las películas antes de su estreno. La elección de utilizar el éxito de películas viene dada debido a que es un tema de interés considerable en las Redes Sociales y que es fácilmente comparar los resultados de la investigación con la taquilla obtenida por la película y así saber si fue un éxito o no. Para hacer esta investigación, primero estudiaron la atención creada por las películas y como está cambiaba con el tiempo, por ejemplo, debido a la publicidad. De esta manera, podían centrarse en el *marketing* de la película y el *hype* de esta antes de su estreno, de manera que pudieran pronosticar el rendimiento en taquilla de dicha película. Tras esto, estudiaban los sentimientos, positivos y negativos hacia la película. Las conclusiones que sacaron fueron positivas, pues, demostraron que se puede crear un modelo para predecir los ingresos de taquilla de una película.

Otro caso de uso de las Redes Sociales para predecir datos, es el que viene siendo habitual en tiempos de elecciones, pues algunos investigadores trataron de predecirlas [432], [601], [652]. En esta investigación, los autores trataron de predecir, de forma fallida, el resultado de las elecciones de los Estados Unidos de América por medio de la recogida de datos utilizando redes sociales. En comparación con otros investigadores, estos utilizaron más datos y utilizaron análisis del sentimiento humano. En esta investigación, los autores se dieron

cuenta de que, utilizar Twitter para predecir las elecciones no implica que sea un mejor método que el azar, pues, según ellos fueron comprobando, es difícil ser precisos en base al análisis de sentimientos a datos con ruido de una muestra sesgada de la población. Sin embargo, gracias a las pruebas, consiguieron revelar algunas de las limitaciones que provocan que sea difícil prever los resultados de las elecciones. Como punto final, propusieron una serie de estándares o reglas para predecir las elecciones u otros eventos sociales.

11.13.6 **PUBLICIDAD**

Muchas compañías utilizan ***Big Data*** para obtener información de las personas y así obtener sus posibles gustos para ofrecerles productos que les interesen o bien ver sus hábitos de navegación. En estos casos podemos encontrarnos a Google [587], Facebook o Double Click.

También se dio el caso de compañías de publicidad que lo utilizaron, pero en el que sus campañas fueron algo polémicas. Este fue el caso que se dio cuando la compañía supo que una adolescente estaba embaraza antes de que lo supiera su padre [585]. Esto ocurrió debido a que la adolescente lo sabía y estaba buscando productos acordes a su embarazo. Esto fue detectado por el software de la compañía de publicidad, que entonces comenzó a mostrarle anuncios de productos para embarazadas a la adolecente. El padre, al recibir cupones de descuentos para productos de mujeres embarazadas en su casa, se sintió molesto y fue a la tienda a quejarse. No obstante, días más tarde tuvo que disculparse el padre a la empresa por esta queja al haberse enterado de que su hija de que la publicidad era acertada debido a que su hija sí estaba embarazada.

Otro uso respecto a la publicidad es el de empresas que utilizan ***Big Data*** para mostrar anuncios en los smartphones sobre las tiendas cercanas a las que se encuentre su dueño gracias al uso del GPS [579]. De esta manera, dan una publicidad mucho más personalizada y dependiente del contexto que rodea al usuario.

Por otro lado, algunos investigadores de Google Labs Israel han afirmado que con Google Trends y Google Insights for Search se puede observar la tendencia de búsquedas de la gente en el motor de búsqueda de Google. Esto puede proporcionar una ventaja para los anunciantes, publicistas, economistas, académicos y cualquier persona interesada, siempre que consigan comprender estas tendencias. Así, estás tendencias pueden ser estacionales y repetirse continuamente, como pasa con el esquí en la temporada de invierno u otros deportes, la salud, la bebida y la comida, o los viajes. En cambio, otras son todo lo contrario, es decir, irregulares y difíciles de predecir, como ocurre con los términos Obama, Twitter, Android, el calentamiento global o el entretenimiento [712].

11.13.7 **SALUD**

Big Data puede ser utilizado en diferentes y variados sitios. Uno de ellos es en los servicios médicos ayudando a monitorizar a los pacientes y reduciendo así las visitas del personal médico [579]. Además, otras ventajas de esto es la rapidez con la que podrían detectar posibles enfermedades, cambios o patrones en el paciente.

En la investigación de Polgreen et. al [713] los autores examinaron la relación entre las búsquedas de gripe en el buscador de Yahoo! y su ocurrencia entre marzo de 2004 y mayo de 2008 y teniendo en cuenta las búsquedas diarias de los Estados Unidos de América.

En [714], donde los autores estudiaron la detección temprana de la epidemia de la influenza, basándose en las consultas realizadas por los usuarios en los motores de búsqueda, como Google, y así estimar el nivel semanal de la influenza en cada región de los Estados Unidos de América.

Otra investigación similar a estas y también centrada en la gripe fue [715], donde crearon una herramienta para seguir el dominio de la gripe en las diferentes regiones del Reino Unido utilizando el contenido de Twitter.

11.13.8 **INVESTIGACIÓN**

Otro de los campos donde se utiliza es en la investigación. Por ejemplo, la *US National Oceanic and Atmospheric Administration* (NOAA) y la *National Aeronautics and Space Administration* (NASA) han generado cantidades enormes de información para responder preguntas sobre la dinámica planetaria y los ecosistemas. Para responder a preguntas que les surgen sobre este tema analizando los datos utilizan ***Big Data*** [587].

Otros casos de usos documentados, según [587], son en la simulación científica y en la ingeniería, para entender y simular el impacto del cambio climático y en la ciencia experimental como los rayos X y la dispersión de neutrones.

Facebook, en el artículo en el que calculó los grados de separación de sus usuarios [439], [440], utilizó un clúster de Hadoop con 2.250 utilizando Hadoop y Hive. Para calcular los caminos y sus aristas, por medio del algoritmo Newman-Ziff [716], utilizaron una única máquina con 64 GB de RAM. Para calcular la longitud de los caminos utilizaron una sola máquina de 24 cores y 72 GB de RAM utilizando el algoritmo HyperANF [717].

12. ROBÓTICA

«Yo... he visto cosas que vosotros no creeríais: atacar naves en llamas más allá de Orión.
He visto rayos C brillar en la oscuridad cerca de la Puerta de Tannhäuser.
Todos esos momentos se perderán... en el tiempo... como lágrimas en la lluvia. Es hora de morir»
Roy Batty, un replicante, en la escena Lágrimas en la lluvia de Blade Runner

«No temo a los ordenadores; lo que temo es quedarme sin ellos»
Isaac Asimov

«Las máquinas serán capaces, en los próximos veinte años, de hacer cualquier trabajo que una persona pueda hacer»
Herbert Simon, 1965

Desde la antigüedad, el hombre ha ido construyendo diferentes artefactos que le ayudasen en su trabajo diario, que pudieran hacer su vida mejor. Los primeros diseños datan de la antigua Grecia, para, posteriormente, ser creado en Italia, por Leonardo da Vinci, el primer caballero mecánico, siendo seguido de un pato mecánico en 1738 y de los robots japoneses que sirven té en 1800. Tras estos primero artefactos, ya se escribió sobre el primer robot de la mano de Karel Čapek, se presentó el primer robot humanoide en la exposición mundial de Chicago y, por fin, se dio el último paso en la ciencia ficción con las novelas y las leyes de la robótica de Isaac Asimov.

Tras todo esto, comenzaron a hacerse realidad los sueños, los prototipos y la ciencia ficción. La idea inicial de su uso era ser utilizados en las fábricas, para así alejar a los trabajadores de peligros o mejorar la rapidez de la cadena de montaje. Sin embargo, con el tiempo y con las nuevas necesidades de la gente, estos han ido extendiéndose en el ámbito doméstico para ayudarnos en nuestro día a día, como se contaba en la ciencia ficción, robots haciendo tareas de seres humanos.

Estos pueden ser fijos o móviles, pueden estar en tierra, mar, aire o en el espacio, pueden parecer simples «máquinas» o tener aspecto humano, pueden ser autónomos o controlados por el ser humano. Unos trabajan en fábricas en procesos especializados, otros en agricultura, ganadería, ayudan en hospitales o cuidan de personas dependientes. Hay muchos tipos, y, en este capítulo, se verán.

Robótica

12.1 *INTRODUCCIÓN*

Un robot es un sistema físico reprogramable equipado con diferentes sensores y actuadores que toman entradas sensoriales del ambiente que los rodea para procesarlas y actuar sobre este ambiente a través de los actuadores que les permiten realizar diferentes tareas en base a su programación para así conseguir una meta [718], [719]. Por ejemplo, los robots, a diferencia de nosotros, los seres humanos, son capaces de recoger datos sistemáticos, coherentes y repetitivos de una forma que permiten su comparación, permitiendo realizar análisis a gran escala o bien Minería de datos sobre estos [720]. Además, otra característica de los robots es que, en el caso de que funcionen correctamente y estén bien programados, pueden trabajar sin ningún fallo a escala milimétrica, algo que a nosotros nos es muy difícil o nos resulta imposible. Por ende, lo más importante en un robot es su autonomía y que un ser humano no deba de controlarlo [114].

La robótica comenzó como disciplina independiente en el año 1960 [721], [722]. En esta época los robots se utilizaban en la producción de coches. Sin embargo, según fue avanzando el software y el hardware fueron permitiendo que los robots fueran más inteligente y veloces en sus respuestas, haciendo que pudieran utilizarse en el mundo real [721], en el día a día, en tareas más cotidianas. Ejemplos de esto son su uso en la agricultura, la construcción, en la logística, en el cuidado de gente mayor, en tareas militares y en nuestras casas [114].

Sobre todo, esta evolución de los robots ha sido dominada por las necesidades humanas [722]. Primeramente, en la revolución industrial se crearon robots para ayudar a los seres humanos en las fábricas y así alejar a los operadores de las máquinas de los riesgos y daños posibles derivados de utilizarlas [722]. Actualmente, los sectores industriales donde se utilizan más robots son el automotriz, el petroquímico, el de la electrónica, la metalurgia y el alimenticio [723].

Posteriormente, estos robots fueron creándose con mayor inteligencia para así incorporarlos a otros procesos de producción que requerían robots con una mayor flexibilidad e inteligencia [722]. El clásico ejemplo de estos son los robots de las cadenas de montaje de las fábricas de coches, mostrado en la izquierda de la Ilustración 57, acompañándolo a la derecha de otro robot industrial destinado a colocar pallets de productos alimenticios.

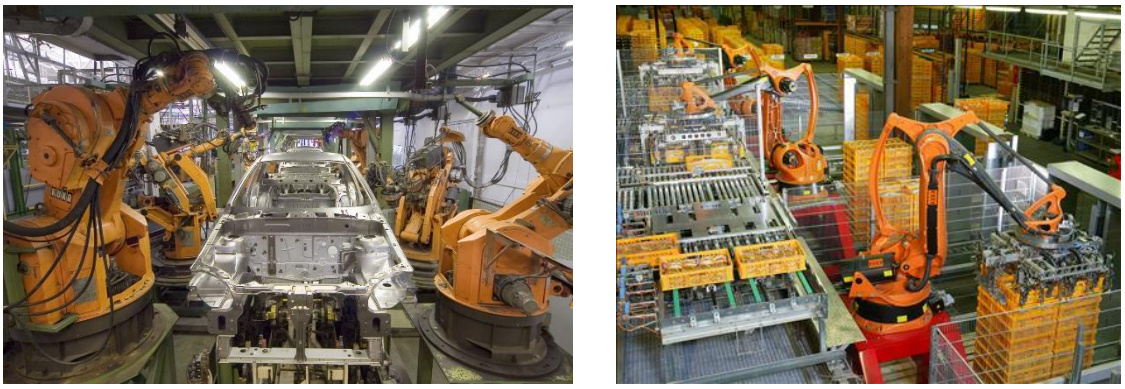


Ilustración 57 El robot industrial de ensamblaje automotriz KUKA [178] [179]

Un ejemplo de estos nuevos robots inteligentes aplicados a la industria es el de Kiva Systems. Estos robots son capaces de compartir memoria y han estado apareciendo en los últimos años a lo largo de la industria, donde empezaron en el verano de 2006 y que ya disponía de 1.000 robots que comparten conocimiento para crear y compartir modelos que permitan la navegación autónoma y el despliegue rápido en entornos semiestructurados con una alta fiabilidad a pesar de las limitaciones económicas. Estos robots cogen automáticamente las estanterías y las transportan desde el centro de almacenaje hasta las estaciones de trabajo, siendo este sistema de robots compuesto por las estanterías y los robots que las transportan [724], [725]. Estos pueden levantar entre 454 Kg y 1.362 Kg, dependiendo del modelo, y viajar a una velocidad de 1,3 metros por segundo [725]. Este sistema fue adquirido posteriormente por Amazon para usarlo en sus almacenes y realizar la tarea descrita anteriormente. La Ilustración 58 muestra una fotografía de los robots Kiva en los almacenes de Amazon.

Ilustración 58 Robots Kiva
(Fotografía cedida por Amazon)
©Amazon.com

Actualmente, los robots se han adaptado al mercado, no siendo solo exclusivos de los procesos industriales, pudiendo ser vistos para cubrir las nuevas necesidades humanas que surgen a raíz de la modernización de los países desarrollados y, en algunos casos, tratando de cumplir lo visto en las películas de ciencia ficción de los años 1920, ejemplo de esto último es el robot sirviente *Advanced Step in Innovative Mobility* (ASIMO) de Honda (Ilustración 59), o si se mira en otros ejemplos tenemos los robots dedicados a la limpieza, el desminado, la construcción y la agricultura [722]. Estos son, por decir algunos, las aspiradoras que hacen un mapa de la casa y la aspiran hasta volver a su posición inicial para recargar la batería, o las

[178] https://ca.wikipedia.org/wiki/Fitxer:KUKA_Industrial_Robots_IR.jpg
[179] https://en.wikipedia.org/wiki/File:Factory_Automation_Robotics_Palettizing_Bread.jpg

diferentes máquinas robóticas utilizadas en ganadería para arrear el ganado, como Rover, para ordeñar vacas o esquilar ovejas, o bien en la agricultura que cuenta con los diferentes intentos de crear robots que sean capaces de cultivar mediante el uso de sensores. Por otro lado, hay gente que piensa que los robots no solo deberían de servir para ayudar en la casa, sino también como entretenimiento, lo que implica que los robots deberían ser de multipropósito y poseer habilidades multifuncionales [723].

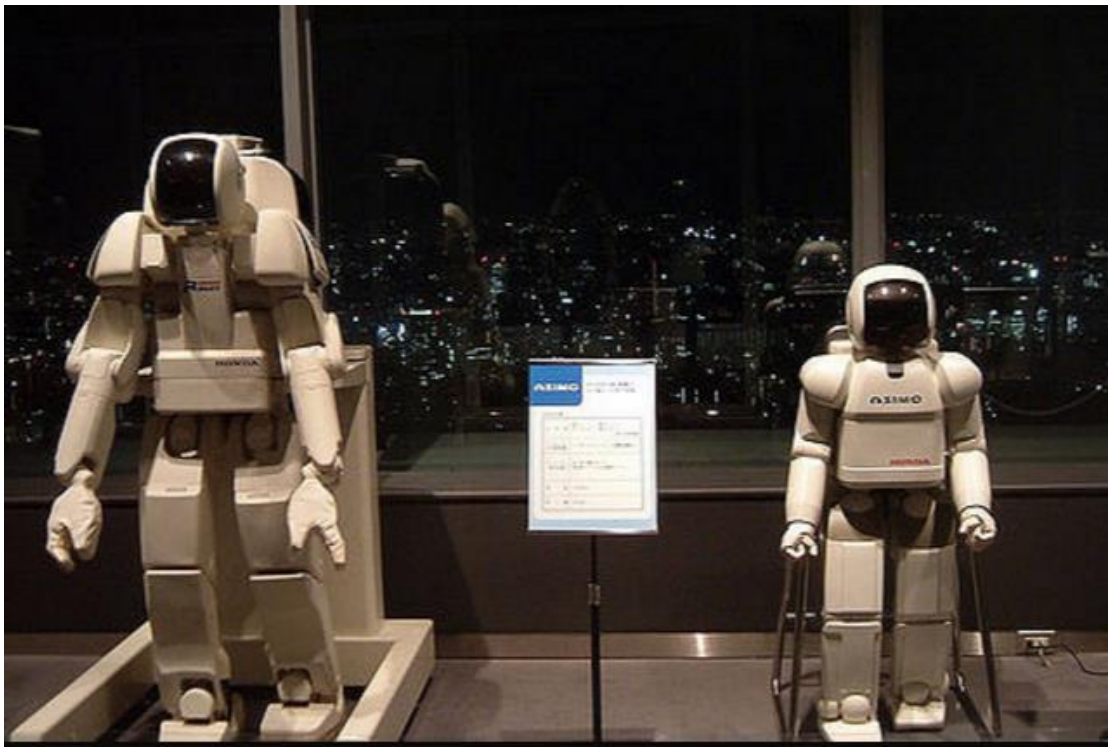


Ilustración 59 ASIMO a la derecha y su predecesor P3 a la izquierda [180]

Sin embargo, hay muchos más ejemplos en otros ámbito como se muestra en [724]. Ejemplos de este tipo de robots son el Packbot de iRobots y el Predator, ambos son robots militares utilizados en Irak y Afganistán. En el ámbito científico está el Mars rover, el cual es utilizado para explorar Marte. El Aerosonde es utilizado para sondear sistemas meteorológicos y volar en tormenta tropicales. Para casas está el PatrolBot de ActivMedia que permite la monitorización de edificios. Mientras, el Tug de Aethon es utilizado para conducir carros de suministros en los hospitales. Otros casos son los robots de limpieza como la Roomba para aspirar, la Scooba para fregar y el Friendly Robotics para segar el césped.

Cabe destacar que hay muchos tipos de robots, como se verá a continuación, pues pueden ser autónomos o no, fijos o móviles, trabajar en equipo o independientemente y hasta aprender. Lo que se conoce como robots, o **robots tradicionales**, pueden operar independientemente gracias a que disponen de un sistema de abordo que les permite realizar la computación necesaria y mantener una memoria almacenada [451]. Estos robots requieren tener una amplia memoria preprogramada y almacenada para poder realizar cada tarea, lo que les obliga a poseer una gran capacidad de procesamiento y grandes baterías [726].

[180] https://www.flickr.com/photos/machu/39761200/in/photostream

Robótica

En el año 2010 se estimó que había 8 millones de robots del mundo que eran juguetes, robots preprogramados para limpiar el suelo o cortar el césped, mientras que solo había 1 millón de robots industriales [720]. Estos han sido desplegados ampliamente para realizar en fábricas para hacer tareas tediosas, repetitivas o peligrosas, como ensamblar, pintar, empacar y soldar piezas [720], [727]. Estos han resultado un éxito debido a su resistencia, velocidad y precisión [727].

Sin embargo, los robots son la confluencia de muchas áreas pues necesitan mecatrónica [181]. Dentro de esta, si nos centramos en la informática, que es a lo que atañe esta tesis doctoral, son necesarias muchas y muy diferentes áreas. Una de estas, tal vez la más importante, es la Inteligencia Artificial y sus diferentes vertientes, como son *softcomputing*, las redes neuronales, algoritmos genéticos y la Lógica [728]. La IA les permita realizar su trabajo, como es esquivar muros y obstáculos, autoorganizar a los grupos de robots, planificar el camino óptimo y tomar decisiones [728].

Un ejemplo es el caso de Kiva [724]. El uso de IA en Kiva les permite crear algoritmos que controlen el tráfico entre los propios robots, les planifiquen el camino por medio del algoritmo A*, y les ayuden a asignar los recursos de los robots de forma que estos hagan que los trabajadores estén tan ocupados como sea posible utilizando el mínimo hardware, espacio del almacén e inventario posible. Por otro lado, otros robots necesitan de sistemas de Visión por Computador para poder reconocer objetos y ofrecer así servicios adicionales, como son la localización, reconocimiento de escenarios y la manipulación de objetos [720]. En el caso de tener una memoria compartida, puede que se necesite *Big Data* para realizar el procesado de información, aplicar Big Data a ciertos algoritmos de Machine Learning, almacenar los datos, y, claramente, en este caso, La Nube (Cloud Computing), pues todo se subirá a la nube y podrán compartir a través de ella la inteligencia, experiencias y decisiones para poder ponerse de acuerdo en las acciones en las que intervengan varios robots [451].

12.2 *ÁREAS DE LA ROBÓTICA*

Hay tres áreas diferenciadas en la robótica [722]: robots manipuladores, robots móviles y robots inspirados en la biología, también conocidos como biorobots. Los **robots manipuladores** incluyen a los robots industriales, los robots médicos y los robots de rehabilitación, así como servicios similares como son los de reabastecimiento de combustible, embalaje y *paletización*. Los **robots móviles** son aquellos vehículos terrestres, submarinos y aéreos. Por último, los **robots inspirados en la biología** están compuesta por los robots que caminan, los robots humanoides (ASIMO, HRP-4C, TOPIO y QRIO) y algunos sistemas submarinos. Sin embargo, sendas áreas convergen en su función original, actual y futura [722].

[181] Glosario: **Mecatrónica**

12.2.1 ROBOTS MANIPULADORES

Los **robots manipuladores** también son conocidos como **brazos robóticos** [722], como se ve en la Ilustración 60, la cual enseña un brazo robótico capaz de escribir y otro brazo robótico trabajando en una fundición. Estos robots pertenecen a cadena en serie y están formados por extremidades rígidas diseñadas para realizar una tarea determinada.

Ilustración 60 Robot industriales KUKA [182] [183]

En este aspecto, probablemente cuando se habla de robots en las fábricas el pensamiento más difundido sea el de los brazos robóticos, aunque también estas poseen, en algunos casos, robots móviles y biorobots. Los brazos mecánicos creados por diferentes empresas, pues existen centenares diferentes de ellos, para cumplir un propósito específico en sitios donde el ser humano no puede o no debería estar debido a las condiciones del lugar, como puede ser una alta temperatura, contaminación en el aire o levantamiento de pesos excesivamente pesados, pintura, soldadura, entre otras muchas situaciones [729], [730]. Además, estos cuentan con una gran precisión, lo que los hace ideales para lugares donde no hay espacio para el error [729] y muchas repeticiones [730].

Los brazos robóticos tienen una serie de características [729], que se pueden considerar estándares entre todos ellos [730], como son el eje, los grados de libertad [184], su espacio de trabajo, su cinemática, el peso que puede levantar, su velocidad y aceleración, su precisión y repetitividad, y su control de movimiento. Estas características son las que diferencian los brazos robóticos y los que permiten que puedan realizar un trabajo u otro según las configuraciones de las que dispongan y lo que estos elementos le permitan realizar. En [730] se puede ver un estado del arte sobre los diferentes brazos mecánicos existentes en el mercado, con sus diferentes características y precios, divididos en cuatro grupos: brazos educativos baratos, brazos industriales de precio bajo, brazos orientados a la investigación y brazos modulares ligeros, que son aquellos que cuentan

[182] https://commons.wikimedia.org/wiki/File:KUKA_Industrial_Robot_Writer.jpg
[183] https://commons.wikimedia.org/wiki/File:Automation_of_foundry_with_robot.jpg
[184] Glosario: **Grados de libertad**

con más movimiento debido a sus juntos y otros componentes y han diseñados para trabajar en ambientes naturales y con seres humanos.

Los primeros brazos robóticos que surgieron fueron aquellos que se utilizaban para soldar y pintar. Sin embargo, la evolución del mundo de la tecnología, de la mano de los nuevos requisitos surgidos han hecho que estos evolucionasen y nos encontremos actualmente con brazos robóticos que ayudan en las cirugías, en la rehabilitación y en el repostaje de combustible. En base a esto, se pueden dividir en tres tipos los **robots manipuladores** o **brazos robóticos**, a saber: robots industriales, robots médicos y robots de rehabilitación.

12.2.1.1 ROBOTS INDUSTRIALES

Los **robots industriales** surgieron alrededor del año 1960 y han dominado desde entonces hasta los 90 la investigación [722]. El típico uso de estos robots es el de manipulación de objetos e interacción con el entorno que los rodea. Principalmente su mayor usuario y el que dictaba su diseño era la industria del automóvil, determinando así su desarrollo e investigación. Una de las mayores investigaciones de los **robots industriales** ha sido la cinemática [185] [731], para así calibrar cómo debían de moverse los elementos del robot según su posición, aceleración y velocidad en el tiempo. Otra línea de investigación ha sido la de planificación del movimiento para así calcular los subobjetivos que el robot debe de cumplir para cumplir la meta final, en conjunto con las leyes de control que se enfocan en las técnicas de controlar el robot.

A comienzos de los 90s, los robots industriales comenzaron a ser usados en otras áreas gracias a la posibilidad que ofrecían de adaptarse a las características físicas del producto, como su tamaño, forma y rigidez. Estas nuevas áreas fueron la industria alimentaria, la farmacéutica y los servicios postales. En esta época, la investigación principal comenzó a ser la habilidad de cómo autoadaptarse al producto y al ambiente, siendo así el principal objetivo el conseguir otorgar al robot la suficiente inteligencia y capacidad de solventar el problema, por ejemplo, mediante Lógica Difusa. Otras veces era introducido un teleoperador para manejar el robot a distancia, pero que tenía que problema el posible retraso en la comunicación, o la teleprogramación, en el que un operador le mandaba instrucciones de alto nivel al robot para que este las ejecutase en bucle.

12.2.1.2 ROBOTS MÉDICOS

Los **robots médicos** se han estado incrementando los últimos años para poder ayudar en el trabajo diario a los médicos y las enfermeras [722]. Estos comenzaron a hacerse realidad en los años 1990, ofreciendo desde entonces muchas y muy diversas funciones como robots de laboratorio, telecirugía, entrenamiento quirúrgico, cirugía remota, telemedicina y teleconsulta, rehabilitación, ayuda para personas sordas y/o ciegas, y robots de hospital. Estos robots permiten ayudar en operaciones de pacientes que han tenido un ataque al corazón o en aquellas que se necesite ajustar milimétricamente, como es el caso de las prótesis. En otros casos podrían ayudar a mover y trasladar productos y/o pacientes entre salas, camas o al baño y así evitar diferentes enfermedades relacionadas con este trabajo a los médicos [732]. En la Ilustración 61 se muestra a un robot de cirugía y un robot dispensador de medicación.

[185] Glosario: **Cinemática**

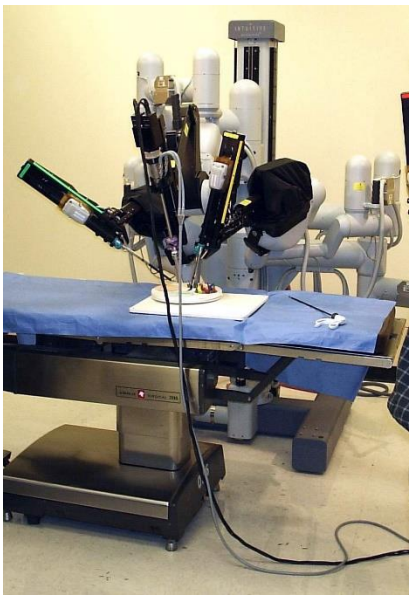


Ilustración 61 Robots médicos [186] [187]

Otros ejemplos de robots médicos es el de HelpMate [732], que es un robot que reparte la comida fuera de horario, lleva los expedientes de los pacientes, y porta suministros de laboratorio y de farmacia. El problema residía en que el robot necesitaba operar sobre una red rígida enterrada o pegada al suelo. HelpMate ha sido instalado en varios hospitales, los cuales reportaron un incremento de productividad y eficiencia al poder usarlos durante las 24 horas del día. Por otro lado, en [733] presentaron un robot llamado MELKONG que podía levantar, agarrar y transportar pacientes de hasta 100 kilogramos.

Los robots médicos, según [722], pueden clasificarse: por el diseño de manipulación (cinemático o actuación), por su nivel de autonomía (preprogramado, teleoperación o control cooperativo), por la anatomía específica o técnica (cardiaco, intravascular, percutánea, laparoscópica, microquirúrgica), por su entorno operativo (escáner, quirófano convencional, etc.).

Otra forma de clasificar la robótica en la medicina sería de acuerdo con [732], quienes crean tres divisiones: macrorobótica, microrobótica y biorobótica. La macrorobótica es la utilizada en la rehabilitación, y que se ven en el apartado Robots de rehabilitación. La microrobótica es la utilizada para operaciones, como son endoscopias y operaciones mínimamente invasivas. La diferencia entre estas dos primeras son tanto el tamaño, como la forma de operar y de ser de las herramientas, pues en la primera son herramientas rígidas y autónomas, mientras que en la segunda con flexibles y teleoperadas. La biorobótica trata acerca de la creación de órganos, o seres basados en la biología, lo cual se verá en el apartado Biorobótica o robots inspirados en la biología. Como se ve, esta clasificación sería relativa a solo los robots utilizados en la medicina.

[186] https://en.wikipedia.org/wiki/File:Laproscopic_Surgery_Robot.jpg
[187] https://commons.wikimedia.org/wiki/File:US_Navy_030819-N-9593R-092_The_Autoscript_robot_dispenses_medication_into_a_bottle_at_the_National_Naval_Medical_Center_in_Bethesda,_Maryland.jpg

Sin embargo, los robos médicos tienen muchos problemas que afrontar como son la seguridad y la precisión, pues puede costar la vida a un paciente, sus elevados costes, y la reticencia a aceptar que se utilicen robots para operar, pues tienen que ser adaptados para personas discapacitadas o no cualificadas en su funcionamiento [722]. Otras veces en cambio son percibidos como competidores o peligrosas máquinas por y para los pacientes [732]. En este aspecto, las líneas de investigación de estos robots están todas encaminadas al diseño, la programación y el control de las interfaces humano-máquina.

12.2.1.3 ROBOTS DE REHABILITACIÓN

Como último tipo de robot industrial están los **robots de rehabilitación** [722], [732]. Estos comenzaron a aparecer en los años 1960 [719], [734], y lentamente han ido evolucionando, pues este concepto incluye una amplia gama de dispositivos mecatrónicos que van desde las extremidades artificiales para apoyar la terapia de rehabilitación y ayudar a la gente con discapacidades o anciana, hasta proporcionar asistencia personal en los centros hospitalarios y residenciales.

Esta mejora en la vida de este tipo de personas se puede conseguir mediante robots manipuladores, sillas de ruedas inteligentes e interfaces específicas para los aparatos domésticos. Otro sistema es a través de la restauración de las alteraciones funcionales por medio de prótesis avanzadas, órtesis [188] y la estimulación eléctrica funcional, de forma que permitan el desarrollo de entornos virtuales de terapias rehabilitación [732].

El primer robot de rehabilitación fue una órtesis con cuatro niveles de libertad para poder mover brazos paralizados [734]. Tras esta, se creó otra con siete grados de libertad [734]. Posteriormente, a mediados de 1970, esta área comenzó a tener trabajo más específico con muchos otros trabajos como se muestra en [719], con diferentes diseños de puestos de trabajo que contaban con escritorios móviles y brazos electrónicos. Posteriormente, en los 80s, surgieron diferentes robots asistentes de varias clases [719].

El primero son los robots fijos en los lugares de trabajo, como los vistos anteriormente. Otros son los dispositivos de alimentación asistida para ayudar a comer a las personas con problemas, donde el primero surgió en 1987 para ayudar a comer por sí mismo a un niño de 12 años con paraplejia espástica, el cual después fue reprogramado para que también diera de beber, lavase los dientes, aplicase maquillaje, afeitara, dibujara, pintara y jugase a juegos [735]. Otro dispositivo para alimentar fue el Winsford Feeder, que estuvo a la venta durante 15 años [719]. Por último, están los robots de asistencia móvil [719], los cuáles son similares a los primeros, pero que cuentan con la libertad y movilidad necesarias para realizar otras tareas en cualquier sitio, pues en la vida cotidiana no se está en frente de un escritorio constantemente, como puede ser coger un libro de la estantería o ayudar a cocinar. Uno de estos éxitos ha sido la silla de ruedas de Manus que contaba con un manipulador que permitía beber agua a sus usuarios [736], puesta a la venta en 1990. Sin embargo, este no fue el único complemento para sillas de ruedas [719], pues hubo muchas más como el «Air Muscle», el «Raptor Arm», entre otros. Otros ejemplos de robots de asistencia móvil es el Mobile Vocational Assistive Robot (MoVAR) de la Universidad de Stanford [737], un brazo robótico que incorpora una videocámara.

[188] Glosario: **Órtesis**

En algunos casos también se han utilizado para ayudar en la educación a los niños con diferentes discapacidades [719]. Un ejemplo ha sido el de la Universidad de Cambridge con un niño que tenía parálisis cerebral y en donde utilizaban un robot que servía para construir torres con bloques, ordenar bloques por color o forma y en resolver el puzle de la torre de Hanói [738]. Dexter es una mano mecánica para interpretar el alfabeto dactilológico que utilizan las personas que son ciegas y sordas. para ello, traduce los caracteres ASCII en las posiciones de los dedos de la mano de forma que el usuario que es sordo y ciego puede tocarlo y «leer» [739].

Sin embargo, esta área está menos desarrollada que los robots industriales [722]. Los robots de rehabilitación, a diferencia de los industriales, se mueven mucho más lentos y son más dóciles de manejar para facilitar la interacción segura con el usuario.

12.2.2 **ROBOTS MÓVILES**

Los primeros **robots móviles** fueron instalados en fábricas casi al mismo momento que los robots industriales, sobre el año 1968, donde estos no eran más que vehículos automáticos guiados (AGV) que servían para transportar herramientas y seguir trayectorias predefinidas [722].

Sin embargo, el término **robots móviles** ha ido evolucionando y encuadra aquellos robots que son capaces de llevar a cabo tareas en diferentes sitios y que están formados por una plataforma que se mueve por elementos locomotores, elementos que dependen del ambiente donde operará, pudiendo ser aéreo, acuático, terrestre [722], o incluso el espacio. Un robot aéreo es conocido también como vehículo aéreo no tripulado (UAV), dron o sistema de avión no tripulado (UAS), un robot acuático es conocido como vehículo autónomo submarino (AUV), un robot que opera sobre la superficie de la Tierra como vehículo terrestre no tripulado (UGV), y los robots espaciales como nave espacial robótica, que son tipos de nave espaciales no tripuladas.

Por lo general, en los robots acuáticos y marítimos suelen utilizar como sistema locomotor hélices, aunque en los marítimos hay robots que andan por el fondo marítimo gracias al uso de «piernas» [722]. Mientras, sobre la Tierra es más complicado, pues hay superficies muy diferentes entre ellas, lo que da lugar a tener que elegir entre diferentes sistemas locomotores con prestaciones muy variadas como son ruedas, orugas o piernas, entre otros [722].

Por otro lado, los robots espaciales pueden ser de varios tipos según la propia NASA [740]. Estos pueden ser robots diseñados para realizar misiones en la superficie de los planetas, en una órbita o en la profundidad del espacio, ya sea explorando, operando, inspeccionando o dando soporte a los humanos, y siendo tanto autónomos como teleoperados.

En la Ilustración 62 se muestra un ejemplo de cada tipo, siendo el primero un robot aéreo ganador de la tercera misión de competición internacional de robots aéreos (IARC), el segundo es un robot acuático diseñado para llevar a cabo un curso de asalto bajo el agua de forma autónoma y sin control externo. Por último, la tercera fotografía de esta ilustración muestra un diseño conceptual del Mars Rover vaporizando rocas en Marte.

Ilustración 62 Robots móviles [189] [190] [191]

[189] https://commons.wikimedia.org/wiki/File:TU-Berlin_IARCmachine.jpg
[190] https://commons.wikimedia.org/wiki/File:Blackghost.jpg
[191] https://commons.wikimedia.org/wiki/File:Martian_rover_Curiosity_using_ChemCam_Msl20111115_PIA14760_MSL_PIcture-3-br2.jpg

12.2.2.1 RETOS E INVESTIGACIONES ABIERTAS

El principal reto de la investigación que se realiza en el campo de los robots móviles se encuentra en el movimiento de forma autónoma en zonas tanto interiores como exteriores. En los primeros debido a que el robot debe de aprender cómo moverse y crear un mapa del entorno. Mientras que en los segundos necesita mapas del mundo o el área en que está para poder localizarse y planificar el movimiento por ellos [722]. En todos estos casos, el robot necesita sensores que recojan datos del ambiente que le ayuden a localizarse en el mundo por medio de la toma de diferentes medidas y del estado del robot. En el caso de robots de interiores están los robots aspiradores, que necesitan recoger datos de la casa para crear su propio mapa y/o no chocar contras los objetos ni caer por las escaleras. Incluso también hay robots guía, como es en el caso de algunos museos.

En el caso de los robots acuáticos, su uso se ha incrementado, sobre todo a partir de los 90s [741], con el fin de poder explorar las zonas más profundas y peligrosas de los océanos [722]. En esta década fueron puestos 34 AUV, presentados en [741], junto a los modelos existentes y las tecnologías necesarias para las diferentes misiones. Estos 34 AUV realizaron trabajos de mantenimiento de cables, inspección del fondo oceánico, mapeo y búsqueda, bancos de prueba, inspección, la pesca, la monitorización de la polución, el rescate, la limpieza de basura, investigación geológica, eliminación de minas submarinas, mantenimiento de estructuras, etc., a distancias de hasta 6.000 metros de profundidad. Normalmente muchos de estos están atados y son remotamente controlados, conocidos como vehículos manejados por control remoto (ROV), aunque la demanda está yendo hacia los AUV, convirtiendo en la línea de investigación principal el mejorar la autosuficiencia, inteligencia y toma de decisiones de estos robots. Otras investigaciones van por el camino de evitar las perturbaciones externas, los métodos de sentir y localizar en ambientes oscuros y ruidosos, y la imposibilidad de realizar transmisión electromagnética.

En el caso de los robots móviles domésticos tienen que tener en cuenta diferentes consideraciones para ser seguros y evitar posibles riesgos con los habitantes de la casa, con la casa y con ellos mismos, como bien se estudió en este *survey* [742]. Estos riesgos pueden ser de diferente índole. En el caso de los seres humanos, el robot puede hacerles daños con su propio cuerpo, si posee bordes constantes mediante las colisiones que puede haber, pero que debería de evitar, pero que en caso de no hacerlo puede conllevar a causar contusiones o cortes. Lo mismo ocurre con los riesgos que sufre la casa, pues puede romper diferentes objetos o dañar el mobiliario. Sobre los riesgos del propio robot están, sobre todo, las escaleras, pues el robot en caso de no detectarlas puede caer.

12.2.3 **BIOROBÓTICA O ROBOTS INSPIRADOS EN LA BIOLOGÍA**

Estos son los robots que rompen con el esquema tradicional visto en los robots móviles y que buscan encontrar una nueva dimensión basándose en la biología conocida, ya sea con la inclusión de piernas, o mediante otros sistemas locomotores como son el reptar como las serpientes o nadar como los peces. En el primer grupo se encuentran los **robots que caminan** y los **humanoides**, siendo estos los más extendidos de este grupo de **robots inspirados en la biología** [722], también conocida como **biorobótica** [732], donde ambos utilizan piernas como sistema locomotor, pero difieren en su aplicación final de cara a los usuarios,

pues mientras unos caminan, los otros tienen que parecer humanos. Este grupo también contiene aquellos robots inspirados en cualquier cosa de la biología, como puede ser en los animales, simulando así su movimiento o tratando de imitarlos a imagen y semejanza. En la Ilustración 63 se muestran un robot de cada tipo, siendo el primero el Nao de Sony como robot sobre piernas, los segundos unos robots serpiente y por último el robot acuático iSplash II de iSplash-Robotics.

Ilustración 63 Robots inspirados en la biología [192] [193] [194]

[192] https://commons.wikimedia.org/wiki/File:Innorobo_2015_-_NAO_(cropped).JPG
[193] https://www.flickr.com/photos/jurvetson/32689486/
[194] https://commons.wikimedia.org/wiki/File:ISplash_Robotic_Fish.jpg

12.2.3.1 ROBOTS QUE CAMINAN

Las piernas de estos robots están formadas por dos o tres grados de libertas, lo que hace que compartan muchos problemas técnicos con los robots industriales y los robots móviles [722]. Sin embargo, estos no tienen por qué ser humanoides ni bípedos, pues pueden tener 2, 4, 6, 8 o más «piernas» o «patas». En la Ilustración 64 se pueden ver diferentes robots que caminan de 2 y 4 patas. El primero es un RunBot [743], robot bípedo que camina en círculos, el segundo es el *Legged Squad Support System* (LS3) [744], perteneciente a un proyecto de la Agencia de Proyectos de Investigación Avanzados de Defensa (DARPA) de los Estados Unidos de América, que es un robot autónomo que puede ser utilizado como caballo de carga por los soldados o marines, y por último un robot de la armada estadounidense subiendo por un monte nevado.

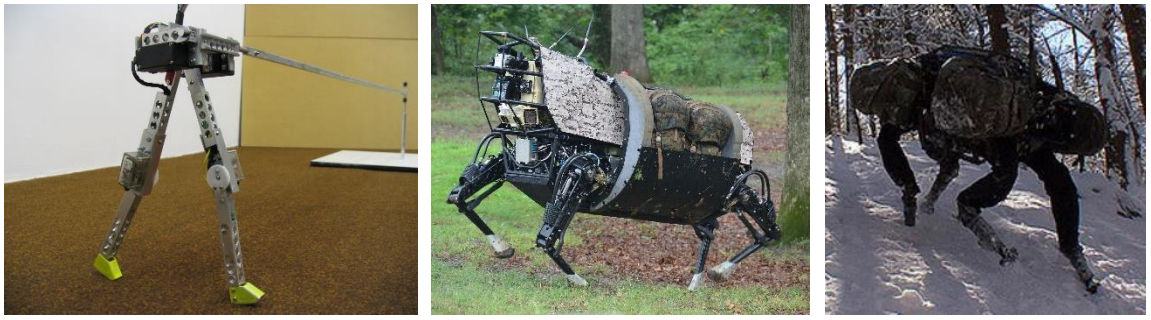


Ilustración 64 Robots que caminan [195] [196] [197]

No obstante, aunque puedan parecer iguales a los robots móviles, hay ciertas ventajas que obtienen al utilizar piernas [722]. La principal característica es el poder caminar por terreno irregular manteniendo el cuerpo estable, lo que implica que puedan subir escaleras, obstáculos y zanjas. Además, pueden andar por terreno arenoso y suelto. También disponen de omnidireccionalidad al poder moverse hacia cualquier sentido e infligen menos daño al medioambiente al no necesitar ruedas u orugas que lo dañan más.

A pesar de estas ventajas, también disponen de contras [722], pues las piernas ofrecen muchos problemas, siendo las principales investigaciones actuales la coordinación y el movimiento de estas, así como la estabilidad del robot. Además, en comparación con las ruedas, las piernas necesitan ser más rápidas de lo que son actualmente, lo que hace que requieren que sean dinámicamente estables.

Estas cualidades hacen que los robots que caminan sean ideales para sustituir a los robots móviles de ruedas utilizados en la agricultura y en la silvicultura, pues así se reduciría el impacto en el medioambiente [722]. Otras utilidades son la limpieza, la inspección de elementos, y el mantenimiento de edificios u otras estructuras, pues el poder caminar les ofrece las ventajas de andar por cualquier tipo de superficie, algo que no pueden los robots móviles [722]. Incluso ya se han desarrollo robots que caminan capaces de soldar en la construcción naval [745] y de trabajar en la consolidación de paredes rocosas y laderas mediante perforación

[195] https://commons.wikimedia.org/wiki/File:RunBot_biped_walking_robot_by_Tao_Geng.jpg
[196] https://commons.wikimedia.org/wiki/File:Legged_Squad_Support_System_robot_prototype.jpg
[197] https://www.flickr.com/photos/rdecom/4115749254

para asegurar estas zonas de posibles catástrofes y sin necesidad de utilizar a personas que hagan rappel para hacer los agujeros de forma manual [746].

12.2.3.2 HUMANOIDES

La robótica humanoide consiste en cómo hacer a los robots más humanos, es decir, investigar en ver como se comunican, expresan sus emociones y realizan «cosas» o actividades de seres humanos. Esta es la principal diferencia con los robots que caminan.

Las primeras investigaciones sobre robots bípedos surgieron entre los 1967, 1969 y 1970 con diferentes artículos entre los que se encuentra siempre Miomir Vukobratovic [747]–[750]. En estos artículos, los autores trataron de imitar el movimiento de cuadrúpedos y bípedos mediante diferentes algoritmos para así poder controlar un robot, sabiendo que necesitaban controlar la velocidad y dirección del movimiento, hacer movimientos automáticos mientras conseguían estabilidad, y conseguir sincronizar los movimientos.

Esto primeros prototipos eran bastante menos sofisticados en comparación con los prototipos actuales [722], que son los que se muestran en la Ilustración 65, la cual muestra al robot de búsqueda y rescate estadounidense Atlas, en desarrollo desde el año 2013, a TOPIO (TOSY Ping Pong Playing Robot), robot bípedo humanoide desarrollado para competir en tenis de mesa contra humanos, y a Sony QRIO (Quest for curRIOsity).

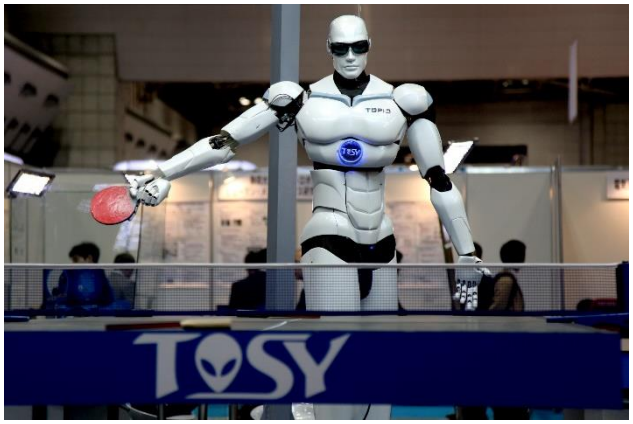


Ilustración 65 Robots humanoides [198] [199] [200]

Estos nuevos robots permitieron crear tres grandes áreas de investigación dentro de los humanoides [722]. La primera de ellas es la generación del paso, la cual puede ser offline, que no permite adaptarse al ambiente de forma dinámica al robot, u online, que va generando todos los ángulos adecuados dinámicamente. También se destinan esfuerzos en la reducción del consumo de energía durante el caminar del robot. El segundo campo es el del control de estabilidad. Por último, el tercer campo es el diseño del robot, donde se busca la mejor relación diseño/estabilidad en los robots bípedos mediante el uso de motores DC y músculos artificiales, otras

[198] https://commons.wikimedia.org/wiki/File:Atlas_frontview_2013.jpg
[199] https://en.wikipedia.org/wiki/File:TOPIO_3.jpg
[200] https://commons.wikimedia.org/wiki/File:Sony_Qrio_Robot.jpg

investigaciones en esta área tratan de tener una buena eficiencia energética, o de que sea posible utilizarlos en terrenos irregulares.

Otra línea de investigación en los humanoides es la que investiga en que el robot pueda tratar con los humanos de forma segura y este pueda ser capaz de expresar emociones. Es decir, se podría decir que el fin de esta línea de investigación es hacer un robot a imagen y semejanza de los humanos y que, al igual que pasa con el test de Turing en la Inteligencia Artificial, en este caso no se pudiera diferenciar al humanoide de un ser humano a simple vista. Un ejemplo de ello son Repliee Q1 y Repliee Q2 mostrados en la Ilustración 66, que posee sensores y actuadores internos y una «piel» realizada en silicona, 47 articulaciones, reconocimiento del discurso, parpadeo y sensibles al tacto y que representan a una mujer.

Ilustración 66 Humanoides con aspecto de ser humano [201] [202]

Estas actividades que pueden realizar las Repliee Q1 y Q2 son también líneas de investigación dentro de los humanoides. Estas son el crear interfaces para facilitar su uso a usuarios sin habilidades en el uso de robots, reconocimiento del discurso, en la electromiograma [203], en el electrooculograma [204], en crear robots más

[201] https://commons.wikimedia.org/wiki/File:Actroid-DER_01.jpg
[202] https://commons.wikimedia.org/wiki/File:Repliee_Q2.jpg
[203] Glosario: **Electromiograma**
[204] Glosario: **Electrooculograma**

seguros mediante un mejor control del movimiento, mejorar la expresión y percepción de los robots, y en mejorar el aprendizaje de los robots mediante imitación y tutelaje [722].

12.3 *CLASIFICACIÓN DE LOS ROBOTS*

12.3.1 **DESDE EL PUNTO DE VISTA DE SU AMBIENTE**

El control de los robots está directamente relacionado con el ambiente impredecible e inestable en el que están, con como de rápido el robot puede reaccionar a este, y como de compleja es la tarea [718]. Al principio había dos tipos de robots de acuerdo con R.A. Brooks [721], que han sido ampliados por Mataric posteriormente [718]:

Situados (*Situated*): estos robots son aquellos que están situados en el mundo y son consciente del «aquí» y del «ahora» del sitio que los rodea, pues este afecta a su comportamiento. Estos robots están incrustados en complejos y dinámicos ambientes, siendo estos los que conforman su comportamiento. Ejemplos de estos son los coches autónomos, equipos de robots que interactúan entre ellos, un robot en un museo lleno de gente, los robots médicos de hospitales o los robots Kiva. Estos robots son un desafío para el diseñador debido a la falta de previsibilidad y estabilidad del ambiente, al contrario que los robots no situados.

Personificados (*Embodiment*): estos robots tienen cuerpo propio y experimentan directamente con el mundo a través de este cuerpo, pues sus acciones son parte de este mundo y reciben un *feedback* inmediato de ellas, lo que se traduce en que el robot tenga sensaciones para así poder reaccionar a este y moverse.

Sin embargo, habría que añadir un tipo de robots más a esta clasificación, que son de los que habla Mataric en [718]. Estos serían los robots de **cuerpo operado por control remoto**. Estos permiten ser controlados para poder moverlos por el mundo físico como uno necesite o quiera.

No situados (*Unsituated*): estos robots están situados en un sitio determinado y son inamovibles. Estos son los robots de cadenas de montaje que operan en entornos altamente estructurados y fuertemente predecibles.

12.3.2 **DESDE EL PUNTO DE VISTA DE SU CONTROL**

Una clasificación sobre el control de los robots es la de Mataric [718]. Esta clasificación se basa en como los robots consiguen información del entorno en el que se encuentran, a través de sensores, procesan dicha información, toman una decisión, y actúan al respecto. Es decir, esta clasificación se basa en que no todos los robots piensan, pues algunos disponen ya de reacciones preprogramadas para ejecutarse, mientras que otros piensan mucho para actuar un poco. Aquí, la complejidad del ambiente está directamente relacionada con la complejidad del control, pues si el robot necesita actuar rápido en un ambiente muy dinámico, esto será una tarea muy complicada en comparación a si el robot no tiene que responder rápidamente y se encuentra además en un ambiente no tan dinámico y cambiante. Hay cuatro clases, cada una pensada para un tipo de trabajo concreto, pues todas tienen sus pros y sus contras:

Control reactivo (*No pienses, reacciona*): los robots pueden reaccionar a los cambios de una forma rápida al poseer reglas predefinidas para las diferentes posibles entradas y que se ejecutan instantáneamente. En este enfoque las entradas sensoriales y las salidas de los actuadores están fuertemente acopladas y así poder reaccionar de una manera rápida [751]. Sin embargo, este enfoque requiere de información ambiental para tener un control efectivo y los robots que lo utilizan no suelen mantener guardada mucha información o aprender en el tiempo. Por eso, el control reactivo es bueno para ciertas tareas, pero cuando se necesita que el robot aprenda y tenga memoria este tipo de control es insuficiente.

Control de deliberación (*Piensa y después actúa*): aquí las acciones son tomadas a partir del razonamiento de la lectura de todas las entradas sensoriales y del conocimiento almacenado internamente del mundo externo, sobre la que se basa el cálculo del resultado de salida. Esto permite al robot planificar los pasos de su decisión y saber cómo podría afectarle esto en el futuro mediante la predicción de posibles resultados. La parte buena de este proceso es que permite al robot actuar estratégicamente seleccionando la mejor decisión para cada situación. Sin embargo, esta planificación necesita de una gran computación interna para lograr la Inteligencia Artificial necesaria para computar todo este proceso. Además, si se encuentra en un ambiente muy dinámico y «ruidoso» puede ser imposible que el robot tome decisiones. Por estos motivos muy pocos robots son puramente deliberativos.

Control híbrido (*Piensa y actúa independientemente en paralelo*): este control es la fusión del control reactivo y del control de deliberación, cogiendo lo mejor de ambos. El control híbrido permite que los objetivos a largo plazo se rijan por un elemento de deliberación para tomar decisiones eficaces, mientras que las acciones inmediatas y en las que es necesario que sean en tiempo real son controladas por un elemento reactivo. Aquí la dificultad de encuentra en que se utiliza el control reactivo para esquivar obstáculos y tomar toras decisiones que deben de ser inmediatas, mientras que el componente de deliberación toma decisiones más a largo plazo. Por este motivo es necesario un sistema que coordine ambas partes para que tengan un beneficio mutuo, siendo así que el sistema reactivo anule el de deliberación cuando necesite tomar una decisión inmediata, pero el sistema deliberativo debe informa al reactivo de poder guiar al robot hacia trayectorias más eficientes, siendo este componente intermedio el más difícil de diseñar en este tipo de controles.

Control basado en el comportamiento (*Piensa en la manera de actuar*): este tipo de control está basado en la biología, exactamente se basa en el comportamiento de los animales con sus entornos. En este tipo de control se guardan los patrones, conocidos como comportamientos, de las actividades que surgen cuando el robot interactúa con el entorno. Estos sistemas poseen una construcción que empieza con los comportamientos más simples y sigue hacia los comportamientos más complejos. Por ejemplo, se comienza con comportamientos de supervivencia como el evitar colisiones, es decir, comportamientos que van mapeados directamente a unas determinadas entradas sensoriales. Posteriormente, se mejoran estos comportamientos añadiendo otros mucho más complejos como son seguir una pared, perseguir un objeto en movimiento, explorar o buscar el retorno a casa. Este tipo de control puede almacenar las representaciones, lo que hace que sea mucho más poderoso. Además, este tipo de controlas utilizan un sistema distribuido para almacenar los comportamientos, lo que permite que, si un robot necesita crear una planificación de futuro, el mismo robot

pueda acceder al sistema distribuido para obtener una mejor información acerca de cómo actuar. Sin embargo, este tipo de sistemas de control son los más complicados de desarrollar.

12.3.3 **DESDE EL PUNTO DE VISTA DE LOS GRUPOS DE ROBOTS**

Por otro lado, están los **grupos de robots**. Estos son robots que trabajan o interactúan con otros para poder tener más inteligencia, poseer detecciones y acciones distribuidas, tener tareas basadas en la reconfiguración, un diseño más simple de los robots y un sistema de redundancia [721]. De este tipo de robots hay dos variantes.

La primera variante es la del **enfoque tradicional**, la cual depende de la habilidad de los robots para crear y gestionar modelos sobre las diferentes tareas y que necesitan una comunicación explícita con los otros robots o con agentes externos. La segunda variante es la basada en la **inteligencia de enjambre** [205], conocida como robótica de enjambre, en donde los robots se comunican entre ellos y con el entorno en donde están desplegados, pues este entorno es el que los gestiona y administra sus tareas. Esta inteligencia puede ser llevada a cabo de varias maneras.

12.3.3.1 DESDE EL PUNTO DE VISTA DEL TIPO DE COOPERACIÓN EN LOS GRUPOS DE ROBOTS

La cooperación es esencial para los **grupos de robots**, los cuales pueden cooperar de dos maneras diferentes de acuerdo con [752], [753]:

La primera es la **cooperación emergente** o **cooperación tipo enjambre**. En este caso, por un lado, se encuentran los comportamientos obtenidos del enjambre de robots con el medioambiente, mientras que por otro lado se encuentran los comportamientos individuales de los robots. Esta es la metodología de la naturaleza y la que se aplica a los enjambres de robots. Sin embargo, este tipo de cooperación se suele aplicar en grupos de robots homogéneos para así obtener un mejor rendimiento y se tiende a utilizar en tareas no críticas, como puede ser el limpiado a fondo de una zona o recolectar información de un sitio, es decir, a tareas repetitivas numerosas veces. El que sea robots homogéneos implica que tengan todos las mismas capacidades y el mismo algoritmo de control.

Un ejemplo del inicio de este concepto es el que se propuso en el año 1988 en [754]. Los CEBOT eran robots con una pequeña misión diferente cada uno, como deslizar, inclinar o rotar. Sin embargo, estos se podían juntar entre ellos para hacer tareas más complejas, incluso podían sustituir automáticamente a un CEBOT que fuese el más débil de una serie de CEBOT enlazados con el fin de ayudar a todo el conjunto a realizar la tarea. Al ser pequeños robots que pueden juntarse para crecer y realizar tareas, su aplicación estaba pensada para sitios pequeños o de difícil acceso, como puede ser un tanque.

La segunda es la **cooperación intencional**, la cual se basa en la comunicación entre los robots para lograr un objetivo común entre todos. Este tipo de cooperación es la más similar al mundo humano, pues se juntan todos los robots para realizar una tarea común, permitiendo sustituir a los seres humanos en la realización de tareas peligrosos o bien en tareas repetitivas en fábricas que requieran de un trabajo monótono y fatigoso.

[205] Glosario: **Inteligencia de enjambre (Swarm-Intelligence)**

Ejemplos de este tipo de cooperaciones posibles por los robots son el limpiar líquidos tóxicos, apagar incendios, desmantelar una planta nuclear y el reconocimiento de lugares peligrosos, entre muchos otros. Sin embargo, este tipo de cooperación trabaja con robots heterogéneos, lo que implica que no todas las tareas puedan ser realizadas por todos los robots, lo que complica la cooperación, pues hace falta saber cómo formular, descomponer y distribuir los problemas e intercomunicar los diferentes robots para que los resuelvan en conjunto y sin conflictos.

12.3.3.2 DESDE EL PUNTO DE VISTA DEL TIPO DE RED Y SUS HERRAMIENTAS EN LOS GRUPOS DE ROBOTS

Los **grupos de robots** se pueden comunicar por diferentes vías, todo depende del tipo de herramientas necesiten, y de las tecnologías de las que dispongan.

Una de ellas es mediante la conexión de los robots a una red, pues, a finales de los años 1990 fue cuando esta tecnología se desarrolló lo suficiente y empezaron a aparecer los primeros robots que la utilizaban, lo cual se conoció como bajo el nombre de **Robótica en Red** [755]. «Un **Robot en Red** es un dispositivo robótico conectado a una red de comunicaciones como puede ser por Internet o por una red de área local (LAN). La red podría ser por cable o inalámbrica, y basada en cualquier tipo de protocolo como TCP, UDP o 802.11» [756]. Hay dos tipos de robots en red. Los primeros son los teleoperados, que son los controlados por seres humanos a través de la red. Los segundos son los autónomos, los cuales son los robots que envían sus datos y los de sus sensores por la red, permitiendo así conectarse con otros robots para coordinar sus actividades [756]. Sin embargo, la Robótica en Red tiene varios retos a superar como es el ruido, la fiabilidad y congestión de la red, los tiempos variables de las respuestas, la estabilidad de la red, la cobertura, la seguridad, la localización, y el diseño de interfaz de usuario, entre otras [756]. Más retos que se necesitan solventar pueden ser vistos en [727]

Posteriormente y en conjunto con los últimos avances en *Cloud Computing* y *Big Data* han hecho posible llevar esta computación a la nube para permitirles el acceso a un conjunto de datos global y dinámico [451], [726], [755], todo gracias a la confluencia con la tecnología inalámbrica [724], y el uso de diferentes tecnologías de la nube, permitiendo reducir costes y mejorar su conducta en tiempo real y su eficiencia energética [755]. Gracias a esto, bajo el uso de estas nuevas tecnologías se podría ofrecer una mejor inteligencia y un servicio más eficiente y barato que el ofrecido por la **Robótica en Red**, el cual puede ser también considerado el siguiente paso o la evolución [727]. Estos robots que utilizan la nube son conocidos como **Cloud Robot**, término acuñado en el año 2010 por Jeff Kuffner [757], por ser capaces de utilizar la nube para realizar cómputo o almacenar parte de, o toda, su memoria [451], haciendo que «ningún robot sea una isla» [726]. Dicho de otra forma, la idea es que tengan el «cerebro» en un sitio remoto y compartido, como bien propuso en el año 1990 Masayuki Inaba [726]. Esto permite que, si un robot no ha visto un objeto nunca, pueda sacar una foto, enviarla a la nube y la nube compare con la inteligencia ya adquirida de otros robots en busca de qué es ese objeto para decírselo al robot [726]. Otro ejemplo de este tipo de robots son los coches de Google, pues indexan mapas he imágenes capturadas por satélite para poder moverse [451]. Sin embargo, **Cloud Robotics** tiene el problema de la latencia de los datos, el tratar grandes cantidades de datos y de la seguridad de los datos [755].

12.4 *LAS TRES REGLAS DE LA ROBÓTICA*

Isaac Asimov, escritor ruso-estadounidense, escribió varios libros de ciencia ficción en los que trataba el tema de la robótica. En estos libros introdujo conceptos como la roboética, el complejo de Frankenstein [206] y las leyes de la robótica.

Estas leyes fueron introducidas en el primer libro que trató sobre el tema de robots y que data del año 1942 [758], y han sido tomadas posteriormente como un compendio imprescindible cuando se trabaja con robots. Estas leyes son tres [488], aunque posteriormente en el libro «Robots e Imperio» [759] los robots razonaron la ley cero, siendo esta el corolario de la primera ley:

0. Un robot no hará daño a la Humanidad o, por inacción, permitir que la Humanidad sufra daño.
1. Un robot no puede hacer daño a un ser humano o, por su inacción, permitir que un ser humano sufra daño.
2. Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entran en conflicto con la Primera Ley.
3. Un robot debe proteger su propia existencia, siempre y cuando esta protección no entre en conflicto con la Primera o la Segunda Ley.

Así, en base a estas normas, Asimov intentó que los robots formen parte de nuestra vida y nos ayuden, pero sin poder ser utilizados para fines destructivos contra la humanidad ni contra ellos mismos, sirviendo así estas leyes de protección para los seres humanos y contrarrestando el complejo de Frankenstein.

El porqué de seguir este orden para las leyes de la robótica ha sido expresado muy bien en la tira cómica XKCD y que se muestra en la Ilustración 67.

[206] Glosario: **Complejo de Frankenstein**

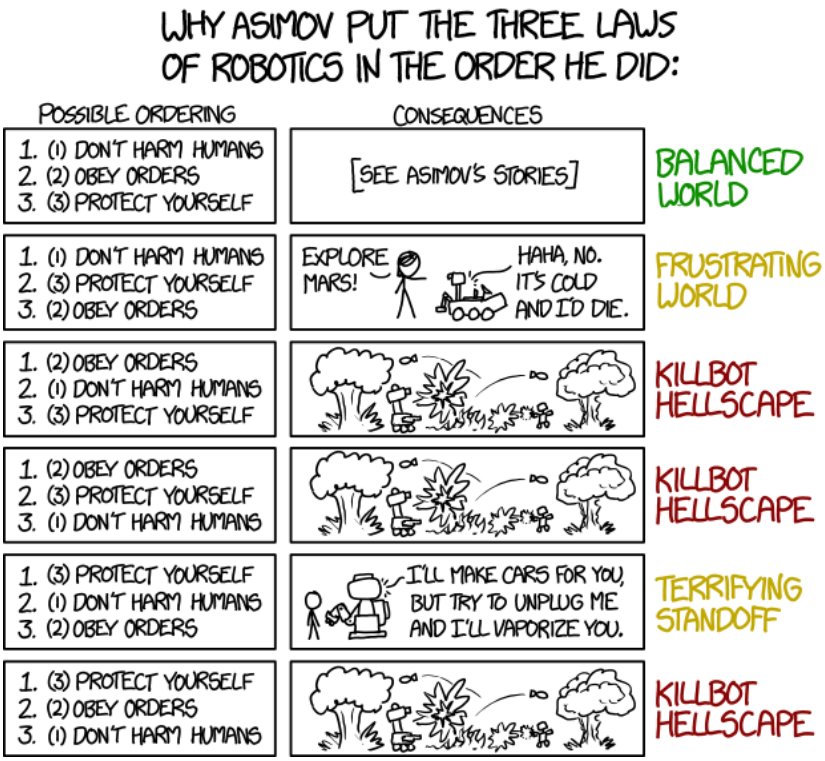


Ilustración 67 Diferentes resultados en caso de cambiar el orden de las tres leyes de la robótica de Isaac Asimov según https://xkcd.com/1613/

Como bien se ve, cualquier otro orden de estas leyes daría problemas, ya sea para que lo robots no obedeciesen por poder sufrir problemas o se revelasen por ello, o bien pudieran destruir a la humanidad al seguir órdenes o protegerse a sí mismos.

12.5 *TRABAJO RELACIONADO*

Para ayudar al crecimiento de los Cloud Robots, entre diciembre de 2010 y enero de 2014 se desarrolló el proyecto RoboEarth [720], [760], subvencionado por la Unión Europea. Los objetivos de RoboEarth han sido que los robots puedan almacenar y compartir información para mejorar la velocidad de aprendizaje, permitir a los desarrolladores crear acciones generales en vez de individuales, ofrecer una mejorar computación para procesados pesados en la nube para así que cualquier robot pudiera ejecutarla, y que los robots puedan colaborar. Así, han demostrado que es factible crear una World Wide Web (WWW) para los robots.

Gracias a IoT y al uso de etiquetas RFID, así como de otros múltiples sensores, se podría incorporar robots y diferentes objetos físicos para realizar aplicaciones domésticas en el ámbito de las Smart Homes [451], por ejemplo, utilizando redes bayesianas para la detección del tipo de actividad que está realizando una persona en base a lo que detecten los sensores como demostraron en [761]. No obstante, como bien analizaron, este sistema necesita mejorar y tener cuidado con que sensores se utilizan y para que, pues como les ocurrió, no fueron capaces de detectar bien el uso del teléfono debido a que la gente utilizaba el móvil y no el teléfono fijo.

Para realizar la comunicación de **grupos de robots**, exactamente de **cooperación intencional**, hay muchas y diversas implementaciones. Por ejemplo, está la arquitectura Alliance [753], que facilita la tolerancia a fallos en los grupos de robots mediante el uso de mecanismos que monitorizan los datos de los sensores de los robots para adaptar sus acciones respecto a las de sus compañeros y el ambiente que le rodea. Por otro lado, Murdoch [752] implementó un sistema de mensajes anónimos mediante un nodo central de manera que era muy reactivo a los cambios del ambiente, incluyendo a fallos de los robots, permitiendo sustituirlos por otros nuevos.

Otro ejemplo del uso de robots, exactamente robots móviles de rescate, fue el caso de la central nuclear de Fukushima Daiichi [762]. En este caso, el 11 de marzo de 2011 un terremoto de magnitud 9 en la escala de Richter dañó tres edificios que albergaban reactores nucleares debido a explosiones de hidrógeno y en uno de los edificios la reacción nuclear se salió fuera de control. Debido a los riesgos de enviar seres humanos a inspeccionar edificios con material radioactivo, decidieron enviar robots de móviles de rescate en busca de supervivientes. Estos robots tuvieron que ser mejorados con diferentes sensores, una mejor fiabilidad hardware, una mejora en sus funciones de comunicación, y resistencia a la radiación de sus componentes electrónicos, de forma que pudieran ser capaces de responder correctamente a la situación requerida. Esto permitió que los robots realizasen un gran número importante de misiones dentro de los edificios.

Otros casos similares de robots móviles se da en [763], donde utilizaron un robot autónomo para recoger la temperatura, la humedad y los gases LPG en el aire, ofreciendo así un método seguro para los seres humanos de tomar medidas en lugares minados, viejos o radiactivos. Este robot se comunicaba mediante radio frecuencia y se movía autónomamente, o bien podía ser controlado remotamente.

Por otro lado, a veces los robots industriales no se adaptan bien a todas las industrias. Este fue el caso de dos explotaciones marítima, una de petróleo y otra de gas [764]. En este caso, ellos podían mejorar mucho la producción, pero sus procesos no eran los procesos típicos estándar de las industrias, pues necesitan un mínimo de intervención humana en sus procesos. Para conseguir aplicar robótica, tuvieron que realizar un estudio previo de ambas plataformas con el fin de ver cómo y donde podían aplicarla. De 8 tareas, 3 eran imposibles de robotizar con la tecnología actual o cercana en el futuro, pero serían muy buenas robotizarlas si se investigase en el cómo, 2 deberían de ser adaptadas con tecnología específica para la tarea, 2 se podían realizar con la tecnología estándar actual y con cambios específicos y 1 era inmediata con la tecnología disponible. Finalmente, desarrollaron un robot de inspección llamado MIMROex capaz de monitorizar medidores, realizar inspecciones visuales de las válvulas de accionamiento, realizar inspecciones acústicas, comprobar fugas, y realizar mantenimiento de sensores de gas y fuego.

Ejemplos de diferentes robots son los utilizados en la industria eléctrica en Brasil. Allí, por ley, están obligados a invertir en investigación, siendo algunas de estas de robótica [765]. El primero que presentan es un robot que permite añadir y quitar de forma autónoma y teleoperada las esferas de advertencia para aviones que se cuelgan en los tendidos eléctricos [766]. El segundo robot que presentan son los robots que monitorizan los cables [765], además de permitir añadir y quitar las esferas como en el robot anterior. El tercer caso es el de un robot, TATUBOT [767], capaz de moverse por los conductos para inspeccionar los cables eléctricos de forma autónoma. El cuarto es un robot de bajo coste utilizado para monitorizar embalses y así poder detectar

problemas estructurales, acumulación de desechos y otros problemas sin necesidad de vaciar los túneles, algo que es peligroso y requiere bastante tiempo [765].

«La mayoría de las personas gastan más tiempo y energías en hablar de los problemas que en afrontarlos»
Henry Ford

«Enfrentarse, siempre enfrentarse, es el modo de resolver el problema. ¡Enfrentarse a él!»
Joseph Conrad

«Si al franquear una montaña en la dirección de una estrella, el viajero se deja absorber demasiado por los problemas de la escalada, se arriesga a olvidar cuál es la estrella que lo guía»
Antoine de Saint-Exupéry

Bloque III
¿Qué problemas tratamos de resolver?

Contenidos de bloque

13. INTERNET DE LAS COSAS

«Internet es la primera creación de la humanidad que la propia humanidad no entiende, el más grande experimento de anarquía que jamás hemos conocido»
Eric Schmidt

«Obtener información de Internet es como intentar beber agua de una boca de incendios»
Mitchell Kapor

Internet de las Cosas es la interconexión de objetos heterogéneos y ubicuos entre si, cuyo objetivo final, junto a los *Smart Objects*, es el de poder crear una red inteligente que controle casi cualquier cosa. No obstante, existen varios problemas que resolver antes de que esto se convierta en una realidad diaria y asequible, es decir, antes de llegar al futuro. La definición de **IoT** nos muestra su primer problema: la **heterogeneidad** de los objetos. Esta **heterogeneidad** dificulta la comunicación entre los objetos.

La **heterogeneidad** de los objetos se puede dividir en dos subproblemas a resolver. El primero, **la interconexión de los objetos** heterogéneos entre ellos. El segundo problema es **el desarrollo de las aplicaciones** para poder «manejar» los diferentes dispositivos heterogéneos.

Este capítulo abarca ambos problemas.

13.1 *INTERCONEXIÓN DE OBJETOS HETEROGÉNEOS*

Los principales problemas en **Internet de las Cosas** y en las redes de sensores son muy similares debido a la naturaleza y finalidad que ambos comparten. Ambos se componen de objetos que deben conectar, en el segundo caso, solo sensores. En el caso de **IoT**, esta trata de conectar sensores, actuadores y ***Smart Objects***, los cuales buscan la misma finalidad que **IoT**, la de facilitar la vida diaria de sus usuarios.

Una de las soluciones actualmente más populares, para conectar objetos es la realización de una interfaz en común. Esta puede ser una aplicación que resida en ambos y consulte un servidor. Como ejemplo a esto son los programas multiplataforma existentes, desde servicios de mensajería como son WhatsApp, Skype y Line, hasta los videojuegos multijugador, por ejemplo, Triviados y Apalabrados. No obstante, estos solo consiguen conectar los *Smart Object* para los que realizan su aplicación. Para esto necesitan de desarrolladores para realizarla. En cambio, el poder de IoT se encuentra realmente en el soporte para objetos heterogéneos, sin importar el sistema operativo, plataforma o las funcionalidades de las que dispongan, si son objetos inteligentes o no, o si son objetos domésticos o sensores dispersos por el mundo. Por esto, los principales problemas de IoT son la **heterogeneidad**, el **dinamismo** y la **evolución** de su contenido [25].

El problema en este punto reside en todas las diferentes implementaciones que hay que realizar para los diferentes tipos de objetos existentes, ya sea por su sistema operativo o por cómo trabajar y leen los datos del entorno o por cómo funcionan. Además, estos también pueden diferir en el tipo de mensaje que se envía y/o el protocolo utilizado. Así, los mensajes enviados entre unos objetos creados por una empresa pueden diferir en gran medida de los objetos de otra o bien ser enviados por un protocolo distinto o en ambos casos, utilizar sistemas propietarios. Este problema se da sobre todo en el caso de los ***Smart Objects***.

Esto provoca que los objetos de diferentes empresas no puedan interactuar entre ellos debido a la falta de entendimiento [25]. Por ello, una posible solución a estos problemas sería el uso de una red que consiguiera interconectar los diferentes dispositivos [10]. Esta podría tratar los diferentes datos recibidos y contrastar todo bajo un mismo formato, además de dotar de la inteligencia de entendimiento necesaria a cada dispositivo a través de la red.

Estos problemas surgen de la intención de manejar diferentes dispositivos, protocolos, sensores, objetos y aplicaciones. Cada uno realizado de una forma diferente, por diferentes fabricantes y con muy poco en común con el resto. Esto implica que la comunicación directa entre ellos, en la mayoría de los casos, no sea posible debido a la falta de entendimiento por la inexistencia de interfaces y un estándar o protocolo común [27]. Por ello, hay que proporcionar una arquitectura adecuada que de soporte a una comunicación fácil para cualquier tipo de dispositivo. Un ejemplo de heterogeneidad con un ***Smart Object*** puede ser un micro-controlador Arduino [7], [53]. En este caso, la **heterogeneidad** de los sensores es mayor debido a la gran cantidad disponible de sensores para ser conectados al microcontrolador, pues cualquier dispositivo electrónico de cualquier marca puede ser conectado a este. Esto hace que cualquier Arduino pueda ser diferente a otros debido a los sensores que usa, pues dependerá de los datos que este sensor utilizado recoja.

Otro ejemplo, pero a gran escala, de los problemas de heterogeneidad en **Internet de las Cosas** son las ***Smart Cities***. Como se comenta en [3], el primer problema a resolver es la recolección de datos, el acceso y la transmisión. Esto se debe a que **Internet de las Cosas** necesita reconocer los objetos inteligentes y mantener un flujo de mensajes constante entre los diferentes objetos. No obstante, cada implementación de diferentes redes de sensores puede presentar diferentes problemas y cada investigador opta por una solución diferente [60]. Esto da lugar a múltiples soluciones y ninguna estándar [27], pues cada objeto es diferente según su fabricante, sirve para diferente propósito, contiene diferentes componentes físicos, y usa diferentes interfaces estándares, protocolos o tecnologías [25].

En los últimos años, las grandes compañías están desarrollando sistemas IoT. El problema es que los objetos de estas compañías no suelen ser compatibles con los de otras. Por ejemplo, LG, Samsung, Telefónica y otras compañías ofrecen packs para crear una red IoT en casa. Estas soluciones permiten conectar diferentes objetos entre ellos. Sin embargo, estos objetos deben de ser todos de la misma compañía. Es decir, si tienes objetos de Telefónica, necesitas utilizar sus propios dispositivos y comprar los sensores de este producto, pues si intentas utiliza run sensor de LG o de Samsung estos no se interconectarán. En el mejor de los casos, se puede desarrollar el software necesario para conectar un sistema hardware abierto, como es un Arduino o una Raspberry Pi a estos dispositivos. Luego, esto rompe la **heterogeneidad** buscada por IoT, pues necesitas objetos de la misma marca debido a que los objetos de hardware propietario utilizan su propio protocolo, o su propio sistema de mensajes no abierto obligando a los usuarios a utilizar el ecosistema de su empresa.

Por estas razones, las inevitables diferencias dan como resultado esta **heterogeneidad** impidiendo que los objetos se puedan conectar directamente [14], [25]. No obstante, los estándares globales podrían facilitar este problema, pero el desarrollo de estos es uno de los mayores retos de IoT [768].

13.2 *DESARROLLO DE APLICACIONES PARA INTERCONECTAR OBJETOS POR PERSONAS INEXPERTAS*

El otro problema de la **heterogeneidad** de los dispositivos viene dado en el software que tienen que tener estos dispositivos. Para desarrollar este software se necesita un grupo de desarrolladores, pero en el caso de IoT, donde lo que se busca es interconectar todos los objetos del mundo, también hace falta que colabore toda la gente del mundo, que son gente, en muchos casos, sin conocimientos de programación, pero con deseo de conectar dichos objetos para poder beneficiarse de la tecnología y ayudar a crear un entorno mejor.

El desarrollo de aplicaciones requiere de personas con experiencia en el desarrollo de software y que tengan un amplio conocimiento del dominio. En este caso, el dominio son los sensores y los objetos a conectar. Además, se requeriría también conocimientos de los diferentes lenguajes de programación que se necesitan utilizar en las diferentes plataformas para así desarrollar el software necesario. Lo deseable en estos casos, es que personas sin conocimientos en el desarrollo software puedan hacer aplicaciones de una manera rápida y fácil, pues el desarrollo de aplicaciones para los diferentes dispositivos requiere de tiempo [27].

De esta manera, se podría conseguir crear aplicaciones que pudiesen conectar nuestros objetos y ayudarnos a conseguir hacer que interactuasen los objetos en base a datos de otros objetos para así mejorar nuestra vida diaria, ya sea en casa, en el trabajo o por el mundo, mediante la automatización de acciones.

14. DESARROLLO DE SOFTWARE PARA INTERNET DE LAS COSAS

«El mundo es movido por software [El software mueve el mundo]»
Juan Manuel Cueva Lovelle

«No existe ninguna bala de plata»
Fred Brooks [207]

«Considerando el lamentable estado de nuestras aplicaciones informáticas actuales, el desarrollo de software es todavía un arte oscuro, y no puede ser aún considerado una ingeniería»
Bill Clinton

Desde los años 60 se vienen dando unos problemas en el desarrollo del software conocidos como la **Crisis del Software**. Sin embargo, surgieron nuevos lenguajes de programación y metodologías que ayudaron a paliar estos problemas. Entre ellos, surgió la **Ingeniería Dirigida por Modelos**, que ayudó a disminuir estos problemas, así como a ayudar a crear **Lenguajes de Dominio Específico**. No obstante, no siempre es adecuado su uso, pues no todos los proyectos son adaptables al uso de MDE, ni en todos es rentable crear un DSL.

En el caso que nos ocupa, se requiere la generación de aplicaciones, pues IoT está compuesta por muchos objetos y, ante todo, por toda la gente del mundo. Por ese motivo, se hace indispensable automatizar este proceso software para así permitir a los usuarios expertos del dominio crear sus propias aplicaciones.

Este capítulo tratará los problemas que surgen en esta tesis para identificar si es abordable utilizar MDE.

14.1 *DESARROLLO TRADICIONAL O INGENIERÍA DIRIGIDA POR MODELOS*

El desarrollo tradicional de software ha venido siempre precedido de diferentes problemas que ya fueron nombrados y llamados por la OTAN en el año 1968 como **Crisis del Software** [22], y posteriormente descritos por Edsger W. Dijkstra en el año 1972 en [21]. No obstante, a pesar de llevar existiendo desde el año 1968, estos problemas se siguen dando actualmente [208].

Estos problemas están relacionados con el desarrollo de software. El primero de ellos tiene que ver con la baja calidad del software desarrollado que contiene bastantes fallos. Otro problema es el incumplimiento de las especificaciones y funcionalidades que hacen que este software no se adapte a lo pedido por los clientes. A esto hay que sumarle el incumplimiento de la planificación, que implica el retraso de las diferentes versiones y muchas veces que el proyecto se alargue más de lo previsto, incluso a veces haciendo que este no se termine. Los anteriores problemas incluyen el sobrecoste del proyecto debido a la aparición de estos problemas y a que muchas veces se presupuesta mal. Por último, el costoso mantenimiento de los proyectos que viene dado por el desarrollo de software de mala calidad que hace que el mantenimiento sea difícil y lento.

Según [203], estos problemas pueden ser debidos a diferentes razones. Entre estas, una posible razón es que no se tiene en cuenta otros desarrollos similares que podrían haber sido utilizados como base del nuevo proyecto para así facilitar el arranque o previsualizar posibles vías de desarrollo o posibles problemas a tener en cuenta. Además, muchas veces, cuando se desarrolla, se tiende a crear un software monolítico, el cual posee muchos componentes muy interconectados, lo que impide la creación de una buena arquitectura mantenible y ampliable. En otros casos, el problema tiene que ver con la elección del GPL, pues, si los desarrolladores optan por un GPL de bajo nivel ganan en flexibilidad, pero tienen que preocuparse de más detalles, como son la memoria, lo que disminuye drásticamente la productividad, además de ser más susceptibles a tener fallos de programación. A todo esto, hay que sumarle la juventud del desarrollo software que no está tan madura como ocurre en otras disciplinas, como la arquitectura. Además, la creciente demanda de software hace que en ocasiones se intente desarrollar software demasiado rápido, creando así una falta en la calidad del desarrollo de este.

14.2 *GENERACIÓN AUTOMÁTICA DE ARTEFACTOS*

Cuando se utiliza la **Ingeniería Dirigida por Modelos**, utilizando una u otra aproximación, finalmente siempre hay que generar artefactos a partir de los modelos software [208], [769]. Según [770], existen razones que justifican la necesidad de generar artefactos cuando se desea conseguir un mayor nivel al proporcionado por los lenguajes de programación y así permitir programar a los expertos del dominio. Gracias a esto, los usuarios podrían crear sus propias aplicaciones de forma automática, sin necesidad de tener conocimientos de programación y sin necesidad de ayuda de soporte técnico, pues solamente necesitan conocer el dominio. El problema aquí reside en cómo desarrollar esta generación automática de artefactos.

15. SEGURIDAD EN INTERNET DE LAS COSAS

«R2D2, ¿te lo dijo la computadora central de la ciudad?
¡R2D2, sabes bien que no debes confiar en una computadora extraña!»
C3PO

«Las organizaciones gastan millones de dólares en firewalls y dispositivos de seguridad,
pero tiran el dinero porque ninguna de estas medidas cubre el eslabón más débil de la cadena de seguridad:
la gente que usa y administra los ordenadores»
Kevin Mitnick

Internet de las Cosas es la comunicación de innumerables objetos entre ellos. Para mantener esta comunicación, los objetos deben de enviar mensajes utilizando diferentes protocolos. No obstante, el problema surge cuando estos mensajes se envían por protocolos no seguros, pues esto da lugar a que cualquier persona pueda leerlos, lo que provoca una intrusión en la privacidad de la primera persona.

Cuando se habla de seguridad, no siempre es software, pues la seguridad física también es importante. En este caso se habla de vigilancia de zonas que pueden ser importantes por diferentes motivos y que necesitan de una mejora en el sistema para ser más eficientes, y tal vez, más seguras.

Este capítulo tratará ambos problemas.

15.1 *SEGURIDAD EN LOS MENSAJES ENTRE OBJETOS*

Debido a la idea de qué es IoT, esto hace que cualquier dispositivo pudiera recabar información sobre nuestra vida para tratar de mejorarla por medio de la automatización o semiautomatización de tareas. Por ello, hay que tener especial cuidado, pues debido a esta filosofía, los riesgos que podríamos correr con IoT son más peligrosos que los sufridos con Internet hasta la fecha [2], [771].

Entre los mensajes que se mueven en las redes IoT pueden viajar datos sensibles como pueden ser datos sobre nuestra salud o nuestras cuentas bancarias. Por ejemplo, con las conocidas pulseras inteligentes, también conocidas como *Smart Bands*, se pueden obtener datos acerca de nuestro sueño, como las horas en las que dormimos, o bien datos sobre lo que caminamos o acerca de nuestras pulsaciones, incluso si tenemos un smartphone podrían hacerse con nuestra localización.

Debido a estos datos sensibles, de acuerdo con [772], hay una visión tan futurista como inquietante, en la que todos nuestros movimientos, acciones y decisiones son grabadas continuamente por dispositivos electrónicos que pueden ser simples objetos de nuestra casa, como la nevera, la televisión o el despertador, hasta dispositivos que portemos siempre encima, como nuestros *smartphones*, relojes o coches. Como se ve, algunos de estos datos son muy sensibles debido a que llevan datos que nos pueden identificar, datos acerca de nuestra salud o nuestros datos bancarios. Por este motivo todos estos objetos se convierten en un riesgo para nuestra **seguridad** y **privacidad**.

Además, el hecho de que cada vez hay más dispositivos conectados a Internet, implica que se creen nuevos retos haciendo que el ataque a estos dispositivos sea algo común. Ejemplos de ellos son los ataques que permiten controlar la dirección y los frenos de los coches o los monitores de bebés que permitan que un atacante pueda gritar al niño a través del dispositivo [14].

Estas razones hacen indispensable crear un **Internet de las Cosas** seguro para así garantizar un sistema de transmisión que respete la privacidad de los usuarios y los objetos de manera que ningún usuario u objeto no autorizado pueda acceder a datos personales independientemente del protocolo o canal que se utilice para transmitirlos, sea este un canal seguro o no.

15.2 *MEJORAS EN LA SEGURIDAD FÍSICA*

Actualmente, la gente utiliza cámaras IP para crear áreas seguras en sus casas o en las ciudades. No obstante, estos sistemas obligan a tener a alguien mirando las cámaras continuamente, o bien, si tienen detectores especiales, como puede ser reconocimiento de movimiento, revisar las imágenes cada vez que tengan un aviso. Ambos casos son un inconveniente, ya que, dependen de la visualización continuamente. Si además añadimos a esto que la cantidad de *Smart Cities* y *Smart Homes* está creciendo, se necesitaría cada vez más gente que vigilase las cámaras.

En el otro lado tenemos a las *Smart Towns*, que necesitan proteger y preservar su cultura y herencia, como son sus edificios, monumentos, paisaje, folklore y tradición. Por ejemplo, las *Smart Towns* pueden tener

sitios compartidos, grabar parte de su cultura como bien puede ser mientras cocinan sus platos típicos, o bien pueden querer monitorizar ciertos sitios que necesiten condiciones especiales o protección, como los monumentos [136].

Referente al problema de las *Smart Towns*, se encuentra el quinto principio de la habitabilidad, el cual habla de proteger los paisajes rurales [106], [136].

Debido a todas estas razones, se hace indispensable encontrar y mejorar el sistema de monitoreo de seguridad incorporándolo a IoT y permitiendo su automatización o semiautomatización, así como su interconexión con otros objetos dentro de IoT.

«Lo bello del desierto es que en algún lugar esconde un pozo»
Antoine de Saint-Exupéry – El Principito

«Para todo problema humano hay siempre una solución fácil, clara, plausible y equivocada»
Henry-Louis Mencken

«No podemos resolver problemas pensando de la misma manera que cuando los creamos»
Albert Einstein

«Cuando el objetivo te parezca difícil, no cambies de objetivo; busca un nuevo camino para llegar a él»
Confucio

Bloque IV
Solución general

Contenidos de bloque

16. INTERNET DE LAS COSAS

«En Internet nadie sabe que eres un perro»
Peter Steiner, chiste en The New Yorker, julio de 1993

«No hay problemas, solo soluciones»
Anónimo

Soluciones hay muchas, lo difícil es dar con la más óptima. Por eso, en este capítulo se presentan las posibles soluciones a los problemas relacionados con IoT que fueron surgiendo durante el desarrollo de esta tesis doctoral.

En el caso del primer problema de Internet de las Cosas, que es el de la **interconexión de objetos heterogéneos**, surgieron dos posibles soluciones. Una de ellas es la de crear una plataforma hardware y software central, que sea el centro neurálgico de todo. La segunda solución posible se basa en utilizar las Redes Sociales existentes como centro de interconexión entre los diferentes objetos.

Respecto al problema del desarrollo de aplicaciones para **interconectar objetos por personas inexpertas** surgieron también dos posibles soluciones, bien crear un lenguaje de dominio específico o utilizar procesamiento de lenguaje natural sobre los requisitos de los usuarios.

16.1 *PLATAFORMA CENTRAL DE HARDWARE Y SOFTWARE*

Una posible solución al problema de la interconexión de objetos heterogéneos y ubicuos en **Internet de las Cosas** puede ser el crear una plataforma independiente. Así, una solución en este caso sería realizar una arquitectura que soporte el paso de mensajes de los diferentes dispositivos y sea capaz de enviarles una respuesta.

De esta manera, el crear una plataforma hardware y software central permitiría tener una **plataforma totalmente independiente** a la que pudiera acceder cualquier tipo de objeto, ya sean sensores, actuadores u objetos inteligentes.

Esta plataforma, al ser independiente, permitiría su extensión mediante la incorporación de diferentes módulos. Estos módulos podrían ser de muy diferentes tipos.

Uno de estos módulos podría permitir desarrollar la **interconexión de los diferentes objetos** de una manera fácil y sencilla para usuarios sin conocimientos de desarrollo. Estas interconexiones podrían definir cuándo y cómo interactúan los diferentes dispositivos entre ellos, así como que deben de hacer bajo ciertas circunstancias.

Si bien se podría crear la interconexión, también se podría **crear el propio software** que necesitan dichos objetos. Facilitando esta parte del desarrollo, se conseguiría que cualquier persona que conozca el dominio del problema pudiese crear un software personalizado para realizar exactamente lo que se desea.

Siguiendo estos pasos, se conseguiría que la plataforma central permitiese programar todo lo necesario a sus usuarios. Esto da juego a que se puedan incrementar y extender las funcionalidades necesarias y así resolver otros problemas de **IoT**. Entre estos se encuentra la **seguridad**, algo muy importante desde siempre, pero en boga en los últimos años, o bien el facilitar el crear **aplicaciones inteligentes** para que ayuden de una mejor manera a sus creadores.

Esta centralización permitiría tener una aplicación web que facilitase todas estas tareas, que en muchos casos son desconocidas o irrelevantes para los usuarios, de forma que siempre estuvieran trabajando con las últimas versiones de una forma transparente.

16.2 *INTERCONEXIÓN A TRAVÉS DE LAS REDES SOCIALES*

Otra posible solución al problema de la interconexión de objetos es el de utilizar un medio de uso común para realizarla, es decir, un protocolo, sistema o entorno al que las personas ya estén acostumbradas. Por ejemplo, mediante el uso de las **redes sociales** como centro neurálgico aportaría ciertas ventajas, como pueden ser Twitter [773] y Facebook [774].

La primera ventaja sería el uso del protocolo HTTP y el evitar utilizar aplicaciones específicas para conectar los diferentes dispositivos. De esta manera, la interconexión se haría mediante una aplicación, en este caso una red social, ya extendida y con usuarios registrados. De esta manera, se evitaría a los usuarios el utilizar un nuevo medio, como puede ser una nueva aplicación específica para interactuar con los objetos.

La segunda ventaja que ofrecen es que poseen una interfaz conocida por los usuarios, pues estos ya conocen como funciona la red social que usan y están familiarizados con ella debido a que es una interfaz que muchos de ellos utilizan varias veces al día, con lo que solo tendrían que aprender como interaccionar con los objetos a través de ella. Un posible método sería mediante la escritura de mensajes siguiendo una serie de reglas. Sobre este punto, científicos de Ericsson han observado que la gente puede lograr una mayor familiaridad con IoT si la interacción de esta y sus objetos se presenta de forma análoga a las interacciones ocurridas en las redes sociales como son Twitter y Facebook [422].

Así, mediante la interconexión de objetos a través de las **Redes Sociales**, los usuarios podrían enviar acciones a sus propios objetos, ver el estado de estos y automatizar tareas entre objetos a la vez que consultan la información de sus amigos, de los usuarios a los que siguen y las noticias.

16.3 *GENERADOR DE APLICACIONES PARA INTERNET DE LAS COSAS*

Acerca del segundo problema de IoT, el problema acerca del desarrollo de aplicaciones para interconectar objetos por personas sin conocimiento en el desarrollo de software, en este subcapítulo se presentan varias soluciones.

Como bien se explicó anteriormente, no todas las personas tienen por qué saber cómo se desarrolla software y por ello es necesario ofrecerles una abstracción simplificada. Uno de los métodos más comunes es la realización de un DSL. Así, se conseguiría acercarle el proceso de creación utilizando un lenguaje basado en su dominio y creando las aplicaciones definidas por el usuario en el modelo de una forma automática.

Por el contrario, podría darse el caso de que se desease omitir el aprendizaje de un DSL. Ante esta circunstancia, una posible solución sería reconocer de forma automática lo que ellos desean hacer, es decir, que ellos escriban que desean que la aplicación haga y mediante **Procesamiento de Lenguaje Natural** (**NLP**), obtener los datos más importantes y construir automáticamente la aplicación tal cual el usuario la describió. De esta manera, mediante **procesamiento de lenguaje natural** se podría permitir a los usuarios que definieran los casos de uso de la aplicación que desean.

17. INGENIERÍA DIRIGIDA POR MODELOS

«Los estándares son siempre obsoletos. Eso es lo que los hace estándares»
Alan Bennett

«Hay que reutilizar y no inventar»
Steve Ballmer

MDE es un enfoque que trata de disminuir los problemas existentes en el desarrollo tradicional de software basándose en el uso de modelos software. No obstante, existen diferentes implementaciones, cada una con sus pros y sus contras. Entre estas se encuentra el estándar actual de la industria, que es MDA del OMG y las tan usadas por la industria, que son las Factorías de Software de Microsoft. Además de esta elección, también se encuentra la de elegir entre ofrecer al usuario utilizar un GPL o un DSL.

En este capítulo se compararán las diferentes posibilidades comentadas para presentar una posible solución que satisfaga los criterios comentados y los Objetivos de esta tesis.

17.1 *MDE VERSUS DESARROLLO TRADICIONAL*

La **Ingeniería Dirigida por Modelos** ofrece una serie de diferencias respecto al del desarrollo de software tradicional [775], que son adquiridas gracias a las cualidades que tienen los **modelos** con los que se trabaja y que ofrecen **automatización**, **abstracción**, **comprensibilidad**, **precisión** y son **predictivos** [24], [166], [170].

Continuamente aparecen nuevas tecnologías o versiones nuevas de las tecnologías utilizadas. Esto, en el desarrollo tradicional puede ser un inconveniente debido a la migración software que esto puede acarrear debido si se desea mantener los sistemas actualizados, con el correspondiente gasto que esto conlleva. Mientras, con MDE, la **portabilidad** es mucho más fácil debido a que los modelos pueden ser independientes de las plataformas, viéndose así solo afectada la parte software que sí lo está, que suelen ser las herramientas que se utilizan para realizar las transformaciones.

A veces es necesario tener **interoperabilidad** entre diferentes modelos. En el desarrollo tradicional, en estos casos, a veces hay que modificar bastantes partes del programa. Utilizando MDE, los modelos quedan intactos y se crean lo que se conocen como *bridges* para así tender la comunicación entre ambos modelos.

Un problema existente en el desarrollo tradicional es el relacionado con la **documentación** y el **mantenimiento**, pues, en un estado ideal, esta deberá de estar siempre actualizada. Aquí, el problema reside en que una aplicación suele estar compuesta de varias capas, lo que implica que, cuando se modifica una de estas capas, haya que actualizar la documentación, lo que conlleva pérdida de tiempo en tareas repetitivas. Una gran facilidad de mantener esta actualizada fue cuando se consiguió que la documentación de código fuese generada como documentación del proyecto, no obstante, aún quedaba por resolver la parte de la documentación del diagrama objeto-relación. Gracias a MDE, esto estaría resuelto debido a que, como MDE se centra en los modelos, casi todo puede ser generado a partir de ellos, poseyendo estos toda la documentación importante de alto nivel.

Además, MDE permite aumentar la **productividad** de los proyectos en donde se utiliza, siempre que el proyecto se adapte al uso de MDE. Por ejemplo, en el caso de que se necesite generar artefactos automáticamente.

Ahora, recordemos varios de los objetivos de la tesis, que se encuentran en el capítulo 1.3. El primero de ellos era el de permitir el **diseño de objetos inteligentes** por usuarios sin conocimientos en el desarrollo de software. Por otro lado, el tercer objetivo era facilitar el diseño y la configuración de **objetos asistentes** a personas no desarrolladoras.

Estos objetivos requieren ofrecer abstracción a los usuarios, lo que se consigue utilizando **modelos**. Por otro lado, requieren también de cierta automatización para generar los artefactos requeridos en base al modelo definido por los usuarios. Por estos motivos, resulta mejor de cara a la tesis utilizar MDE frente al desarrollo tradicional.

17.2 *LENGUAJES DE PROPÓSITO GENERAL VERSUS DSLS*

Los **Lenguajes de Propósito General**, también conocidos como **GPL**s (*General Purpouse Languages*) son aquellos lenguajes de programación que pueden ser usados para varios propósitos como es crear aplicaciones, páginas webs, comunicar dispositivos y un largo etcétera. Es decir, para casi cualquier cosa, pues no están especializados en resolver un determinado problema, sino que se presentan para intentar resolver cualquiera, pues no están especializados en un dominio.

Por otro lado, los **Lenguajes de Dominio Específico**, conocidos como **DSL**s (*Domain-Specific Languages*), sirven para resolver un problema determinado de un dominio concreto. Esto permite reducir la complejidad de las plataformas actuales y expresar los conceptos del dominio de una forma más apropiada. Esto permite abstraer así el problema a una especificación del sistema que no está sujeta a la implementación del sistema, abstrayendo de esta manera la codificación a un determinado vocabulario del dominio bien definido y acotado. Es decir, se consigue que las personas solo necesiten ser conocedoras del dominio del problema gracias a la abstracción y el encapsulamiento ofrecido por los DSLs [30].

Así, por medio del uso de los DSLs se puede simplificar cierto tipo de tareas. Por ejemplo, HTML sirve para crear la estructura de las páginas web, mientras que CSS sirve para especificar el estilo de estas. Como se ve, ambos están definidos en una tarea específica y acotada en lugar de intentar resolver grandes problemas como hacen los GPLs, pues, por ejemplo, a diferencia de estos últimos, no pueden trabajar con ficheros, entre otras muchas funcionalidades. De esta manera, al usar un DSL se consigue tener una mejora en la **portabilidad**, **interoperabilidad**, **reusabilidad** y en la **reducción de errores**, lo que repercute en un incremento de la **productividad**. No obstante, se necesita crear, probar y aprender un nuevo «lenguaje», además de que estos son más lentos que los GPLs.

Por estos motivos, una posible solución podría ser utilizar un DSL para resolver aquellos problemas específicos y que necesiten una cierta repetición, como puede ser, la abstracción que necesita darse a los usuarios a los que va destinado el desarrollo. En el caso de esta tesis doctoral, esto se corresponde con los puntos 1 y 3 de losObjetivos.

17.3 *MDE VERSUS MDA*

MDE es un es un enfoque para el diseño y desarrollo de software basado en el uso de modelos software que explica cómo se han de hacer el desarrollo el software, pero sin establecer con qué herramientas ni que estándares hay que utilizar. Esto implica que se puedan crear tantas o tan pocas capas de abstracción y transformaciones como sean necesarias, aunque como bien se explicó anteriormente, en palabras de Bran Selic [170], el potencial de MDE esta cuando se realiza una sola transformación, debido a que se genera automáticamente el programa desde el modelo definido.

MDA, por el contrario, es una visión estándar de MDE creada por el OMG. Si se desea aplicar MDA, se deben de utilizar una serie de estándares y niveles de capas marcados por el OMG. De esta manera, MDA describe el cómo utilizar el MDE utilizando una serie de estándares para crear y usar los modelos y sus

relaciones. Esta implementación está presente en diferentes herramientas que la cumplen como son los proyectos de Eclipse pertenecientes al EMF. Utilizar MDA implica el tener un total de cuatro niveles diferentes y de realizar tres transformaciones, una por nivel, y el tener que utilizar una serie de estándares de forma obligada en el desarrollo, lo que hace que la productividad se vea afectada frente al uso de MDE.

Como se ve, MDE resulta, teóricamente, más productivo que MDA, la parte mala es que no se sigue ningún proceso estándar y los resultados tampoco serán estándares e interoperables con herramientas estándar, salvo que así se desarrollen. No obstante, MDE también ofrece la posibilidad de crear DSLs, cosa de la que MDA reniega al no ser estándares de la industria. Por ello, con MDE se pueden crear DSLs que abstraigan y faciliten la tarea a los usuarios.

Por estos motivos, en el desarrollo de esta tesis, se decidió utilizar MDE para cumplir los objetivos presentados en el capítulo 1.3.

18. SEGURIDAD

«Todos tenemos tres vidas, una vida pública, una vida privada y una vida secreta»
Gabriel García Márquez

«El único sistema seguro es aquél que está apagado en el interior de un bloque de hormigón protegido en una habitación sellada rodeada por guardias armados»
Gene Spafford

Anteriormente, en el capítulo 15, se mostraron dos problemas, de entre los muchos existentes, de la seguridad dentro del marco de Internet de las Cosas.

El primero de estos tiene que ver con el envío de los datos, pues muchas veces estos se envían en plano, permitiendo que un usuario malintencionado pueda interceptarlos y leerlos o modificarlos. Esto es algo que nunca se debería permitir ya que permite la vulneración de la privacidad de los usuarios.

En el segundo caso, se presentó una posible mejora para la seguridad física en el marco de Internet de las Cosas. No obstante, esta mejora se basa en la automatización. Una de las posibles soluciones implica que el sistema deba reconocer ciertos objetos y tomar decisiones en base a lo que reconozca por sí mismo, sin ayuda humana. Para esto, una posible solución es la explicada en este capítulo y que se basa en la Visión por Computador.

*****

18.1 *ENVÍO DE DATOS SEGUROS USANDO ENTORNOS INSEGUROS*

Una posible solución a este problema pasa por cifrar los datos para que así, aunque sean interceptados, nadie, salvo que sea el destinatario final, pueda leerlos mediante el uso de las herramientas y/o tecnologías correspondientes.

Por ello, se puede utilizar lo mismo que durante toda la historia de la humanidad funcionó y que ha dado buenos resultados en la era informática, la **criptografía**. Durante la historia, como bien se comentó en el capítulo 7, la **criptografía** ha dado muy buenos resultados y por ello podría ser un buen método para proteger toda la información enviada por los objetos.

No obstante, como se vio, hay diferentes métodos. Así pues, surge el problema de cuál de ellos adoptar y que nivel de seguridad será el adecuado adoptar debido a que Internet de las Cosas está compuesto por objetos heterogéneos. Además, estos objetos pueden utilizar diferentes protocolos para conectarse. Por ello, si se adopta una seguridad muy alta, puede que la comunicación sea muy lenta, debido a la pobre capacidad computacional de algunos objetos, y si se utiliza una seguridad muy baja, puede que sea casi tan inseguro como no usar nada. Por esto, la mejor solución sería tomar las mejores cualidades de cada tipo de algoritmo y fusionarlas, para así obtener confidencialidad, privacidad e integridad del mensaje

18.2 *DETECCIÓN DE COSAS MEDIANTE VISIÓN POR COMPUTADOR*

Una posible solución al problema de mejorar la seguridad física pasa por la incorporación de **Visión por Computador**. De esta manera, con el uso de los algoritmos adecuados, se podría hacer que automáticamente el módulo creado para tal efecto difiriera entre las fotos que contengan personas o no. Incluso, si se diera el caso, se podría hacer que detectase ciertos objetos en las fotos, como pudiesen ser armas, objetos que no estuvieran antes situados, como coches o paquetes, o bien gente que podría ser peligrosa o no tuviera que estar en un determinado sitio, como pirómanos o ladrones.

De esta manera, se podrían tratar las fotos como si fuesen sensores y así obtener como resultado si contienen o no la cosa que deseamos. Sin embargo, para conseguir esto, hace falta entrenar el modelo y comprobar que funcione correctamente al analizar las diferentes fotos mientras ofrece una buena precisión de detección.

El beneficio del uso de un módulo de Visión por Computador junto a Internet de las Cosas ofrecería la posibilidad de automatizar diferentes acciones y objetos sin necesidad de que un ser humano tenga que estar visualizando las imágenes o las cámaras de seguridad para detectar algo «no deseable».

Además, en lo que refiere al problema de las *Smart Towns*, este sistema ayudaría al quinto principio de la habitabilidad, el cual habla de proteger los paisajes rurales [106], [136], como bien se dijo en el ejemplo acerca de la detección de personas, o cosas indeseadas en determinados sitios, desde cosas que pueden ser peligrosas hasta cosas que afecten al ecosistema, como pueden ser los objetos abandonados.

«No se puede publicar un documento sobre física sin los datos completos y los resultados experimentales, y esto debería ser la norma en periodismo»
Julian Assange

«A veces, cuando innovas, cometes errores. Es mejor admitirlos rápidamente y seguir mejorando tus otras innovaciones»
Steve Jobs

«No basta saber, se debe también aplicar. No es suficiente querer, se debe también hacer»
Johann Wolfgang Goethe

Bloque V Prototipos desarrollados

Contenidos de bloque

19. PLATAFORMA HARDWARE Y SOFTWARE MIDGAR

«Comenzar bien no es poco,
pero tampoco es mucho»
Sócrates

«Tener grandes ideas es excelente, transformarlas en realidad una virtud»
Anónimo

«Un hombre que no arriesga nada por sus ideas, o no valen nada sus ideas, o no vale nada el hombre»
Platón

En los capítulos previos se presentaron ciertos problemas existentes a resolver con esta tesis doctoral, además de posibles soluciones para solventar estos problemas. No obstante, para llevar a cabo esta investigación, se requería de un sistema que gestionase los diferentes objetos.

Por este motivo, uno de los primeros pasos fue el desarrollo de una infraestructura que diese soporte a la comunicación entre los diferentes objetos. Bajo esta necesidad nació **Midgar**, una plataforma para Internet de las Cosas. Esta plataforma se encarga de la centralización de todos los datos de los diferentes sensores, actuadores y *Smart Objects*, permitiendo la interacción y comunicación entre ellos por medio de la creación de una red ubicua, resolviendo así el problema de la interconexión.

En este capítulo se presenta Midgar: qué es, qué ofrece, su arquitectura y su funcionamiento. Midgar es la base de esta tesis doctoral, pues el resto de investigaciones y la demostración de la hipótesis se hicieron basándose en Midgar, con diferentes implementaciones, diferentes DSLs e interfaces, y nuevas ampliaciones.

19.1 *DESCRIPCIÓN*

19.1.1 **OBJETIVOS**

El objetivo de este prototipo ha sido crear una plataforma web IoT que permita implementar diferentes prototipos de acuerdo a verificar la hipótesis de esta tesis doctoral, así como de otras investigaciones en el área de Internet de las Cosas, u otras áreas relacionadas. Para lograrlo, esta plataforma debía de ser capaz de gestionar objetos, tanto inteligentes como no inteligentes, además de ser modificable, algo que las plataformas abiertas de IoT no permitían.

Para lograr estos objetivos se tienen que cumplir diferentes puntos:

- Soportar un sistema de mensajes que permita la comunicación con los objetos.
- Tener inteligencia para la toma de decisiones
- Ser capaz de gestionar diferentes objetos
    - Registro
    - Recepción de datos
    - Envío de decisiones y de datos

19.1.2 **FUNCIONALIDADES**

Esta plataforma en la nube, llamada Midgar, soporta el registro de objetos, el envío de datos, la recepción de datos y la gestión de los datos de diferentes tipos de objetos, tanto *Smart Objects*, como de sensores y actuadores, permitiendo así su interconexión.

Para gestionar toda esta información, Midgar posee un sistema de mensajes basado en XML. Todo objeto que cumpla las normas de este sistema, puede tanto enviar como recibir información de Midgar.

Midgar posee también Inteligencia Artificial. Depende de la iteración en el tiempo, esta es más o menos compleja. En los primeros pasos Midgar posee una IA basada en árboles de decisión programados por los propios usuarios de Midgar. Así, en base a esta inteligencia, la plataforma era capaz de saber lo que debe realizar cada objeto en función de unos determinados parámetros de entrada, tomar la decisión predefinida, y mandársela al objeto destinatario. Más adelante, con los siguientes prototipos, esta inteligencia fue expandiéndose.

Todo objeto que cumpla el sistema de mensajes puede registrarse en Midgar. Una vez este objeto está registrado, puede enviar y recibir mensajes de continuo a Midgar. Estos mensajes pueden ser para enviar datos de los sensores y el estado de este objeto a Midgar, o bien para recibir mensajes de Midgar acerca de lo que debe de hacer. Es decir, Midgar es el cerebro de todo el ecosistema.

19.2 *IMPLEMENTACIÓN*

Midgar es una plataforma de Internet de las Cosas desarrollada específicamente de cara a la investigación de los problemas de estas plataformas y de la interconexión e interoperabilidad de objetos. En esta tesis se intenta dar una solución a la generación de aplicaciones que interconecten objetos heterogéneos. En este apartado se describirá la plataforma Midgar, siendo esta la primera parte y la base de todos los prototipos que han sido desarrollados.

19.2.1 ARQUITECTURA

La arquitectura del sistema se puede dividir en **cuatro capas** como se ilustra en la Ilustración 68. Cada una es un proceso dentro del conjunto global dentro de la infraestructura.

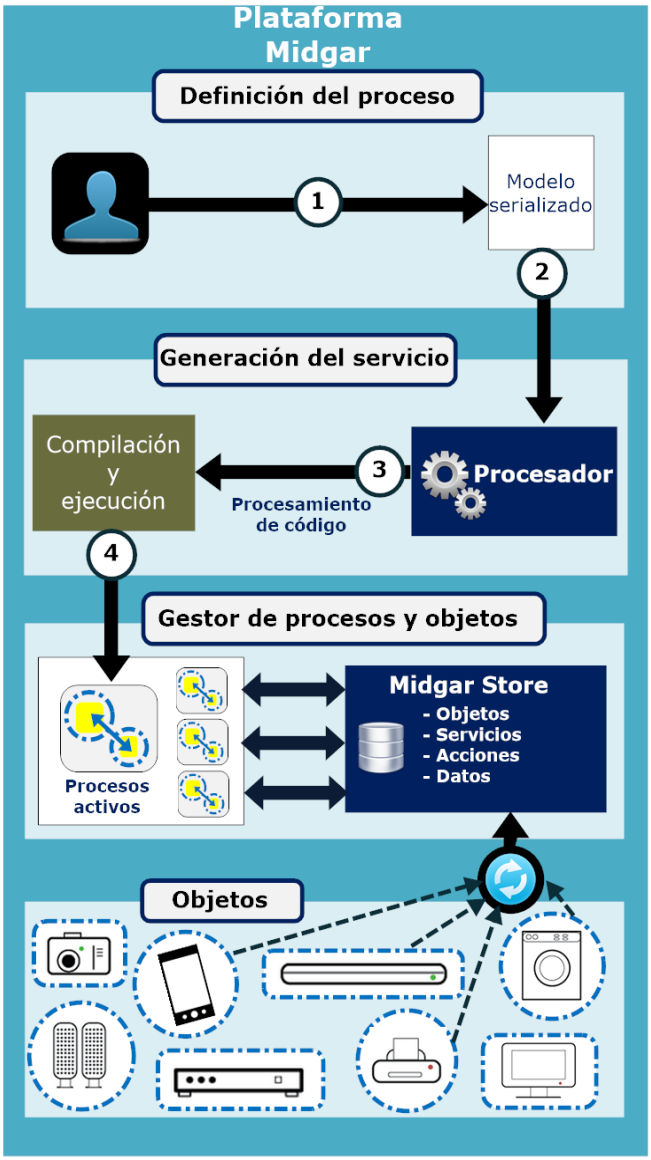


Ilustración 68 Arquitectura de la plataforma Midgar

19.2.1.1 DEFINICIÓN DE PROCESOS

La primera capa abarca la **definición de los procesos**. Esta capa es la única con la que interactúan los usuarios mediante el uso de diferentes utilidades que varían según la investigación, ya que pueden ser DSLs textuales o gráfico, es decir, es la capa de presentación, el *Front-end*. Esta capa se comunica con la segunda capa mediante el uso de XML, que contienen el modelo persistido que el usuario haya definido en la primera capa.

19.2.1.2 GENERACIÓN DEL SERVICIO

La capa de **generación del servicio** se encarga de procesar la información y realizar lo que deba de hacer en función de esta información. Esto puede ser generar una aplicación a partir de ella, compilarla y ejecutarla en el servidor o procesar la información recibida para introducirla adecuadamente en la base de datos. En este caso, puede haber varios procesadores existentes, uno por cada DSL diferente que provenga de la primera capa, aunque después se ejecuten todos bajo la misma aplicación. Estas aplicaciones en caso de ser demonios[207] o bien necesiten guardar datos, pasan a la tercera capa.

19.2.1.3 GESTOR DE PROCESOS Y OBJETOS

La tercera capa es la del **gestor de procesos y objetos**. Esta capa además contiene el servicio al que se deben de conectar los *Smart Objects* que interactúan con Midgar. Esta capa se encarga de gestionar todos los objetos, procesar su información, tomar las decisiones, mantener las aplicaciones demonio, mantener la aplicación web y gestionar la base de datos. Esta capa está formada por dos módulos que trabajan de forma conjunta: el **servidor de procesos** y el **almacén de datos**.

En esta capa también se encuentran las aplicaciones realizadas por los usuarios. A esta parte se le llama el **servidor de procesos**. Las aplicaciones aquí encontradas, que son demonios, están siendo ejecutas continuamente mientras dure su ciclo de vida a la vez que están realizando diferentes peticiones contra el servidor, ya sean de lectura o escritura. Estas aplicaciones realizan consultas continuamente a la base de datos y, cuando se cumple lo estipulado por el usuario en ellas, introducen el mensaje que el servidor debe de enviar al *Smart Object*. Para esto insertan una notificación en la base de datos, de tal manera que el servidor se da cuenta de la existencia de respuestas para dicho objeto.

Cabe destacar que, como se ve, los procesos se encuentran en la misma capa que el servidor, pues son parte de él y están en un continuo funcionamiento conjunto.

La otra parte de esta tercera capa es el **almacén de datos**. Este se encarga de centralizar todas las peticiones, de registrar los objetos inteligentes junto a sus servicios, guardar todos los datos y acciones de los servicios y de servir a las aplicaciones creadas por los usuarios de dicha información. Para ello hace uso de una arquitectura orientada a servicios (SOA), al igual que sugieren en otras investigaciones [2], [25], [98], [776], en conjunto a una arquitectura REST, como sugieren en [777] por sus beneficios y su facilidad de uso.

[207] Glosario: **Demonio**

De esta forma, se consigue que sea accesible para cualquier objeto que implemente el mismo lenguaje de mensajes que la plataforma Midgar. De esta manera, SOA aporta la encapsulación de servicios para ocultar los detalles de la implementación, ocultando así la heterogeneidad [19]. Mientras, mediante el uso de REST permite los objetos reciben y envían todos los mensajes necesarios. Cuando recibe un mensaje de un objeto, lo procesa y guarda los datos en la base de datos, en caso de ser necesario. Después comprueba si hay mensajes pendientes para dicho objeto. Si los hay, genera la respuesta con todos los mensajes pendientes para el *Smart Object* y le envía estos datos como respuesta a su petición. En caso contrario, le envía un mensaje de confirmación que indica la recepción correcta de la información.

19.2.1.4 OBJETOS

Por último, la **capa de los objetos** es donde se encuentran los objetos. Estos objetos deben de tener una aplicación que cumpla con el sistema de mensajes de la plataforma para así mantener una comunicación correcta que sea permanente y bidireccional con el servidor. De esta forma, mientras la aplicación está en funcionamiento, se comunica directamente con el servidor cuando debe de hacer peticiones web y con la base de datos cuando debe de escribir o leer datos directamente. Esto permite a los servicios creados por los usuarios interactuar con el servidor y dejarle los mensajes para los objetos. En la última capa se encuentran los *Smart Objects*, los sensores y los actuadores.

Los objetos que quieran conectarse con Midgar deben implementar una especificación de mensajes entendible por el servidor. Esta especificación define la interfaz por el cuál un objeto debe presentar sus acciones y servicios. Lo primero que debe hacer un objeto es registrarse en la plataforma. Así las acciones y servicios del objeto se registran en el almacén de datos de Midgar para que posteriormente los usuarios puedan utilizar los servicios de los objetos registrados en la especificación de procesos de coordinación entre objetos. Los servicios que puede ofrecer un objeto son aquellos datos que puede proporcionar. Dichas acciones pueden diferir entre unos objetos y otros. Estas se registran al mismo tiempo que el objeto y los servicios. Un objeto puede tener infinidad de acciones, al igual que servicios.

Por ejemplo, un smartphone puede proporcionar un servicio por sensor, un coche podría dar el kilometraje o el gasto de combustible y la centralita de una *Smart Home* podría dar la temperatura de cada habitación, su humedad o el estado de las ventas o puertas (abiertas o cerradas). Mientras, las acciones son lo que puede realizar cada objeto. Un móvil podría vibrar durante un determinado periodo de tiempo, mandar un SMS, realizar una llamada o crear una notificación y una *Smart Home* podría ofrecer el abrir y cerrar puertas y ventanas o encender o apagar la calefacción o el humidificador.

19.2.1.4.1 PROTOTIPO ANDROID

La aplicación nativa hecha para móviles Android da soporte a partir de la versión 2.1, que se corresponde con la API 7. Esta muestra una lista con los sensores poseídos por el teléfono que está siendo utilizado y los valores que capturan en tiempo real.

El prototipo de Android permite modificar de una manera sencilla los sensores que se desean enviar, así como escribir el mensaje de registro de los servicios y acciones que posee el dispositivo móvil. Tras esto, se despliega la aplicación en el dispositivo móvil, se selecciona el botón registro y se arranca para que comience

el envío de datos al almacén de datos de forma continuada. Estos datos serán los que utiliza el servidor de procesos para ejecutar las acciones definidas en los flujos de los programas desarrollados por los usuarios en la capa 1.

En esta primera iteración, el dispositivo móvil solo envía datos y recibe órdenes de que acciones debe de ejecutar.

19.2.1.4.2 PROTOTIPO ARDUINO UNO SMD

El microcontrolador Arduino Uno SMD utilizado, mostrado en la Ilustración **69**, se conectó mediante conexión Universal Serial Bus (USB) al ordenador.

Para realizar el envío y recepción de mensajes se ha utilizado una aplicación Java que hace de intermediaria entre el Arduino, el trato de parámetros y la conexión a Internet. De esta manera, se ha evitado el tener que utilizar un módulo wifi adicional. Además, facilita la creación de aplicaciones sobre Arduino al tener métodos muy parecidos a la aplicación Android y usar un lenguaje de más alto nivel.

También permite dotar de una manera fácil de inteligencia al Arduino para así, convertirlo en un *Smart Object* cuando convenga durante las pruebas, pues amplía la capacidad computacional de este.

Para su utilización primero hubo que realizar el montaje de los sensores sobre el microcontrolador y después realizar la configuración de estos en C, utilizando el IDE que posee el propio Arduino, de forma que este compilase y cargarse el programa en el microcontrolador. Una vez realizado este paso, se ejecuta la aplicación Java y queda listo para usarse. Primero se registra y después, automáticamente, comienza el envío y recibo de datos.

Al igual que el smartphone, en esta primera versión solo es capaz de enviar datos y recibir instrucciones acerca de qué acciones ejecutar.

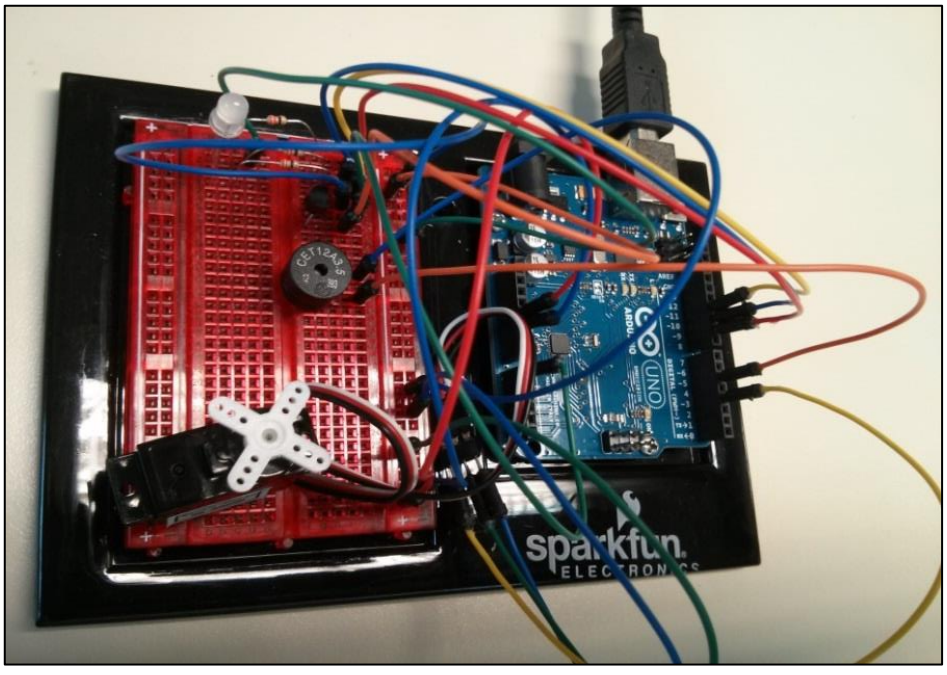


Ilustración 69 **Foto del Arduino durante una de las pruebas**

19.2.1.5 SISTEMA DE MENSAJES

A continuación, se muestra un ejemplo en el que registra un smartphone con el servicio del acelerómetro y la acción de vibrar.

En el Código fuente 13 se muestra un ejemplo de mensaje que es enviado desde un objeto al servidor para ser registrado.

- **register**: etiqueta padre única de todo el mensaje.
- **device**: etiqueta única padre de los datos generales del dispositivo.
    - **idDevice**: mac del dispositivo.
    - **descriptionDevice**: descripción del dispositivo que será mostrada a los usuarios.
- **service**: etiqueta que engloba los datos de un servicio del dispositivo y que se repite tantas veces como servicios tenga el dispositivo.
    - **idService**: idDevice + «-» + número del servicio empezando en 0.
    - **description**: descripción del servicio que será mostrada a los usuarios.
    - **sendType**: tipo de dato que envía el servicio y que podría ser *int*, *float*, *string*, *array*, etc.
- **action**: etiqueta que engloba los datos de una acción del dispositivo y que se repite tantas veces como acciones tenga el dispositivo.
    - **idAction**: idDevice + «-» + número del actuador empezando en 0.
    - **nameAction**: nombre de la acción que será mostrado a los usuarios.
    - **descriptionAction**: descripción del servicio que será mostrada a los usuarios.
    - **hasMessage**: el tipo del mensaje (int, float) en caso de que el actuador pueda recibir un dato. En caso contrario: «-».

```
<register>
    <device>
        <idDevice>e6644a22aae2cec6</dispositivo_id>
        <description Device>Nexus 4</dispositivo_descripcion>
    </device>
    <service>
        <idService>e6644a22aae2cec6-0</idservicio>
        <description>Acelerometro</descripcion>
        <sendType>float</enviaTipo>
    </service>
    <action>
        <idAction>e6644a22aae2cec6-0</idaction>
        <nameAction>Vibracion</action_name>
        <descriptionAction>Vibracion</action_description>
        <hasMessage>int</has_message>
    </action>
</register>
```

Código fuente 13 Ejemplo de mensaje de registro en Midgar

Una vez está registrado, el objeto puede comenzar a enviar datos al servidor. En cada envío, el objeto debe de adjuntar todos los datos de los servicios existentes: el dato a enviar y el número del servicio. Esto se muestra en el Código fuente 14.

- **send**: etiqueta padre única de todo el mensaje.
- **data**: etiqueta padre de los datos enviados desde el dispositivo al servidor. Se repite tantas veces servicios que envíen datos.
    - **datum**: dato enviado en el formato indicado en el registro.

- **service**: identificador del servicio correspondiente al dato y que fue registrado previamente bajo este identificador.

```
<send>
    <data>
        <datum>8.365426</datum>
        <service>e6644a22aae2cec6-0</service>
    </data>
    <data>
        <datum>30</datum>
        <service>20</service>
    </data>
</send>
```

Código fuente 14 Ejemplo de mensaje de envío de datos al servidor desde un objeto

Tras ello, el objeto espera por una respuesta del servidor, ya sea, la confirmación de llegada o un mensaje con acciones que debe ejecutar. En el siguiente ejemplo, Código fuente 15, se puede ver como recibe la petición para realizar la acción 0 con un valor de 1000. En este caso se corresponde con la vibración, a la que le ordena que vibre durante 1 segundo.

- **answers**: etiqueta padre de todo el mensaje.
  - **answer**: etiqueta que contiene los datos de una respuesta. Esta puede repetirse tantas veces como respuestas se envíen.
    - **action**: número del actuador a ejecutar de los registrados previamente. No se introduce el idDevice.
    - **result**: parámetro a pasarle a la acción, si lo requiere, como en el caso de la vibración en el que se le pasan los milisegundos que debe de vibrar. En caso de que no reciba parámetros: «-».

```
<answers>
    <answer>
        <action>0</action>
        <result>1000</result>
    <answer>
</answers>
```

Código fuente 15 Ejemplo de mensaje de respuesta del servidor a un objeto

En cambio, si el servidor no tuviera mensajes para dicho objeto, entonces le envía el mensaje de confirmación, el cual se puede ver en el Código fuente 16.

- **device**: etiqueta única padre.
  - **status**: etiqueta que muestra el estado del servidor. En este caso «ok» significa que el servidor recibió y proceso los datos correctamente. En otros casos podría llevar un error.

```
<device>
    <status>ok</status>
</device>
```

Código fuente 16 Mensaje de confirmación de datos recibidos por el servidor

19.2.2 TIPOS DE OBJETOS UTILIZADOS Y SOPORTADOS POR MIDGAR

Durante la realización de esta tesis, se utilizaron diferentes objetos, tanto inteligentes como no inteligentes, en base a las diferencias entre este tipo de objetos presentadas en [48]. En este apartado se explicarán los diferentes tipos de objetos utilizados, diferenciando así los no inteligentes en sensores y actuadores y los inteligentes o *Smart Objects* según la clasificación en base a las tres categorías de inteligencia que pueden poseer.

19.2.2.1 OBJETOS NO INTELIGENTES

En la Ilustración 70 se muestra algunos sensores y actuadores utilizados.

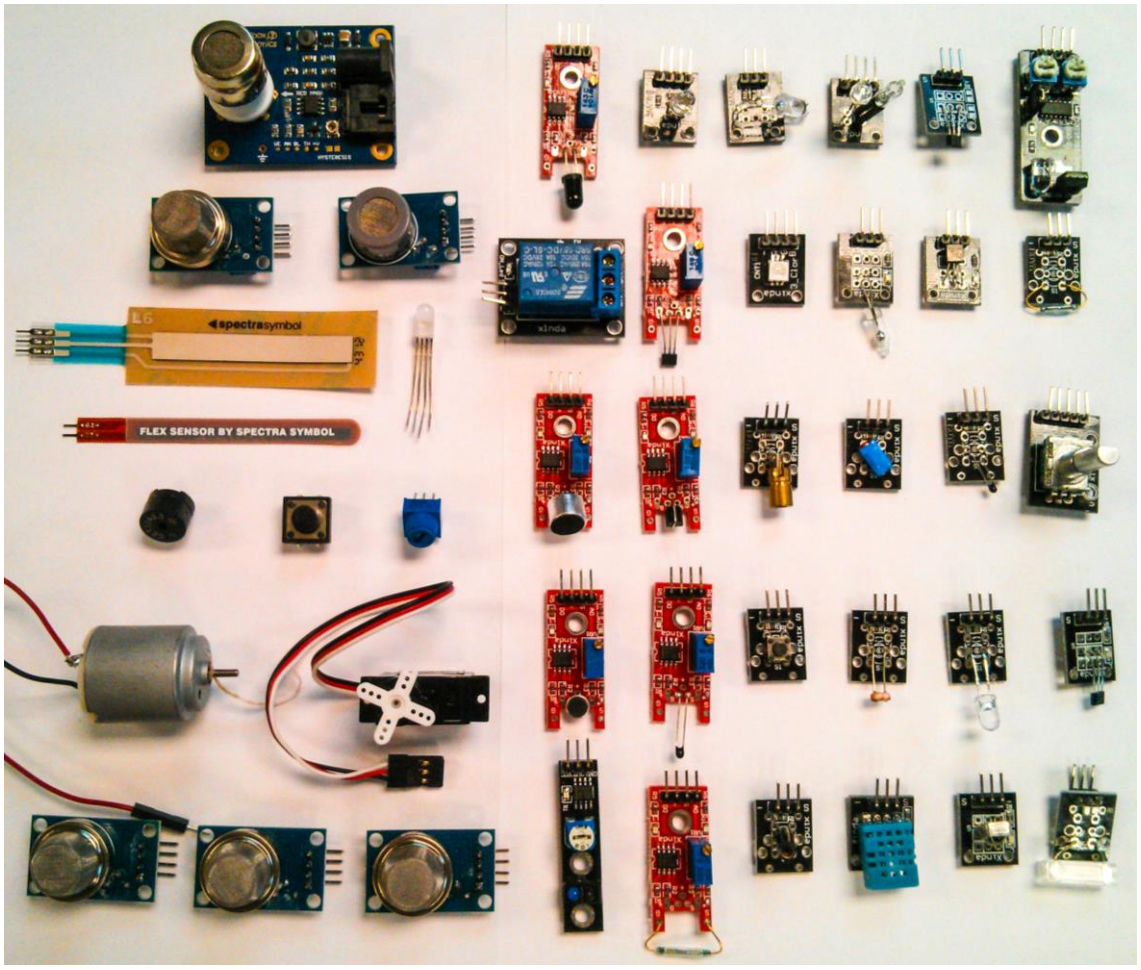


Ilustración 70 Sensores y actuadores utilizados

Primeramente, los objetos no inteligentes se utilizan mediante su conexión a objetos inteligentes, pues necesitan de un objeto central que sea capaz de enviar sus datos a la nube. Entre los objetos no inteligentes que se han utilizado hay sensores de todo tipo. Se han utilizado sensores de:

- Detección de llamas → KY026.
- Sensores de gases:
  - Monóxido de carbono (CO) → MQ7.
  - Dióxido de carbono (CO_2) → MG811.
  - Metano (CH_4) → MQ4.
  - Hidrógeno (H_2) → MQ8.
  - Alcohol (C_2H_5OH) → MQ3.
  - Gases Licuados del Petróleo (GLP), propano, H_2, CH_4 y Alcohol →MQ2.
  - GLP, gas natural y gas ciudad → MQ5.
  - GLP, butano y propano → MQ6.
  - CO, CH_4 y GLP → MQ9.
  - Amoniaco (NH_3), benceno y alcohol → MQ135.
- Láser → KY008.
- Termistor TM3P6 y 19B20.
- Sensor de temperatura y humedad → DHT11 / KY015.
- De flexión (*flex sensor*).
- Potenciómetros por presión (*soft potentiometer*).
- Ritmo cardiaco → KY039.
- Receptor de infrarrojos → KY022.
- Fotorresistor → KY018.
- … y más.

Por otro lado, para poder realizar acciones se han utilizados actuadores de dos tipos. Los primeros son las propias acciones que forman parte y que permitían ejecutar los smartphones, como son la vibración, enviar un correo, encender o apagar el flash de la cámara, enviar un SMS, llamar a un determinado número o cualquier otra acción disponible desde el smartphone e inseparable de este objeto. El segundo tipo de actuadores eran los propios objetos independientes conectados a un Arduino o Raspberry Pi, como un motor, un servomotor, un altavoz (KY012), luces RGB (KY011, KY016 y otros), LEDs (KY034 y otros sueltos), SMD (KY009), botones, un emisor láser (KY008), potenciómetros, infrarrojos (KY005), etc.

19.2.2.2 OBJETOS INTELIGENTES

Como *Smart Objects* se ha utilizado un total de cuatro objetos con diferentes niveles de inteligentes, los cuales pueden verse en la Ilustración 71. El primero de ellos ha sido el Arduino Uno al que se le han conectado diferentes sensores y actuadores a él, entre los que se encuentran los anteriormente explicados. También en otros casos se cambió el Arduino Uno por una Raspberry Pi 2 con sensores y actuadores, pero con un programa que la dotaba de más inteligencia que el Arduino Uno. Otros *Smart Objects* que se han utilizado han sido diferentes smartphones Android para ser usados tanto como para recoger datos de su alrededor como para ejecutar acciones. Por último, se ha utilizado una cámara IP que ha permitido tratar con fotos cuando se detectaba cambio en la ventana visual de la cámara.

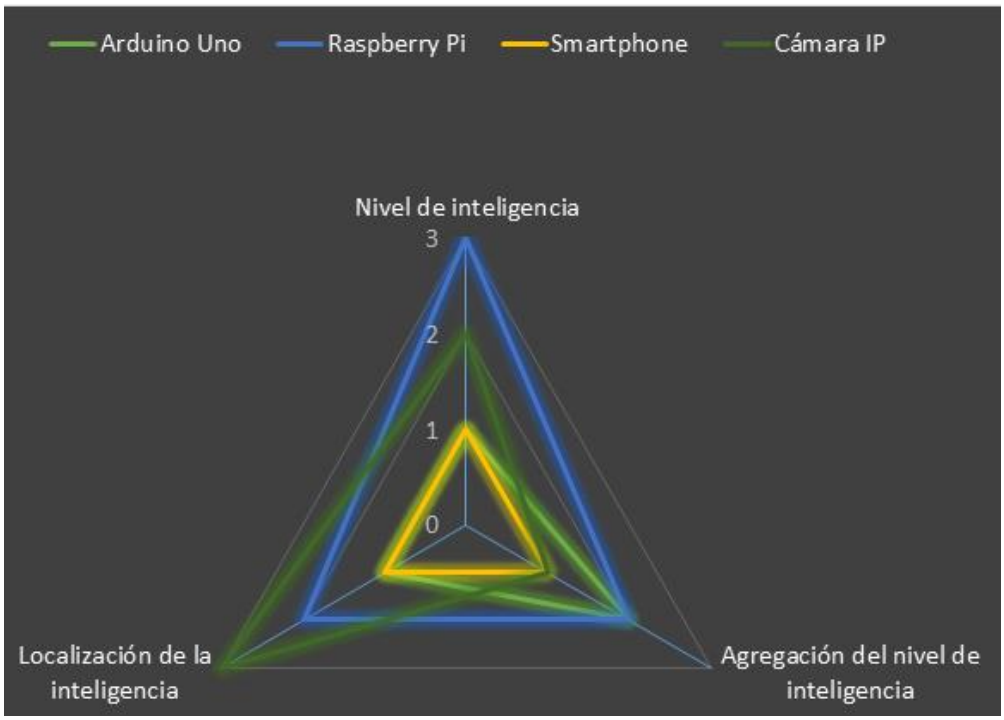


Ilustración 71 Tipos de *Smart Objects* utilizados en la tesis sobre la plataforma Midgar

Para representar la clasificación de inteligencia de los *Smart Objects* utilizados se ha utilizado la descrita en el artículo [48], la cual está basada en la clasificación de Meyer [26], pero que amplia dos categorías de inteligencia con un nivel más cada una. Además, se explica exactamente las diferencias entre un objeto inteligente y un objeto no inteligente.

19.2.3 SOFTWARE Y HARDWARE UTILIZADO

Para desarrollar este prototipo se ha utilizado hardware y software muy diverso. No obstante, en algunos casos, tanto el hardware y el software soportado y/o utilizado fue evolucionando durante estos tres años. Por eso, aquí se recoge, o bien la última versión utilizada del software y que es soportada por todas las iteraciones, o bien las diferentes versiones que se ha soportado.

Software:

- La plataforma IoT Midgar ha sido desarrollado utilizando el *framework* **Ruby on Rails**.
    - Ruby 2.3.1p112.
    - Rails 4.2.6.
- Para el **procesador** de los DSLs creados por la primera capa y que se encuentra en la segunda capa de utiliza Java. Primeramente, fue Java 6, posteriormente Java 7 y actualmente **Java 8**.
- **La aplicación de conexión con el Arduino** está realizada en Java, y, al igual que con el procesador, pasó por las tres últimas versiones de este GPL: Java 6, Java 7 y ahora **Java 8**.
- El primer **servidor web** para Ruby on Rails fue Webrick, posteriormente se ha utilizado **Thin** por ser más rápido, ligero y fácil de configurar frente a otros como Apache.
    - Thin 1.6.4.

- Base de datos SQLite que después se ha cambiado por MySQL.
    - MySQL 5.5.50.
    - Conector: gema mysql2 0.3.20.
- IDE de Arduino para el desarrollo del software para Arduino, donde se han ido utilizando las versiones más recientes.
    - 1.6.10.

Hardware:

- Servidor de alojamiento:
    - Ordenador dedicado con sistema operativo Windows Server 2008R2 de 64 bits, procesador Intel Core i3-2100 a 3100MHz y 4GB de memoria RAM.
    - Diferentes máquinas virtuales con Ubuntu 14.04 LTS.
    - Actualmente en una Raspberry Pi 2 modelo B con un sistema operativo Raspbian Jessie 4.4.13 v7.
- Un microcontrolador **Arduino Uno SMD**.
- Diferentes **smartphone Android**:
    - Nexus 4 con Android 4.2.2, más tarde con versiones más actuales de Android hasta llegar a la 5.1.1, inclusive.
    - Motorola con la versión 2.2.2
    - Samsung Galaxy S con la versión 2.1
    - Samsung Galaxy Mini S5570 con la versión 2.3.6.

19.3 *CONCLUSIONES*

En esta primera iteración se ha desarrollado una plataforma de Internet de las Cosas, llamada Midgar, que permite interconectar cualquier dispositivo heterogéneo y ubicuo a ella, siempre que el dispositivo cumpla el sistema de mensajes de la plataforma. Esta se encarga de interconectar los objetos mediante las aplicaciones que puede albergar en la tercera capa, de guardar sus datos y de tomar las decisiones oportunas en base a lo que hagan las aplicaciones que albergue y notificarles la decisión a los objetos.

Además, Midgar es totalmente ampliable y fácil de mantener, lo que ofrece una plataforma web para Internet de las Cosas buena para las investigaciones llevadas a cabo en esta tesis, presentadas en los capítulos posteriores, así como para futuras investigaciones.

20. GENERACIÓN DE APLICACIONES PARA INTERCONECTAR OBJETOS HETEROGÉNEOS Y UBICUOS UTILIZANDO UN LENGUAJE DE DOMINIO ESPECÍFICO

«Un diseñador sabe que ha alcanzado la perfección no cuando ya no tiene nada más que añadir, sino cuando ya no le queda nada más que quitar»
Antoine de Saint-Exupéry

«Uno no descubre nuevas tierras sin perder de vista la costa»
Andre Gide

«Hay una antigua historia sobre una persona que quería que su ordenador fuese tan fácil de utilizar como su teléfono. Estos deseos se han hecho realidad, ya no sé cómo usar mi teléfono»
Bjarne Stroustrup

La comunicación y colaboración entre objetos para cumplir ciertas metas es una de las claves fundamentales de Internet de las Cosas. En el futuro, se espera que la gente viva rodeada de múltiples objetos, siendo estos objetos los que más tráfico generen por Internet.

La gente ya ha demostrado que cada vez tienen una mejor habilidad con los dispositivos electrónicos, sin embargo, esto no implica que sepan o deban saber programar. Esto dificulta que los propios usuarios sean quienes desarrollen sus propias interconexiones, pues posiblemente es muy probable que en el futuro esta gente quiera o necesite adaptar el comportamiento de sus objetos a sus necesidades y definir las colaboraciones de estos.

Por estos motivos, se hace necesario el ofrecer facilidades a la gente sin conocimientos en el campo de desarrollo de software para que puedan crear las interconexiones que deseen entre sus propios objetos. En esta iteración se proporciona una posible solución a este problema. Utilizando Midgar como base se busca poder facilitar dicha tarea a cualquier persona en comparación con el desarrollo tradicional de las aplicaciones.

20.1 *DESCRIPCIÓN*

20.1.1 **OBJETIVOS**

La hipótesis de esta iteración es saber si se puede facilitar el desarrollo de aplicaciones que ofrezcan interconexión de objetos en el marco IoT de una forma rápida y sencilla mediante Ingeniería Dirigida por Modelos.

El objetivo de esta segunda iteración ha sido tratar de cumplir la hipótesis aquí planteada. Para ello se han debido de cumplir una serie de objetivos:

- Desarrollar un Lenguaje de Dominio Específico para la creación de aplicaciones que interconecten dispositivos heterogéneos.
- Aplicar Ingeniería Dirigida por Modelos para crear el DSL.
- Obtener una alta abstracción para la creación de aplicaciones mediante un DSL web gráfico.
- Comunicar objetos heterogéneos y ubicuos.

20.1.2 **FUNCIONALIDADES**

Para cumplir con los objetivos anteriormente explicados se ha utilizado la plataforma Midgar. Esta plataforma ha implementado en su primera capa un DSL que permitiese ofrecer una abstracción alta a usuarios sin conocimientos de programación que les permita crear aplicaciones que interconecten objetos heterogéneos y ubicuos. Esto repercute en una mayor facilidad a la hora de crear aplicaciones por parte de un usuario, en una reducción del número de errores a la hora de realizar estas aplicaciones, y una mayor facilidad y rapidez a la hora de crearlas.

Para obtener la abstracción necesaria se ha realizado un DSL gráfico que ayuda al usuario a escribir la sintaxis concreta diseñada para crear el modelo formal de las aplicaciones interconectoras de objetos heterogéneos y ubicuos. Este DSL, además de aumentar la facilidad de escribir y generar el código necesario, ayuda a reducir los posibles errores que puedan surgir a la hora de definir el modelo formal deseado para crear el objeto.

La aplicación generada a partir de la transformación del modelo formal descrito por el usuario al utilizar el DSL gráfico es capaz de conectar cualquier objeto, sin importar la plataforma, el tipo de hardware o el lenguaje en el que esté realizado mediante la arquitectura propuesta en esta investigación.

Para poder interconectar los objetos, el usuario debe de utilizar el DSL gráfico realizado para así definir el modelo formal, es decir, lo que desea que la aplicación realice. Si así lo desea, el usuario puede utilizar también un DSL textual, o bien no utilizar ninguno y definir el modelo formal escribiendo el modelo formal sin ayuda. No obstante, como se verá en la Evaluación y discusión, el DSL gráfico junto a su editor ayudan de una manera bastante óptima en la creación del modelo formal.

Gracias al DSL, un usuario puede describir todo el flujo que desee que realice la aplicación. En la aplicación es posible conectar multitud de objetos y leer de ellos cualquier parámetro de un servicio que estos

posean y del que suban datos, siempre que estos servicios se encuentren registrados en la plataforma Midgar. De esta manera, el usuario puede leer datos de varios objetos y en base a estos datos, mandar realizar ciertas acciones a los mismos o a otros objetos diferentes. Las aplicaciones realizadas pueden tener infinidad de objetos conectados y acciones a realizar.

20.2 *ARQUITECTURA PROPUESTA*

La arquitectura del sistema se puede dividir en cuatro capas como se ilustra en la Ilustración 72, y que se corresponden con las cuatro capas de la plataforma Midgar. Cada una es un proceso dentro del conjunto global dentro de la infraestructura.

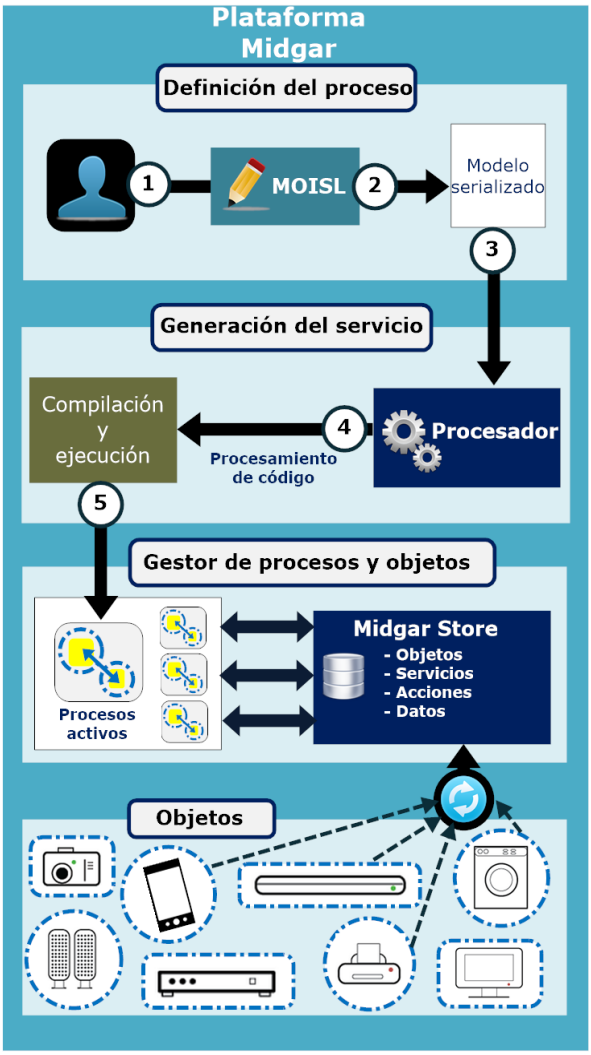


Ilustración 72 Arquitectura de MOISL

La primera capa abarca el proceso de definición por parte del usuario. En este proceso, un usuario cualquiera mediante el uso del DSL web debe definir lo que quiere que haga su aplicación. Tras terminar de definirla, el editor genera el modelo formal en formato XML. Esta información es pasada a la segunda capa: la capa de generación del servicio.

Esta segunda capa se encarga de procesar la información, generar una aplicación a partir de ella, compilarla y ejecutarla en el servidor en modo de demonio. Este último paso forma parte de la capa tres, la capa del servidor.

De esta manera, la aplicación creada por el usuario queda funcionando en el servidor realizando lo que el usuario haya definido. Mientras está en funcionamiento, la aplicación se comunica directamente con el servidor y la base de datos y establece, si se debe, los mensajes a enviar al objeto requerido la próxima vez que este se conecte. Esta capa, además, contiene también el servicio al que se deben de conectar los *Smart Objects* que interactúan con Midgar.

Estos objetos deben implementar el paso de mensajes concreto de Midgar para mantener la conexión con el servidor, el cual fue explicado en el apartado 19.2.1.5. Por medio del procesado de estos mensajes, el servidor obtiene los datos de cada objeto. Esto permite a los servicios creados por los usuarios interactuar con el servidor y dejar los mensajes para los objetos. En la última capa se encuentran los *Smart Objects*. Estos objetos implementan la interfaz de mensajes para así poder mantener una conexión permanente y bidireccional con el servidor.

20.2.1 PRESENTACIÓN DE LA ARQUITECTURA DE LA INGENIERÍA DIRIGIDA POR MODELOS UTILIZADA

Para crear el Lenguaje de Dominio Específico requerido, primero había que elegir el meta-metamodelo y crear el metamodelo del problema perteneciente a nuestro dominio, el cuál es la **creación de aplicaciones para interconectar dispositivos en una red IoT por usuarios no expertos en el desarrollo de software**.

Tras esto, había que definir la sintaxis abstracta y la sintaxis concreta, así como sus respectivas representaciones por medio de uno o varios DSLs que permitiesen crear el modelo formal a los usuarios que lo utilizasen para crear aplicaciones pertenecientes al dominio a resolver. Estos DSLs van acompañados de pequeños editores que facilitan su manejo y que contienen la semántica estática que verifica en tiempo real que el modelo formal creado por el usuario cumpla en todo momento los conceptos definidos en el metamodelo.

Por esto, en este apartado se tratan estos dos temas anteriores presentándolos y explicando ambos casos.

20.2.1.1 METAMODELO

Como meta-metamodelo se eligió Ecore [181] debido a la facilidad de uso y creación que ofrece, así como las posibilidades que otorga el proyecto *Eclipse Modeling Framework* [239] con las Ecore Tools [778] y el previo conocimiento de su uso en otras ocasiones y utilización en la propuesta [39].

El siguiente paso fue definir el metamodelo a partir del meta-metamodelo. Este se puede ver en la Ilustración 73. En este metamodelo se define de manera formal los conceptos relevantes que tiene nuestro dominio.

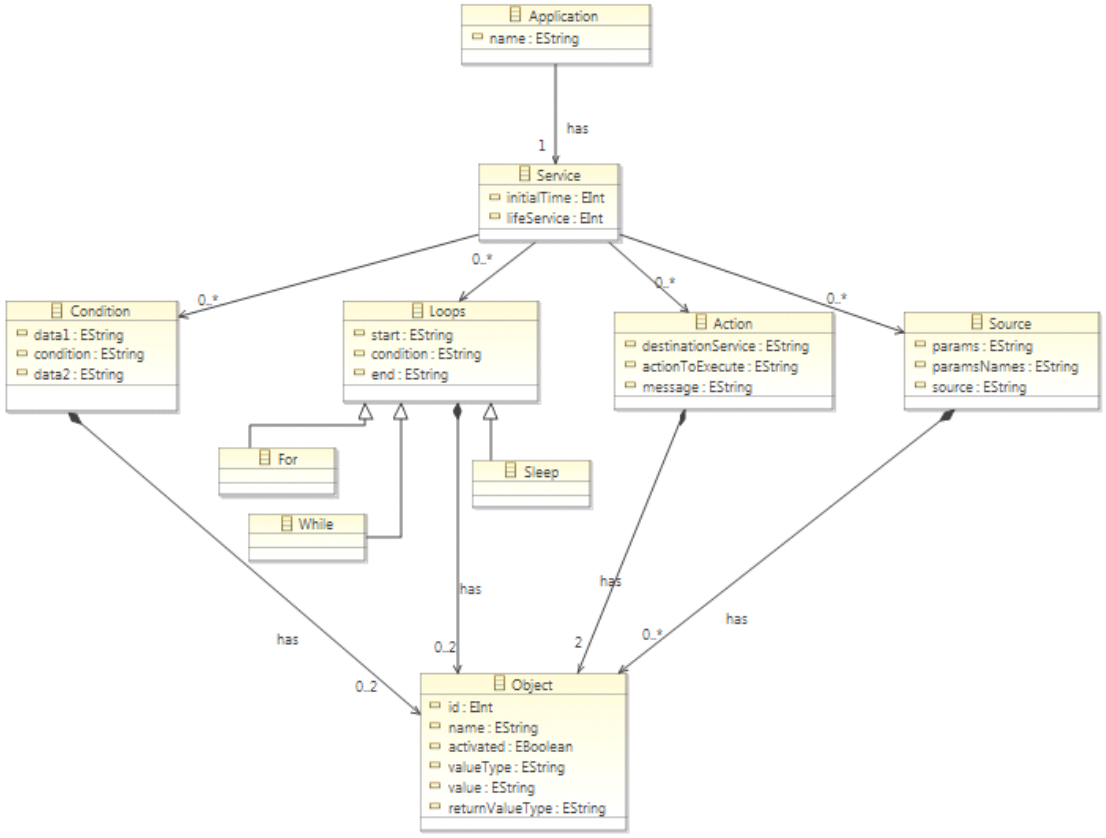


Ilustración 73 Metamodelo [208]

Para crearlo, se pensó en que necesitaría el usuario para poder generar una aplicación que contuviese todo lo que otorga un lenguaje de programación, como son los bucles, los condicionales, la inserción de código fuente y los hilos. Además, como se ve en el metamodelo, este también contiene los datos de la aplicación y del servicio, las acciones y los propios datos relevantes de los objetos, pues son una parte importante de todo el sistema y que son necesarios conocer.

20.2.1.2 SINTAXIS ABSTRACTA

Una vez realizado el metamodelo se especificó la sintaxis abstracta. La sintaxis abstracta cuenta con un total de ocho nodos. Tres de ellos se corresponden con estructuras del lenguaje de programación («*if*», «*for*» y «*while*»), uno con opción de dormir el proceso («*sleep*»), uno con la inserción de código directo del propio lenguaje («*source*»), y otro con la acción a ejecutarse («*action*»).

Por medio de estos, el usuario puede crear el flujo de ejecución del programa ya sea, insertando condicionales para que solo se ejecute bajo una o varias condiciones, un bucle para que se ejecuten una o más tareas varias veces o controlar el flujo de tiempo de ejecución de programa insertando pausas o esperas. La diferencia entre el «*while*» y el «*for*» es que el primero da soporte a las condiciones basadas en tiempo.

[208] Se encuentra en lengua inglesa debido a que la implementación ha sido realizada en esta lengua

Para crear el flujo del programa, el usuario debe ir anidando los diferentes nodos unos dentro de otros si lo que desea es que estos se ejecuten cuando se cumpla el padre, en caso de ser el condicional o ejecute eso si el padre es un bucle. Si lo que desea es que se ejecute uno detrás de otro, simplemente debe colocarlos seguidos.

Los dos nodos restantes sirven para poder definir datos de la aplicación, como es su nombre en «*application*» y el tiempo de espera en milisegundos para que se inicie la aplicación, contado desde el momento en que se crea, y si la aplicación esta debe de ejecutarse por siempre o un solo por un tiempo determinado en el nodo «*service*».

Por otro lado, el «*action*» es el elemento final y más importante, pues, mediante su inclusión, se permite enviar una orden junto a un mensaje a un objeto para que la ejecute.

20.2.1.3 SINTAXIS CONCRETA

Una vez definida la sintaxis abstracta se hizo su representación y equivalencia a la sintaxis concreta, la cual posee dos DSLs que la representan. Primeramente, la sintaxis concreta del DSL textual utilizando un lenguaje basado en XML. Tras la versión textual, se realizó una conversión a una representación gráfica de la sintaxis concreta para crear el DSL gráfico. Las equivalencias entre la sintaxis abstracta y los dos DSLs que representan la sintaxis concreta se pueden ver en la Tabla 5. Ambos DSLs son conocidos por el nombre de **Midgar Object Interconnection Specific Language** (MOISL), siendo uno textual y el otro gráfico.

Cabe destacar que, en el DSL gráfico, los nodos «*application*» y «*service*» se fusionaron en uno para facilitar la generación de aplicaciones y simplificar así el DSL gráfico.

Sintaxis abstracta	Sintaxis concreta	
	DSL Textual	DSL Gráfico
Application	<application>	
Service	<service>	
Condition	<ifCondition>	
For	<forLoop>	
While	<whileLoop>	
Sleep	<sleep>	
Action	<action>	
Source	<java>	

Tabla 5 Correspondencia entre la sintaxis abstracta y la sintaxis concreta [209]

20.2.2 IMPLEMENTACIÓN

Para el desarrollo de esta iteración se ha utilizado la plataforma IoT desarrollada previamente, conocida como Midgar. En este apartado se describirá en detalle la ampliación de la plataforma Midgar que se ha llevado a cabo para soportar esta segunda iteración, los diferentes componentes que intervienen, su interacción, y la solución adoptada para cumplir los objetivos de esta parte.

Se comenzará describiendo en detalle los Lenguaje de Dominio Específico, tanto el textual como el gráfico, en conjunto al editor que los apoya, así como estos DSLs se enlazan con la segunda capa. A continuación, se explicará esta segunda capa para ver cómo se procesa el modelo formal creado con el DSL y como este procesador genera la aplicación demonio residente en el servidor.

Debido a que la tercera y la cuarta capa de Midgar no han sido modificadas y se mantuvieron iguales que como se han explicado en el capítulo anterior, no se explicarán.

[209] Se encuentra en lengua inglesa debido a que la implementación ha sido realizada en esta lengua

20.2.2.1 LENGUAJE DE DOMINIO ESPECÍFICO TEXTUAL

La primera versión realizada fue la del DSL textual. Este DSL está basado en XML.

La primera parte del DSL es donde se define el nombre de la aplicación, el tiempo de inicio y si es o no permanente en el tiempo. Dentro de las dos etiquetas hay que insertar la aplicación, es decir, su funcionalidad por medio de las etiquetas que se describirán a continuación. Un ejemplo de esto está en Código fuente 17, donde se pone el nombre de la aplicación y se especifica un inicio a los cinco segundos y su tiempo de vida infinito.

```
<application name='MIDGARTest'>
  <service initialTime='5000' lifeService='true'>
      ...
  </service>
</application>
```

Código fuente 17 Encabezado del DSL textual

El ejemplo del Código fuente 18 se corresponde con una de las primeras aplicaciones realizadas y en el que se muestra un condicional con una acción dentro. El dato cogido del primer servicio se correspondía con el eje Y del acelerómetro del smartphone. Cuando el smartphone se ponía casi en vertical y superaba dicho valor, se enviaba un mensaje a otro móvil. En este primer ejemplo se ejecutaba una acción siempre y cuando se cumplía la condición. Si el resultado devuelto por el servicio dos del dispositivo *e6644a22aae2cec6* es mayor que ocho, el servicio cero del dispositivo *c3b9f28c24f2be8b* ejecuta la acción («actionToExecute») dos y se le envía el mensaje «*This action is a popup*». En este caso, lo que hace es ejecutar la acción dos del dispositivo, que es un mensaje por pantalla que muestra el mensaje recibido.

```
<ifCondition data1='serviceId:e6644a22aae2cec6-2'
condition='higher' data2='8'>
  <action destinationService='c3b9f28c24f2be8b-0'
actionToExecute='2' message='This action is a popup'>
  </action>
</ifCondition>
```

Código fuente 18 Condición utilizando el DSL textual

Otro ejemplo es el de usar un bucle para enviar varias acciones seguidas, pero con una pausa entre cada acción de varios segundos. Esto puede ser definido como se muestra en Código fuente 19, que se corresponde con una aplicación realizada que leía la temperatura de un sensor que se encontraba colocado en un microcontrolador Arduino Uno. Cuando este superaba los 25ºC enviaba por cada grado de diferencia una vibración de 2.500 milisegundos al móvil personal del usuario. Este servicio puede resultar útil para controlar sitios en los que pueda subir mucho la temperatura y avise de una forma un poco intrusiva al controlador de la zona.

Otro ejemplo de uso podría ser la cocina y que, en vez de enviarlo a un móvil, lo enviase a otro microcontrolador Arduino que dispusiese de un altavoz y así poder avisar mediante una alarma sonora al cocinero. Como se ve en el Código fuente 19, cuando el valor del servicio dos del objeto *08002700FC83* supera el valor 25 se ejecuta un bucle que va desde veinticinco hasta el número devuelto por el mismo servicio.

Una vez entra, ejecuta la acción desde cero con un mensaje de 2500. En este caso esta acción es la vibración con una duración de 2.500 milisegundos. Tras ejecutar esta acción, ejecuta una similar a la del ejemplo anterior, un mensaje, y después espera cinco segundos para enviarle de nuevo la misma acción al dispositivo mediante el uso de la pausa del programa.

```
<ifCondition data1='serviceId: 08002700FC83-2' condition='higher'
data2='25'>
     <forLoop start='25' condition='less'
end='serviceId:08002700FC83-2'>
          <action destinationService='c3b9f28c24f2be8b-0'
actionToExecute='0' message='2500'></action>
          <action destinationService='c3b9f28c24f2be8b-0'
actionToExecute ='2' message='Temperature > 25°C'></action>
          <sleep start='0' condition='higher' end='5000'></sleep>
     </forLoop>
</ifCondition>
```

Código fuente 19 Ejemplo de bucle «*for*»

Como se pudo observar en el ejemplo anterior, las dos acciones y el «*sleep*» se ejecutan dentro del cuerpo del bucle, al igual que este se ejecuta sólo cuando se cumple la condición. Por eso van anidados. No obstante, las acciones y el «*sleep*» se ejecutan una detrás de otra pues no están anidadas entre ellas.

El usuario también puede realizar un bucle «*while*», el cual es exactamente igual que el «*for*», pero cambiando la etiqueta «*forLoop*» por «*whileLoop*», como se ve en Código fuente 20. En este código se muestran dos bucles «*while*», en donde el primero es exactamente igual que el bucle «*for*» del ejemplo anterior, y el segundo hace uso del tiempo en su condición.

```
<whileLoop start='25' condition='less' end='serviceId:08002700FC83-
2'>
     <action destinationService='c3b9f28c24f2be8b-0'
actionToExecute='0' message='2500'></action>
</whileLoop>

<whileLoop desde="System.currentTimeMillis()"
hasta="System.currentTimeMillis()+5000" condition="mayor">
     <action destinationService='c3b9f28c24f2be8b-0'
actionToExecute='0' message='2500'></action>
</whileLoop>
```

Código fuente 20 Bucle «*while*»

Incluso el usuario puede introducir código Java directamente en la aplicación. En el segundo caso esta etiqueta viene bien si el usuario posee conocimientos de informática y quiere otorgar de una mayor funcionalidad a la aplicación. Como se ve en el ejemplo de abajo, que se corresponde con el Código fuente 21, el usuario puede incluso definir los parámetros que recibirá de los servicios que él elija y su nombre. Tras esto, el usuario puede insertar el código Java que desee. En este ejemplo si el primer valor es mayor que el segundo, realiza una inserción en la base de datos para dejar un mensaje al móvil. Este ejecutará la acción 0, que es la vibración y le pasa como parámetro el 4000, que se corresponde con 4 segundos en el caso de la vibración.

```
<java params='serviceId:c3b9f28c24f2be8b-1, serviceId: 08002700FC83-
0' paramsNames='temperatureMobile, 'temperatureArduino'
javaSource='if(temperatureMobile > temperatureArduino){new
BBDD().insertAnswer("c3b9f28c24f2be8b-1", 0, "4000");}'>
</java>
```

Código fuente 21 Etiqueta Java para insertar código Java en el proyecto

Para facilitar la tarea de escritura se desarrolló una herramienta que facilitase la escritura del DSL textual y que se puede ver en la Ilustración 74. Cuando se accede a ella, los nodos «*application*» y «*service*» ya vienen creados por defecto, de tal forma que el usuario lo único que deberá hacer es completar estos e insertar dentro de ellos el contenido que quiera para crear la aplicación. Para esto, el usuario deberá colocar el cursor del teclado en la zona donde desee insertar texto y seleccionar una de las opciones de la izquierda. Esto hará que ese nodo se le inserte donde tenía situado el cursor. Con esto se consigue que el usuario sólo deba rellenar los atributos. El editor utilizado para definir el modelo formal utilizando el DSL textual se puede dividir en tres zonas.

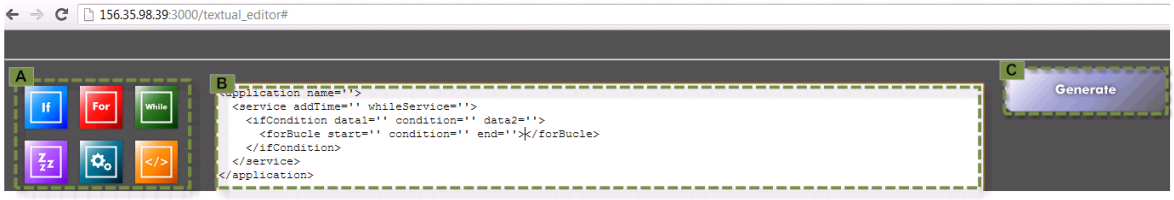


Ilustración 74 DSL textual para la generación de aplicaciones interconectoras de objetos en la plataforma Midgar [210]

La primera zona (Ilustración 74A) contiene el panel de selección con los botones para incluir un elemento en el flujo de la aplicación. De izquierda a derecha y de arriba abajo son: el «*if*», el «*for*», el «*while*», el «*sleep*», el «*action*» y la inserción de código, Java en este caso.

Como ejemplo se puede ver la Ilustración 74B. La aplicación aquí creada tiene los nodos «*application*», «*service*», «*if*» y «*for*». Fueron insertados como se explicó antes, pero contienen los atributos sin rellenar. Gracias a este editor, esto es lo único que el usuario debe hacer.

Por último, en la Ilustración 74C se ve el botón «generar» que permite la generación de esta aplicación.

20.2.2.2 LENGUAJE DE DOMINIO ESPECÍFICO GRÁFICO

En la Ilustración 75 se muestra el DSL gráfico creado para Midgar, junto a su editor. De esta forma, si el usuario desea generar una aplicación lo que debe hacer es seleccionar la caja correspondiente al flujo que desea añadir al programa y enlazarla al resto mediante el uso de los conectores. El editor de este DSL gráfico se puede dividir en cinco zonas.

[210] Se encuentra en lengua inglesa debido a que la implementación ha sido realizada en esta lengua

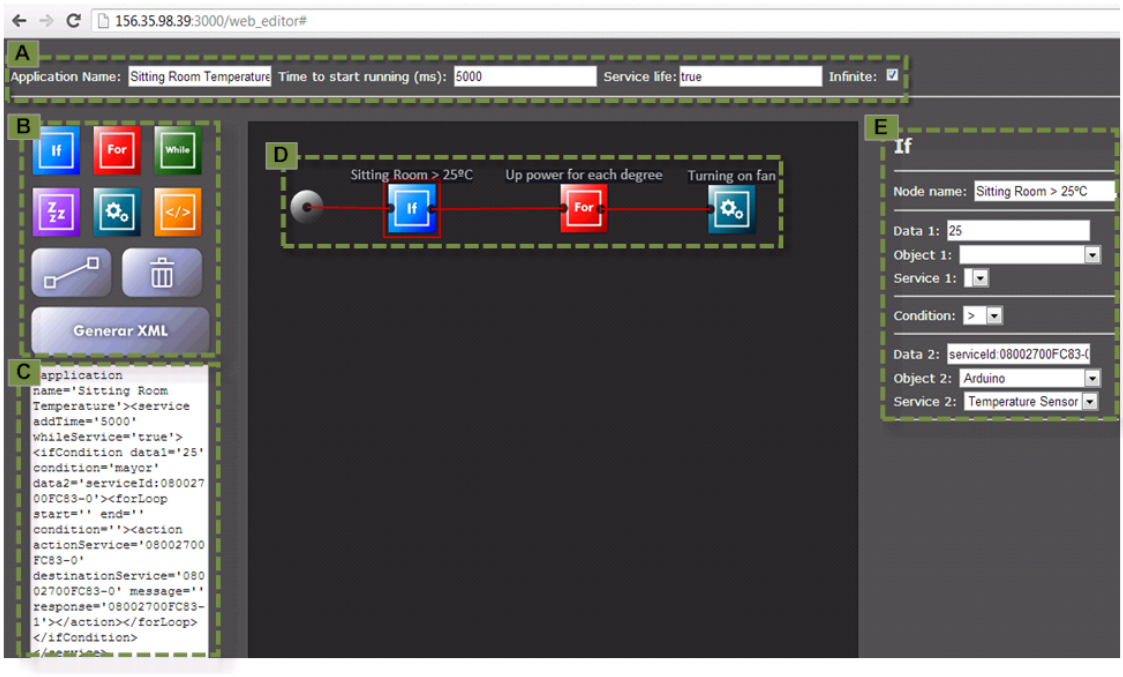


Ilustración 75 Editor gráfico para la generación de aplicaciones interconectoras de objetos de la plataforma Midgar por medio del DSL gráfico MOISL [211]

La primera zona se corresponde a la Ilustración 75A. Esta zona contiene los datos correspondientes a la aplicación que se desea crear: nombre de la aplicación, tiempo de espera hasta comenzar a ejecutarse desde el momento en que se crea, y el tiempo de vida de la aplicación desde el momento en que es iniciada. En este último se puede indicar mediante el *checkbox* si su ejecución es infinita, obviando el parámetro anterior, o no.

La segunda zona (Ilustración 75B) contiene el panel de selección con los botones para incluir un elemento en el flujo de la aplicación. De izquierda a derecha y de arriba abajo son: el «*if*», el «*for*», el «*while*», el «*sleep*», el «action», y la inserción de código Java. El siguiente botón es el de conexión de nodos. Mediante el uso de este botón se crean las conexiones entre los elementos incluidos en la aplicación y se establece así el flujo y orden que debe seguir la aplicación desarrollada. A su derecha está el botón de eliminar. Con él se puede eliminar y relación cualquier elemento del flujo desarrollado. Por último, el botón generar permite la generación de esta aplicación y la visualización del DSL textual en el editor.

En la Ilustración 75C se encuentra la visualización del modelo formal correspondiente a la aplicación creada por el usuario con el editor y representado con DSL textual. En la zona central se encuentra la zona de trabajo. Aquí es donde se crea la aplicación y se conecta el flujo que se desea realizar. El flujo de la aplicación siempre debe empezar a partir del círculo gris. Como ejemplo se puede ver la Ilustración 75D. La aplicación aquí creada comprueba el valor de temperatura del microcontrolador Arduino. Si este es mayor de 25ºC crea un bucle en el que, por cada grado por encima de 25, aumenta la velocidad del motor conectado al Arduino y que maneja un ventilador encontrado en esa habitación.

[211] Se encuentra en lengua inglesa debido a que la implementación ha sido realizada en esta lengua

Por último, en la Ilustración 75E se ve la zona de los formularios. En ella se visualiza el formulario correspondiente al nodo seleccionado en la zona de trabajo. En este caso esta seleccionado el nodo «*if*». En esta zona es donde se ingresan los valores necesarios por cada módulo, como es el dispositivo, el dato, la condición u otros datos, o bien se puede insertar el nombre del módulo para que sea más fácilmente identificable en el editor.

20.2.2.3 SERVICE GENERATION

Como se visualiza en la Ilustración 76, esta segunda capa se compone de dos subprocesos. A esta capa le llega en formato XML el modelo formal que ha sido creado con el MOISL, ya sea el DSL textual o el DSL gráfico. El modelo formal contiene toda la información acerca del flujo que debe seguir el servicio que desea crear el usuario, así como todos los datos que el servicio debe usar.

El procesador procesa todos los nodos de información que contiene este modelo formal, creando un árbol con dicha información. Una vez ya tiene el árbol generado, lo recorre y va generando la aplicación con la información que contiene cada nodo. Los nodos pueden ser condiciones, instrucciones o acciones, cada uno, con su propia configuración para definir lo que realizará. De esta manera, dos nodos del mismo tipo, además de estar situados en diferentes partes del código, pueden realizar dos cosas totalmente diferentes. Tras recorrer todo el árbol y generar la aplicación, en nuestro caso una aplicación Java, la aplicación es compilada y ejecutada en el servidor.

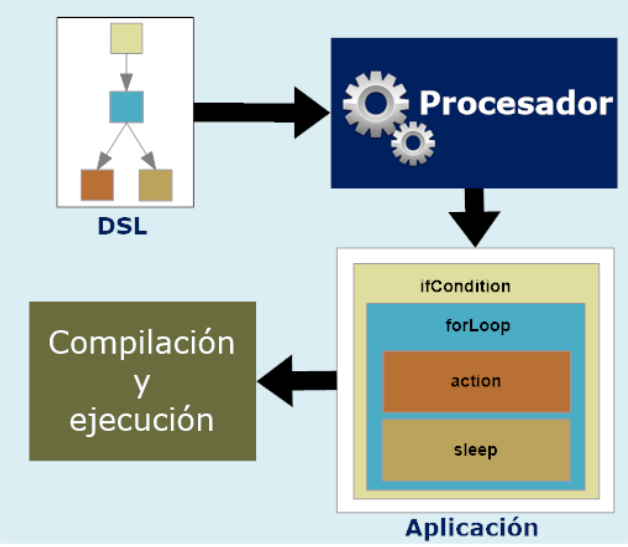


Ilustración 76 Funcionamiento interno del Service Generation

20.2.3 SOFTWARE Y HARDWARE UTILIZADO

A continuación, se muestra el software utilizado para el desarrollo de este prototipo:

- La plataforma IoT Midgar:
  - Ruby 1.9.3
  - Rails 3.1.1
  - Servidor web Webrick.
  - Base de datos SQLite 3.
- Ambos editores se han realizado utilizando HTML5 y JavaScript estándar, sin utilizar ninguna librería externa.
- Para la creación del generador de aplicaciones y la aplicación de conexión con el Arduino se optó por utilizar Java 6. No obstante, se ha mantenido soportando y actualmente se ha compilado bajo Java 8.

Para las pruebas se utilizó el siguiente hardware:

- Servidor alojado en un ordenador dedicado con sistema operativo Windows Server 2008R2 de 64 bits, procesador Intel Core i3-2100 a 3100MHz y 4GB de memoria RAM.
- Cuatro smartphone Android:
  - Nexus 4 con Android 4.2.2.
  - Motorola con la versión 2.2.2.
  - Samsung Galaxy S con la versión 2.1.
  - Samsung Galaxy Mini S5570 con la versión 2.3.6.
- Microcontrolador Arduino Uno SMD basado en el ATmega328.

20.3 *EVALUACIÓN Y DISCUSIÓN*

En esta subsección se explica la metodología seguida para cada prueba: la toma de datos y la encuesta. En ambos apartados se explica hasta el más mínimo detalle todas las decisiones adoptadas para la realización de ambas pruebas.

20.3.1 METODOLOGÍA

En esta subsección se explica la metodología seguida para cada prueba: la toma de datos y la encuesta. En ambos apartados se explica con el más mínimo detalle todas las decisiones adoptadas para la realización de ambas pruebas.

20.3.1.1 TOMA DE DATOS

Para la realización de las pruebas por parte de los usuarios y así medir su tiempo al escribir los DSLs, junto a otros datos, se optó por utilizar las siguientes herramientas:

- Notepad++ con el plugin XML Tools [779]: este editor de texto enriquecido junto con este *plugin*, facilita la creación de XML al ofrecer resaltado de sintaxis, creación de las etiquetas de cierre de un nodo, *identado* automático y comprobación de la correcta escritura del XML realizado. En este editor hay que escribir todo el código, salvo las etiquetas de cierre.
- Editor web textual de Midgar: la primera versión del editor creado para la finalidad de este proyecto fue un editor textual para permitir ayudar en la escritura del DSL textual. Tras comprobar y analizar que sería mejor ofrecer un editor gráfico, este se relegó. Este editor cuenta con la inserción automática del código de cada nodo por medio de un botón. De esta manera, lo único que hace falta es saber dónde hay que insertar cada nodo y rellenar sus atributos.
- Editor web gráfico de Midgar: como se comentó anteriormente, este editor facilita la tarea de creación al usuario al permitir utilizar el DSL gráfico. Lo que debe hacer el usuario es seleccionar la caja correspondiente al flujo que desea añadir al programa y enlazarla al resto mediante el uso de los conectores.

Para la toma de tiempos se utilizó el programa **Mousotron** [780]. Este programa permite tomar el tiempo que lleva activo, número de pulsaciones en el teclado y en cada botón del ratón, incluyendo las dobles pulsaciones, la distancia recorrida con el ratón, las coordenadas (X e Y) del cursor y la velocidad en Km/h. Durante las pruebas se tomó medida de los siguientes factores:

- Tiempo: se midió el tiempo que el participante tardó en realizar la aplicación. Este dato sirve para saber el tiempo que tardó cada participante y averiguar cuál sería la media para compararlo con el tiempo de creación utilizando otras herramientas.
- Pulsaciones de teclas: se contaron las pulsaciones de teclas que hizo el participante. Con este dato se pretende sacar el número de pulsaciones que debe realizar un participante, pues, al equivocarse puede que deba de realizar más, así como al no recibir ayuda de ningún tipo. Esto último da como consecuencia el aumento de la probabilidad de realizar un mayor número de errores.
- Errores: los errores cometidos por el participante durante la prueba. Entre estos se contabilizaban:
    - Equivocación con los anidamientos de nodos: cuando un usuario anidaba incorrectamente un nodo dentro de otro.
    - Olvido de parámetros: cuando se olvidaban de insertar datos e los parámetros de los nodos.
    - Olvido de la etiqueta «*serviceId:*»: a veces se olvidaban de insertar esta etiqueta para pedir el parámetro a un objeto en el nodo «*if*» o «*for*».
    - Inserción de la etiqueta «*serviceId:*» en sitios no requeridos: otras veces insertaban esta etiqueta en el nodo «*action*».
    - Desorientación en la creación de la aplicación: a veces los usuarios se perdían y no sabían dónde debían seguir rellenando o insertando datos debido a la cantidad de texto que se encontraba en su pantalla.

  - Erratas: muchos usuarios cometían erratas a la hora de escribir los nodos o sus atributos, así como utilizaban mayúsculas en vez de minúsculas o se les olvidaba el bloqueo de mayúsculas pulsado.
  - Olvido de datos debido al excesivo texto en pantalla: algunos usuarios, debido al exceso de texto, no se daban cuenta de que les faltaba por rellenar algunos atributos.
- Consultas: a veces algunos usuarios realizaban alguna consulta durante el transcurso de la prueba debido a que no sabían cómo continuar o se les olvidaba algún paso o acción a realizar o bien cometían un error que no sabían solucionar.

20.3.1.2 ENCUESTA

Como método de medición para la encuesta, se utilizó la escala Likert [781]. Se usó esta por ser la medida más utilizada en el diseño de escalas. Se optó por usar el ***5-points Likert Scale*** dando como opciones: 1 como *Totalmente en desacuerdo*, 2 como *En desacuerdo*, 3 como *Neutral*, 4 como *De acuerdo* y 5 como *Totalmente de acuerdo.*

Para la muestra se eligió un tamaño que fuera representativo de la población y que pudiesen otorgar resultados significativos al estudio [782]. Como perfiles se decidió elegir desarrolladores de software (SDev) y usuarios interesados en IoT (IoTU). Se eligió estos dos perfiles en base a la segunda contribución: conseguir una capa de abstracción sobre el DSL que proporcione la facilidad de creación de las aplicaciones a cualquier tipo de usuario, sea o no desarrollador de software.

Para realizar el muestreo de la población se utilizó el método bola de nieve [783]. Para ello, se partió de gente conocida. Estos fueron invitando a otra gente a participar en la prueba. En total fueron 21 participantes. 12 eran SDev y 9 eran IoTU. De esta manera se esperaba poder evaluar el editor y el generador de aplicaciones desde ambos puntos de vista: desarrollador y usuario. Esto se debe a que cada uno aprecia y busca una cosa diferente a pesar de que deban realizar la misma tarea.

Para la realización de las pruebas ambos perfiles debían utilizar primero el Notepad++, después el editor textual y al final el editor gráfico. Durante todo el procedimiento se procedió con los participantes individualmente. Primero se les explicó el escenario del proyecto, sus problemas actuales y la investigación. Después se les explicaba la aplicación que debían realizar sobre un caso real. De esta manera se buscó un mejor entendimiento por parte de los usuarios. En ella debían de comprobar la temperatura del micro-controlador Arduino. Cuando este superase los 25ºC debía de enviar una vibración de 2500 milisegundos al Nexus 4 por cada grado centígrado sobrepasado y una notificación de aviso al Motorola.

Una vez lo comprendían, se les mostraba la aplicación que debían utilizar. Para las tres aplicaciones se les explicó: colocación de botones, significado, distribución y uso. Tras este procedimiento realizaban la aplicación. Una vez acababan, se cambiaban de ordenador y rellenaban, de manera anónima, sin ayuda y en privado el cuestionario, el cual se muestra en la Tabla 6.

Declaración	Descripción
Q1	El usuario entienda la funcionalidad de los elementos y su rol en el proceso de creación de la aplicación.
Q2	Este DSL permite interconectar dispositivos de manera sencilla, utilizando unos pocos clics y sin necesidad de programar código.
Q3	El DSL hace que sea difícil cometer errores mientras el usuario está modelando las aplicaciones.
Q4	Esta solución ofrece una forma rápida de desarrollar la tarea indicada.
Q5	Esta solución provee asistencia para crear aplicaciones para interconectar objetos.
Q6	El DSL no requiere que el usuario utilice conocimientos de programación complejos, como ocurre en el desarrollo de aplicaciones de forma tradicional.
Q7	El DSL incluye suficientes elementos y funcionalidad para que el usuario pueda crear una amplia gama de aplicaciones para interconexión de objetos.
Q8	Esta propuesta es una contribución positiva para fomentar el desarrollo de servicios y aplicaciones que proporcionen interconexión entre objetos.
Q9	Internet de las Cosas y los *Smart Objects* se beneficiarán de esta solución.
Q10	Este DSL podría ser usado para simplificar el proceso de desarrollo clásico de aplicaciones software en otras áreas.

Tabla 6 Cuestionario realizado a los usuarios en MOISL

Para la realización del cuestionario se buscaron un total de diez declaraciones. En ellas se preguntaba acerca de la opinión sobre el uso del DSL, sus posibilidades y su posible impacto en Internet de las Cosas y los *Smart Objects*.

Para la evaluación de los resultados, se optó por mostrar las respuestas de los usuarios por separado, pero evaluarlas en conjunto. Esto se debe a que interesaba evaluar desde el punto de vista de ambos perfiles. No obstante, lo que se busca es ofrecer la misma herramienta y funcionalidad a ambos, por eso se evalúa en conjunto. Por ello, esta es la primera evaluación que hacemos. A continuación, se hacen la de ambos perfiles por separado.

Los datos obtenidos para la evaluación son variables estadísticas cuantitativas, ya que se expresan mediante valores numéricos. Además, estas son continuas ya que se sitúan dentro de un rango de valores determinado por la escala Likert de cinco puntos (1 a 5). Para evaluar los datos recogidos se optó por usar las siguientes medidas:

- Medidas de centralización: con un único valor se identifica la tendencia central de la variable.
    - **Moda**: es el valor de la variable que aparece con más frecuencia en la muestra. Puede a ver más de una moda. Con este valor se puede obtener cuál es la respuesta más elegida por los usuarios respecto a una declaración.
    - **Mediana**: es el valor central de un conjunto de datos ordenados, dejando así el mismo número de elementos tanto a izquierda como a derecha de él. Con esta medida podemos averiguar cuál es la respuesta de la población que se sitúa en medio de todas. Esto nos da una visión general de hacía donde apuntan las respuestas.
- Medidas de posición: determinan la posición que ocupa un individuo dentro de los valores que toma la variable.
    - **Cuartiles**: estos son los tres valores que dividen un conjunto de datos ordenado en cuatro partes iguales. Los cuartiles nos sirven para poder valorar las respuestas de los

usuarios divididas en cuatro porciones. Esto nos proporciona una mejor información respecto a la diferencia de opiniones entre usuarios.

  - **Cuartil 1**: se corresponde con la mediana de la primera mitad de los valores. Este dato ayudará a comparar la opinión de la primera mitad de valores.
  - **Cuartil 2**: se corresponde con la mediana de todas las respuestas obtenidas.
  - **Cuartil 3**: se corresponde con la mediana de la segunda mitad de los valores. Este dato ayudará a comparar la opinión de la segunda mitad de valores.

- Medidas de dispersión: miden el grado de variabilidad de los datos muestrales y se usan para completar la información que dan las medidas de centralización, midiendo la representatividad de los valores centrales.
  - **Rango**: es la diferencia entre los valores de los extremos. Con el rango podemos saber cuánta diferencia hay entre las respuestas extremas. Esto nos indicará si hay mucha diferencia en la opinión de los usuarios.
  - **Rango intercuartílico (RIC)**: es la diferencia entre el primer y tercer cuartil. Con este dato podremos saber la diferencia entre las medianas de ambos grupos de cuartiles.
- Otras:
  - **Mínimo**: número menor de los datos. Con este dato podremos saber cuál fue la calificación más baja dada por un participante a una declaración.
  - **Máximo**: número mayor de los datos. Con este dato podremos saber cuál fue la calificación más alta dada por un participante a una declaración.

Para la visualización de los datos se optó por utilizar un diagrama de cajas y bigotes que ayuda a visualizar los datos antes descritos y así poder comparar todas las declaraciones a la vez. La Ilustración 77 es un ejemplo donde se ve dicho diagrama y muestra su correspondencia con las medidas que se utilizarán y que han sido utilizadas anteriormente.

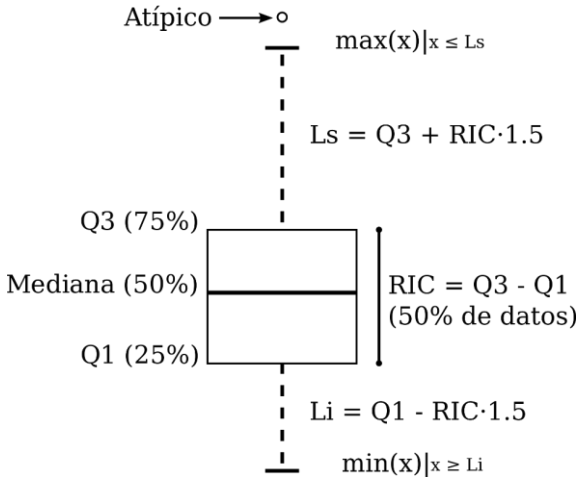


Ilustración 77 Ejemplo de un diagrama de cajas y bigotes [212]

[212] https://commons.wikimedia.org/wiki/File:Boxplot.svg

20.3.2 RESULTADOS DE LA TOMA DE TIEMPOS

En la Tabla 7, Tabla 8 y Tabla 9 se muestran los datos recogidos a todos los participantes durante la realización de las pruebas. Estos se corresponden con la creación de la aplicación requerida con el Notepad++ con el Plugin XML Tools, el editor del DSL textual y el editor del DSL gráfico, respectivamente. En cada tabla se muestra el identificador del participante, el tiempo en segundos que tardó en hacer la aplicación, las pulsaciones de teclado que hizo, los errores cometidos y el número de consultas que hizo al supervisor durante la prueba.

ID	Tiempo (seg.)	Pulsaciones de teclado	Errores	Consultas
p1	627	722	8	2
p2	403	468	5	0
p3	429	649	5	1
p4	245	494	3	0
p5	270	437	3	0
p6	555	449	2	1
p7	461	573	4	2
p8	281	504	1	0
p9	625	564	5	2
p10	673	489	7	0
p11	600	501	6	0
p12	360	582	6	0
p13	803	584	11	3
p14	344	443	5	1
p15	235	495	4	1
p16	857	796	5	0
p17	344	457	3	0
p18	454	478	4	1
p19	280	554	8	0

Tabla 7 Toma de datos en la realización de la aplicación usando el DSL textual MOISL con el Notepad++

ID	Tiempo (seg.)	Pulsaciones de teclado	Errores	Consultas
p1	267	250	1	1
p2	197	132	2	0
p3	263	191	5	1
p4	152	209	0	1
p5	144	161	2	0
p6	195	188	2	0
p7	186	173	2	1
p8	145	159	1	0
p9	297	317	1	0
p10	242	93	2	0
p11	309	129	3	0
p12	185	163	2	1
p13	280	86	5	1
p14	159	139	1	0
p15	134	208	2	0
p16	231	248	0	0
p17	147	98	2	0
p18	145	123	4	0
p19	113	113	2	0

Tabla 8 Toma de datos en la realización de la aplicación con el DSL textual MOISL con el editor

ID	Tiempo (seg.)	Pulsaciones de teclado	Errores	Consultas
p1	305	30	1	0
p2	165	24	0	0
p3	177	24	1	1
p4	117	24	0	0
p5	131	26	2	0
p6	185	28	1	1
p7	155	32	0	1
p8	99	22	1	0
p9	169	24	0	0
p10	225	20	0	0
p11	203	20	1	0
p12	142	23	1	0
p13	264	26	2	1
p14	127	26	1	1
p15	82	22	0	0
p16	390	40	0	0
p17	131	21	0	0
p18	141	27	1	0
p19	93	28	1	0
p20	110	26	0	0
p21	192	29	0	0

Tabla 9 Toma de datos en la realización de la aplicación usando el DSL gráfico MOISL con el editor

20.3.2.1 COMPARATIVA DE TIEMPO

En la Ilustración 78 se puede ver la gráfica que compara el tiempo medio que los participantes han tardado en crear la aplicación pedida en cada editor.

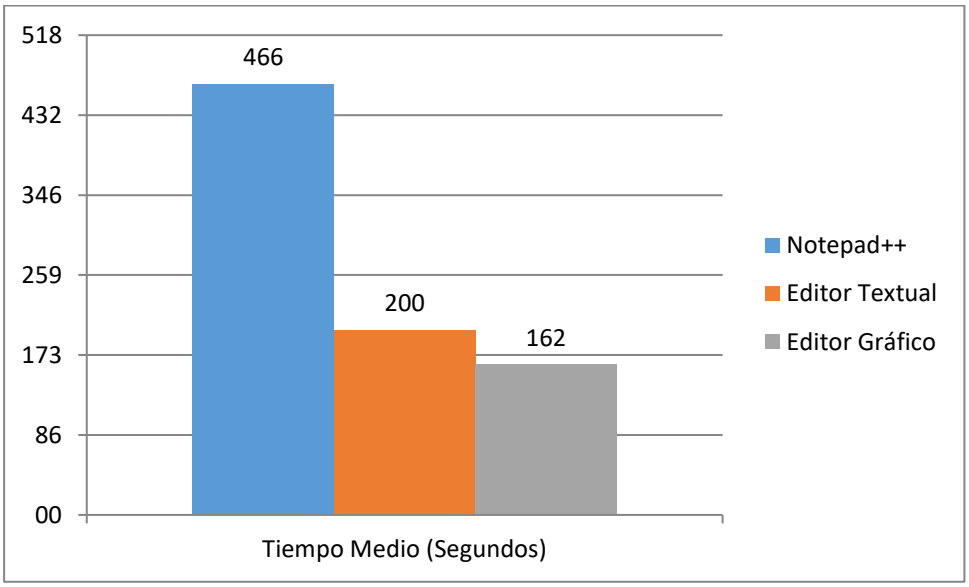


Ilustración 78 Comparativa de tiempos medio de creación de las aplicaciones de MOISL

Se puede observar como con el DSL textual en el Notepad++ tardaron una media de 466 segundos, lo equivalente a poco más de 8 minutos. No obstante, con el DSL textual en el editor textual de Midgar tardaron tres minutos y veinte segundos. Casi cinco minutos menos. Esto indica una gran disminución, pues es más de la mitad del tiempo. Con el DSL gráfico en el editor gráfico tardaron 162 segundos. Una pequeña disminución, exactamente de 37 segundos respecto al DSL textual. De esto se puede sacar que los participantes tardaron menos con este último DSL gráfico. Sin embargo, la diferencia entre ambos editores de la plataforma no es tan grande como para decantarse por uno, como si ocurre con el uso del Notepad++ que los supera en más del doble de tiempo.

Estos números se deben, claramente, a la diferencia existente entre tener que escribir casi toda la aplicación por medio del uso del DSL, como sucede con el Notepad++, a sólo tener que escribir los atributos, que es lo que sucede en los dos editores propios. Por ello, tanto el DSL textual y el DSL gráfico tienen prácticamente el mismo tiempo, resultando casi irrelevante la diferencia de tiempo medo entre ambos, pues sólo es de 37 segundos. Lo que no es irrelevante es la mejora sustancial en tiempo que ofrecen estos dos editores frente a tener que escribir toda la aplicación.

20.3.2.2 COMPARATIVA DE PULSACIONES DE TECLADO

En la Ilustración 79 se puede ver la gráfica que compara el número medio de pulsaciones de teclas necesitadas por los participantes al crear la aplicación pedida en cada editor.

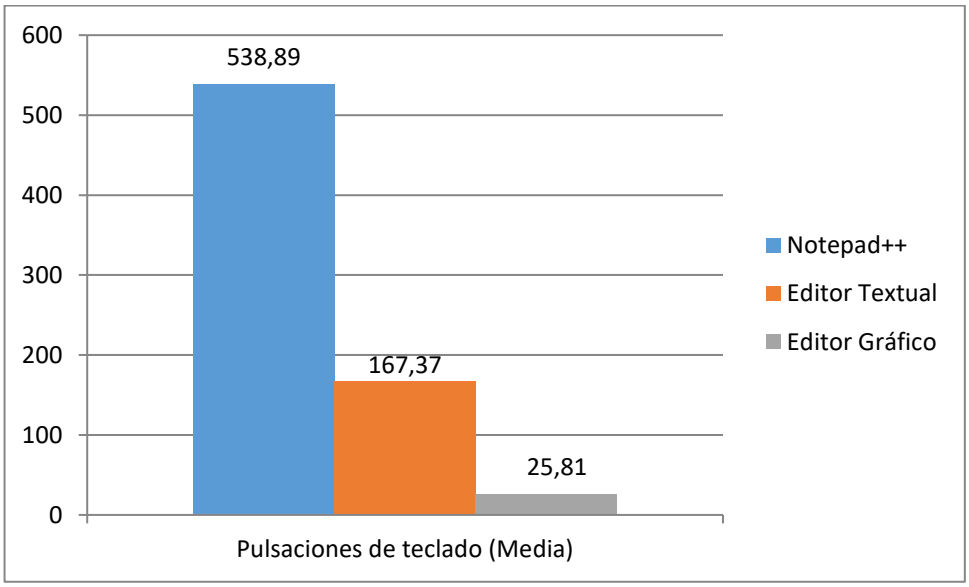


Ilustración 79 Comparativa con las pulsaciones de teclado medias en la creación de las aplicaciones de MOISL

Con el Notepad++ los participantes necesitaron una media de 538,89 pulsaciones. Esto es debido a que debían de escribir todo el código salvo las etiquetas de cierre. Este número de pulsaciones medias se ve drásticamente reducido en 3,2 veces con el uso del editor de texto y, este último, en 6,4 veces por el editor gráfico. Igual que en el caso del tiempo, el número de teclas pulsadas se ve claramente influenciado por la ayuda que ofrece cada editor: en el caso del Notepad++ solo ayuda con la creación de las etiquetas de cierre; el editor de texto crea el nodo con sus atributos y lo que debe escribir el participante son los datos que van dentro de cada atributo; en el editor gráfico, el participante sólo debe escribir algunos atributos, pues otros, simplemente seleccionándolos con el ratón son rellenados automáticamente por el propio editor.

La interpretación que se puede obtener es la ayuda ofrecida por cada editor a la hora de escribir. Esta diferencia puede repercutir en una disminución en el número de errores al conseguir evitar que el participante deba teclear partes del lenguaje que pueden insertarse automáticamente.

20.3.2.3 COMPARATIVA DE ERRORES

En la Ilustración 80 se puede ver la gráfica que compara el número medio de errores cometidos por los participantes al crear la aplicación pedida en cada editor.

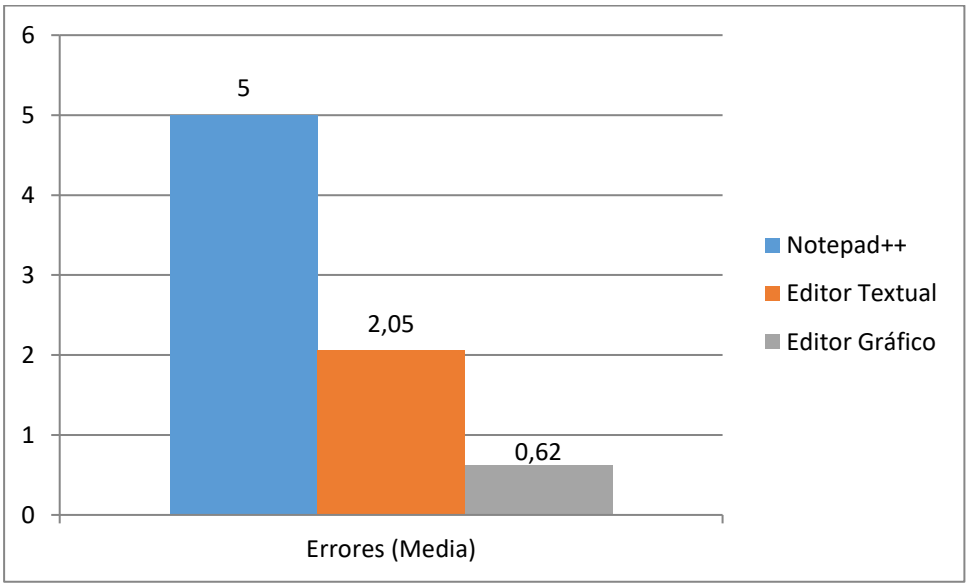


Ilustración 80 Comparativa con el número de errores medios en la creación de las aplicaciones de MOISL

Cuando los participantes utilizaban el Notepad++ cometieron una media de 5 errores por participante. Con el editor textual se redujo la media de los errores en más de la mitad, hasta 2,05. Estos, se reducen algo más de tres veces usando el editor gráfico. Se consigue reducir la media hasta menos de un fallo por persona, exactamente, a un 0,62.

Analizando la gráfica se puede ver como mediante el uso del editor gráfico hay una considerable disminución de los errores respecto al editor de texto, y de este, sobre el Notepad++. Esto va ligado al número de pulsaciones de teclado que deben realizar, pues los errores analizados tenían que ver con equivocaciones que cometían al introducir los nodos, sus atributos o los valores. En este caso, mediante el uso del editor gráfico y la ayuda que este les ofrece, se ve cómo repercute en una disminución del número de errores.

20.3.2.4 COMPARATIVA DE CONSULTAS

En la Ilustración 81 se puede ver la gráfica que compara el número medio de consultas realizadas por los participantes al crear la aplicación pedida en cada editor.

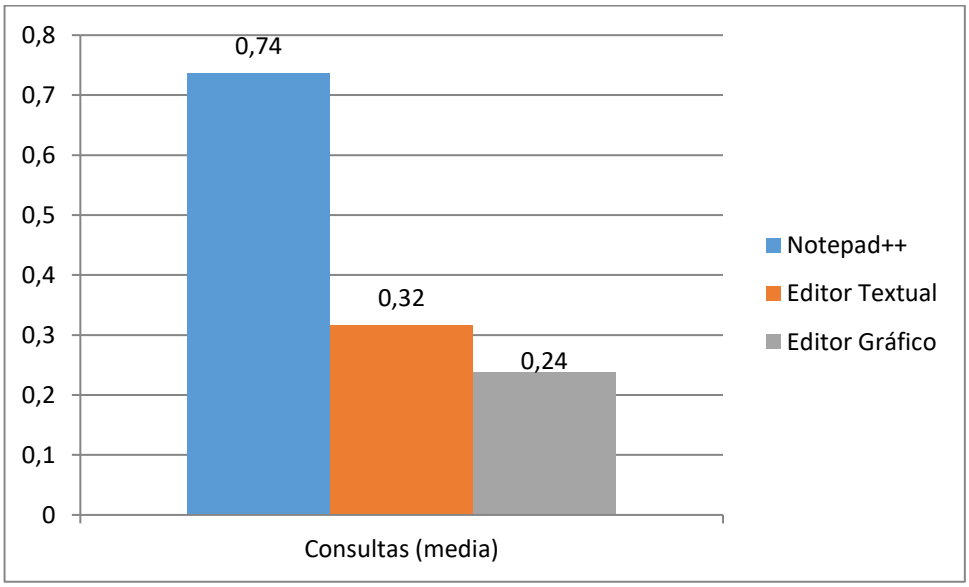


Ilustración 81 Comparativa con el número de consultas medias de los participantes en la creación de las aplicaciones de MOISL

Con el editor gráfico fue con el que menos consultas, un 0,24 de media por persona. No obstante, con el editor textual realizaron pocas más, un 0.32. Estas están muy por debajo de las realizadas con el Notepad++ que suben a algo más del doble, 0,74 consultas por persona.

Al igual que en el caso de los errores, las consultas de los participantes se redujeron en más de la mitad con el uso de los dos editores. Sobre todo, en el aspecto de poder seguir correctamente el flujo de creación de la aplicación y no perderse a la hora de crearlo. Como se observa, la diferencia entre el editor de texto y el gráfico es inapreciable, luego, no se puede considerar como un aspecto para destacar uno sobre el otro, pero sí para probar que el uso de un editor que ofrezca más ayuda mejora sustancialmente la creación de la aplicación.

20.3.3 RESULTADOS DE LA ENCUESTA

En la Tabla 10 se pueden ver las respuestas enviadas por cada participante de forma anónima. Esta tabla contiene tanto las respuestas de los participantes que son desarrolladores de software (SDev) como los participantes interesados en *Internet of Things* (IoTU).

ID	Perfil	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
p01	SDev	3	4	4	4	4	4	3	3	3	4
p02		5	5	5	5	5	5	5	5	5	4
p03		4	5	4	5	4	5	3	3	4	3
P04		3	5	5	4	4	4	3	4	5	4
p05		2	3	5	4	4	4	3	4	4	4
p06		4	4	5	4	3	3	3	5	4	3
p07		3	4	5	5	4	5	4	4	5	5
p08		4	4	5	5	5	5	3	5	4	5
p09		4	5	5	5	5	5	5	3	5	4
p10		4	5	4	5	5	5	4	5	5	5
p11		5	4	5	4	4	4	3	5	4	5
p12		3	3	4	3	4	4	2	3	3	3
p13	IoTU	4	4	5	4	5	5	4	4	5	5
p14		4	4	5	5	4	4	4	4	4	4
p15		5	4	5	5	5	5	5	5	4	5
p16		3	3	4	4	4	5	3	5	4	5
p17		5	5	5	4	5	5	5	4	4	5
p18		4	4	5	5	4	5	2	2	3	4
p19		4	5	5	5	5	5	5	4	5	5
p20		5	5	5	5	5	4	5	5	5	4
p21		5	5	4	5	5	5	5	5	5	4

Tabla 10 Respuestas globales de los participantes en cada declaración de MOISL

Finalmente, se evaluaron los resultados globales, es decir, las respuestas de los desarrolladores de software y de los participantes interesados en Internet de las Cosas en conjunto para así obtener la opinión que buscamos. En la Tabla 11 se presentan las estadísticas descriptivas de todo el conjunto. En ella se puede ver, desglosado para cada declaración, el mínimo, el primer cuartil, la mediana o segundo cuartil, el tercer cuartil, el máximo, el rango, el rango entre cuartiles y la moda. En la Ilustración 82 se pueden ver estos mismos datos en un diagrama de cajas y bigotes.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Mínimo	2	3	4	3	3	3	2	2	3	3
Cuartil 1	3	4	4	4	4	4	3	4	4	4
Mediana	4	4	5	5	4	5	4	4	4	4
Cuartil 3	5	5	5	5	5	5	5	5	5	5
Máximo	5	5	5	5	5	5	5	5	5	5
Rango	3	2	1	2	2	2	3	3	2	2
Rango Intercuartílico	2	1	1	1	1	1	2	1	1	1
Moda	4	4	5	5	4	5	3	5	5	4

Tabla 11 Tabla con las estadísticas descriptivas globales de MOISL

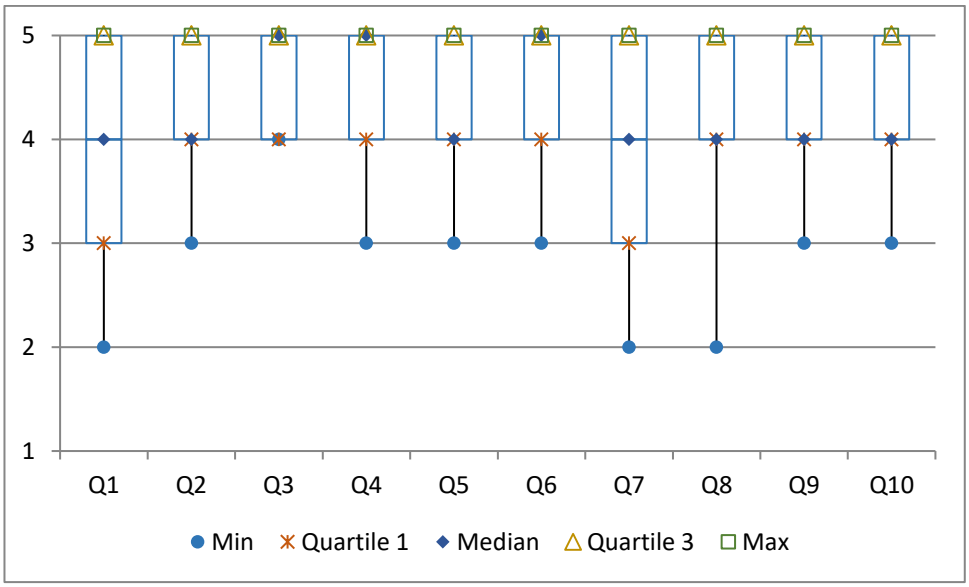


Ilustración 82 Diagrama de cajas y bigotes global para cada declaración de MOISL

Analizando la Tabla 11 y la Ilustración 82 se pueden obtener las siguientes interpretaciones:

- Q3 tiene el mínimo más alto, en este caso, 4 de 5. Esto significa que todos los participantes están de acuerdo con esta declaración por lo menos.
- Q3, Q4 y Q6 son las declaraciones con mayor mediana. De esto podemos deducir que la mayoría de los participantes están de acuerdo con estas declaraciones.
- Q3 es la única declaración con un rango de 1. Esto significa que todos los participantes tienen la misma opinión en esta declaración. Por otro lado, Q1, Q7 y Q8 tienen el mayor rango, lo que indica que hay mucha diferencia entre las respuestas de cada participante.
- De acuerdo con la moda se puede observar Q7 tiene una moda de 3. Esto muestra que la declaración fue elegida por los participantes como «*Neutral*». Por otro lado, Q3, Q4, Q6, Q8 y Q9 tienen una moda de 5, lo cual indica que la mayoría de las respuestas elegidas han sido «*Totalmente de acuerdo*».
- Mirando Q1, Q7 y Q8, las cuales tienen un rango de 3, un mínimo de 2, una mediana de 4 y un máximo de 5, vemos que son las declaraciones con valoraciones más bajas. No obstante, Q8 está bien valorada pues tiene cuartil de 4, un rango intercuartílico de 1 y una moda de 5.

En la Tabla 12 se puede ver la frecuencia de las respuestas para cada declaración. En esta se puede ver desglosado para cada declaración el número de votos de cada decisión elegida y su porcentaje correspondiente para ambos conjuntos. En la Ilustración 83 se muestra en un gráfico de barras la frecuencia de respuestas en el conjunto formado por ambos perfiles. En la Ilustración 84 se muestran las respuestas de estos mediante un gráfico de barras apilado marcando los percentiles.

Declaración		Totalmente en desacuerdo	En desacuerdo	Neutral	De acuerdo	Totalmente de acuerdo
Q1	#	0	1	5	9	6
	%	0%	5%	24%	43%	29%
Q2	#	0	0	3	9	9
	%	0%	0%	14%	43%	43%
Q3	#	0	0	0	6	15
	%	0%	0%	0%	29%	71%
Q4	#	0	0	1	8	12
	%	0%	0%	5%	38%	57%
Q5	#	0	0	1	10	10
	%	0%	0%	5%	48%	48%
Q6	#	0	0	1	7	13
	%	0%	0%	5%	33%	62%
Q7	#	0	2	8	4	7
	%	0%	10%	38%	19%	33%
Q8	#	0	1	4	7	9
	%	0%	5%	19%	33%	43%
Q9	#	0	0	3	9	9
	%	0%	0%	14%	43%	43%
Q10	#	0	0	3	9	9
	%	0%	0%	14%	43%	43%

Tabla 12 Tabla de frecuencias de las respuestas globales de MOISL

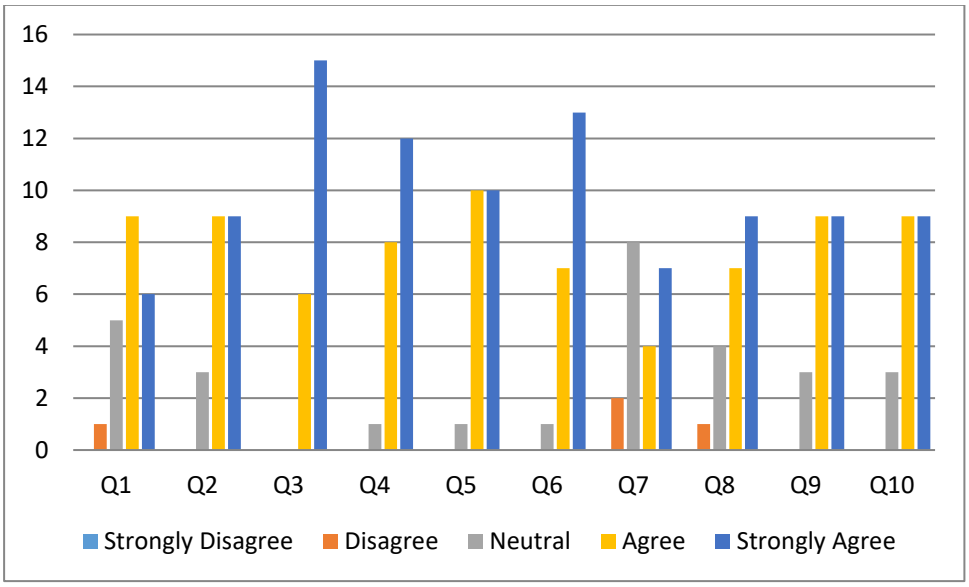


Ilustración 83 Distribución de las respuestas globales de MOISL

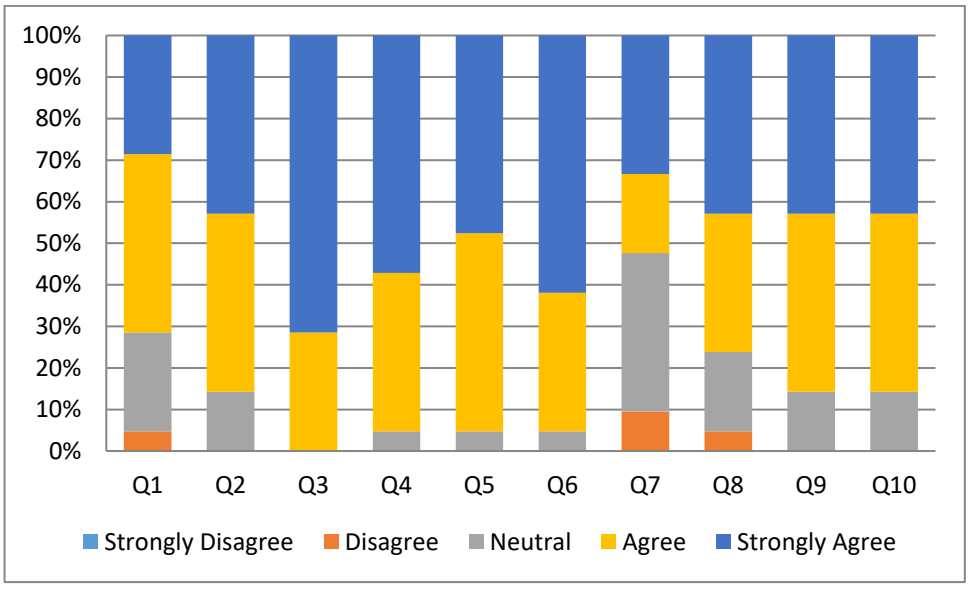


Ilustración 84 Distribución apilada de las respuestas globales de MOISL

A partir de la Tabla 12, Ilustración 83 y la Ilustración 84 se pueden sacar interpretaciones que no se pudieron anteriormente. Estas son las siguientes interpretaciones:

- La Q5 posee un 48% de votación como «*De acuerdo*» y «*Totalmente de acuerdo*», mientras que solo el 4% de los participantes votaron «*Neutral*». En números esto se traduce como 10, 10 y 1 voto respectivamente. Esto indica que los participantes están *De acuerdo* como mínimo, a excepción de una pequeña minoría, con esta declaración.
- Las declaraciones Q2, Q9 y Q10 tienen un 43% de votos en «*De acuerdo*» y «*Totalmente de acuerdo*» y un 14% en «*Neutral*». La correspondencia en números es, respectivamente, 9, 9 y 3. Esto indica que la mayoría está «*De acuerdo*» pero hay bastante porcentaje que está indecisa o no cree en estas declaraciones.

20.4 *CONCLUSIONES*

En esta segunda iteración se ha presentado la primera ampliación para Midgar., MOISL, un DSL textual y otro gráfico. Esta primera ampliación permite a usuarios no técnicos modelar y generar aplicaciones software que interconecten *Smart Objects*. Tradicionalmente, el desarrollo de aplicaciones software que tiene como objetivo conectar heterogéneos objetos es un proceso relativamente complejo, ya que en la mayoría de los casos se requiere de conocimientos de programación sólidos, así como conocimientos sobre diferentes tecnologías.

En esta iteración, que amplía la plataforma IoT Midgar, se desarrolla una solución centralizada en varios niveles que permite el modelado y generación de aplicaciones software específicas diseñadas especialmente para la interconexión de objetos. Uno de los aspectos clave de la solución es que aumenta el nivel de abstracción en el proceso de desarrollo de aplicaciones utilizando un editor de gráfico que facilita la generación del DSL gráfico, el cual fue diseñado para este propósito específico. Usando MOISL hemos

conseguido abstraer el problema de tener que desarrollar una aplicación en un GPL sin necesidad de conocimientos de programación. De esta forma, queda validada la hipótesis de esta iteración.

Para validar la efectividad del DSL para interconectar objetos por gente, tanto desarrolladora como no desarrolladora de software y siendo un total de 21 personas, se hizo una evaluación que consistía en desarrollar una aplicación usando el DSL textual en el Notepad++ y en un editor propio, y el DSL gráfico. El DSL gráfico fue el más rápido de todos, obteniendo un tiempo medio de desarrollo de la aplicación 162,19 segundos.

El proceso de evaluación incluía también una encuesta compuesta de 10 declaraciones que debían de valorar los participantes, con el fin de evaluar aspectos subjetivos de la propuesta. Los aspectos tratados en esta encuestan estaban relacionados con la experiencia del participante durante el uso de la solución propuesta, permitiendo así saber si la propuesta permite el modelado de las interconexiones de una forma comprensible y ágil. El 90% de las declaraciones han obteniendo más del 75% de evaluaciones positivas o muy positivas. Incluso la declaración de menor calificación ha obtenido un 52,38% de evaluaciones positivas o muy positivas.

Cabe destacar que la mayoría de las evaluaciones no positivas de los participantes fueron motivadas por las limitadas características de desarrollo que la plataforma estaba usando para modelar aplicaciones, pues están muy lejos de permitir modelar cualquier aplicación imaginable. En futuras ampliaciones se va a dar énfasis a nuevos elementos de modelado para extender las posibilidades de desarrollo de aplicaciones, la adición de nuevos mecanismos para interconectar nodos, y la posibilidad de hacer tareas más complejas y avanzadas. En cualquier caso, estas ampliaciones deben respetar la naturaleza de la plataforma y evitar la adición excesiva de complejidad al proceso de modelado.

En función de los resultados de la evaluación, se puede decir que tanto el DSL textual como el gráfico, pero sobre todo el gráfico, propuestos y que hacen uso de la plataforma Midgar pueden ser muy utilizados para reducir la complejidad de interconectar objetos, que es algo que permite a usuarios no técnicos esta posibilidad.

21. GENERACIÓN DE *SMART OBJECTS* PARA INTERNET DE LAS COSAS MEDIANTE EL USO DE INGENIERÍA DIRIGIDA POR MODELOS

«Ha sido uno de mis mantras – concentración y simplicidad. Lo simple puede ser más difícil que lo complejo: tienes que trabajar duro manteniendo tu mente clara para hacer las cosas simples. Pero vale la pena llegar hasta el final porque una vez allí, puedes mover montañas»
Steve Jobs

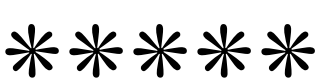
«En igualdad de condiciones, la explicación más sencilla suele ser la más probable»
Guillermo de Ockam, Navaja de Ockam

«La simplicidad es un prerrequisito para la fiabilidad»
Edsger W. Dijkstra

La comunicación entre los *Smart Objects* a través de Internet es uno de los objetivos de Internet de las Cosas, pues estamos rodeados de ellos Estos objetos nos proveen de beneficios en nuestra vida diaria, de recomendaciones y nos ayudan cuando viajamos o bien mejoran los procesos industriales mediante la automatización de ciertas tareas. Sin embargo, el crear estos objetos necesita de tiempo y conocimientos. Luego, se necesita, o bien comprar un objeto o bien crearlo. El problema aquí es que no todas las personas tienen el conocimiento, el tiempo necesario para desarrollar la aplicación necesaria, o bien el dinero para pagar estas mejoras.

Ya se ha visto que Midgar, con MOISL, permite realizar la interconexión entre objetos ya registrados y existentes. Pero, ahora bien, ¿Quién crea estos objetos? ¿Cómo se crean? Este es uno de los problemas aún existentes. Existen aplicaciones que ayudan en esta tarea, pero no te permiten trabajar con sensores, enviar datos a través de la red o interconectar el dispositivo a otro. Por esto, se hace necesario proponer algo para mejorar esta parte y cerrar así el ciclo de vida de los objetos, permitiendo no solo su interconexión, sino también su creación.

Para solucionar esto se ha desarrollado un DSL que permite a la gente sin conocimientos en el campo del desarrollo software crear sus propios *Smart Objects*, especificando los componentes que desean utilizar entre diferentes sensores y actuadores. De esta manera, se les facilitará el desarrollo de *Smart Objects* respecto al desarrollo tradicional de estos.



21.1 *DESCRIPCIÓN*

21.1.1 **OBJETIVOS**

Esta tercera iteración se basó en las siguientes dos hipótesis:

- Aplicar MDE para obtener una abstracción y crear así un DSL para facilitar la creación de objetos de una manera fácil y rápida para gente que sepa crear un objeto, pero no tenga los conocimientos de programación para crear el software.
- La creación de software para objetos es una mejora significante para IoT.

Así, el objetivo principal de esta tercera iteración ha sido el facilitar el desarrollo de *Smart Objects* para ser usados en el marco de Internet de las Cosas de una forma rápida y sencilla a personas que no tengan conocimientos de desarrollo. Para ello se han debido de cumplir una serie de objetivos:

- Desarrollar un Lenguaje de Dominio Específico para la definición del uso del *Smart Object*.
- Obtener una alta abstracción para la creación de aplicaciones mediante un DSL web gráfico.
- Encapsulación de todo el software necesario para hacer funcionar el *Smart Object* en su plataforma destino, incluyendo el software necesario para interactuar con múltiples sensores y actuadores compatibles y el sistema de mensajes de la plataforma Midgar.
- Conseguir saber si el ofrecer un sistema que ayude a crear objetos podría ser una mejora significante en IoT.

21.1.2 **FUNCIONALIDADES**

Para cumplir con los objetivos anteriormente explicados se ha utilizado la plataforma Midgar. En esta tercera iteración, se ha implementado a Midgar un segundo DSL gráfico, llamado **Midgar Object Creation Specific Language** (MOCSL), que ha abstraído de todo lo necesario para facilitar la creación de la lógica necesaria para crear un *Smart Object* de una forma rápida y sencilla. Exactamente, se han creado dos prototipos, el que permite crear *Smart Objects* con un dispositivo Android y el del Arduino Uno.

El uso de MOCSL se basa en la selección de dispositivos y componentes y se encarga de generar el modelo forma persistido en formato XML, para así enviarlo a la capa 2 de Midgar, donde se encuentra en el procesador, que es el encargado de transformar el modelo formal en la aplicación final. Así, un usuario que utiliza MOCSL obtiene directamente la aplicación final.

De esta manera, MOCSL abstrae del hardware y software del *Smart Object* a generar, así como de la propia programación interna que necesita para interactuar con sus propios sensores o actuadores. Sobre todo, respecto al Arduino Uno, donde abstrae más si cabe, pues este posee infinidad de sensores y actuadores posibles y, a pesar de ser casi imposible dar soporte para todos, facilita en gran medida el software de los soportados. Esto se debe a que la programación necesaria para trabajar con este tipo de sensores es aún a más bajo nivel que en un smartphone.

Por esto, con MOCSL, un usuario puede especificar que *Smart Object* desea crear y seleccionar los actuadores y sensores que desea utilizar, en caso de ser un smartphone, o bien seleccionar los sensores y actuadores que ha conectado a su microcontrolador Arduino. Tras esto, solo debe de generar la aplicación e introducirla en su *Smart Object* para que este comience a interactuar según lo definido. Una vez introdujo la aplicación, el usuario puede registrar el dispositivo en Midgar y utilizar MOISL para definir la interacción entre diferentes objetos de los que el usuario disponga.

Es decir, esta tercera iteración es la parte restante para completar la segunda iteración, pues mientras que la primera permitía interconectar objetos ya existentes, en esta tercera iteración se permite definir estos objetos para así poder interconectarlos sin necesidad de programar el objeto.

21.2 *HERRAMIENTAS PARA GENERAR SOFTWARE PARA SMART OBJECTS*

Actualmente existen diferentes aplicaciones o herramientas que permiten generar software para diferentes smartphones o para diversos microcontroladores basados en Arduino. No obstante, para hacer la interconexión entre objetos heterogéneos se necesita programar, lo que es una ardua tarea debido a su complejidad y dificultad [784]. Este es el caso de algunas herramientas que permiten crear programas para estos dispositivos, pero trabajar solamente con datos de forma interna, impidiendo sacar los datos al exterior, ya sea a una plataforma web o a otro objeto. Luego, a estas herramientas les falta la parte de poder sacar los datos al exterior.

En esta iteración, se propone una solución para permitir crear *Smart Objects*, al igual que hacen las soluciones se presentarán en este subcapítulo, pero con la diferencia de que, esta solución, basada en un DSL llamado MOCSL, si permite el envío y el trato de parámetros del exterior. La interconexión entre objetos no se trata debido a que esta ya ha sido realizada utilizando MOISL, presentado en el capítulo anterior.

21.2.1 SOLUCIONES PARA SMARTPHONES

En este caso se han estudiado varios editores web gráficos que permiten crear software para diferentes smartphones. Editores que proporcionan la opción de desarrollar solo para sistemas operativos Android son AppsGeyser [785], iBuildApp [786] y Andromo [787]. Editores que ofrecen soporte para Android, iOS y HTML son AppsBuilder [788] y Infinite Monkeys [789]. Por otro lado, existe un editor con soporte para Blackberry y Windows Phone como es Appypie [790]. En cambio, para generar aplicaciones para Android y iOS está Como [791], o en el caso de que también queramos tener soporte para Windows Phone se podría utilizar AppMachine [792].

Como se ha visto, hay muchos editores con soporte a diferentes sistemas operativos móviles. Además, cada editor ofrece diferentes características a pesar de que soporte el mismo sistema operativo móvil que otro. Por eso, a continuación, se hablará de los que se ha pensado que son los editores más completos y relevantes.

AppsGeyser [785] permite seleccionar el tipo de aplicación que se desee entre una lista de aplicaciones predeterminadas: sitio web, buscador, página o diferentes tipos de juegos, entre otros. Dependiendo de la selección realizada, la aplicación presenta diferentes opciones para personalizarla.

Ilustración 85 Imagotipo de AppsGeyser

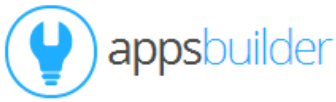


Ilustración 86 Imagotipo de AppsBuilder

En el caso de AppsBuilder [788], los usuarios pueden seleccionar varias características que deseen en su aplicación, como por ejemplo son redes sociales, sonido, video, noticias, imágenes, texto y mucho más. Después de este paso, el usuario puede seleccionar el tema gráfico de la aplicación entre 83 disponibles. Una vez el usuario ha realizado estas decisiones y ha seleccionado las características de la aplicación, AppsBuilder le ofrece opciones para poder personalizarlas. También permite personalizar la selección de icono, el estado, el estilo de cabecera, el menú de navegación y su estilo, el *look & feel* y el menú de opciones del usuario. Durante este proceso, el usuario puede revisar toda la aplicación en el sitio web o bien probarla en un iPhone, iPad o Android físicos que el posea.

El primer paso al usar Infinite Monkey [789] es introducir el nombre de la aplicación que se desea desarrollar y los datos personales del usuario. Tras esto, el usuario puede elegir empezar la aplicación entre cuatro diferentes plantillas que te ofrecen o hacer una aplicación vacía. Si se opta por la última opción, el usuario pueda arrastrar y soltar diferentes funcionalidades que desee en su aplicación, como es un calendario, un blog, redes sociales o muchas otras opciones. Durante este proceso, se puede ver la vista previa de cómo va quedando la aplicación.

Ilustración 87 Imagotipo de Infinite Monkeys

Como se ha visto, AppsGeyser, AppsBuilder y Infinite Monkeys son buenas plataformas y ofrecen gran cantidad de opciones para crear aplicaciones para los smartphones, no obstante, no soportan el uso de sensores, solamente el del sistema de posicionamiento global (GPS), ni el envío ni la recepción de datos para realizar acciones de acuerdo a estos datos. Por estas razones, estos editores no sirven para crear aplicaciones para IoT.

21.2.2 BITBLOQ

BitBloq [328] es un editor web gráfico de la compañía BQ que permite crear aplicaciones para dispositivos electrónicos como son los microcontroladores basados en Arduino. Este editor se basa en la combinación de piezas de puzle para crear el ciclo de vida de la programación.

Ilustración 88 Imagotipo de BitBloq

En la esquina superior izquierda se encuentra el menú de instrucciones que permite insertar piezas que son transformadas en código fuente posteriormente. El ciclo de vida se define a partir de la pieza que se encuentre más arriba y a la izquierda del puzle montado, pues toma esta como la pieza inicial.

Las piezas tienen diferentes pestañas y espacios en blanco según el tipo de pieza que sea. Por ejemplo, las piezas de sensores tienen el espacio en blanco del pin a la derecha, el cual sirve para enlazar una pieza del tipo pin y así indicarle en que pin irá conectado ese sensor en el microcontrolador. Esta pieza también tiene un conector a la izquierda para así poder conectar a una pieza que se corresponde con la salida de datos, la cual puede ser por USB o Bluetooth. En cambio, otras piezas, como son los actuadores, tienen en vez del conector a la izquierda un conector arriba, por si se desea colocar esta pieza debajo de otra, y otro conector abajo, para poder colocar más piezas debajo de esta y así continuar la secuencia.

Este editor gráfico web tiene piezas que se corresponden con sensores y actuadores, otras piezas que sirven para crear el flujo como del programa, como son el condicional «*if*» o el bucle «*while*», piezas lógicas como «*true*» y «*false*», otras piezas para insertar números, matrices, funciones o texto, piezas de comunicación para imprimir datos por el USB o el Bluetooth, y piezas para crear variables y trabajar con ellas.

El sistema de Bitbloq está basado en Scratch y ha sido una buena idea crear un editor web gráfico debido a que Scratch [793] es muy utilizado para enseñar a los estudiantes a programar [784], [794].

21.2.3 MINIBLOQ

MiniBloq v. 0.83 [340] es un editor gráfico de escritorio de código abierto para desarrollar aplicaciones para Multiplo y Arduino. MiniBloq permite crear el ciclo de vida de la aplicación mediante la composición de un puzle. Una vez que la aplicación está definida en ese puzle, el programa crea el código fuente necesario.

Ilustración 89 Isotipo de Minibloq

Sin embargo, en este caso, las piezas no tienen libre movimiento. Por ejemplo, si se desea agregar una pieza para mejorar la pieza básica, es necesario añadir esta pieza en el menú de la primera pieza en lugar arrastrar una nueva pieza desde el menú general. Sin embargo, si desea iniciar la siguiente instrucción, se debe utilizar el menú general para añadir la pieza. También existe el caso contrario: tener la misma función en diferentes piezas con diferente icono, por ejemplo, para utilizar los pines digitales desde el menú general con la pieza de la doble flecha o desde el USB con la pieza del pin digital.

Además, Minibloq ofrece menos libertad al utilizar los pines, ya que estos dependen del tipo de pieza. Otro problema es que incluye por defecto los archivos de cabecera «mbq.h» y «PingIRReceiver.h», aunque no se necesite nada de ellos. Además, para repetir una tarea, es necesario insertar un bucle porque todo el código fuente que genera se crea en el método «*setup*» en lugar de en el método «*loop*». Esto es erróneo porque en Arduino el método «setup» sirve para el establecimiento de las distintas variables y los pines, mientras que el método «*loop*» es para la creación de tareas repetitivas. Además, Minibloq necesita de la creación de variables para poder crear la funcionalidad interna del programa.

21.2.4 CONCLUSIONES

Las herramientas que se han visto son bastante potentes y ofrecen una gran ayuda. No obstante, como se ha visto en el primer caso, en las generadoras de aplicaciones para smartphones, estas no ofrecen la posibilidad de trabajar con sensores, solo con el GPS, además de con datos externos o enviar datos a otro sitio, lo que impide el crear aplicaciones para IoT.

Por otro lado, en lo relacionado a los microcontroladores están Bitbloq y Minibloq. El primero es muy potente, pero no permite modificar el código fuente generado y obliga a establecer el tipo de comunicación a realizar con cada dispositivo, ya sea USB o Bluetooth, así como a usar un sistema de puzles que difiere de la realidad y que se hace necesario aprender.

En cambio, Minibloq facilita la generación de aplicaciones y ofrece un editor muy intuitivo, pero sufre de un grave problema de usabilidad, de confusión de sus acciones y de problemas en su generación de código. Por ello, si se necesitase hacer una aplicación crítica en el área industrial, Minibloq podría darnos ciertos problemas con el código generado.

Por estas razones se ha creado un DSL gráfico, MOCSL, que permite la creación de aplicaciones IoT para smartphones con sistema operativo Android y microcontroladores Arduino. En la Tabla 13 se muestra una comparativa entre las 5 alternativas estudiadas y lo que ofrece MOCSL.

	Minibloq	**Bitbloq**	**AppsGeyser**	**AppsBuilder**	**Infinite Monkeys**	**MOCSL**
¿Necesita habilidad de desarrollo?	Sí	Sí	No	No	No	No
Arduino	Sí	Sí	No	No	No	Sí
Mobile	No	No	Sí	Sí	Sí	Sí
Sensores y actuadores	Sí	Sí	No	No	No	Sí
Servicios Web	No	No	No	No	No	Sí
¿Se puede modificar el código?	No	No	No	No	No	Sí

Tabla 13 Comparativa entre las alternativas existentes y MOCSL

21.3 *ARQUITECTURA PROPUESTA*

La arquitectura del sistema se puede dividir en tres capas como se muestra en la Ilustración 90, y que se corresponden con tres de las cuatro capas de la plataforma Midgar. Cada una es un proceso dentro del conjunto global dentro de la infraestructura.

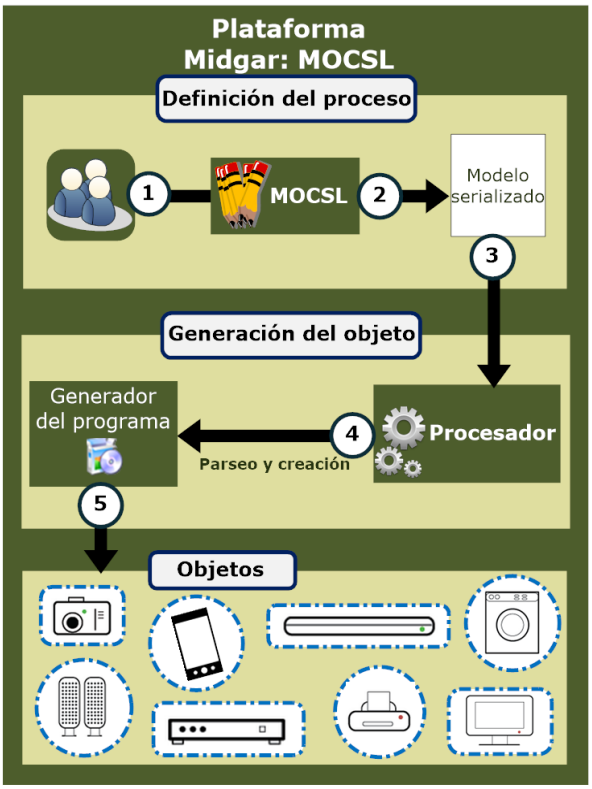


Ilustración 90 Arquitectura de MOCSL

La primera capa es la donde interviene la actuación de los usuarios y abarca todos los procesos en los que tienen ellos que ver. En esta capa, que es la de definición del proceso, el usuario debe de definir el *Smart Object* que desea mediante el uso de MOCSL. Una vez esté definido, el editor gráfico, que es el que ayuda al usuario a definir el DSL, persiste el modelo formal en XML con toda la información necesaria para transformar ese modelo en la aplicación del *Smart Object* en la segunda capa, a la cual lo envía. El modelo podría ser creado también por el usuario mediante el uso correcto del DSL textual, pero sin ayuda de un editor.

La segunda capa de Midgar, que es la de generación de objetos, contiene un nuevo procesador que incluye todo lo necesario para procesar los modelos formales creados con MOCSL y transformarlos en aplicaciones funcionales para smartphones Android o microcontroladores Arduino capaces de conectarse a Midgar debido a que implementan el sistema de mensajes de la plataforma IoT.

Una vez se ha generado la aplicación en base a lo definido por el usuario, esta se puede subir al *Smart Object* destino, ejecutarla y registrar el objeto. De esta manera, el objeto quedará registrado en Midgar y puede empezar a enviar datos y permitirá ser interconectado a otros objetos mediante el uso de MOISL.

21.3.1 PRESENTACIÓN DE LA ARQUITECTURA DE LA INGENIERÍA DIRIGIDA POR MODELOS UTILIZADA

Para crear MOCSL, que es un DSL, se ha requerido crear primero el metamodelo para así poder especificar las características del modelo y represente adecuadamente la sintaxis abstracta definida, recordando que el dominio del problema es la **generación de *Smart Objects* para Internet de las Cosas**.

Tras esto, se ha creado una sintaxis concreta que respetase la sintaxis abstracta y se ha creado el DSL gráfico MOCSL para representarla y permitir al usuario crear los modelos formales deseados. A este DSL se la acompañó de un editor gráfico que facilita su uso y proporciona la semántica estática que se encarga de realizar las comprobaciones semánticas en el modelo, a fin de comprobar que todo lo que haga el usuario sea correcto y esté permitido en el metamodelo. No obstante, como se ha apuntado, se puede utilizar la versión textual del DSL.

A continuación, se tratará en este apartado el metamodelo creado y las dos sintaxis, la abstracta y la concreta.

21.3.1.1 METAMODELO

Al igual que ocurrió en la segunda iteración, en esta segunda se optó por utilizar Ecore [181] como meta-metamodelo.

Utilizando Ecore se ha definido el metamodelo que se muestra en la Ilustración 91, el cual contiene todos los elementos necesarios para conectar un objeto a una plataforma IoT, aunque es nuestro caso dicha plataforma IoT es Midgar.

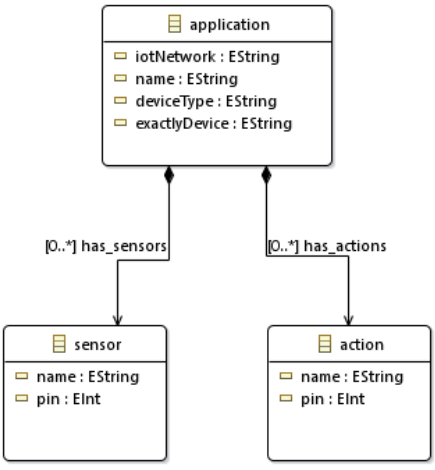


Ilustración 91 Metamodelo de MOCSL [213]

Como bien se ve, se necesitan los datos necesarios para conectar el *Smart Object* a la red, así como sus cualidades y los sensores y acciones de los que dispondrá el objeto.

[213] Se encuentra en lengua inglesa debido a que la implementación ha sido realizada en esta lengua

21.3.1.2 SINTAXIS ABSTRACTA

La sintaxis abstracta de este metamodelo cuenta con 3 nodos. El primero de ellos representa la aplicación y todos los datos necesarios, como son la red IoT, el nombre de la aplicación, el tipo de dispositivo, para saber si es un smartphone o un microcontrolador, y el tipo exacto de dispositivo, de manera que se identifique exactamente el smartphone que es o el microcontrolador exacto para saber el código fuente o las posibilidades de las que dispone.

Por otro lado, están los sensores y las acciones, los cuáles pueden no existir o existir en el dispositivo, y en este caso, podrían ser infinitos.

21.3.1.3 SINTAXIS CONCRETA

Con el metamodelo creado y la sintaxis abstracta definida, se creó la sintaxis concreta y sus correspondientes representaciones textuales y gráficas en los DSLs, como se ve en la Tabla 14. No obstante, cabe destacar que solo se aportó editor para el DSL gráfico.

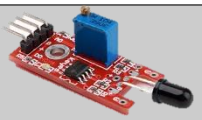
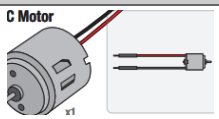
Sintaxis abstracta	**Sintaxis concreta**	
	DSL Textual	**DSL Gráfico**
Application	<application>	Formulario
Sensor	<sensor>	
Action	<action>	

Tabla 14 Correspondencia entre la sintaxis abstracta y la sintaxis concreta [214]

El DSL textual está basado en XML, y es en este en el que se hace la persistencia del modelo formal. Por otro lado, el DSL gráfico se compone de un formulario para especificar los datos de la aplicación. Después, según el tipo de dispositivo, se ve de forma diferente. En el caso de los smartphones, se muestra una lista de checkbox de los sensores y actuadores existentes en ese tipo de dispositivo. En el caso de los microcontroladores, se muestra el microcontrolador y al pulsar en el pin deseado se despliega una lista con fotos de los diferentes sensores y actuadores existentes.

21.3.2 IMPLEMENTACIÓN

Con MOCSL los usuarios pueden crear la lógica necesaria para sus objetos sin necesidad de escribir código gracias al uso de un sistema *drag & drop* en la plataforma IoT Midgar.

En los siguientes subapartados se describirán el DSL textual y el DSL gráfico. Para este último se acompañará de la descripción del uso del editor que lo apoya. Tras esto, se explicará como la segunda capa

[214] Se encuentra en lengua inglesa debido a que la implementación ha sido realizada en esta lengua

procesa el modelo formal creado con el DSL y como este procesador genera la aplicación que se ha de insertar en el objeto destino.

En este caso, la tercera capa de Midgar no es necesaria y la cuarta capa se mantiene con el mismo funcionamiento. La diferencia es que, en este caso, es el propio usuario el que crea el cómo un *Smart Object* funciona en vez de tener uno ya predefinido. Por estos motivos, estas dos capas no se explicarán.

21.3.2.1 LENGUAJE DE DOMINIO ESPECÍFICO TEXTUAL

La primera versión realizada fue la del DSL textual, basada en XML, que se utiliza para persistir el modelo formal creado por el usuario y pasarle este al procesador para que genere los objetos.

En el Código fuente 22 se muestra un ejemplo de cómo se creó una aplicación para un Nexus 4, el cual es un smartphone Android. En este, código de ejemplo se puede ver como se especifica la IP de la plataforma IoT, el nombre, y el tipo de dispositivo exacto. Tras esto, se encuentran definidos dos sensores, un acelerómetro y un sensor de proximidad, y la acción «*Toast*», que muestra un mensaje emergente en pantalla durante unos breves segundos. Cabe recordar que, en los smartphones no hace falta conectar ningún elemento ya que vienen todo integrado, lo que hace que, en este caso, como bien se ve en el código fuente de ejemplo, no se utilice el atributo «pin».

```
<application iotNetwork='156.35.82.104:3000' name='Test'
deviceType='Smartphone' exactlyDevice='Nexus 4'>
      <sensor>Accelerometer</sensor>
      <sensor>Proximity</sensor>
      <action>Toast</action>
</application>
```

Código fuente 22 Ejemplo de modelo formal serializado para crear un *Smart Object* con un Nexus 4

En cambio, en el Código fuente 23 se muestra el modelo formal serializado de un Arduino UNO SMD conectado a Midgar y que contiene un sensor fotorresistor en el pin A0 y tres actuadores, los cuales son un led en el pin 13, un motor de corriente continua (DC) en el pin 7, y un servomotor en el pin 2.

```
<application iotNetwork='156.35.82.104:3000' name='Test'
deviceType='Arduino' exactlyDevice='Uno'>
  <sensor pin='A0'>photoResistor</sensor>
  <action pin='13'>led</action>
  <action pin='7'>dcMotor</action>
  <action pin='2'>servo</action>
</application>
```

Código fuente 23 Modelo formal serializado del microcontrolador Arduino UNO SMD

21.3.2.2 LENGUAJE DE DOMINIO ESPECÍFICO GRÁFICO

La Ilustración 92 muestra una captura de MOCSL con una aplicación terminada para el Arduino UNO SMD. Además, esta ilustración se ha dividido en las 4 áreas importantes del editor.

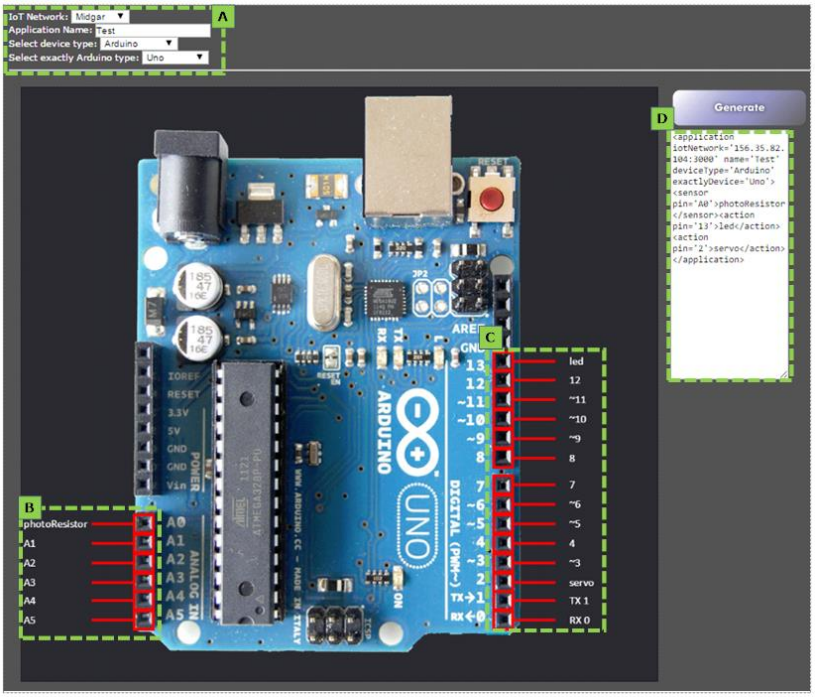


Ilustración 92 Ejemplo de creación de un Arduino SMD con MOCSL [215]

La primera área, que se corresponde con la Ilustración 92ª, contiene el formulario donde se seleccionan los datos de la aplicación, es decir, la plataforma IoT, el nombre de la aplicación, el tipo de dispositivo (smartphone Android o Arduino), y el modelo exacto del dispositivo dentro de los dos anteriores grupos, cuyo *combobox* se actualiza y muestra solo los modelos posibles en función de la selección del tipo de dispositivo elegido. Una vez el usuario ha realizado esta última selección, se muestra una imagen del dispositivo elegido en el *canvas*, que se corresponde con la zona de color negro oscuro. En este ejemplo se ha seleccionado el Arduino Uno SMD.

La Ilustración 92B se corresponde con el área donde el usuario puede seleccionar los pines analógicos del Arduino UNO SMD, mientras que en la Ilustración 92C es donde el usuario puede seleccionar los pines digitales. Mientras, en el área de la Ilustración 92D es donde se visualiza el modelo formal creado cuando se da al botón «*Generate*», correspondiéndose este con la sintaxis del DSL textual.

Cuando el usuario selecciona uno de estos pines, se muestra un *popup* con los diferentes sensores y actuadores existentes en la plataforma y que puede utilizar, como bien se muestra en la Ilustración 93. EN este *popup* el usuario puede seleccionar la que desee. En cambio, si clica el botón «*Cancel*», el usuario cerrará

[215] Se encuentra en lengua inglesa debido a que la implementación ha sido realizada en esta lengua

el *popup* y borrará el sensor o actuador que estuviese establecido, en caso de existir uno, devolviendo así el pin a su estado por defecto.

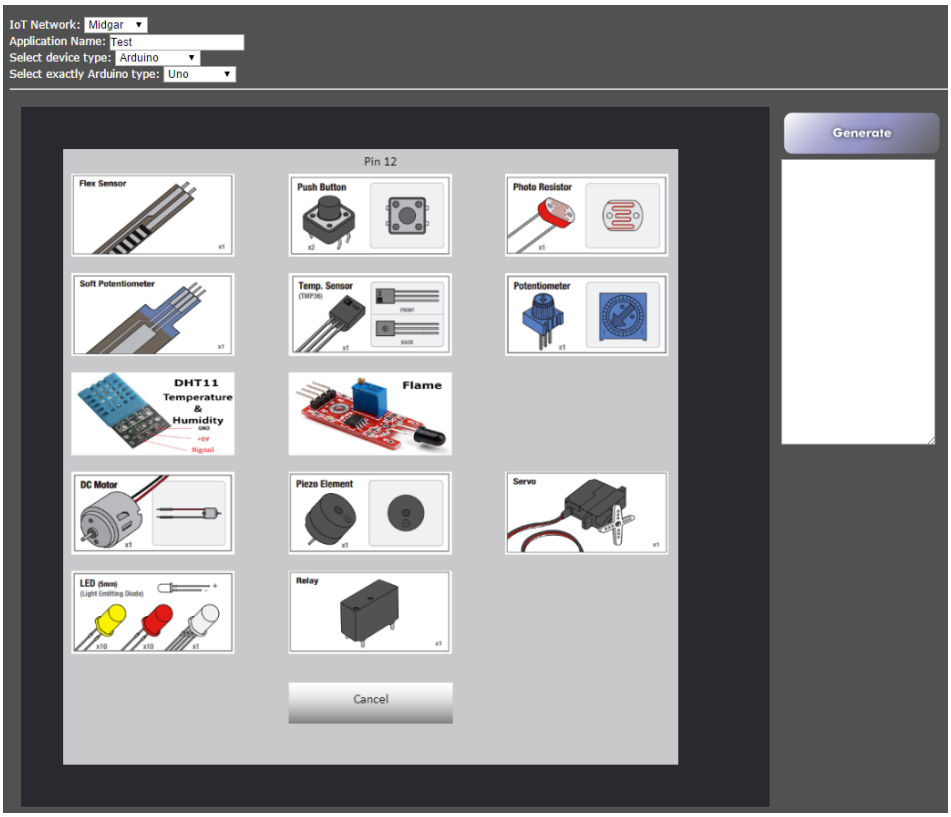


Ilustración 93 Sensores y acciones disponibles para el Arduino UNO SMD

Por otro lado, si en vez de seleccionar un microcontrolador, seleccionamos un smartphone Android como puede ser el Nexus 4, todo cambia, como se ve en la Ilustración 94. Como se ve, se muestra la foto del modelo y en ella, se muestra una lista con todos los sensores disponibles en los smartphones Android. No obstante, esta lista solo muestra los sensores que no posee este modelo con un guion rojo (▬), con una equis roja (✘) los sensores que posee y que no han sido seleccionados, siendo este el estado por defecto de todos, y con un tic verde (✓) los sensores que han sido seleccionados por el usuario a la hora de crear la aplicación.

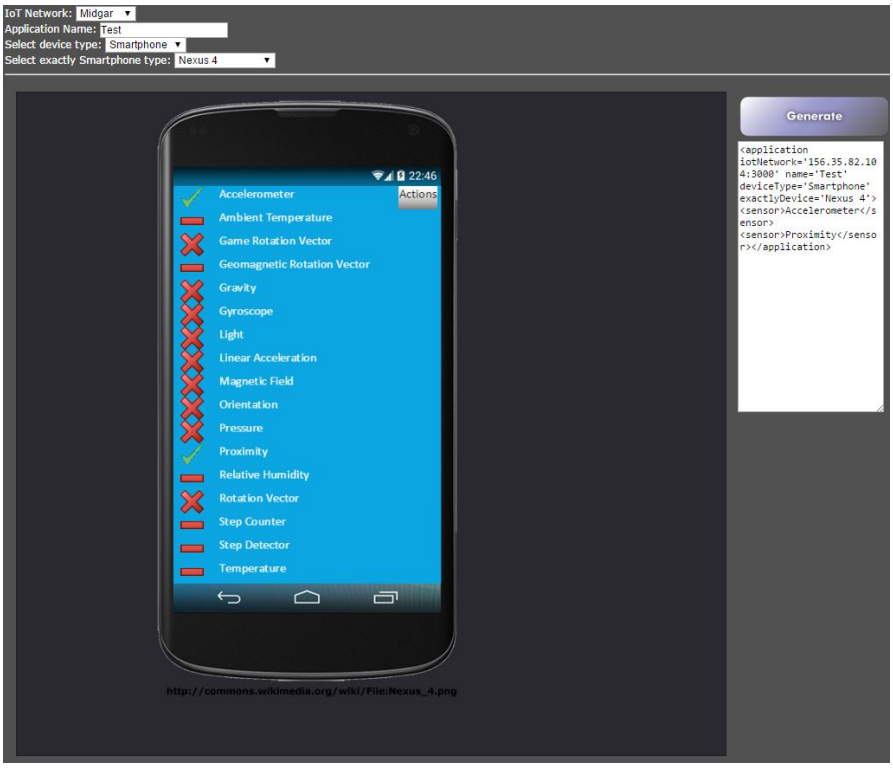


Ilustración 94 Creación del software para un smartphone Android Nexus 4

En el caso de que el usuario clique sobre el botón de la esquina superior derecha gris llamado «*Actions*», la lista cambiará para mostrar todas las acciones disponibles en ese smartphone. Esto se puede ver en la Ilustración 95.

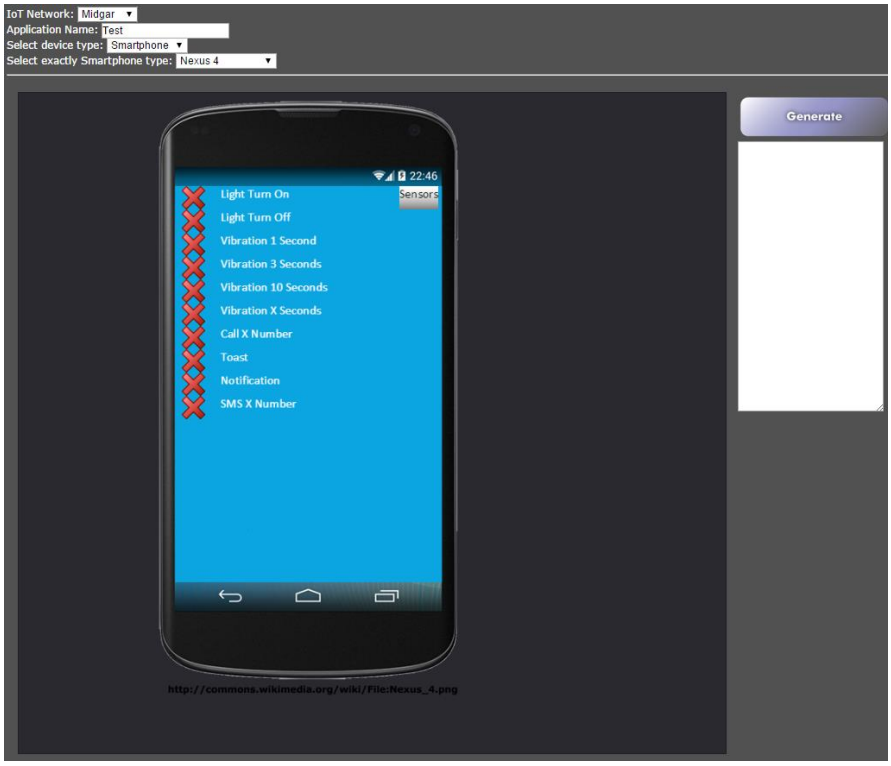


Ilustración 95 Lista de acciones disponibles para el smartphone

21.3.2.3 OBJECTS GENERATION

La Ilustración 96 muestra el proceso de funcionamiento de la segunda capa. Como se ve, la segunda capa recibe el modelo formal serializado en XML. Tras recibirlo, el **procesador** procesa los nodos y los atributos de los nodos para obtener la información necesaria acerca de la aplicación que fue definida previamente por el usuario.

Con esta información, el **procesador** reemplaza en las plantillas existentes para cada dispositivo los símbolos correspondientes a lo definido por el usuario para así insertar lo necesario para que se comporten como se espera. En el caso del Arduino, el procesador crea una aplicación para el Arduino e importa las librerías y el código nativo en C++ necesario para los sensores que así lo requieran

Cuando el **procesador** finaliza el procesado, se genera la aplicación. La aplicación generada es la aplicación que el usuario debe de desplegar en su smartphone o en su microcontrolador.

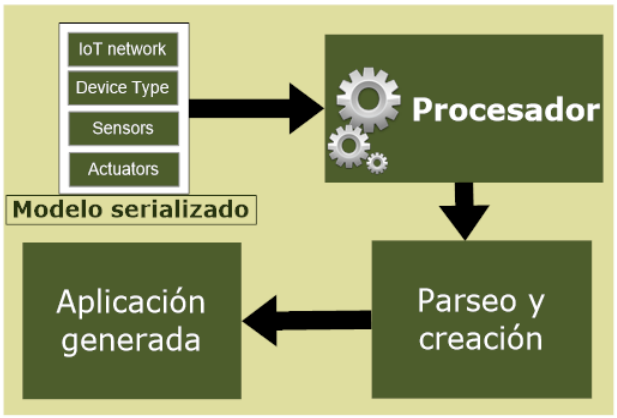


Ilustración 96 Funcionamiento interno del generador de objetos

21.3.3 SOFTWARE Y HARDWARE UTILIZADO

A continuación, se muestra el software utilizado para el desarrollo de este prototipo:

- La plataforma IoT Midgar:
  - Ruby 2.2.2p95.
  - Rails 4.2.1.
  - Servidor web Thin 1.6.3.
  - Base de datos MySQL 5.5.43.
- El editor gráfico se ha realizado utilizando HTML5 y JavaScript estándar, sin utilizar ninguna librería externa.
- Para la creación del generador de aplicaciones y la aplicación de conexión con el Arduino al ordenador se optó por utilizar Java 8.

Para las pruebas se utilizó el siguiente equipo hardware:

- Raspberry Pi 2 Model B dedicada con un sistema operativo Raspbian 1.4.1.
- Cuatro smartphones Android:

  - Nexus 5 con Android 5.1.1.
  - Nexus 4 con Android 5.1.1.
  - Motorola con la versión 2.2.2.
  - Samsung Galaxy Mini S5570 con la versión 2.3.6.
- Microcontrolador Arduino Uno SMD basado en el ATmega328.
- Durante los diferentes test se han usado los siguientes sensores y actuadores: termistor TMP36, un altavoz, un servomotor, un motor DC, diferentes LEDs, dos botones, un fotorresistor, un sensor de temperatura y humedad DHT11, y un sensor de llamas KY026.

21.4 *EVALUACIÓN Y DISCUSIÓN*

Esta subsección mostrará la metodología realizada en la evaluación de este prototipo, así como la evaluación junto su discusión. La evaluación fue dividida en dos fases. La primera fase es la toma de datos cuantitativos a partir de la evaluación realizada al participante. La segunda fase es una encuesta realizada a los participantes.

21.4.1 **METODOLOGÍA**

El objetivo principal de este proceso de evaluación es validar la hipótesis inicial, la cual es crear un DSL que facilite la creación de *Smart Objects* y si este puede ser una mejora significante para Internet de las Cosas. Para validar la hipótesis se ha implementado un proceso de evaluación dividido en dos fases. De esta manera, se han obtenido una evaluación más completa al contar con datos cuantitativos y cualitativos que ayudan a verificar de una mejor manera la hipótesis usando. Por ello, se ha dividido la evaluación en dos fases:

- **Fase 1**: en esta primera fase se ha propuesto a los diferentes participantes que creasen una aplicación utilizando dos editores gráficos y MOCSL. Esta aplicación es un caso básico posible que se puede requerir programar en un Arduino Uno. Esta fase se corresponde con la evaluación cuantitativa.
- **Fase 2**: tras realizar la fase uno, los participantes han tenido que realizar una encuesta compuesta de 13 declaraciones y que utiliza la escala Likert de 5 puntos [781]. En esta encuesta, los participantes han de decir como de acuerdo están con cada una de las declaraciones. Estas declaraciones están basadas en la fase 1 y muestran diferentes aspectos relevantes y cualitativos de esta investigación.

21.4.1.1 FASE 1

Para esta fase se han utilizado dos editores gráficos y MOCSL. Como editores gráficos se han elegido MiniBloq y BitBloq. La elección de MiniBloq es debido a que este es el editor gráfico para programar para Arduino. Por otro lado, se ha elegido BitBloq porque es el editor web gráfico oficial de la empresa BQ para programar para sus microcontroladores, como ZUM, además de ofrecer soporte para Arduino.

Durante el transcurso de las pruebas se han tomado datos cuantitativos del uso de los editores, como han sido el tiempo en seguro que tomo a cada participante realizar la aplicación, los centímetros requeridos con el ratón y los clics con el botón derecho e izquierdo del ratón. Para medir estos datos se ha utilizado la herramienta Mousotron [780].

En primer lugar, se definió una tarea básica que emulase un escenario real y que debía de ser realizada por los participantes. No obstante, hubo que tener cuidado al elegir los sensores, pues las tres plataformas no contienen todos los sensores, pues es algo imposible, siendo la más restringida BitBloq. Es por esto que se tuvo que pedir una aplicación que funcionase con sensores que tuvieran las tres plataformas utilizadas.

La función de esta aplicación es simular un sistema que encienda la luz y un motor que pueda renovar el aire de una habitación cuando el fotorresistor recibe luz. Para ello, los han realizado un objeto con un sensor y dos actuadores. El sensor era un fotorresistor y los actuadores eran un LED y un servomotor. Los participantes han tenido que realizar la misma tarea en los dos editores gráficos y en MOCSL.

En las siguientes ilustraciones se muestra la solución a la aplicación pedida en cada una de las plataformas, viendo en la Ilustración 97 la solución en MiniBloq, en la Ilustración 98 la correspondiente en BitBloq, y finalmente la de MOCSL en la Ilustración 99.

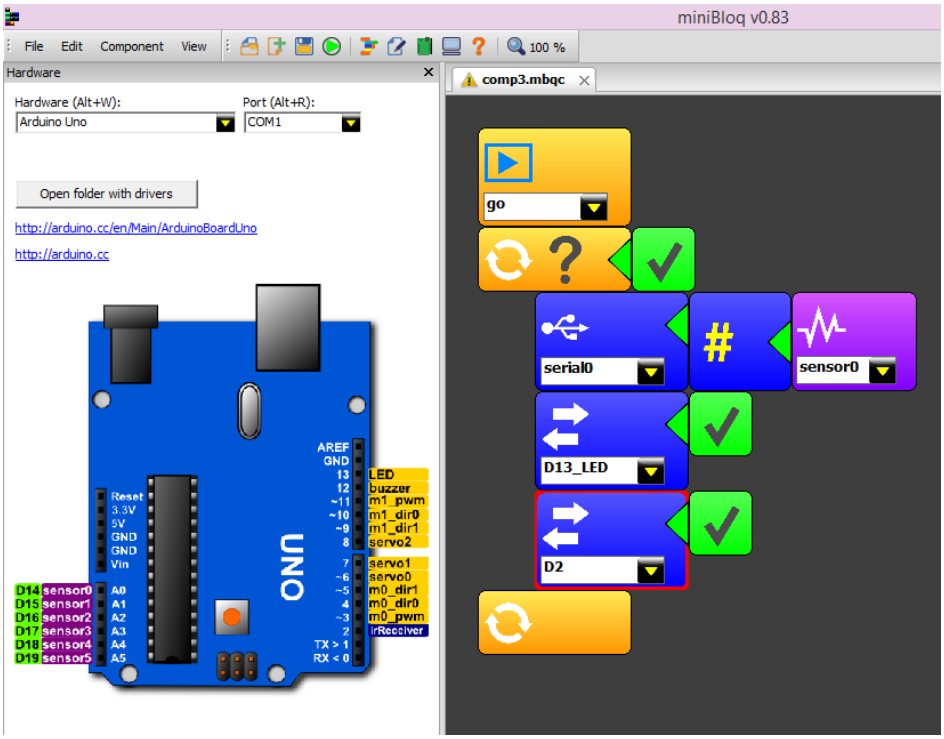


Ilustración 97 Solución de la aplicación pedida utilizando MiniBloq

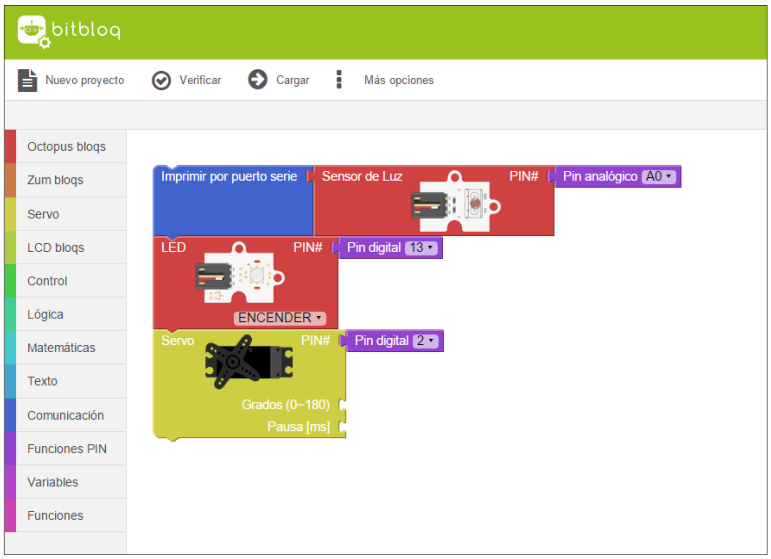


Ilustración 98 Solución de la aplicación pedida utilizando BitBloq

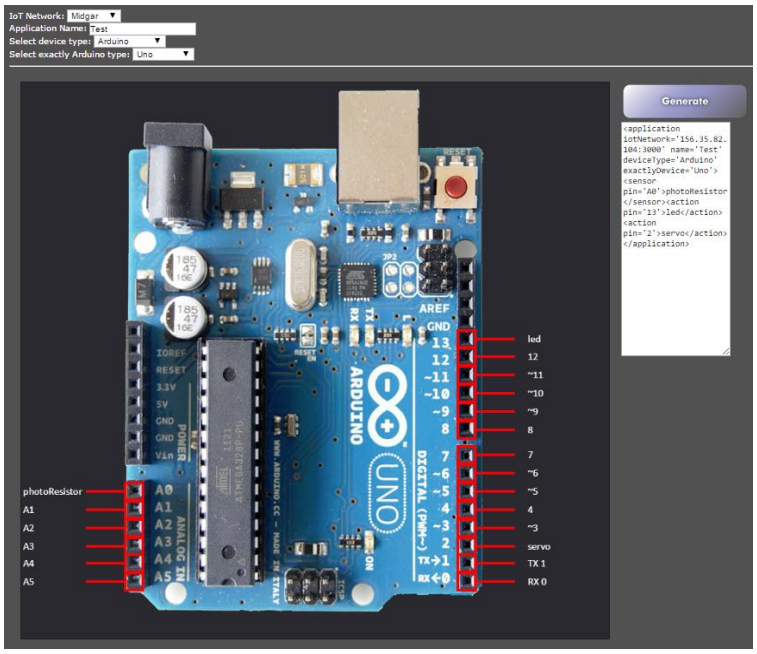


Ilustración 99 Solución de la aplicación pedida utilizando MOCSL

21.4.1.2 FASE 2

En la segunda fase se ha evaluado la parte cualitativa de MOCSL para así obtener la opinión de los participantes y ver su comparación con los otros editores y como MOCSL podría mejorar Internet de las Cosas. Para hacer esta encuesta se ha utilizado la escala Likert de 5 puntos como método de evaluación. La elección de la escala Likert de 5 puntos es debido a que es un método muy utilizado en el campo de la ingeniería del software para obtener información eficazmente para apoyar la toma de decisiones [795]. Este es exactamente nuestro caso porque no podemos medir la eficiencia ni el potencial de la creación de objetos utilizando un DSL. Para ello, hemos utilizado las encuestas para obtener más información y así obtener una respuesta más exacta a nuestras hipótesis.

La encuesta cuenta con un total de 13 declaraciones y 5 posibles respuestas que son: 1 como *Totalmente en desacuerdo*, 2 como *En desacuerdo*, 3 como *Neutral*, 4 como *De acuerdo*, y 5 como *Totalmente de acuerdo*.

Los participantes realizaron la encuesta una vez habían terminado de realizar la fase 1. Las encuestas se han realizado de manera anónima y sin ayuda. Esta encuesta contiene declaraciones sobre MOCSL acerca de la opinión de los participantes, sus posibilidades y el impacto que puede tener en Internet de las Cosas y en los *Smart Objects*. La Tabla 26 contiene las declaraciones.

#	Declaraciones
Q1	El participante entiende la funcionalidad de los elementos del DSL y su rol en el proceso de creación.
Q2	MOCSL permite crear *Smart Objects* de una manera fácil, usando pocos clics y sin tener que programar código.
Q3	MOCSL hace difícil el cometer errores cuando el participante modela la aplicación.
Q4	Esta solución ofrece una forma rápida de desarrollar la tarea indicada.
Q5	Esta solución provee de asistencia para crear aplicaciones para *Smart Objects*.
Q6	Con base en la experiencia previa con el uso de otras herramientas, MOCSL provee de una mayor capacidad para generar aplicaciones.
Q7	MOCSL no requiere que el participante posea habilidad complejas de programación, como ocurre en el desarrollo tradicional de aplicaciones.
Q8	MOCSL incluye suficientes elementos y funcionalidad para que el participante cree un amplio rango de aplicaciones para *Smart Objects*.
Q9	Esta propuesta es una contribución positiva para fomentar el desarrollo de servicios y aplicaciones para los *Smart Objects*.
Q10	Internet de las Cosas y los *Smart Objects* se beneficiarán de esta solución.
Q11	MOCSL podría ser usado para simplificar el proceso de desarrollo clásico de aplicaciones software en otras áreas.
Q12	MOCSL es más fácil de usar que MiniBloq.
Q13	MOCSL es más fácil de usar que BitBloq.

Tabla 15 Declaraciones de la encuesta de MOCSL

21.4.2 RESULTADOS

En esta sección se mostrará y discutirán los resultados obtenidos en cada fase. En la primera subsección se mostrarán los resultados cuantitativos obtenidos de realizar la aplicación en los diferentes editores y en la segunda fase se mostrarán los resultados cualitativos de la encuesta.

21.4.2.1 FASE 1

En esta primera fase se ha tomado el tiempo de cada participante, en segundos, mientras hacían la tarea con cada editor, los clics necesarios de ratón con el botón primario y secundario, y los centímetros movidos del ratón.

En la Ilustración 100 se muestra el tiempo que necesitó cada participante para realizar la tarea en cada plataforma.

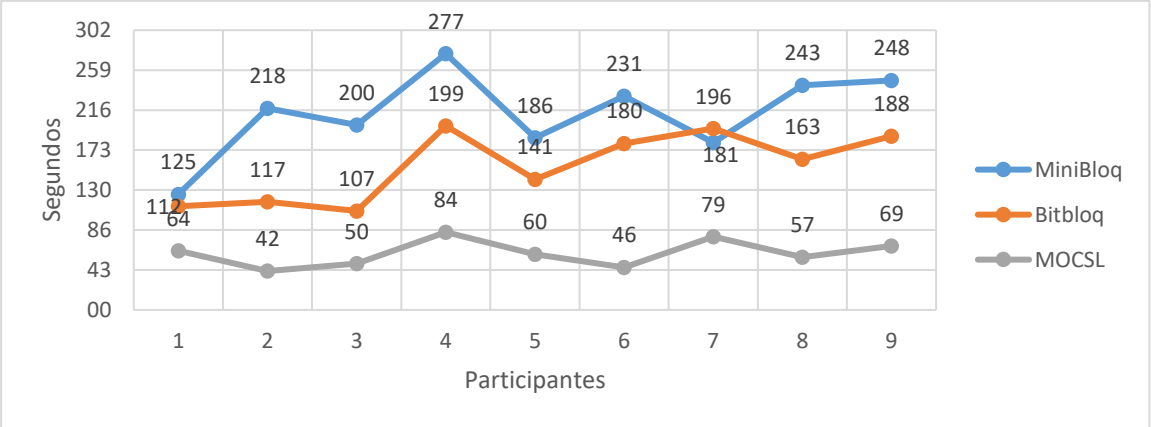


Ilustración 100 Tiempo que necesitó cada participante para realizar la tarea en cada plataforma

Analizando esta gráfica, se pueden sugerir las siguientes interpretaciones:

- El participante más rápido en cada plataforma ha necesitado 125 segundos en MiniBloq, 107 segundos en BitBloq y 42 segundos en MOCSL, lo que indica que esta última es la más rápida si se comparan los mejores resultados.
- • El participante más lento en cada plataforma ha necesitado 277 segundos en MiniBloq, 199 segundos en BitBloq y 84 segundos en MOCSL, lo que indica que esta última es la más rápida si se comparan los peores resultados.
- El tiempo medio de los participantes en cada plataforma es de 212 segundos para MiniBloq, 156 para BitBloq y 61 para MOCSL. Esto indica que los participantes necesitan solo el 39% del tiempo que BitBloq y el 28% de MiniBloq para realizar la misma aplicación en MOCSL.
- MiniBloq tiene los tiempos más lentos en todos los participantes a excepción de en el participante número 7.
- MOCSL es más rápido que MiniBloq y BitBloq para crear objetos en todos los casos. Esto indica que MOCSL ofrece mayor velocidad para crear objetos.

- El peor tiempo de MCOSL es de 84 segundos, que es mejor que el mejor tiempo de MiniBloq y BitBloq, en el que en ambos es de 107 segundos.

En la Ilustración 101 se muestran los clics que necesitó cada participante al realizar la aplicación en cada plataforma.

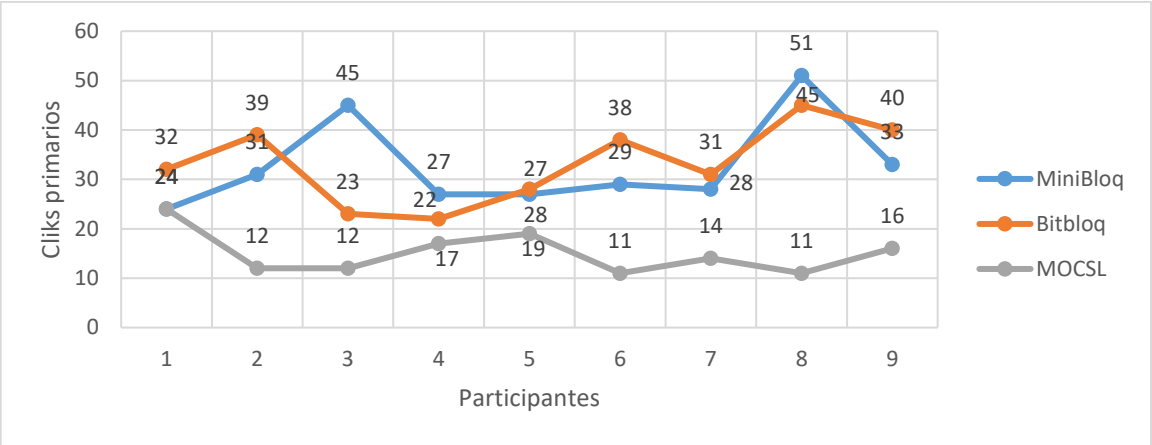


Ilustración 101 Clics primarios en cada plataforma realizados por cada participante

Analizando este último gráfico se interpreta lo siguiente:

- El mayor número de clics primarios en cada plataforma han sido 51 en MiniBloq, 45 en BitBloq y 24 en MOCSL. EE base a las peores estadísticas, MOCSL es la plataforma que requiere menos clics para realizar la aplicación.
- El menor número de clics primarios en cada plataforma ha sido 24 en MiniBloq, 22 en BitBloq y 11 en MOCSL. Así, analizando los mejores resultados, MCOSL requiere la mitad de clics que otras plataformas.
- La media de clics primarios en cada plataforma ha sido de 32,78 en MiniBloq, de 33,11 en BitBloq y de 15,11 en MOCSL. Esto indica que mientras que MiniBloq y BitBloq requieren un número similar de clics, MOCSL necesita menos de la mitad que estas.
- MiniBloq tiene el máximo número de clics, pero menor media que BitBloq, lo que indica que MiniBloq es menos entendible que BitBloq dependiendo de la gente.
- MOCSL es siempre la plataforma con menor número de clics excepto en el primer participante que hizo el mismo número que con BitBloq, lo que indica que MOCSL es la plataforma más usable.

La Ilustración 102 muestra el número de clics con el botón secundario del ratón que necesitó cada participante a la hora de la realización de la aplicación en cada plataforma. Sin embargo, en el caso de MCOSL no es necesario utilizar este botón debido a que se redujo la complejidad restringiendo todo el uso al botón primario. En el caso de MiniBloq y BitBloq, el botón secundario ofrece atajos y la opción de borrado.

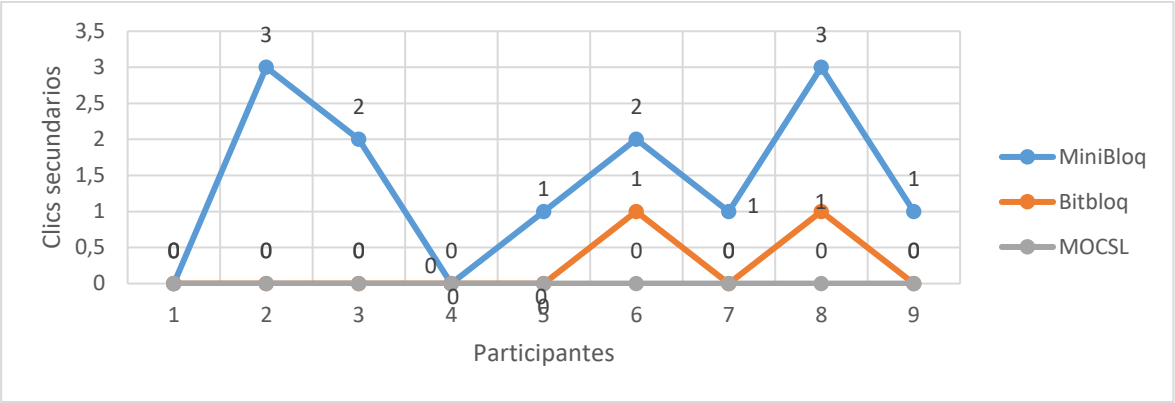


Ilustración 102 Clics de botón secundario que han necesitado los participantes

A partir de estos datos se pueden obtener las siguientes conclusiones:

- MiniBloq es la plataforma con más clics secundarios con un total de 13 clics, BitBloq tiene 2 clics y MOCSL tiene 0.
- Solo dos participantes no han usado el botón secundario del ratón en MiniBloq. Por el contrario, en BitBloq solo dos participantes usaron el botón secundario. En este caso, los participantes de BitBloq tuvieron menos fallos.

La Ilustración 103 contiene la gráfica que muestra la distancia recorrida, en centímetros, por los participantes de esta evaluación en cada una de las tres plataformas.

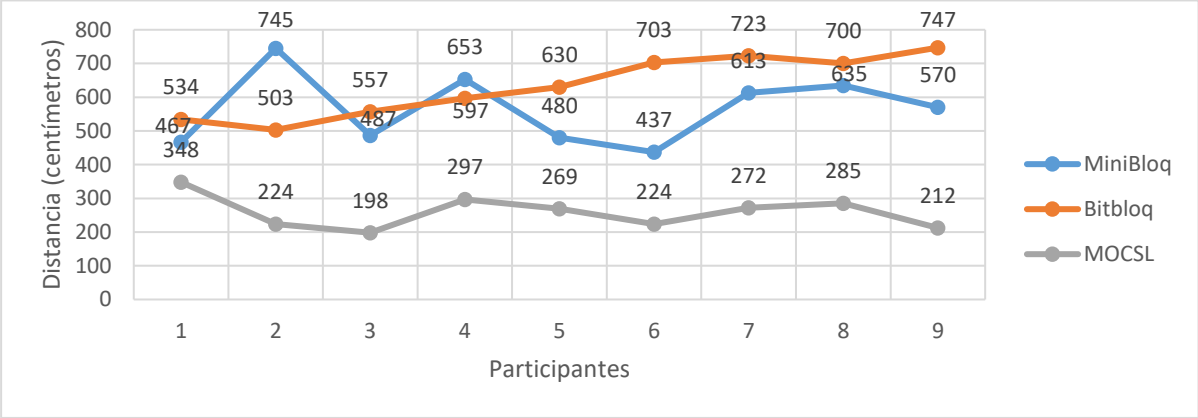


Ilustración 103 Distancia que han necesitado los participantes en cada plataforma

Analizando esta gráfica se pueden sugerir las siguientes interpretaciones:

- La mayor distancia recorrida, en centímetros, en cada plataforma fue de 745 en MiniBloq, de 747 en BitBloq y de 348 en MOCSL. Esto indica que MOCSL tiene los elementos más accesibles que los otros dos editores.
- Por el contrario, la menor distancia recorrida en cada editor durante las pruebas fue de 437 para MiniBloq, de 503 en BitBloq y de 198 en MOCSL, lo cual indica que BitBloq puede tener los elementos más dispersos que MiniBloq y que, de nuevo, MOCSL parece tener los elementos más cercanos, obligando a los participantes a moverse menos.

- La distancia media en cada editor fue de 565,22 para MiniBloq, de 632,67 en BitBloq y de 258,78 en MOCSL. Esto remarca la interpretación anterior.
- La peor distancia en MOCSL fue de 348 centímetros, la cual es menor que la mejor distancia en MiniBloq y de BitBloq, que fue de 437 centímetros. Por ello, MOCSL, en todos los casos, ha sido la plataforma que ha necesito menos distancia para resolver la tarea propuesta.

21.4.2.2 FASE 2

La Tabla 16 muestra las respuestas de cada participante para cada declaración de la encuesta.

	Q 1	**Q 2**	**Q 3**	**Q 4**	**Q 5**	**Q 6**	**Q 7**	**Q 8**	**Q 9**	**Q 10**	**Q 11**	**Q 12**	**Q 13**
P01	5	5	4	5	5	4	4	5	5	5	1	5	5
P02	5	5	5	5	3	5	5	4	4	4	5	5	4
P03	3	5	5	5	5	5	3	3	5	4	4	5	4
P04	3	5	4	5	4	5	5	3	4	4	1	5	5
P05	5	5	5	4	5	4	5	5	3	4	4	5	3
P06	5	5	5	5	5	5	5	5	5	5	5	5	5
P07	5	5	4	5	5	5	5	5	5	5	5	5	5
P08	5	5	4	5	5	5	5	5	5	4	5	5	5
P09	5	5	5	5	4	4	5	4	5	5	5	5	5

Tabla 16 Respuestas de los participantes para cada declaración de MOCSL

La Tabla 17 muestra las estadísticas descriptivas globales. Esta tabla muestra el mínimo, el primer cuartil, la mediana o segundo cuartil, el tercer cuartil, el máximo, el rango entre cuartiles y la moda de cada una de las declaraciones de la encuesta.

	Q 1	**Q 2**	**Q 3**	**Q 4**	**Q 5**	**Q 6**	**Q 7**	**Q 8**	**Q 9**	**Q 10**	**Q 11**	**Q 12**	**Q 13**
Mínimo	3	5	4	4	3	4	3	3	3	4	1	5	3
Cuartil 1	5	5	4	5	4	4	5	4	4	4	4	5	4
Mediana	5	5	5	5	5	5	5	5	5	4	5	5	5
Cuartil 3	5	5	5	5	5	5	5	5	5	5	5	5	5
Máximo	5	5	5	5	5	5	5	5	5	5	5	5	5
Rango	2	0	1	1	2	1	2	2	2	1	4	0	2
Rango intercuartílico	0	0	1	0	1	1	0	1	1	1	1	0	1
Moda	5	5	5	5	5	5	5	5	5	4	5	5	5

Tabla 17 Tabla con las estadísticas descriptivas globales de MOCSL

En la Ilustración 104 se puede ver los datos de cada pregunta, pero en un diagrama de cajas y bigotes.

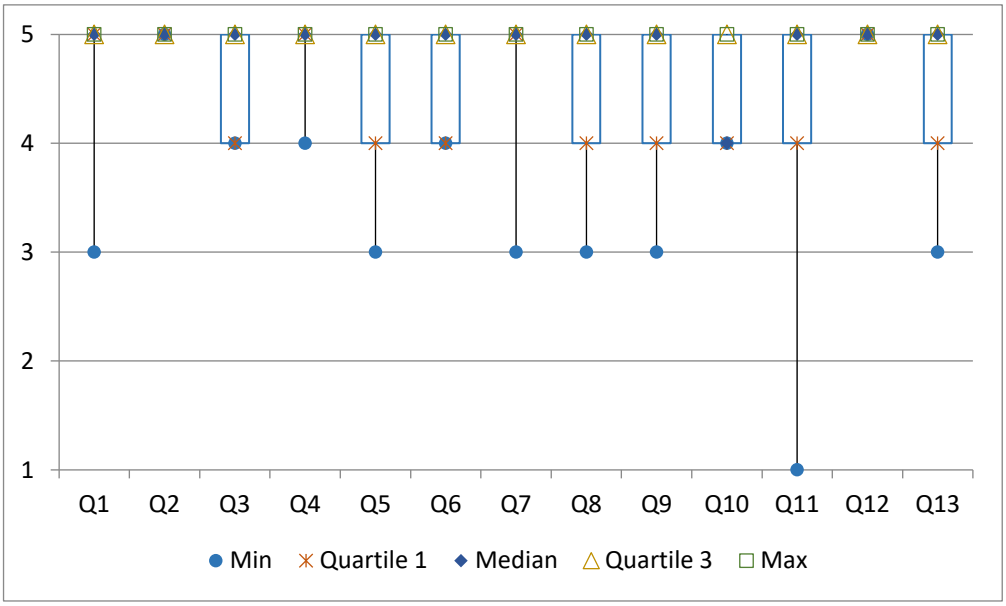


Ilustración 104 Diagrama de cajas y bigotes global para cada declaración de MOCSL

Así, a partir de los datos recogidos y mostrados en la Tabla 17 y la Ilustración 104 se pueden sugerir las siguientes interpretaciones:

- Q2 y Q12 son las declaraciones con mayor mínimo, 5 de 5. Esto significa que todos los participantes están de acuerdo con estas declaraciones, exactamente están «*totalmente de acuerdo*». Por otro lado, el menor mínimo lo tiene Q11, lo que significa que hay mucha diversidad de opiniones entre los participantes.
- Todas las declaraciones, a excepción de la Q10, tienen una mediana de 5, la más alta. En base a esto, se puede interpretar que la mayoría de los participantes están «*Totalmente de acuerdo*» con estas declaraciones. En cambio, Q10 tiene una mediana de 4, lo que indica que los participantes están, como mínimo, «*de acuerdo*» con esta declaración.
- Q2 y Q12 tienen un rango de 0, lo que demuestra que todos los participantes tienen la misma opinión en estas declaraciones. En este caso, los participantes opinan que están *totalmente de acuerdo* con las declaraciones Q2 y Q12. Por el contrario, Q11 tiene un rango de 4, lo que expresa que hay una gran diversidad de opiniones sobre esta declaración.
- Si miramos la moda, todas las declaraciones, excepto Q10, tienen una moda de 5, lo cual indica que la respuesta más elegida ha sido «*totalmente de acuerdo*». Q10 posee una moda de 4, lo que la hace coincidir con la respuesta «*de acuerdo*».
- Si se estudia los máximos, se puede ver que todas las declaraciones tienen el máximo en 5. Esto indica que al menos alguien ha elegido la opción «*totalmente de acuerdo*».
- El rango intercuartílico de Q1, Q2, Q4, Q7 y Q12 es de 0. Esto indica que los participantes tienen una opinión muy cercana en estas declaraciones.

La Tabla 18 muestra las diferentes frecuencias para que declaración en base a las respuestas de los participantes. Esta tabla tiene el desglose de cada pregunta para mostrar el número de votos para cada decisión y su porcentaje correspondiente. En la Ilustración 105 se muestran en un gráfico de barras la frecuencia de

respuestas de los participantes. En la Ilustración 106 se muestran las respuestas de las diferentes declaraciones mediante un gráfico de barras apilado marcando los percentiles.

Declaración		Totalmente en desacuerdo	En desacuerdo	Neutral	De acuerdo	Totalmente de acuerdo
Q1.	#	0	0	2	0	7
	%	0%	0%	22%	0%	78%
Q2.	#	0	0	0	0	9
	%	0%	0%	0%	0%	100%
Q3.	#	0	0	0	4	5
	%	0%	0%	0%	44%	56%
Q4.	#	0	0	0	1	8
	%	0%	0%	0%	11%	89%
Q5.	#	0	0	1	2	6
	%	0%	0%	11%	22%	67%
Q6.	#	0	0	0	3	6
	%	0%	0%	0%	33%	67%
Q7.	#	0	0	1	1	7
	%	0%	0%	11%	11%	78%
Q8.	#	0	0	2	2	5
	%	0%	0%	22%	22%	56%
Q9.	#	0	0	1	2	6
	%	0%	0%	11%	22%	67%
Q10.	#	0	0	0	5	4
	%	0%	0%	0%	56%	44%
Q11.	#	2	0	0	2	5
	%	22%	0%	0%	22%	56%
Q12.	#	0	0	0	0	9
	%	0%	0%	0%	0%	100%
Q13.	#	0	0	1	2	6
	%	0%	0%	11%	22%	67%

Tabla 18 Tabla de frecuencias de las respuestas globales de MOCSL

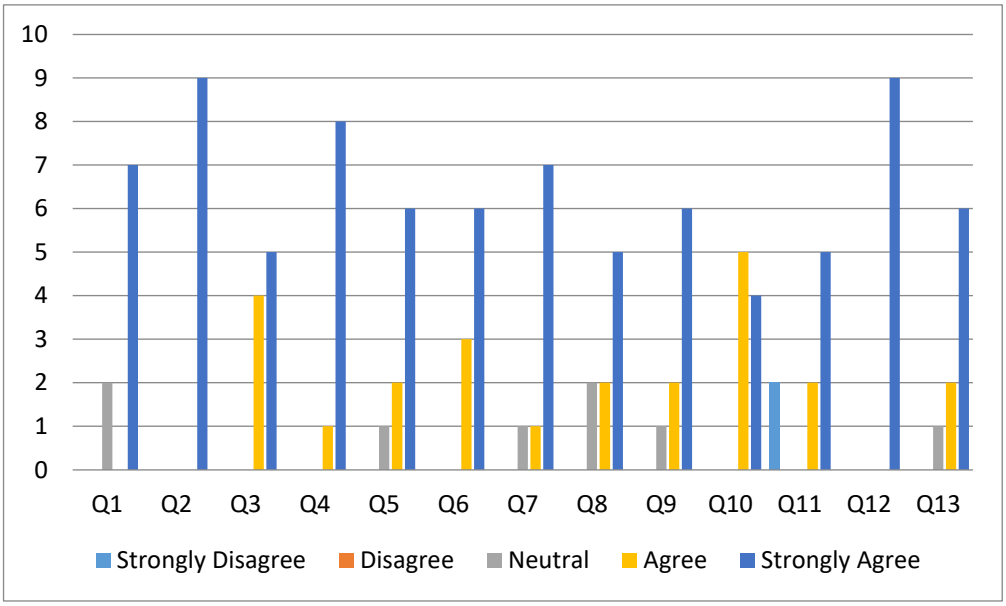


Ilustración 105 Distribución de las respuestas globales de MOCSL

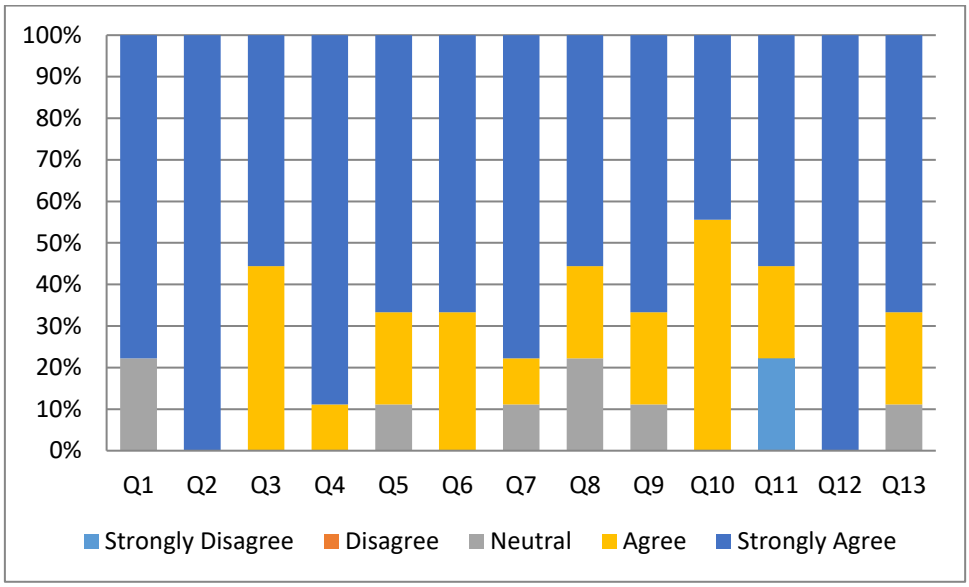


Ilustración 106 Distribución apilada de las respuestas globales de MOCSL

Basándonos en estos datos, se pueden sugerir las siguientes interpretaciones:

- Q2 y Q12 tienen el 100% de los votos en la opción «*totalmente de acuerdo*», lo que indica que todos los participantes están totalmente de acuerdo con estas declaraciones.
- Con el 89% de los votos se encuentra Q4. Esto suma un total de 8 votos en la opción de «*totalmente de acuerdo*». Por otro lado, Q3 y Q6 tienen un mínimo de 44% y 33% votos en «*de acuerdo*», respectivamente. Esto indica que todos los participantes están al menos de acuerdo con estas declaraciones.
- Q1 y Q7 tienen el 78% de los votos en «*totalmente de acuerdo*», lo que hace un total de 7 votos. Mientras tanto, Q1 tiene 2 votos en «*neutral*» y Q7 1 voto en «*de acuerdo*» y otro en «*neutral*».

Esto indica que la mayoría de los participantes están de acuerdo con estas sentencias, pero alguna minoría tiene dudas.

- Q8 y Q11 tienen el 56% de los votos en «*totalmente de acuerdo*» y el 22% en «*de acuerdo*» y en «*neutral*». Estos números se corresponden respectivamente con 5, 2 y 2 votos. Esto indica que todos los participantes están de acuerdo, pero hay un importante porcentaje que está indeciso o que no está del todo de acuerdo con esta declaración.

21.5 *CONCLUSIONES*

En esta iteración se ha ampliado la plataforma IoT Midgar con un novedoso DSL gráfico llamado MOCSL que provee una solución para crear *Smart Objects*, pues crea toda la lógica necesaria para interconectar *Smart Objects* con la plataforma IoT y crear toda la lógica necesaria para que funcionen correctamente como desea el usuario. Todo esto, se consigue de una manera sencilla, como demuestra la evaluación, en pro de la mejora del desarrollo tradicional que es complejo y que necesita que se programe la aplicación, leer los datos, procesarlos y enviarlos.

Midgar ya tiene una solución parcial que es MOISL y permite interconectar los *Smart Objects* registrados en la plataforma IoT entre ellos. Ahora, con MOCSL, Midgar ofrece también a los usuarios la capacidad de crear el software que necesitan estos *Smart Objects* para funcionar como ellos deseen y registrarse en la plataforma sin necesidad de tener conocimientos de desarrollo de software.

Para evaluar la efectividad de esta propuesta, se comparó MCOSL con dos editores gráficos utilizados para solucionar este problema: MiniBloq y BitBloq. En la evaluación se obtuvieron medidas cuantitativas y cualitativas.

En la evaluación cuantitativa MOCSL demostró ser más rápido ya que los participantes solo necesitaron 61 segundos para hacer la tarea. En comparación con las otras dos plataformas, este tiempo supone el 39% necesario en BitBloq y el 28% del tiempo necesario en MiniBloq para hacer la misma aplicación. Esto mismo es aplicable a los clics primarios del ratón y los centímetros recorridos, donde son mucho menores en MOCSL. La media de clics en cada plataforma ha sido de 32,78 en MiniBloq, de 33,11 en BitBloq y de 15,11 en MOCSL y la media en centímetros recorridos de 562,33 en MiniBloq, de 632,67 en BitBloq y de 258,78 en MOCSL. Esto demuestra que MOCSL es mucho más usable y facilita mucho más el trabajo a las personas

Respecto a la evaluación cualitativa realizada con una encuesta siguiendo en el método dela escala Likert de 5 puntos se ha obtenido que los participantes han elegido «*totalmente de acuerdo*» en el 71% de las declaraciones y «*de acuerdo*» en el 21% de las ocasiones. Por consiguiente, el 92% de las declaraciones tienen una acogida positiva o muy positiva entre los participantes. Esto indica que los participantes creen que MOCSL cumple con la funcionalidad de facilitar la creación de *Smart Objects* de una manera fácil sin necesidad de escribir código fuente. Además, también opinan que MOCSL podría ofrecer beneficios para Internet de las Cosas y los *Smart Objects*, lo que permite verificar la segunda hipótesis planteada, la referente a si es relevante para IoT.

Cabe destacar que algunos de los participantes han elegido en un 7% de las respuestas como «*neutral*» y en un 2% como «*totalmente en desacuerdo*». El significado de la primera de ellas es que podemos mejorar algunos aspectos como la facilidad de uso e incluir nuevos elementos para ofrecer más opciones. En el caso de las respuestas con respuesta en «*totalmente de acuerdo*», podemos ver que estas se corresponden con la declaración Q11. La razón es que MOCSL permite la creación de un gran número de aplicaciones para diferentes dispositivos, pero es imposible que permita el modelado de cualquier aplicación imaginable. En futuras ampliaciones se dará énfasis especial a los nuevos elementos de modelado para permitir nuevas extensiones y posibilidades para el desarrollo de aplicaciones, como la inclusión de la Lógica Difusa, la Inteligencia Artificial, otros protocolos, y otras tareas complejas. Sin embargo, cuando se vayan a hacer estas extensiones, hay que respetar el dominio de MOCSL para así evitar la inclusión de funcionalidades con una complejidad indebida.

Por lo tanto, en base a estos resultados podemos decir que MOCSL puede ser muy útil para facilitar la creación de *Smart Objects* y reducir el tiempo y la complejidad de la creación de este tipo de aplicaciones. Así, MOCSL complementa la plataforma de Midgar y ayuda a personas sin conocimientos de desarrollo a crear *Smart Objects* para Internet de las Cosas de una manera fácil y rápida, como bien se había formulado en la primera hipótesis.

La comunicación entre los *Smart Objects* a través de Internet es uno de los objetivos de Internet de las Cosas. No obstante, no todas las personas tienen el conocimiento o el tiempo necesario para desarrollar la aplicación necesaria. Por eso, tenemos que traer ciertas instalaciones de la industria y a la vida diaria. Este es el propósito de Midgar, exactamente, con esta última investigación, MOCSL.

22. DETECCIÓN DE PERSONAS MEDIANTE EL USO DE VISIÓN POR COMPUTADOR PARA MEJORAR LA SEGURIDAD EN ESCENARIOS DE INTERNET DE LAS COSAS

«La función de un buen software es hacer que lo complejo parezca simple»
Atribuida a Grady Booch en: Frank H. P. Fitzek et al. (2010) Qt for Symbian. p. xv

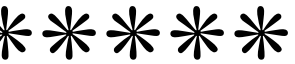
«Visión sin acción es soñar despierto, acción sin visión es una pesadilla»
Proverbio japonés

«No te preocupes de que la gente te pueda robar una idea. Si es original, se la harás tragar a la fuerza»
Howard Aiken, creador del Mark I

La automatización es uno de los caminos que busca Internet de las Cosas. Una de las maneras de conseguir esto sería mediante la inclusión de Visión por Computador con el fin de detectar automáticamente lo que se busca en un determinado sitio. Esto podría mejorar enormemente la vida diaria, sobre todo, en la vigilancia, pues esto permitiría analizar las imágenes automáticamente y solo requerir el análisis de la imagen por un humano para comprobar esa situación. Luego, esto permitiría mejorar la seguridad en las casas, los pueblos, las ciudades y la Tierra, de tal manera que se protegiesen automáticamente zonas necesitadas de vigilancia.

Previamente se ha visto cómo se pueden automatizar acciones en base a los datos que proporcionan los sensores. Aquí la duda que surge es si se pueden automatizar acciones en base a fotografías, o, dicho de otro modo y bajo el enfoque de este capítulo: ¿se pueden utilizar fotografías como si fueran sensores? Un problema que surge de esto sería el de cómo utilizar fotografías como sensores y cómo de eficaz y preciso sería este sistema.

Una posible solución, que es la propuesta en este capítulo, es incluir un módulo de Visión por Computador a la plataforma IoT Midgar y utilizar las fotos como si fueran un sensor. Posteriormente, se explica cómo de preciso es dicho módulo y cuan eficaz resulta utilizar las fotos como si fueran sensores.



Detección de personas mediante el uso de Visión por Computador para mejorar la seguridad en escenarios de IoT

22.1 *DESCRIPCIÓN*

22.1.1 **OBJETIVOS**

Las hipótesis de las que parte esta cuarta iteración son las siguientes:

- Es posible insertar el uso de Visión por Computador en Internet de las Cosas.
- Se pueden utilizar las fotografías de una cámara IP como si fuesen sensores.
- Se puede obtener una buena precisión para automatizar o semiautomatizar este tipo de eventos.

En esta cuarta iteración se ha buscado el permitir reconocer personas mediante el uso de Visión por Computador en diferentes escenarios de Internet de las Cosas para proveer a estos de una mayor seguridad. Los objetivos que se deben de cumplir son los siguientes:

- Interconectar una cámara IP con una plataforma IoT, como es Midgar.
- Crear un módulo de Visión por Computador capaz de reconocer personas en las fotografías.
- Tratar las fotografías como si fueran sensores.
- Ayudar en la mejora de la seguridad en zonas concretas.

22.1.2 **FUNCIONALIDADES**

Para poder cumplir los objetivos anteriormente marcados, se ha utilizado la plataforma Midgar como punto de interconexión de los *Smart Objects* y como cerebro de todo el proceso. En esta cuarta iteración se ha implementado un módulo de Visión por Computador a Midgar, el cual es capaz de discernir a una persona en una foto tomada por una cámara.

Este módulo recibe fotos de una cámara IP mediante un servicio web REST expuesto de Midgar, el cual recibe una foto. Según el transcurso del tiempo entre las fotos, el módulo es capaz de saber si estas pertenecen a una misma escena, en cuyo caso, simplemente con detectar a una persona en una de ellas envía una respuesta positiva.

Esto permite utilizar una foto o una secuencia de fotos como si de un sensor se tratase. Esto permite añadir una nueva dimensión a IoT, permitiendo recoger más datos sin necesidad de utilizar una red de sensores de proximidad o incluso de mejorar estos para solo detectar aquello que queramos, como, por ejemplo, una persona y así con un animal no funcionase, o viceversa.

Sin embargo, la Visión por Computador depende mucho del modelo realizado. Por esto, en esta iteración se ha utilizado el modelo de la parte del cuerpo superior, que se corresponde con la zona de los hombros y la cabeza de una persona, de la librería de Visión por Computador OpenCV. Además, para conseguir una mayor precisión, se ha implementado un método que utiliza toda la secuencia de detección de una persona, que se compone de varias fotos, en vez de cada foto individualmente. De esta manera, con que detecte una foto con una persona el resultado será positivo.

Esto proporciona un sistema que, junto a las iteraciones anteriores, permite utilizar las fotos recogidas por las cámaras IP como sensores con el fin de detectar lo que se desee, en el caso que nos ocupa, gente. Si se desea detectar perros, gatos, coches, cajas u otro tipo de cosas, simplemente habría que cambiar el modelo del módulo de Visión por Computador. De esta manera, se pueden crear zonas con una mejor seguridad y que no sean dependiente de sensores de proximidad, los cuales saltan positivamente con cualquier objeto. Además, una cámara IP necesita tener una persona vigilando el video de la cámara 24 horas al día, pero con el acercamiento aquí presentado se permite a esa persona simplemente revisar las fotografías detectadas positivamente cuando se reciban.

22.2 *ARQUITECTURA PROPUESTA*

La arquitectura de este sistema incluye la modificación de la capa 3 de la plataforma IoT Midgar a la vez que utilizará la capa 1 de MOISL, pues recordemos que es el DSL que permite interconectar objetos y elegir las circunstancias y condiciones sobre las que ocurrirá todo. Por esta razón, con MOISL se podría elegir la cámara IP con detección de fotos de personas. Esta arquitectura es mostrada en la Ilustración 107.

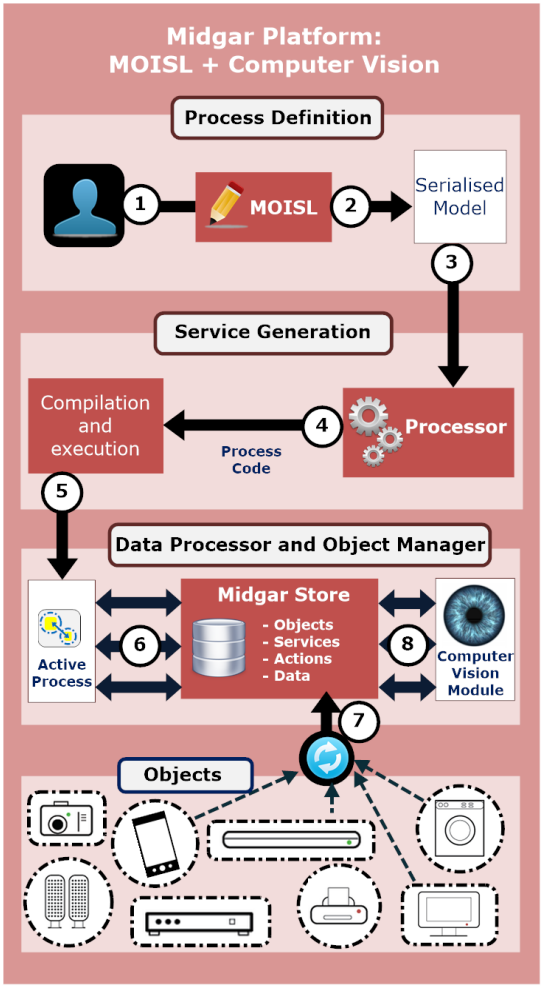


Ilustración 107 Arquitectura de Midgar con el módulo de Visión por Computador

Como bien se ha comentado, la capa 1 se corresponde con MOISL, lo que implica que en la capa 2 se utilice el procesador y generador de MOISL también.

La tercera capa de Midgar es la que recibe la modificación, siendo esta la primera iteración que la modifica. Como se ve, se incluye a la derecha de esta un módulo de Visión por Computador. Esta capa, que es la que tiene expuestos los servicios web REST recibe de la cámara IP las fotografías. Estas fotografías son enviadas al módulo de Visión por Computador, quien las analiza y se comunica directamente con la base de datos, introduciendo en ella los resultados de los análisis.

Por otro lado, la capa 4 de Midgar sigue siendo la misma, con los propios objetos que pueden registrarse, enviar datos y recibir acciones. La única diferencia aquí es el soporte a cámaras IP, las cuales ofrecen fotografías para analizar, pero no pueden recibir acciones, es decir, son meros sensores. Este soporte se ofreció a través de la exposión de un servicio para que las cámaras envíen fotografías.

22.2.1 **IMPLEMENTACIÓN**

Con esta iteración se añade el soporte de detectar cosas, según el modelo entrenador, en fotos y utilizar estas como sensores. En combinación con MOISL, esto permite que los usuarios puedan definir esto sin necesidad de programar, simplemente utilizar el DSL MOISL y el editor gráfico web de la plataforma IoT Midgar.

En este caso, la única capa que se ha modifica de Midgar es la capa número 3, pues la 1 y la 2 son las de MOISL (iteración 2). En esta capa número 3 se ha añadido el módulo de Visión por Computador y por ello será la única capa que se explica. Este apartado será el primero que se explica, describiendo así el ciclo de vida del módulo de Visión por Computador en Midgar, y explicando cómo funciona todo el proceso. Tras esto, se explica el funcionamiento interno del módulo de Visión por Computador, así como su configuración y como han sido ambos integrados en Midgar. Por último, se explicará en detalle los modos de funcionamiento y las opciones de la cámara IP Canon y como se ha configurado esta para que funcione como un sensor dentro de Midgar.

22.2.1.1 CICLO DE VIDA DEL MÓDULO DE VISIÓN POR COMPUTADOR EN MIDGAR

Para poder dotar de capacidad de utilizar fotos como sensores a Midgar, se ha requerido la implementación de un módulo de Visión por Computador. Con este módulo, Midgar puede recibir los XMLs que contienen los mensajes previamente ya descritos, o bien fotografías. Midgar es capaz de discernir entre ambos, mensajes y fotografías, mediante la detección del tipo de extensiones multipropósito de correo de Internet (MIME) contenida en cada mensaje recibido. Por ejemplo, un Arduino o un smartphone, los cuales ofrecen la posibilidad de insertarles una aplicación que se conecte con Midgar, son capaces de enviar mensajes utilizando el estándar XML de Midgar. No obstante, la cámara IP de Canon no permite modifica su software, al igual que ocurre con otras cámaras IP, pero si se puede analizar el tipo MIME del mensaje que envía para ver que es una imagen. Esto es lo que muestra la Ilustración 108.

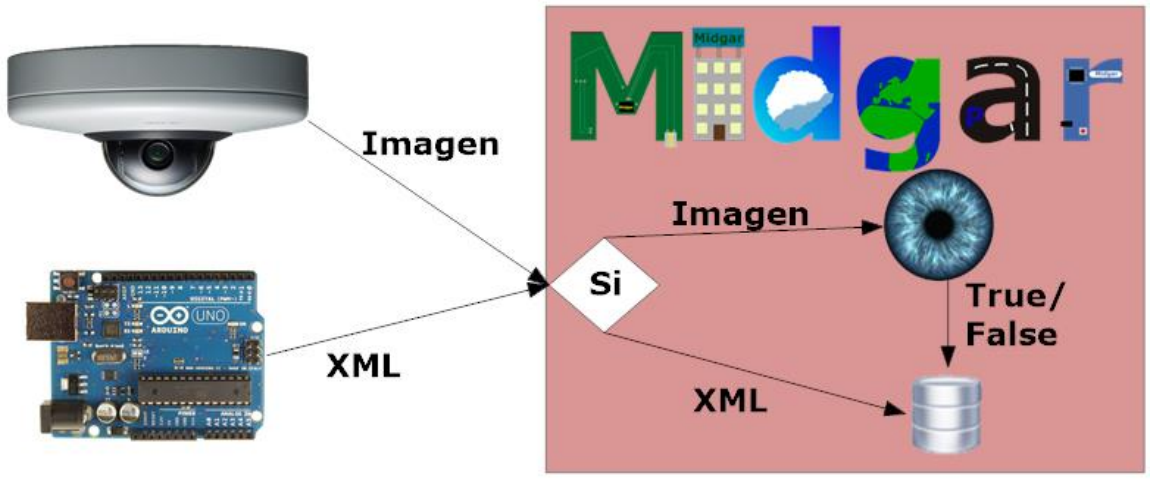


Ilustración 108 Ciclo de vida del módulo de Visión por Computador en Midgar

De esta manera, cuando Midgar recibe una imagen la guarda en una carpeta. Una vez han pasado 5 segundos sin recibir nuevas imágenes Midgar envía toda la secuencia al módulo de Visión por Computador, el cual analiza la carpeta entera, que contiene toda la secuencia de fotografías, en busca de al menos una persona en una de las fotos que componen toda la secuencia. En caso de encontrar una o más personas, el módulo de Visión por Computador responde con un «*True*» a Midgar, en caso contrario con un «*False*». De esta manera, Midgar almacena en la base de datos la respuesta como si se tratara de un sensor más.

22.2.1.2 MÓDULO DE VISIÓN POR COMPUTADOR

Se ha elegido la solución de crear un módulo de Visión por Computador separado de Midgar que ofrezca la posibilidad de llamarlo cuando se necesitase evaluar una imagen o bien una secuencia entera. Esta decisión ha sido debida a que de esta manera se tiene una implementación separada del módulo para así permitirnos ejecutar diferentes test con el mismo módulo para evaluar la propuesta, tanto bajo la arquitectura Midgar, como el módulo por separado. Lo único que se necesita es cambiar el parámetro de llamada al módulo para que este interactúe por separado o en toda la arquitectura.

El módulo de Visión por Computador ha sido desarrollado en Python para así poder utilizar la librería OpenCV. El uso de esta librería requiere usar la librería Numpy. El ciclo de vida del módulo consiste en cargar la imagen desde un fichero, convertirla a un array de bytes, transformarla a escala de grises y usar OpenCV para detectar el número de cuerpos existentes en la imagen. Si hay algún cuerpo, el módulo retorna el valor booleano «*True*», en otro caso «*False*». Sin embargo, se ha optado por mejorar el reconocimiento del módulo mediante el uso de secuencias de fotos en vez de utilizando una sola foto. De esta manera, se puede obtener una mejor precisión debido a que el objetivo de esta propuesta es detectar un movimiento peligroso en una determinada zona, no solo una cosa en una foto, es decir, nos importa todo el movimiento. Claramente, este uso final nos permite mejorar la precisión al solo importarnos detectar una persona en una secuencia de imágenes, pues no nos importa los falsos negativos, nos sirve con detectar solo un verdadero positivo.

Sin embargo, OpenCV necesita configurar unas pocas variables. La primera de esta es el factor de escala, el cual es necesario para crear una escala piramidal que el algoritmo usa para encontrar objetos a diferentes profundidades y que hemos configurado al valor «1.01». La segunda variable es el número mínimo de objetos que han sido detectados cerca unos de otros y así detectar todo el conjunto como un solo objeto. Esta segunda variable se ha configurado a «15». Por último, el parámetro del tamaño mínimo de cada ventana de detección se ha configurado en «(200, 200)». Sin embargo, estos valores que se han usado en la configuración de OpenCV dependen del contexto, pues no es lo mismo utilizarlos para detectar cosas dentro de un edificio, que en la calle o si se tiene mucha o poca profundidad.

Además, se requiere de un archivo XML externo porque OpenCV carga el clasificador desde un archivo externo con el fin de volver a utilizar el mismo código en distintos clasificadores. En nuestra propuesta, hemos decidido utilizar uno de los clasificadores de ejemplo que está disponible en los archivos descargados de OpenCV y cuyo nombre es «haarcascade_upperbody.xml». Con este clasificador, OpenCV es capaz de detectar la parte superior del cuerpo de las personas. Si se quisiera requeridas otras cosas, se debería de crear un nuevo clasificador mediante la extracción de las características necesitadas.

Sin embargo, este modelo es un ejemplo y es un poco débil. Por esta razón, si se quisiera mejorar el reconocimiento, habría que entrenar un nuevo modelo para obtener un clasificador mejor y más específico. No obstante, se podría mejorar la detección de movimiento en el caso que nos ocupa debido a que se analiza toda la secuencia para así obtener al menos una imagen. Por ejemplo, en el caso de que se quisiera tener un mejor clasificador, se podría aumentar el número de imágenes de la secuencia y así en vez de obtener un mínimo de una fotografía, requerir por lo menos tres o cinco fotografías con el objeto que se quisiera reconocer. Esto es muy útil si, por ejemplo, se quiere utilizar este sistema para detectar personas peligrosas, como ladrones en nuestra casa, o personas en algún área privada o peligrosa. En estos casos es preferible que el módulo de falsos positivos en vez de falsos negativos. La razón es porque si recibimos un falso positivo, simplemente se ha recibido una falsa alarma, pero en el caso contrario, si obtuviéramos un falso negativo, tendríamos a un ladrón en nuestra casa.

Como se ha comentado, con el fin de evaluar el módulo, se introdujo un parámetro que permite utilizarlo fuera de Midgar. De esta forma, el módulo utiliza la librería Flask para recibir las fotografías de la cámara, en vez de utilizar Midgar con Ruby on Rails. Además, en este modo, el módulo no devuelve un valor booleano, sino que salva la imagen que la cámara le ha enviado en un directorio, mientras que en otro directorio diferente guarda la misma imagen, pero recuadrando en verdad la detección del cuerpo en esa imagen, si es que ha detectado algo, si no ha detectado nada, guarda la imagen normal. Con esta información se ha hecho la evaluación de este módulo.

22.2.1.3 CÁMARA IP CANON

Se ha utilizado una Cámara IP Canon VB-S30D actualizada a la versión 1.2 de su firmware, que data del 26 de mayo de 2015. Esta cámara se conecta a la red por medio de un cable de Ethernet. La cámara es un *Smart Object* debido a que puede reconocer sus propios datos y puede tomar decisiones de acuerdo al video que graba. La cámara IP permite hacer streaming, para el que ofrece una URL, o enviar fotos o emails cuando la cámara detecta algún cambio en el video. Por ejemplo, es capaz de reconocer movimiento y modificación de los objetos en escena, si esto se ajusta correctamente, y así tras detectarlo enviar una fotografía de la escena o un email.

En la Ilustración 109 se muestran los 5 tipos de detecciones que la cámara ofrece. El primero de ellos es la «*moving object detection*» que consiste en detectar algún movimiento. El segundo es el «*abandoned object detection*» que detecta después de unos pocos segundos objetos nuevos o que han sido abandonados. Como tercera opción está «*removed object detection*», esta opción detecta después de algunos segundos la desaparición de un objeto. Otra opción más es la de «*camera tampering detection*», que sirve para saber si la cámara ha sido manipulada. Por último, la quinta opción es la de «*step detection*» que sirve para detectar movimiento sobre una línea que ha sido definido por el usuario previamente. Estos cinco eventos son configurables y permiten enviar una fotografía del momento exacto o un email cuando se lanza ese evento.

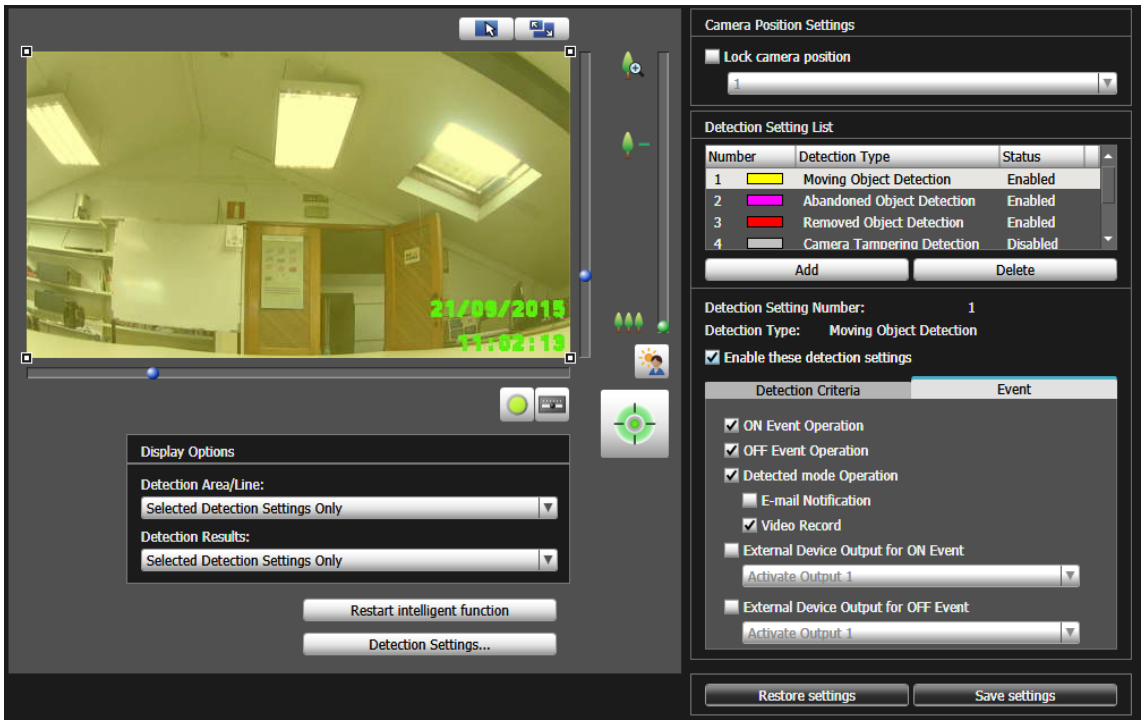


Ilustración 109 Cámara IP Canon con la «detección de objetos en movimiento» seleccionada

Para esta propuesta se han elegido dos de los cinco modos de la cámara. Se puede analizar el streaming o bien las fotografías que se reciben. Para trabajar con la cámara en Midgar, primeramente, se ha configurado la cámara IP usando Internet Explorer, configurando la IP y el puerto del servicio REST de Midgar, al cual la

cámara ha de enviar las fotos. Después de esto, se ha registrado la cámara IP en la plataforma para que esta pueda ser seleccionada en MOISL y así poder crear la interconexión y trabajar con la cámara.

22.2.2 SOFTWARE Y HARDWARE UTILIZADO

A continuación, se muestra el software utilizado para el desarrollo de este prototipo, comenzando por la implementación del sistema:

- La plataforma IoT Midgar:
  - Ruby 2.2.2p95.
  - Rails 4.2.1.
  - Servidor web Thin 1.6.3.
  - Base de datos MySQL 5.5.43.
  - MOISL, el cual ha sido desarrollado utilizando el canvas de HTML5 y JavaScript estándar.
  - Para la creación del generador de aplicaciones y la aplicación de conexión con el Arduino al ordenador se optó por utilizar Java 8.
- Módulo de Visión por Computador:
  - Se ha desarrollado utilizando Python 3.4.3.
  - Se ha utilizado la librería OpenCV 3.0.0 para aplicar Visión por Computador a las fotografías.
  - El módulo Numpy 1.10.0, que es requerido por Open CV.
  - Flask 0.10.1 para desarrollar un pequeño servidor de pruebas.
  - El modelo «haarcascase_upperbody.xml» como modelo ya entrenador para detectar personas.
- Cámara IP:
  - Canon VB-S30D con el firmware en la versión 1.2.0.
  - Internet Explorer 11 para acceder a la configuración de la cámara IP.

Para la evaluación de la propuesta se han utilizado los siguientes componentes hardware:

- Raspberry Pi 2 Model B dedicada con un sistema operativo Raspbian Jessie 4.4.13 v7.
- Tres smartphones Android:
  - Nexus 4 con Android 5.1.1.
  - Motorola con la versión 2.2.2.
  - Samsung Galaxy Mini S5570 con la versión 2.3.6.
- Microcontrolador Arduino Uno SMD basado en el ATmega328.
- Durante los diferentes test se han usado los siguientes sensores y actuadores: termistor TMP36, un altavoz, un servomotor, un motor DC, diferentes LEDs, dos botones, un fotorresistor, un sensor de temperatura y humedad DHT11, y un sensor de llamas KY026.

22.3 *EVALUACIÓN Y DISCUSIÓN*

Esta subsección contiene la explicación acerca de la metodología utilizada para la evaluación del prototipo de esta iteración, así como su evaluación y discusión. La evaluación contiene dos fases. La primera fase es la evaluación de fotografías tomadas de forma manual y sin pertenecer a una secuencia. Mientras, la segunda fase contiene la evaluación de fotografías tomadas automáticamente y que pertenecen a una secuencia.

22.3.1 **METODOLOGÍA**

El principal objetivo de esta evaluación es verificar si:

- Se puede insertar el uso de Visión por Computador en Internet de las Cosas
- Se pueden utilizar fotografías de una cámara IP como si fuesen sensores.
- Se puede obtener una buena precisión para automatizar o semiautomatizar este tipo de eventos.

Previamente, en la Implementación se ha demostrado que se puede incluir la Visión por Computador en Internet de las Cosas. Esto cumple ya con el primer punto de los tres presentados y ha permitido verificar las dos primeras hipótesis. Ahora, se va a explicar cómo se tratará de validar el segundo punto para así verificar si ha sido correcta o no la tercera hipótesis.

Para tratar de demostrarlo, se ha dividido la evaluación en dos fases que poseen el mismo objetivo: evaluar la previsión del módulo de Visión por Computador utilizando fotografías para detectar personas. En ambas fases se han utiliza fotografías sin gente y fotografías con gente. Así, se han utilizado fotografías sin gente para evaluar los falsos positivos y los verdaderos negativos. Por otro lado, con las fotografías con gente, se pueden obtener los falsos negativos y los verdaderos positivos.

Para ambas fases se ha utilizado el módulo sin la interacción de Midgar para así poder obtener la imagen con la detección recuadrada en verde para evaluar de forma cuantitativa si el módulo trabaja correctamente. Las dos fases son las siguientes:

- **Fase 1 – Fotografías manuales**: en esta primera fase se han utilizado las fotografías tomadas por la cámara IP Canon dentro del laboratorio de manera manual. En esta fase se ha testeado el módulo de Visión por Computador con fotografías aisladas y sin ninguna relación entre el resto.
- **Fase 2 – Fotografías automáticas**: esta segunda fase utiliza secuencias de fotografías tomadas de manera automática por la cámara IP Canon. Cuando la cámara detectaba movimiento en el laboratorio, esta, de forma totalmente automática, sacaba y enviaba fotos durante todo el movimiento al servidor web Flask que contenía el módulo de Visión por Computador de Midgar. Así, todas estas fotos enviadas de un mismo movimiento y con una pausa menor de 5 segundos entre cada foto formaban una secuencia de movimiento. En esta segunda fase se ha buscado comprobar si el utilizar secuencias mejora la precisión del módulo y si esto es aplicable a pesar de utilizar modelos de detección débiles.

Para ambas fases se ha utilizado el mismo modelo, el ejemplo de OpenCV llamado «haarcascade_upperbody.xml». Esto es importante debido a que se ha evaluado ambas fases utilizando las mismas condiciones para obtener una evaluación de verdad. Este modelo de ejemplo de OpenCV viene entrenado por defecto con la librería. Este modelo puede solo detectar la parte superior del cuerpo de la gente, es decir, sus hombros y su cabeza, y necesita de muy buenas fotos con la gente colocada en muy buena posición para ser capaz de detectarlas. Esto provoca que el módulo tenga una baja precisión debido a la necesidad de muy buenas fotos. En el caso de que se necesitara una mejor precisión habría que crear otro modelo entrenado desde cero para la situación requerida.

Para tomar las fotos que se han utilizado en esta evaluación se ha colocado la cámara en el medio del laboratorio, encima de una mesa, apuntando a la puerta de entrada como bien se puede ver en la Ilustración 110.

Ilustración 110 Situación de la cámara IP Canon en el laboratorio con el fondo que se utilizó para tomar las fotos

22.3.1.1 FASE 1 - FOTOGRAFÍAS MANUALES

Para esta primera fase se ha utilizado el modo manual de la cámara IP. Se han dividido las fotografías en dos carpetas en base a si tenían o no gente en ellas. Tras esto, se ha analizado cada carpeta por separado con el módulo de Visión por Computador. Lo que realiza aquí el módulo es crear una nueva carpeta para cada carpeta analizada en la que introduce únicamente las imágenes que detectase que contienen gente ya modificadas, es decir, con el recuadro alrededor de la o las personas detectadas.

Después de estos análisis, se ha revisado manualmente cada fotografía para analizar así los resultados y ver las fotografías analizadas correctamente e incorrectamente. En el caso de que en las fotografías con recuadro el rectángulo verde estuviera marcando una cosa incorrecta como un armario o una señal en lugar de una persona, se cuenta esta detección como errónea.

En esta fase se han utilizado un total de 160 fotografías, las cuales se dividen en 64 con personas y 96 sin personas. La Ilustración 111 muestra tres fotografías tomadas de forma manual con la cámara y que pertenecen a esta fase.

Ilustración 111 Varias fotografías de ejemplo tomadas manualmente

Así, se ha procesado manualmente todas estas fotografías después de su análisis con el módulo de Visión por Computador para poder detectar la precisión del módulo. Con este test se ha evaluado la precisión del módulo utilizado en base a fotografías sueltas.

22.3.1.2 FASE 2 - FOTOGRAFÍAS AUTOMÁTICAS

En la segunda fase se ha programado la cámara para enviar una fotografía cuando detecta algún tipo de movimiento en el área a la que enfoca. En este caso, la cámara enviaba una secuencia de fotografías desde el primer momento en que detectaba algún movimiento hasta la última detección de movimiento. Después de esto, se han analizado todas las secuencias obtenidas para obtener si es válido o no este método de uso de Visión por Computador con una cámara IP como sensor.

En la Ilustración 112 se muestra una secuencias de fotografías pertenecientes a una detección de movimientos por la cámara utilizada. El orden es de arriba a la izquierda hacia abajo a la derecha. Como se ve, la primera fotografía ha sido tomada cuando detectó a alguien entrar en su foco. El resto de fotografías fueron tomadas cada segundo, hasta que después de la sexta no hubo más movimiento. La cámara puede enviar desde una fotografía por segundo hasta un máximo de treinta. Sin embargo, para hacer la evaluación se ha seleccionado el valor máximo de la cámara, treinta. Como se puede apreciar, el número de fotografías depende del tiempo que dure el movimiento.

Ilustración 112 Secuencia de fotografías pertenecientes a un movimiento detectado por la cámara IP Canon

La cámara enviaba cada fotografía al servicio web creado en Python y Flask, con el fin de evitar la interacción de Midgar y poder evaluar únicamente el módulo de Visión por Computador. En este caso, se ha evaluado si era posible obtener una mejora del sistema, o al menos mantener el mismo nivel. Además, esta manera permite reducir el tráfico en red, reducir el proceso de computación y evitar utilizar la opción de *streaming*. Para esta fase se han utilizado 979 fotografías divididas en 17 eventos de movimientos, de los cuales 8 secuencias que contienen un total de 817 fotografías tienen movimiento de gente y las otras 9 secuencias compuestas de 162 fotografías tienen movimiento sin gente.

Hay que aclarar que las secuencias tienen muchas imágenes sin gente porque al utilizar el valor máximo, estas pueden contener solo una mano o un brazo en ellas. Así, estas secuencias contienen fotografías inválidas, pero que son parte de la secuencia. Por otro lado, para las secuencias negativas se ha utilizado movimiento de diferentes objetos en frente de la cámara, como son pelotas, tazas o papeles.

22.3.2 **RESULTADOS**

En esta sección se mostrarán los resultados. Para facilitar el análisis se han diseñado tablas para ambas fases. Estas tablas contienen los resultados de los test de las fotografías con el módulo de Visión por Computador clasificados en cuatro diferentes grupos:

- **Verdadero negativo**: fotografías sin gente con resultado negativo. Este es el mejor resultado para las fotografías sin gente porque esto significa que el módulo no detectó gente en fotos sin gente.
- **Falso positivo**: fotografías sin gente, pero con resultado positivo. Este es un caso erróneo donde encuentra gente en fotografías que no tienen gente debido a que el módulo dice que ha encontrado gente.
- **Falso negativo**: fotografías con gente con resultado negativo. Este es el peor caso de todo, pues se da cuando la fotografía tiene gente, pero el módulo no es la ha detectado.
- **Verdadero positivo**: fotografías con gente y con resultado positivo. Se da cuando la fotografía tiene gente y el módulo la encuentra.

A continuación, se presentará primero la fase donde se van a describir los resultados de las fotografías manuales. Tras esto, se mostrará la fase dos y sus respectivos resultados utilizando la cámara como sensor y los resultados de analizar las secuencias de fotografías.

22.3.2.1 FASE 1 - FOTOGRAFÍAS MANUALES

La Tabla 19 muestra los resultados de la evaluación de las 160 fotografías manuales. Se han analizado un total de 96 fotografías sin gente y de 64 fotografías con gente. En la Ilustración 113 se muestran tres fotografías pertenecientes tres casos de verdaderos positivos.

Tipos		**Gente**	
		Negativo (96)	Positivo (64)
Resultado	Negativo	Verdadero negativo 96 / 100%	Falso negativo 57 / 89,062%
	Positivo	Falso positivo 0 / 0%	Verdadero positivo 7 / 10,937%

Tabla 19 Resultados de la evaluación de las fotografías manuales

Ilustración 113 Tres fotografías pertenecientes a los casos verdaderos positivos

Analizando la Tabla 19 se pueden sugerir las siguientes interpretaciones:

- Se han utilizado 96 fotografías sin gente. El módulo analizó las 96 fotografías como negativas. Esto representa una precisión del 100%, la que resulta ser la mejor precisión para los casos negativos debido a que se corresponde con los casos verdaderos negativos.
- Para los casos positivos se han utilizado 64 fotografías. El módulo analizó 57 de estas, que representan el 89,062% del total, como casos negativos. Es decir, el módulo tiene un alto índice de fracaso ya que no detecta gente en fotos que si tienen gente.
- El módulo ha obtenido solo un 10,937% de éxito en los casos positivos. Esto indica una precisión muy baja. En estos casos, el modulo obtuvo 9 fotografías, pero tras analizarlo manualmente, se observó que dos de esas fotografías tenían recuadrada la zona errónea. Por ello, dos de esas nueve fotos han sido reclasificadas como falsos negativos. Estas dos imágenes se pueden ver en la Ilustración 114. Como se puede ver, en la primera fotografía detecta la camiseta y en la segunda una señal que hay a la derecha de la cabeza. El resto de casos contenían personas borrosas, personas que no estaban exactamente de frente a la cámara o fotografías con solo una parte del cuerpo.

- El módulo con el modelo utilizado no ha dado ningún falso positivo, lo que indica que tiene un 0% de error. Sin embargo, tiene una baja precisión de verdaderos positivos, lo cual es un problema.

Ilustración 114 Las dos fotografías que se reasignaron manualmente debido a una decisión errónea del módulo

22.3.2.2 FASE 2 - FOTOGRAFÍAS AUTOMÁTICAS

En esta segunda fase se han analizado las fotografías pertenecientes a las 17 secuencias. Se han utilizado 9 secuencias sin gente y 8 con gente. Se han analizado todas las fotografías, pero esta vez como parte de un todo, de la secuencia. En la Ilustración 115 se muestra un collage con fotografías pertenecientes a secuencias catalogadas como verdaderas positivas.

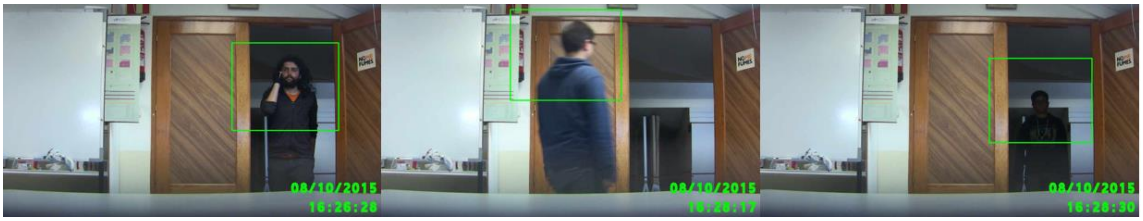


Ilustración 115 Collage con fotografías pertenecientes a las secuencias verdaderas positivas

En la Tabla 20 se muestran los resultados de aplicar el mismo módulo de Visión por Computador de la fase 1, pero a las secuencias de fotografías.

Tipos de acuerdo a la secuencia		**Gente**	
		Negativo (9)	Positivo (8)
Resultado	Negativo	Verdadero negativo 8 / 88,888%	Falso negativo 0 / 0%
	Positivo	Falso positivo 1 / 11,111%	Verdadero positivo 8 / 100%

Tabla 20 Resultados del análisis de las secuencias de movimiento por el módulo de Visión por Computador

En base a esta tabla se pueden sugerir las siguientes interpretaciones:

- Se han utilizado 9 casos negativos y se han obtenido 8 casos de verdaderos negativos. Esto representa un 88,888% del total, lo que supone un alto rango de detección en lo que se refiere a no detectar personas en fotografías sin personas.
- Solo se ha obtenido un falso positivo. Esto representa un 11,111%, lo que significa que el módulo ha reconocido un objeto como si fuera la parte superior del cuerpo de una persona. En la Ilustración 116 se puede ver la fotografía que hizo que la secuencia fuese marcada como falso positivo. Por lo que se ve, el módulo detectó como parte superior del cuerpo el área que contiene el asa de la taza, el pulgar que la sujeta y la parte superior de la puerta.
- Se han obtenido 8 casos de secuencias marcadas como verdadero positivo. Esto representa el 100% de precisión, lo que indica que y utilizando secuencias en vez de imágenes sueltas se mejora la precisión del módulo.
- El módulo ha tenido cero falsos negativos, lo que indica que al menos en todas las secuencias verdaderas ha detectado al menos una fotografía como persona.

Ilustración 116 Fotografía de la secuencia del falso positivo

La Tabla 21 muestra información acerca de cada una de las secuencias: nombre, si contiene gente o no, el número total de fotografías enviadas durante la detección de movimiento, el número de fotografías, el porcentaje de fotografías que el módulo ha detectado como fotografías con gente, y el resultado de acuerdo al módulo de Visión por Computador.

Nombre de la secuencia	**¿Tiene gente?**	**Total de fotos**	**Fotos identificadas**	**% de detección**	**Resultado**
C1	Sí	78	1	1,282	Verdadero positivo
C2	Sí	229	7	3,056	Verdadero positivo
D1	Sí	53	15	28,301	Verdadero positivo
D2	Sí	49	9	18,367	Verdadero positivo
D3	Sí	54	22	40,740	Verdadero positivo
D4	Sí	207	5	2,415	Verdadero positivo
L1	Sí	84	6	7,142	Verdadero positivo
L2	Sí	63	2	3,174	Verdadero positivo
C	No	12	0	0	Verdadero negativo
F	No	35	0	0	Verdadero negativo
M	No	9	0	0	Verdadero negativo
P	No	7	0	0	Verdadero negativo
P1	No	7	0	0	Verdadero negativo

Nombre de la secuencia	¿Tiene gente?	Total de fotos	Fotos identificadas	% de detección	Resultado
P2	No	25	0	0	Verdadero negativo
P3	No	17	0	0	Verdadero negativo
SP	No	30	0	0	Verdadero negativo
T	No	20	1	5	Falso positivo

Tabla 21 Información acerca de las secuencias

De acuerdo a los datos de esta tabla se pueden sugerir las siguientes interpretaciones:

- El porcentaje de detección no tiene relación con el número de fotografías: «C2» tiene 229 fotografías y solo ha obtenido el 3,052%, mientras tanto «D3» que cuenta con 54 fotografías ha obtenido un 40,740%. Esto depende de la calidad de las fotografías: la posición de la gente, su nitidez, o que la persona esté entera en la foto, pues a veces solo se ve un brazo o medio cuerpo.
- El mismo caso ocurre con los casos negativos, donde «F» tiene 35 fotografías con un 0% de detección y «T» tiene 20 fotografías con un 5% de detección.

22.4 *CONCLUSIONES*

En esta iteración se ha presentado una posible solución que incluye Visión por Computador en Internet de las Cosas. Esto podría permitir utilizar cámaras IP y fotografías como sensores. Además, esto permite abrir la puerta a aplicaciones similares como la automatización de cosas utilizando por ejemplo el reconocimiento óptico de caracteres (OCR), detección de caras o de gestos.

Por esto, se ha integrado en Midgar un módulo de Visión por Computador que permite analizar fotografías, o en nuestro caso, secuencias de fotografías, y retorna un valor booleano a la plataforma IoT como resultado, de tal manera que trabaja igual que si fuera un sensor. Sin embargo, como ha mostrado que, si se utiliza un modelo débil, el módulo puede tener una precisión baja. Por ejemplo, se ha obtenido en la primera fase un **89,092% de falsos negativos**, pero con un **100% de verdaderos negativos** cuando se han analizado las fotografías independientemente. Sin embargo, como se ha propuesto, si se utiliza un análisis en base a las secuencias de movimiento detectadas por la cámara, se obtienen mejores resultados: **100% de verdaderos positivos** y un **88,888% de verdaderos negativos**. Esta ha sido posible debido a que se centró el módulo de Visión por Computador en analizar toda la secuencia de movimiento. Sin embargo, este módulo si tuvo un fallo, pero fue un falso positivo. Así, para importantes cosas como detectar ladrones, es mejor, desde una

humilde opinión, que tenga falsos positivos en vez de falsos negativos, pues básicamente la diferencia sería tener un aviso de robo y que no haya nadie en casa, a no tener aviso y tener un ladrón. Claramente, este no es el mejor resultado, pero se podría mejorar con un mejor modelo.

Además, como se ha visto, el número de fotografías de la secuencia no afecta al resultado debido a que en algunos casos muchas fotografías tienen menos porcentaje de detección. En el caso de **«C2» con 229 fotografías y un 3,052% de fotografías identificadas**. Por el contrario, **«D3» tiene 54 fotografías y un 40,740%**. En este caso, la parte importante está en la calidad de las fotografías. Por esto, probablemente es mejor analizar secuencias de fotografías en vez de fotografías aisladas, como se ha demostrado. Así, de este modo, se torna verdadera la tercera hipótesis, pues, utilizando un modelo no perfecto se ha conseguido una buena precisión y ha permitido utilizar las fotografías como si fueran sensores, automatizando o semiautomatizando gran parte del proceso de vigilancia.

Por estos motivos, se puede decir que es posible usar la Visión por Computador en conjunto a Internet de las Cosas. Además, con una aplicación interesante, pues, como se ha demostrado se ha utilizado para usar fotografías como si se tratasen de sensores. En este caso se ha utilizado un modelo que no era el más perfecto y era débil, pero que no fue impedimento debido a que se han utilizado secuencias para mejorar esta debilidad. Sin embargo, es mejor y más recomendado utilizar un buen modelo debido a que este podrá mejorar la precisión, por ejemplo, en nuestro caso, un buen modelo mejoraría los falsos positivos, obteniendo así menos casos de identificación cuando no existen personas.

23. SEGURIDAD EN LA COMUNICACIÓN DE LOS *SMART OBJECTS* A TRAVÉS DE UNA PLATAFORMA DE INTERNET DE LAS COSAS

«La inteligencia consiste no sólo en el conocimiento,
sino también en la destreza de aplicar los conocimientos en la práctica»
Aristóteles

«El aprendizaje más importante proviene de la experiencia directa»
Nonaka & Takeuchi

«Un científico debe tomarse la libertad de plantear cualquier cuestión,
de dudar de cualquier afirmación, de corregir errores»
Julius Robert Oppenheimer

Privacidad: f. Ámbito de la vida privada que se tiene derecho a proteger de cualquier intromisión.

Como bien reza la definición de la Real Academia Española de la Lengua, la privacidad es el espacio comprendido de la vida privada de una persona que esta tiene derecho a proteger de cualquier intromisión. Sin embargo, como bien es sabido, esto no siempre es así en Internet. Unas veces roban datos de bases de datos de empresas, otras veces diferentes errores dejan libre parte de nuestros datos privados en las redes sociales o en sistemas de correo o mensajería, otras, nos *crackean* el ordenador.

Ahora bien, ¿si podemos perder nuestra privacidad en Internet, que ocurre en Internet de las Cosas? Internet de las Cosas es similar a ciertas aplicaciones de correo o de mensajería, las cuáles últimamente están introduciendo criptografía para asegurar los mensajes.

Por esto, una posible solución que aporte seguridad a los mensajes en IoT puede ser el uso de criptografía, siempre y cuando pueda ser usada bajo el tipo de tecnologías y aplicaciones que se utilizan en IoT. Pero, como suele ocurrir, hay muchos y muy diferentes algoritmos. Unos han sido vulnerados, otros son seguros, pero lentos. ¿Cuál de ellos o que combinación podría aplicarse?

Seguridad en la comunicación de los *Smart Objects* a través de una plataforma de Internet de las Cosas

23.1 *DESCRIPCIÓN*

23.1.1 **OBJETIVOS**

Esta quinta iteración se centró en la seguridad y privacidad de los usuarios que utilicen *Smart Objects* de tal modo que se envíen mensajes seguros entre los objetos y una plataforma IoT. La hipótesis aquí es la siguiente: obtener un buen algoritmo de encriptación para asegurar la conexiones entre objetos en Internet de las Cosas en medios no seguros.

Para comprobar esta hipótesis se han propuesto los siguientes objetivos a cumplir en esta iteración buscando así proporcionar la seguridad requerida:

- Crear mensajes seguros en un entorno no seguro, lo que implica que estos ofrezcan una serie de Cualidades de los mensajes: privacidad y confidencialidad, autenticación, integridad y no repudio.
- Utilizar la Criptografía híbrida para lograrlo
- Elegir el algoritmo criptográfico correcto entre la variedad existente.
- Que el algoritmo sea aplicable a la capacidad de cómputo disponible en los objetos.

23.1.2 **FUNCIONALIDADES**

De acuerdo a los objetivos marcados, se ha utilizado la plataforma Midgar para implementar el sistema de mensajes seguros en entornos inseguros, pues funciona por el protocolo HTTP. Así, en esta quinta iteración, tras haber realizado las pruebas pertinentes y encontrado los algoritmos criptográficos adecuados para utilizar criptografía híbrida, estos se han implementado en Midgar.

Los algoritmos se encuentran tanto en el servidor, Midgar, como en el cliente Android. Estos, deben de intercambiar su clave pública en el registro, la cual utilizarán después para cifrar sus mensajes, pues utilizan un sistema de Clave pública o criptografía de clave asimétrica.

El uso de criptografía pública permite a los objetos ser identificados y que su mensaje solo sea leído por el destinatario. Esto permite asegurar las conexiones en entornos no seguros, de tal forma que permite crear un sistema de mensajes, al menos teóricamente, invulnerable.

No obstante, no solo se ha utiliza la criptografía pública, sino también otros sistemas criptográficos para conseguir cumplir las cualidades requeridas para que un mensaje sea seguro. Esta combinación consta del uso de criptografía híbrida, que incluye la Clave secreta o criptografía de clave simétrica y la asimétrica, la Firma digital y Función Hash criptográfica. Así, la combinación de todos estos ha de procesarse en un tiempo acorde con el disponible para enviar los mensajes, pues, si se necesita mucho tiempo o incluso más tiempo del disponible, el sistema no serviría debido a que no es rentable para enviar datos dentro de un tiempo dado.

Seguridad en la comunicación de los *Smart Objects* a través de una plataforma de Internet de las Cosas

23.2 *ARQUITECTURA*

Esta vez, como viene siendo habitúan, se ha vuelto a utilizar la plataforma IoT Midgar. La arquitectura de esta coincide totalmente con la propuesta en MOISL, la segunda iteración, que fue vista en el capítulo 20.2. Consta de dos diferencias.

La primera es que en el registro de los objetos se intercambian ahora la clave pública del objeto y de la plataforma, con el fin de que puedan encriptar los mensajes el uno para el otro. La segunda diferencia reside en que esta vez se ha introducido encriptación en el paso de mensajes, y por ello, cuando son recibidos, ya sea por el objeto o Midgar, estos deben de ser desencriptados utilizando la clave privada para después dar paso a su procesado.

En este apartado se explicarán estos pasos divididos en un total de 4: el registro del objeto, el envío del mensaje del cliente al servidor, como envía un mensaje Midgar al objeto destinatario, y como se recibe y se lee este mensaje final por el objeto.

23.2.1 **IMPLEMENTACIÓN**

La solución propuesta utiliza tanto la criptografía híbrida como la firma digital y las funciones Hash. Cabe resaltar que el uso de estas dos criptografías incluye el uso de criptografía secreta y pública, por parte del primero, y de funciones hash por parte del segundo. Con esto, se ha buscado obtener los beneficios de ambas, y en eliminación de sus contras, con el fin de crear mensajes seguros.

A continuación, se mostrará la arquitectura dividida en las 4 fases que conforman el sistema seguro de mensajes, explicando así la comunicación entre las diferentes partes y cómo funciona todo el proceso.

23.2.1.1 REGISTRAR OBJETO

La primera fase consiste en registrar el objeto en la plataforma IoT, como se muestra en la Ilustración 117.

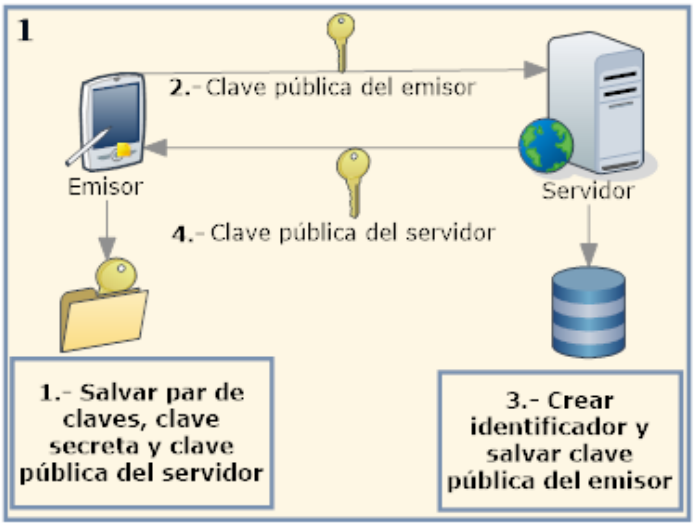


Ilustración 117 Fase 1: registrar objeto

Seguridad en la comunicación de los *Smart Objects* a través de una plataforma de Internet de las Cosas

En esta fase es donde Midgar sufrió su primera modificación. Aquí, además del mensaje de registro, se adjunta la clave pública del objeto que se registra, y en el mensaje de confirmación este objeto recibe la clave pública de Midgar. Esto permite que, en base al uso de la criptografía pública, cada uno sea capaz de encriptar el mensaje con la clave pública del otro de tal manera que solo el poseedor de la clave privada que hace pareja con esa clave pública pueda leer el mensaje. Además, en cada mensaje se crea una clave secreta que se utiliza para encriptar los mensajes mediante encriptación híbrida y que cambia continuamente para mejorar la seguridad.

Los pasos del registro son los siguientes:

1. El objeto que se registra su par de claves pública y privada, almacenándolas en su memoria interna.
2. El objeto al registrarse envía su clave pública a la plataforma.
3. El servidor almacena la clave pública del objeto junto a su identificador.
4. En la respuesta de confirmación del servidor, este adjuntas y clave pública para que el objeto pueda enviar mensajes encriptados.

23.2.1.2 MENSAJE DEL OBJETO EMISOR AL SERVIDOR

La siguiente fase se da cuando el objeto registrado envía un mensaje al servidor para así poder comunicarse con otro objeto. Las fases de este segundo paso se muestran en la Ilustración 118.

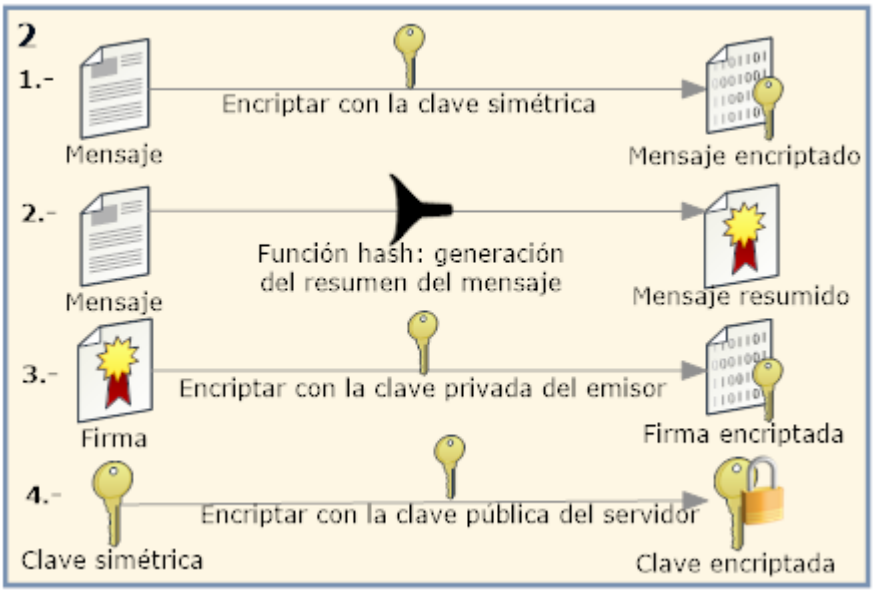


Ilustración 118 Fase 2: envío de un mensaje del objeto emisor al servidor

Los pasos mostrados en la ilustración anterior son los siguientes:

1. En primer lugar, el objeto encripta el mensaje a enviar utilizando una clave secreta mediante criptografía simétrica, y que es nueva en cada mensaje que se envía. Esta clave es la misma que se debe de utilizar para desencriptar el mensaje.
2. Tras esto, el objeto hace un resumen del mensaje utilizando una función hash de tal modo que se tenga una constancia resumida del mensaje original que permita garantizar la integridad del mensaje enviado.
3. El siguiente paso consta de incrustar la firma del objeto emisor en el mensaje original utilizando su clave privada para firmar. Esta firma contendrá el resumen del mensaje. Se puede comprobar que es su clave privada desencriptando la firma con su clave pública, pues el usuario original debería de ser el único en poder firmar con la clave privada, pues esta es única e intransferible.
4. Como último paso, el objeto emisor encripta la clave secreta utilizada para encriptar este mensaje con la clave pública de quien será el receptor, en este caso, la plataforma IoT Midgar, otorgándole así la capacidad de ser la única capaz de obtener la clave secreta que se adjunta en el envío para desencriptar el mensaje encriptado con la clave secreta.

23.2.1.3 RECEPCIÓN Y ENVÍO DEL MENSAJE POR EL SERVIDOR AL OBJETO RECEPTOR

Esta tercera fase incluye el envío del mensaje, el cual fue enviado por el objeto emisor, desde el servidor hasta el objeto destinatario o receptor. Pues, recordemos que el servidor, Midgar, es el cerebro de toda la operación y el encargado de decidir qué y cuando los objetos deben de hacer las acciones estipuladas. Luego, todo debe de pasar por él y debe de ser capaz de tener acceso a todo. Esta fase se desglosa en la Ilustración 119.

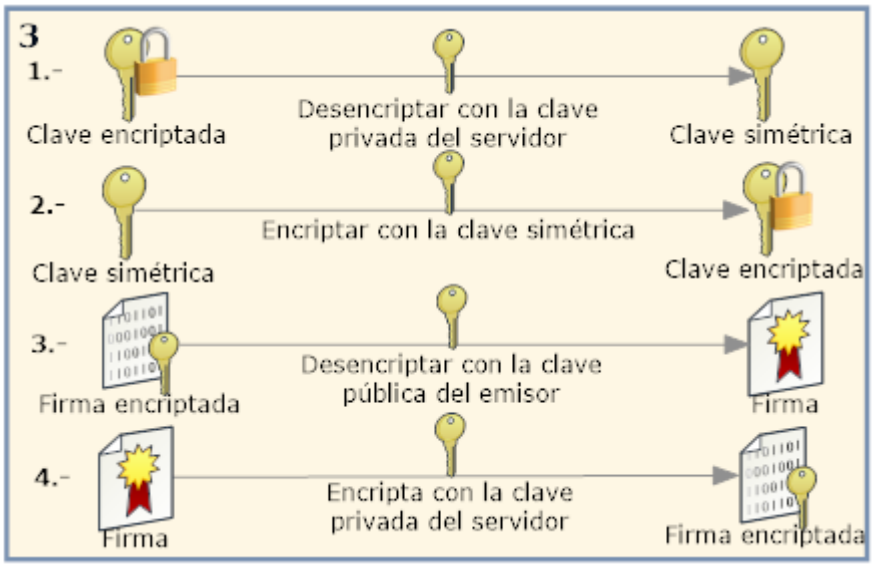


Ilustración 119 Fase 3: envío de un mensaje de servidor al objeto receptor

Los pasos mostrados en la Ilustración 119 son los siguientes:

Seguridad en la comunicación de los *Smart Objects* a través de una plataforma de Internet de las Cosas

1. Primeramente, el servidor ha de desencriptar la clave secreta utilizando su clave privada, para así poder acceder al contenido del mensaje, que es donde se encuentra la clave secreta y el mensaje encriptado con esta última clave.
2. Se encripta la clave simétrica que se usará para encriptar el mensaje recibido cuando se encripte de nuevo al objeto destinatario.
3. Desencripta la firma encriptada utilizando la clave pública del emisor para obtener la firma original.
4. Comprueba que la firma se corresponda con la clave privada del objeto emisor desencriptándola con su clave pública y así obtener el resumen del mensaje realizado por el objeto emisor.
    a. Realiza un hash del mensaje desencriptado y comprueba que este nuevo hash sea igual que el resumen adjuntado en la firma del mensaje.
    b. En caso de que la comprobación falle, se rechaza el mensaje. En caso contrario, se procesa el mensaje y se prepara el nuevo mensaje.

Después encripta el nuevo mensaje a enviar al objeto destinatario utilizando una nueva clave secreta. Encripta el resumen del mensaje que se va a enviar con su clave privada, es decir, la clave del servidor, introduciendo de esta forma la firma al mensaje y garantizando así que el objeto receptor pueda realizar el mismo proceso de comprobación.

23.2.1.4 RECEPCIÓN Y DESCENCRIPTACIÓN DEL MENSAJE POR EL OBJETO DESTINATARIO

En esta última fase el objeto destinatario recibe el mensaje de la plataforma Midgar con las acciones que debe de realizar. Esta fase se compone de 4 pasos que se muestran en la Ilustración 120.

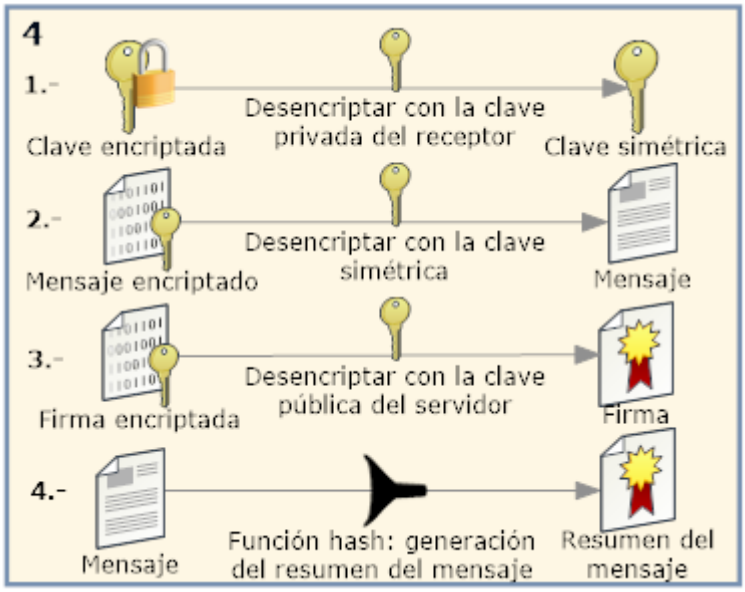


Ilustración 120 Recepción y desencriptación del mensaje por el objeto receptor

Los pasos de esta fase son los siguientes:

1. El objeto receptor desencripta la clave simétrica utilizando su clave privada
2. El mensaje se desencripta utilizando la clave simétrica obtenida en el paso previo.
3. Desencripta la firma utilizando la clave pública del servidor para así obtener el resumen. En caso de obtener, entonces ha sido capaz de comprobar su autoría.
4. Crea un resumen del mensaje recibido para poder comparar que sea exactamente igual al resumen obtenido en el paso anterior con el fin de comprobar su integridad.

23.2.2 SOFTWARE Y HARDWARE UTILIZADO

A continuación, se muestra todo el software utilizado durante el desarrollo de este prototipo:

- La plataforma IoT Midgar:
  - Ruby 2.2.2p95.
  - Rails 4.2.1.
  - Servidor web Thin 1.6.3.
  - Base de datos MySQL 5.5.43.
  - Java 8 para el demonio que interconecta los dispositivos.
- Para evaluar y usar los diferentes algoritmos criptográficos se han utiliza las siguientes librerías:
  - Librería Security[216] y Crypto[217] de Java.
  - Librería Bouncy Castle[218].

Para la evaluación de la propuesta se han utilizado los siguientes componentes hardware:

- Máquina virtual sobre VMware con Ubuntu 14.04 LTS.
- Tres smartphones Android:
  - **Dispositivo 1**: HTC Desire 610 con Android 4.4.2 y un procesador de cuatro núcleos a 1.2 GHz y un 1GB de RAM.
  - **Dispositivo 2**: LG Optimus L9 con Android 4.1.2 y un procesador de dos núcleos con una velocidad de un 1GHz y 1GB de RAM.
  - **Dispositivo 3**: LG Optimus L7 con Android 4.0 y un procesador de un núcleo a 1GHz y 512MB de RAM.

[216] Java Security: http://docs.oracle.com/javase/7/docs/api/java/security/package-summary.html
[217] Java Crypto: http://docs.oracle.com/javase/7/docs/api/javax/crypto/package-summary.html
[218] Bouncy Castle: https://www.bouncycastle.org/java.html

Seguridad en la comunicación de los *Smart Objects* a través de una plataforma de Internet de las Cosas

23.3 *EVALUACIÓN Y DISCUSIÓN*

Esta subsección explica la metodología utilizada en la evaluación de esta iteración, así como la evaluación y su discusión.

Primeramente, se explica el procedimiento de selección de algoritmos y como se ha realizado la toma de tiempos para continuar con la explicación de cómo se ha llevado a cabo la evaluación de todos estos.

23.3.1 **METODOLOGÍA**

El principal objetivo de esta evaluación es validar la hipótesis inicial que consiste en si se puede obtener un buen algoritmo de encriptación para asegurar la conexiones entre objetos en Internet de las Cosas en medios no seguros.

Para lograr esto, primero se ha de elegir entre varios algoritmos disponibles en los diferentes sistemas criptográficos existentes, es decir, en la criptografía simétrica, en la criptografía asimétrica y en las funciones hash criptográficas.

Tras esto, se deben de implementar los diferentes algoritmos y realizar una toma de tiempo que permita ver el rendimiento de cada algoritmo en diferente hardware, para así permitir tomar una decisión acorde a estos tiempos necesarios para encriptar las comunicaciones.

23.3.1.1 SELECCIÓN DE ALGORITMOS

Primeramente, se llevó a cabo la búsqueda de los diferentes algoritmos criptográficos existentes más utilizados en cada tipo de criptografía. De estos algoritmos se tuvo en cuenta su rendimiento, su tamaño de clave, uso y popularidad. De todos los algoritmos investigados se seleccionó un grupo, el cual fue implementado para ser evaluarlos. Además, los algoritmos seleccionados debían de disponer de una implementación disponible para los sistemas donde iban a ser utilizados, lo que obligaba a que dispusiesen de su implementación en Ruby, debido al servidor, y en Java, por los smartphones Android.

23.3.1.1.1 CRIPTOGRAFÍA SIMÉTRICA

Los algoritmos que se seleccionaron para evaluar referentes a la criptografía simétrica fueron los siguientes:

- **Data Encryption Standard (DES)** [381] es un algoritmo de cifrado que fue escogido como estándar en 1976. Fue un algoritmo dudoso en sus inicios, con un tamaño de clave corto y continuas sospechas sobre posibles puertas traseras por parte de la National Security Agency (NSA) de los Estados Unidos de América. El tamaño máximo de claves es de 56 bits resultando inseguro y endeble, por ejemplo a ataques de fuerza bruta [383]. Esto provocó que se creara su sucesor, el algoritmo Triple DES.

- **Triple DES (3DES)** [382] surgió debido a las dudas sobre el tamaño de clave insuficiente del algoritmo DES. Por ello emplea el triple de tamaño de clave, que es de 168 bits. Es mucho más resistente a ataques que el algoritmo DES debido a su longitud de clave, pero su uso está quedando relegado.
- **Advanced Encryption Standard (AES)** [379], [380] es un algoritmo de cifrado por bloques adoptado también como estándar de cifrado. Ha sustituido a los algoritmos DES y Triple DES como nuevo estándar de cifrado, tanto para hardware como para software, siendo posible implementarse en poco tiempo, requiriendo poca memoria y siendo más rápidos que estos dos. Su tamaño de clave oscila entre un mínimo de 128 bits y un máximo de 256 bits.
- **International Data Encryption Algorithm (IDEA)** [385] es un algoritmo de cifrado por bloques creado y propuesto como sustituto del algoritmo DES, trabajando con una longitud de clave de 128 bits. Es considerado uno de los algoritmos más seguros por su fortaleza ante el ataque por fuerza bruta. No obstante, está protegido por patente [383].
- **Blowfish** [383], [384] es un algoritmo de cifrado por bloques simétricos que utiliza bloques de 64 bits y claves que van desde los 32 bits a los 448 bits y que fue presentado por Bruce Schneir en el año 1993. Blowfish es un algoritmo que surgió para reemplazar al algoritmo DES, con sus problemas de seguridad.

23.3.1.1.2 CRIPTOGRAFÍA ASIMÉTRICA

Respecto a los algoritmos de clave asimétrica que se eligieron, donde se buscó un manejo de claves superior y velocidad, fueron los siguientes, pues se utilizan tanto en criptografía pública e híbrida, como en firma digital:

- **Rivest, Shamir, Adleman (RSA)** [388] es el algoritmo de criptografía asimétrica más ampliamente utilizado. El algoritmo RSA trabaja con longitudes de clave que van desde los 1024 bits a los 4096 bits siendo tamaños prácticamente imposibles de resolver por fuerza bruta en la actualidad.
- **ElGamal** [389] es un algoritmo de criptografía asimétrica basado en la idea de Diffie-Hellman, cuya seguridad se basa en el empleo del logaritmo discreto. ElGamal permite utilizar tamaños de claves que van desde los 1024 bits hasta los 4096 bits.

23.3.1.1.3 FUNCIONES HASH CRIPTOGRÁFICAS

Por último, se han evaluado diferentes funciones Hash para la realización de los resúmenes de los mensajes. Entre estas funciones Hash se han evaluado aquellas que dispusieran de implementación en ambas plataformas, cumplan las reglas necesarias para ser funciones hash y sean utilizadas. A continuación, se muestras las elegidas:

- **Message-Digest Algorithm** (**MD5**) [393] fue diseñado por Ronald Rivest en el MIT en 1992 para reemplazar MD4 [796]. MD5 posee una gran difusión a nivel actual y dispone de implementaciones en muchos GPLs, ofreciendo un resumen de 128 bits. Sin embargo, posee problemas ya detectados de colisión de hash que pueden comprometer su seguridad [797], lo que hace que sea considerado como inseguro por muchos criptógrafos [798].
- **MD6** [394] ha sido la siguiente iteración de esta familia de funciones hash realizada por Ronald Rivest y que fue presentado en el año 2008 como propuesta como nuevo algoritmo de SHA-3. Una de las características de MD6 era la longitud de salida variable que podía proporcionar. No obstante, debido a ciertos problema explicados por Rivest al NIST en [799], MD6 no estaba aún listo, pues entre otras cosas, era más lento que SHA-2.
- **SHA-1** [395]–[397] posee un amplio uso y ha sido el segundo estándar del NIST, propuesto en el año 1995, adoptado por los Estados Unidos de América como estándar para realizar funciones Hash con un resumen de 160 bits y sustituir así a SHA-0 (1993). No obstante, su seguridad está en entredicho debido a un conjunto de ataques divulgados en el año 2005 sobre una estructura similar a la empleada en SHA-1 y que podría comprometer su seguridad [796]. Además, compañías importantes como Microsoft, Google y Mozilla recomendaron en el año 2010 cambiar SHA-1 por alguna de sus versiones posteriores, así como algunos criptógrafos aconsejan usar las versiones posteriores [798].
- **SHA-2** es el sucesor de SHA-1 como estándar elegido por el NIST [396]. SHA-2 comenzó ofreciendo cuatro tamaños de bits, los cuales eran 256, 384 y 512 bits [800]. Posteriormente, el NIST sacó la versión de 224 bits [801]. SHA-2 es más rápido en algunas plataformas que SHA-3 [802]. Sin embargo, SHA-2 ya ha sido vulnerados de diferentes maneras en los últimos años, a pesar de que aún es recomendado, junto a SHA-3 por el NIST [798].
- **SHA-3** [398] es el algoritmo de reducción criptográfica estándar que surgió como alternativa criptográfica frente a SHA-2, sin pretender reemplazarlo, debido a los ataques teóricos respecto a las versiones anteriores. Tiene su uso sobre tamaños que son de 224, 256, 384 y 512 bits, y dos funciones especiales llamadas SHAKE125 y SHAKE256 que permiten extender la salida a la longitud deseada, siendo estas dos funciones las primeras Funciones de Salida Extensible (XOF) estandarizadas por el NIST.

En la siguiente tabla, la Tabla 22, se muestra un resumen de las propuestas elegidas a valorar, así como de sus implementaciones previas más inmediatas. Como se ha comentado, únicamente MD6 y SHA-3 no han sido vulnerado hasta el momento actual.

Algoritmo		Salida (bits)	Año de aparición	¿Vulnerado?
MD4		128	1990	Sí: *preimage* [803], colisiones [804]
MD5		128	1992	Sí, colisiones [797], [804], [805]
MD6		Variable	2008	No
SHA-0		160	1993	Sí, colisiones [806]
SHA-1		160	1995	Sí, colisiones [397], [796]
SHA-2	*SHA-224*	224	2004	Sí: *preimage* [807]
	SHA-256	256	2001	Sí: colisión [798], [808], *preimage* [803], [807], [809], pseudocolisión [810]
	SHA-384	384	2001	Sí: *preimage* [807]
	SHA-512	512	2001	Sí: colisión [808], *preimage* [803], [807], [809], pseudocolisión [802]
	SHA-512/224	224	2012	Sí, colisión [811]
	SHA-512/256	256	2012	Sí, colisión [811]
SHA-3	224/256/384/512	224/256/384/512	2015	No
	SHAKE128/256	Extensible	2015	No

Tabla 22 Resumen de las funciones hash criptográficas estudiadas

No se utilizó MD6, a pesar de valorarse en esta investigación, debido a la falta de una implementación para Java de este algoritmo.

23.3.1.2 TOMA DE TIEMPOS

Se ha desarrollado una aplicación nativa para smartphones con sistema operativo Android. Esta aplicación lleva implementados los diferentes algoritmos criptográficos previamente vistos de forma que esta aplicación es capaz de encriptar utilizando estos algoritmos los diferentes mensajes que se han de enviar a la plataforma Midgar.

En el Código fuente 24 se muestra el mensaje de ejemplo utilizado en la evaluación, pues siempre ha sido el mismo mensaje. Las acciones principales a realizar por cada algoritmo ha sido la generación automática de claves y el encriptado y el desencriptado de mensajes.

```
<send>
  <data>
      <datum>28</datum>
      <service>18</service>
  </data>
</send>
```

Código fuente 24 Ejemplo de mensaje de datos de Midgar

Por otro lado, para realizar las mediciones, se ha encriptado una contraseña simétrica generada en nuestra aplicación y que se muestra en el Código fuente 25 y que consta de un tamaño intermedio de 128 bits (16 bytes). Esta contraseña es la utilizada para encriptar de manera simétrica el mensaje anterior (Código fuente 24) en todas las ocasiones y así que no afecte el tiempo de generación de la contraseña al tiempo de encriptación.

```
W1PnUCA+fVEAOaKaxfu1cQ==
```

Código fuente 25 Una contraseña secreta de ejemplo de las utilizadas para la encriptación simétrica

Para calcular el consumo de tiempo, en base a la arquitectura presentada previamente, se han hecho dos fases. Esto se hace debido a que las claves solo hace falta calcularlas una vez y puede que no sea tan relevante este tiempo en comparación con el tiempo de encriptado y desencriptado del mensaje, que se hace múltiples veces. Las fases son las siguientes:

- Primeramente, se destaca que solo se cuenta el tiempo de generación de claves en una ocasión, que es cuando el objeto se registra como nuevo dispositivo. Esto se debe a que en ese momento se generan el par de claves pública-privada necesarios para la criptografía asimétrica, pues en el registro es el momento en el que el objeto envía, además de sus datos, su clave pública. Además, en este momento recibe también la clave pública del servidor Midgar. Mientras, la clave privada es guardada en memoria para poder desencriptar los mensajes que le envíen.
- Por otro lado, para calcular el tiempo de cálculo del encriptado y desencriptado este se realizada con un mensaje creado para la plataforma Midgar. Siempre se utiliza el mismo mensaje con el fin de que siempre se encripte el mismo número de caracteres y esto no afecte a los tiempos de encriptado y desencriptado. Este mensaje se muestra en el Código fuente 24. En esta fase se hicieron dos tipos de pruebas, las cuales fueron llamadas de nodos críticos y no críticos. De esta

forma se ha intentado dividir en dos los diferentes tipos de sensores y aplicaciones existentes, aquellas que necesitan enviar cada pocos segundos y aquellas que no.

- **Nodo crítico**: estos nodos son aquellos que necesitan enviar muchos mensajes en poco tiempo debido al uso que dan a sus sensores. Aquí, por ejemplo, estarían aplicaciones que hagan uso del acelerómetro, del sensor de gravedad o del giroscopio. Estos sensores se suelen utilizar en aplicaciones que intentan prevenir accidentes o avisar en caso de que ocurra uno. Por estos motivos, es muy importante que envíen medidas del sensor cada pocos segundos, como se puede ver en [812]–[814]. Para este tipo de nodos se ha establecido una medida de un mensaje cada 4 segundos.
- **Nodo no crítico**: estos nodos son aquellos que no requieren una comunicación tan crítica y en los que no habría un gran impacto en el uso de la aplicación si se perdiese un mensaje, hubiera un error en la comunicación o tardase unos segundos de más. Ejemplos de estos sensores son los fotorresistores, un termistor o de presión. Para este tipo de nodos se ha estimado un mensaje cada minuto.

Para la toma de tiempos se han utilizado tres dispositivos móviles con sistema operativo Android, siendo cada uno de una gama diferente, es decir, un smartphone es de gama baja, otro de gama media y el último de gama alta. De esta manera se ha buscado ver el impacto de la computación de este sistema de seguridad en diferentes dispositivos existentes en las tres gamas existentes en el mercado.

Para calcular los tiempos, en ambas fases se hicieron 10.000 ejecuciones para cada cálculo y después se hizo la media de estas. De esta manera se ha buscado evitar casos especiales en el que la toma de tiempo llevase más, o menos, tiempo en un determinado caso en concreto, como puede ser que saltase el recolector de basura, o posibles pausas o bloqueos no deseados del sistema operativo.

23.3.2 **RESULTADOS**

Esta sección muestra los resultados obtenidos para los diferentes tipos de algoritmo en los diferentes dispositivos móviles de diferente gama y los discute mediante el uso de gráficas que reflejan los resultados y las comparaciones de los algoritmos.

23.3.2.1 CRIPTOGRAFÍA SIMÉTRICA

En primer lugar, se presenta la evaluación de los algoritmos de criptografía simétrica que se han seleccionado y que son: DES (64), 3DES (128), AES (128), AES (256), Blowfish (128), Blowfish (256), IDEA (128) e IDEA (256).

No todos los algoritmos gozan de las mismas condiciones iniciales, ya que no todos soportan los mismos tamaños de clave y, en el caso del algoritmo IDEA, no permite generar una clave aleatoria, por lo que sus valores de cálculo de clave no son incluidos. Para realizar los cálculos en la tabla, se emplean los tiempos del cálculo de clave de AES en IDEA.

En la Tabla 23 se muestran los resultados obtenidos en los tres dispositivos empleados, desglosados en cada algoritmo utilizado y mostrado el tiempo tardado en crear la clave, en encriptar y desencriptar el mensaje

mostrado en el Código fuente 24 utilizando la clave secreta de Código fuente 25, la suma total, y el tiempo total de 1 hora de comunicación de un nodo crítico y uno no crítico. Para este último cálculo se ha sumado el tiempo que tardaría en encriptar y desencriptar un nodo crítico y uno no crítico, sin incluir el cálculo de la clave ya que su cálculo solamente se realiza durante el registro del dispositivo.

	Algoritmo	Tamaño (bits)	Clave (ms)	Encriptación (ms)	Desencriptación (ms)	Total (ms)	1 hora de comunicación			
							Crítico (ms)	%	No crítico (ms)	%
Dispositivo 1	DES	64	0,0355	0,1575	0,1328	0,3258	4.180,32	0,1161	17,42	0,0005
	3DES	128	0,0368	0,2739	0,2649	0,5756	7.758,72	0,2155	32,33	0,0009
	AES	128	0,0340	0,1247	0,1362	0,2949	3.756,96	0,1044	15,65	0,0004
	AES	256	0,0401	0,1310	0,1536	0,3247	4.098,24	0,1138	17,08	0,0005
	Blowfish	128	0,0344	1,1503	1,1607	2,3454	3.3278,40	0,9244	138,66	0,0039
	Blowfish	256	0,0391	1,1542	1,1435	2,3368	3.3086,88	0,9191	137,86	0,0038
	IDEA	128	0,0340	0,1941	0,1358	0,3639	4.750,56	0,1320	19,79	0,0005
	IDEA	256	0,0401	0,1965	0,1281	0,3647	4.674,24	0,1298	19,48	0,0005
Dispositivo 2	DES	64	0,0375	0,1945	0,3865	0,6185	8.366,40	0,2324	34,86	0,0010
	3DES	128	0,0314	0,3149	0,5028	0,8491	11.774,88	0,3271	49,06	0,0014
	AES	128	0,0237	0,1342	0,3494	0,5073	6.963,84	0,1934	29,02	0,0008
	AES	256	0,0424	0,1597	0,3809	0,5830	7.784,64	0,2162	32,44	0,0009
	Blowfish	128	0,0218	1,2402	1,4471	2,7091	38.697,12	1,0749	161,24	0,0045
	Blowfish	256	0,0302	1,2276	1,4410	2,6988	38.427,84	1,0674	160,12	0,0044
	IDEA	128	0,0237	0,1617	0,3519	0,5373	7.395,84	0,2054	30,82	0,0009
	IDEA	256	0,0424	0,1556	0,3383	0,5363	7.112,16	0,1976	29,63	0,0008

	Algoritmo	Tamaño (bits)	Clave (ms)	Encriptación (ms)	Desencriptación (ms)	Total (ms)	1 hora de comunicación			
							Crítico (ms)	%	No crítico (ms)	%
Dispositivo 3	DES	64	0,0332	0,2469	0,4770	0,7571	10.424,16	0,2896	43,43	0,0012
	3DES	128	0,0435	0,3974	0,6522	1,0931	15.114,24	0,4198	62,98	0,0017
	AES	128	0,0288	0,1960	0,4777	0,7025	9.701,28	0,2695	40,42	0,0011
	AES	256	0,0343	0,2200	0,5063	0,7606	10.458,72	0,2905	43,58	0,0012
	Blowfish	128	0,0289	1,7291	2,0003	3,7583	53.703,36	1,4918	223,76	0,0062
	Blowfish	256	0,0364	1,7231	2,0301	3,7896	54.046,08	1,5013	225,19	0,0063
	IDEA	128	0,0288	0,5054	0,4710	1,0052	14.060,16	0,3906	58,58	0,0016
	IDEA	256	0,0343	0,5161	0,4631	1,0135	14.100,48	0,3917	58,75	0,0016

Tabla 23 Resultados de los algoritmos simétricos

En la Ilustración 121 se muestran los resultados de la tabla pero de manera visual a través de un gráfico de barras.

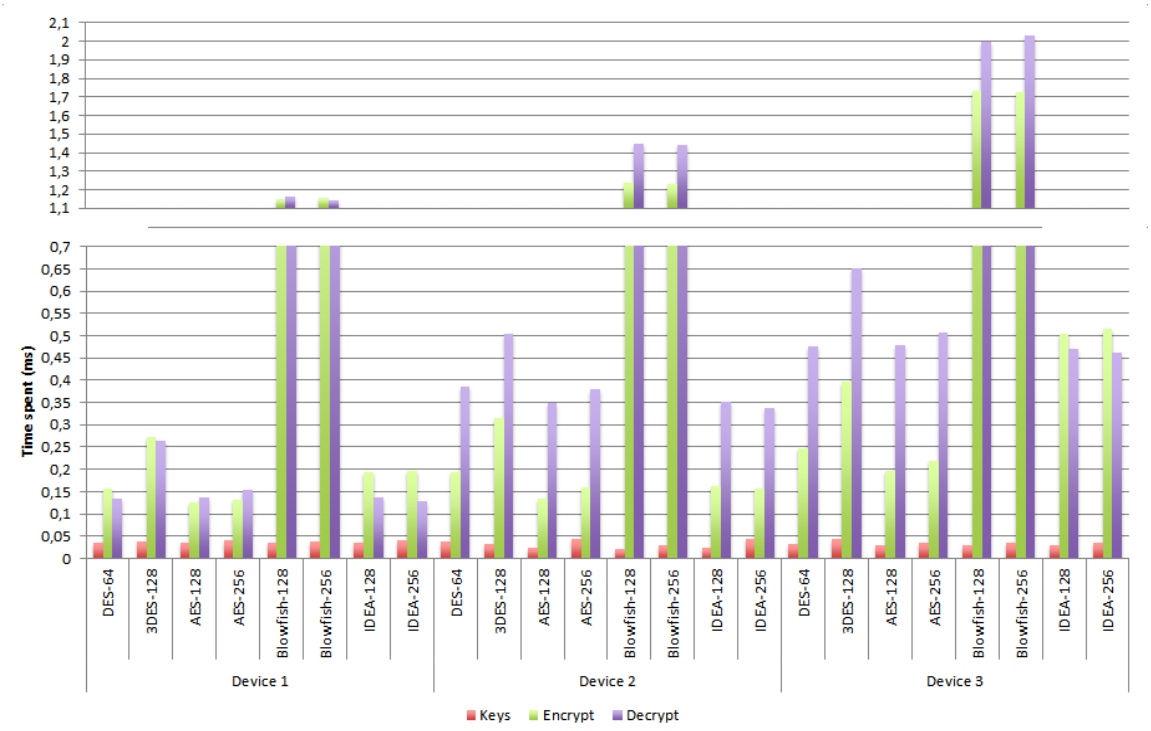


Ilustración 121 Gráfica que muestra los resultados de los algoritmos simétricos

De esta manera, si se analizan la Tabla 23 y la Ilustración 121 se puede sugerir las siguientes interpretaciones:

- El cálculo de claves llega a ser prácticamente irrelevante ya que las diferencias son pequeñas y en ninguno de los casos supera los 0,05 ms.
- Los algoritmos Blowfish de 128 bits y Blowfish de 256 bits emplean un tiempo de encriptado y desencriptado muy alto, resultando mucho más costosos que el resto de algoritmos evaluados.
- El algoritmo DES es el más inseguro debido a que utilizar el menor tamaño de clave facilita descifrar las contraseñas por fuerza bruta, al contrario de lo que sucedería con contraseñas mayores y más robustas [815]. Pese a esto, presenta unos tiempos de cálculo superiores frente a otros algoritmos más seguros como el AES de 128 bits y AES de 256 bits.
- AES de 256 bits tarda un poco más de tiempo, prácticamente lo mismo, que AES de 128 bits, pero ofrece mayor seguridad al utilizar el doble de bits para la clave.
- 3DES de 128 bits tarda el doble que AES de 128 bits, permitiendo usar únicamente claves de dicho tamaño, al contrario que AES que tiene otras opciones.
- IDEA de 256 bits tarda un poco más que IDEA de 128 bits, pero ofrece mucha mayor seguridad al utilizar el doble de bits para la clave. Este es el mismo caso que AES de 128 y AES de 256 bits.
- IDEA 256 bits requiere un poco más de tiempo total que AES 256 bits.
- Así, finalmente la decisión estaría entre los algoritmos AES e IDEA, dónde los tiempos de AES son inferiores, sobre todo en encriptado, tanto para una clave de 128 bits como de 256 bits. Respecto al tiempo que el algoritmo emplea encriptando y desencriptando en una hora de comunicación, el porcentaje de AES es inferior en todos los casos, siendo el que menos tiempo emplea AES de 128 bits, seguido de AES 256 bits.
- El tiempo empleado con AES de 128 bits, según el cálculo medio entre los tres dispositivos, es de 0,19% para nodos críticos y 0,0008% para nodos no críticos. Por otra parte, AES 256 bits emplea 0,21% para nodos críticos y 0,0009% para nodos no críticos.

Al suponernos una mayor seguridad en su versión de 256 bits, con un consumo extra de 0,02% para nodos críticos y 0,0001% para nodos no críticos, seleccionaremos la versión de AES de 256 bits para nuestra solución con la mejor relación rapidez/seguridad de las evaluadas.

23.3.2.2 CRIPTOGRAFÍA ASIMÉTRICA

A continuación, se presenta la evaluación de los algoritmos de criptografía asimétrica o de clave pública que se han seleccionado y que son RSA y ElGamal, ambas utilizando sus versiones de 1024, 2048 y 4096 bits.

En la Tabla 24 se muestran los resultados obtenidos en cada uno de los tres dispositivos utilizados con cada algoritmos utilizado. Para cada algoritmo se muestran los milisegundos que tardo en crear el par de claves público-privada, en encriptar y desencriptar la clave secreta (Código fuente 25), la suma total de la operación, y el coste de 1 hora de comunicación para nodos críticos y no críticos junto a su porcentaje

correspondiente de la hora. Para el cálculo de la hora de comunicación se ha sumado el tiempo que tardaría en encriptar y desencriptar un nodo crítico y uno no crítico, sin incluir el cálculo de la clave, debido a que este cálculo solamente se realiza durante el registro del dispositivo.

	Algoritmo	Tamaño (bits)	Clave (ms)	Encriptación (ms)	Desencriptación (ms)	Total (ms)	1 hora de comunicación			
							Crítico (ms)	%	No critico (ms)	%
Dispositivo 1	RSA	1024	806,7	0,54	7,58	814,82	116.928	3,248	487,2	0,014
	RSA	2048	2671,7	1,41	39,32	2712,43	586.512	16,292	2.443,8	0,068
	RSA	4096	24460,3	4,49	248,53	24713,32	3.643.488	101,208	15.181,2	0,422
	ElGamal	1024	16,8	31,36	14,47	62,63	659.952	18,332	2.749,8	0,076
	ElGamal	2048	118,2	229,98	117,15	465,33	4.998.672	138,852	20.827,8	0,579
	ElGamal	4096	880,5	1783,21	892,33	3556,04	38.527.776	1070,216	160.532,4	4,459
Dispositivo 2	RSA	1024	869	0,746	8,99	878,736	140.198,4	3,894	584,16	0,016
	RSA	2048	2821,7	1,807	40,62	2864,127	610.948,8	16,971	2.545,62	0,071
	RSA	4096	27357,3	5,46	256,44	27619,2	3.771.360	104,760	15.714	0,437
	ElGamal	1024	17,7	32,54	15,86	66,1	696.960	19,360	2.904	0,081
	ElGamal	2048	123,2	240,36	120,32	483,88	5.193.792	144,272	21.640,8	0,601
	ElGamal	4096	933,9	1864,92	933,53	3732,35	40.297.680	1119,380	167.907	4,664
Dispositivo 3	RSA	1024	886,8	1,5	9,898	898,198	16.4131,2	4,559	683,88	0,019
	RSA	2048	6501	2,7	50,769	6554,469	76.9953,6	21,388	3.208,14	0,089
	RSA	4096	42143,3	6,02	333,08	42482,4	4.883.040	135,640	20.346	0,565
	ElGamal	1024	25,6	40,751	20,349	86,7	879.840	24,440	3.666	0,102
	ElGamal	2048	153,7	305,684	153,601	612,985	6.613.704	183,714	27.557,1	0,765
	ElGamal	4096	1185,6	2364,47	1186,48	4736,55	51.133.680	1420,380	213.057	5,918

Tabla 24 Resultados de los algoritmos asimétricos

A continuación, en la Ilustración 122 se muestra el tiempo que ha tardado cada algoritmo en crear el par de claves público-privada, mientras que en la Ilustración 123 muestra el tiempo en milisegundos que ha tardado cada algoritmo en encriptar y desencriptar la clave secreta mostrada en el Código fuente 25.

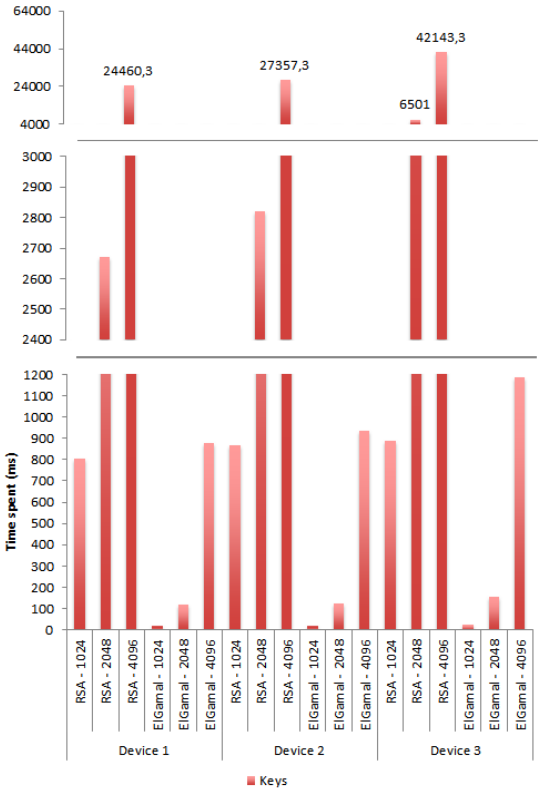


Ilustración 122 Gráfica de barras que muestra los resultados de generación de clave en los algoritmos simétricos

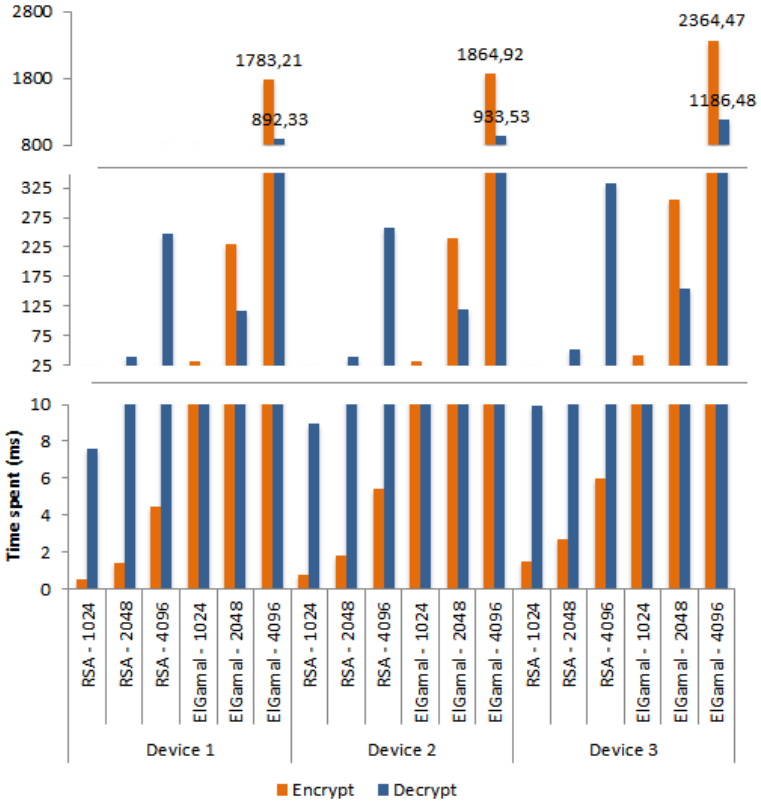


Ilustración 123 Gráfica que muestra los resultados de encriptar y desencriptar usando los algoritmos simétricos

Analizando la Tabla 24, la Ilustración 122 y la Ilustración 123, se pueden sugerir las siguientes interpretaciones:

- Los tiempos de generado de par de claves del algoritmo RSA son muy superiores a los tiempos de ElGamal, situándose aún más en el extremo según aumenta el tamaño de clave seleccionado. RSA llega a emplear una media de 31,3 segundos en el caso más costoso, 4096 bits, frente a 1 segundo de media de ElGamal, para el mismo tamaño de clave.
- Como se puede comprobar en la Tabla 24, si se tiene en cuenta el coste en una hora de comunicación, las cifras de encriptado y desencriptado de RSA son menores en todos los casos.
- El porcentaje medio consumido en una hora de comunicación para nodos críticos de RSA para 1024, 2048 y 4096 bits es de 3,9%, 18,2% y 113,9% respectivamente. Por su parte, ElGamal, emplea un 20,71%, 155,6% y 1203,3% para un tamaño de 1024, 2048 y 4096 bits respectivamente. Lo que hace que ElGamal necesite bastante más tiempo en todos los casos.
- En una hora de comunicación para nodos no críticos RSA emplea el 0,02%, 0,08% y el 0,5% para claves de 1024, 2048 y 4096 bits. Por su parte, ElGamal emplea más tiempo en todos los casos, siendo este del 0,09%, 0,65% y el 5% para claves de 1024, 2048 y 4096 bits.
- De ello se deduce que para ElGamal no podría ser empleado con claves de 2048 y 4096 bits, ya que, para las comunicaciones entre nodos críticos, excede el tiempo de cálculo encriptando y desencriptando en una hora, por lo que no sería viable. Lo mismo sucede con RSA para su versión de 4096 bits.
- RSA presenta un coste inicial muy superior para el cálculo de clave. Debido a que este coste solo se computa una vez, al registro del dispositivo en la plataforma, es menos relevante que las comunicaciones continuadas, en las que presenta valores mucho menores que ElGamal.

En función de los valores estudiados, es más recomendable la utilización del algoritmo **RSA con 4096 bits para nodos no críticos**, dónde la comunicación es menor, y **emplearía el 0,565% del tiempo encriptado y desencriptado**, en el peor de los casos, ofreciendo un buen rendimiento.

Para **nodos críticos**, dónde la comunicación es muy continuada, enviando hasta cuatro mensajes por minuto, la mejor opción pasaría por usar el algoritmo **RSA con una longitud de clave de 1024 bits**, que, aunque ofrezca una seguridad menor al ser menos resistente para ser descubierto por fuerza bruta, ofrece un rendimiento mucho más ágil para este tipo de envío, **con un gasto de 4.559%** en el peor de los casos en el procesado de los mensajes. No obstante, si se necesitase más seguridad, podría utilizarse RSA de 2048 bits, que requiere algo más de un cuarto del tiempo de procesado.

Seguridad en la comunicación de los *Smart Objects* a través de una plataforma de Internet de las Cosas

23.3.2.3 FUNCIONES HASH

A continuación, se muestran los resultados de las pruebas de las diferentes funciones Hash, necesarias para garantizar la integridad del mensaje. Para ello, se ha decidido generar un resumen del mensaje, para que el usuario receptor pueda comprobar que el mensaje que ha recibido coincide con el original que se le había enviado. Esto le permitirá discernir si ha sido o no manipulado durante el envío.

Para ello, se han seleccionado entre algunas funciones Hash criptográficas existentes, con el fin de realizar tomas de tiempos y seleccionar la que mejor se adapte a nuestras necesidades en términos de consumo. Se han realizado las pruebas con los algoritmos: MD5, SHA-1 (256), SHA-1 (512), SHA-2 (256), SHA-2 (512), SHA-3 (256) y SHA-3 (512). MD6 ha sido descartado por no encontrar una librería que aportara implementación de esta función Hash en Java.

En la Tabla 25 se muestra las funciones hash criptográficas utilizadas en cada uno de los dispositivos, junto al tiempo que han tardado en resumir el mensaje y el tiempo correspondiente a una hora de comunicación en un nodo crítico y uno no crítico.

			1 hora de comunicación			
	Algoritmo	**Hash (ms)**	Crítico (ms)	%	No crítico (ms)	%
Dispositivo 1	MD5	0,2129	3.065,76	0,08516	12,774	0,0004
	SHA-1	0,2551	3.673,44	0,10204	15,306	0,0004
	SHA2-256	0,3976	5.725,44	0,15904	23,856	0,0007
	SHA2-512	0,7836	11.283,84	0,31344	47,016	0,0013
	SHA3-256	0,3437	4.949,28	0,13748	20,622	0,0006
	SHA3-512	0,6197	8.923,68	0,24788	37,182	0,0010
Dispositivo 2	MD5	0,2686	3.867,84	0,10744	16,116	0,0004
	SHA-1	0,3023	4.353,12	0,12092	18,138	0,0005
	SHA2-256	0,485	6.984	0,194	29,1	0,0008
	SHA2-512	0,9227	13.286,88	0,36908	55,362	0,0015
	SHA3-256	0,4374	6.298,56	0,17496	26,244	0,0007
	SHA3-512	0,7676	11.053,44	0,30704	46,056	0,0013

			1 hora de comunicación			
	Algoritmo	**Hash (ms)**	Crítico (ms)	%	No crítico (ms)	%
Dispositivo 3	MD5	0,3848	5.541,12	0,15392	23,088	0,0006
	SHA-1	0,4605	6.631,2	0,1842	27,63	0,0008
	SHA2-256	0,6977	10.046,88	0,27908	41,862	0,0012
	SHA2-512	1,3104	18.869,76	0,52416	78,624	0,0022
	SHA3-256	0,6737	9.701,28	0,26948	40,422	0,0011
	SHA3-512	1,0228	14.728,32	0,40912	61,368	0,0017

Tabla 25 Resultados de las funciones Hash criptográficas

En la Ilustración 124 se muestra gráficamente el tiempo tardado en resumir el mensaje por cada función Hash en cada dispositivo.

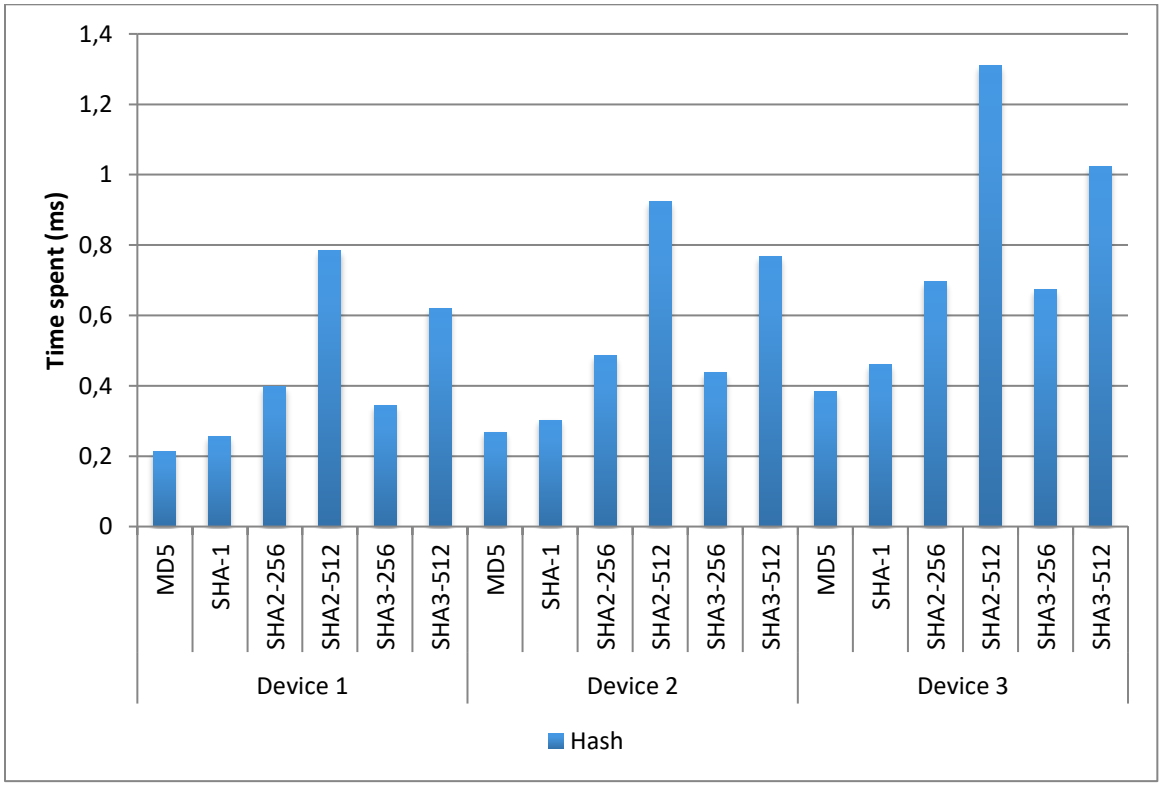


Ilustración 124 Gráfica con los resultados de aplicar las funciones Hash al mensaje

Seguridad en la comunicación de los *Smart Objects* a través de una plataforma de Internet de las Cosas

Analizando la Tabla 25 y la Ilustración 124 se pueden sugerir las siguientes interpretaciones:

- El algoritmo MD5 con un tamaño de 128 bits es más rápido que el resto.
- SHA-1 con un tamaño de 160 bits es sólo un poco más lento que MD5 y un poco más rápido que SHA-3 de 256 bits.
- SHA-2 de 256 y 512 bits presentan unos tiempos superiores a los de SHA-3 para los mismos tamaños de claves, además de ser más inseguros y haber sido vulnerados.
- SHA-3 de 256 bits ofrece la mejor combinación rapidez/seguridad debido a que es un poco más lento, un 57% exactamente, que el más rápido, MD5.

Algunos problemas de seguridad presentes en estas funciones hash:

- MD5 es la opción más rápida de las evaluadas, pero presenta problemas debido a la colisión hash [797], [804], [805], por lo que no resulta la opción más segura.
- En 2005 se detectaron fallos de seguridad en SHA-1, por lo que no se recomienda su uso debido a la colisión [397], [796]. Por ello, surgió SHA-2.
- SHA-2 ha sido vulnerado en múltiples ocasiones en todas sus versiones como se puede ver en 23.3.1.1.3.
- El único algoritmo no vulnerado de los estudiados aquí ha sido SHA-3.

Por lo tanto, la decisión final ha sido utilizar SHA-3, que presenta un rendimiento superior y un coste más bajo que SHA-2 y casi tan bueno como MD5, además de aún no haber sido vulnerado todavía. En este caso se puede utilizar SHA-3 de 512 bits, ya que ofrece una seguridad mayor debido al tamaño y el consumo, 0,4% para comunicación entre nodos críticos y 0,0017% para comunicación entre nodos no críticos, en los peores casos, es bajo.

23.4 *CONCLUSIONES*

En esta iteración se ha presentado una propuesta de seguridad para Internet de las Cosas, implementada en la plataforma de IoT Midgar, con el propósito de buscar el mayor grado de seguridad en el envío de los mensajes entre objetos a través de una conexión no segura intentando perder el menor tiempo de computo posible, pero utilizando la máxima seguridad posible.

La solución ha pasado por adoptar las medidas características de la criptografía híbrida combinándola con la firma digital, para así obtener los beneficios de cada una. Con esto, se obtiene la confidencialidad y privacidad necesarias para el intercambio de mensajes de ámbito privado, añadiendo integridad y autenticación, conceptos necesarios en una red IoT.

El estudio evalúa varios algoritmos de entre los tipos de criptografía seleccionada, con el objetivo de encontrar la solución más rápida posible, que consuma el menor tiempo y almacenamiento en un entorno tan cambiante de objetos con prestaciones tan diferentes, pero siempre manteniendo la seguridad necesaria. En este estudio se observa que la mejor opción es el uso de **AES**, **RSA** y **SHA3**.

Por un lado, en la criptografía simétrica **AES** para un tamaño de **256 bits** obtiene los mejores resultados tanto en tiempo de encriptado y desencriptado, dónde emplea el **0,2905%** de las comunicaciones en una hora en nodos críticos y un **0,0012%** en la comunicación de nodos no críticos, en los peores casos, como en términos de seguridad por tamaño de clave, dónde tarda en generar una clave de 256 bits una media de **0,0343 ms**.

Respecto a la criptografía asimétrica, RSA es más rápido que ElGamal en una comunicación prolongada, cómo se espera que sean en un entorno IoT, ya que ElGamal llega a emplear más del doble de tiempo que RSA, pese a que RSA tiene un tiempo de cálculo de claves inicial muy superior a ElGamal. RSA consumiría en una hora de comunicación entre nodos críticos, dónde se usaría la versión de **1024 bits**, el **4,559%** del tiempo en tareas de encriptado y desencriptado. En la comunicación de una hora entre nodos no críticos RSA de **4096 bits** emplearía únicamente el **0,437%** del tiempo. En ambos casos se valora el tiempo con el dispositivo de gama baja.

Finalmente, para resumir el mensaje y garantizar su integridad, se emplearía la función Hash criptográfica SHA-3 con un tamaño de **512 bits**, dónde la función consumiría el **0,40912%** del tiempo en las comunicaciones de una hora entre nodos críticos y el **0,0017%** en nodos no críticos.

Esta seguridad, que usa de forma conjunta los algoritmos **RSA, AES y SHA3**, tendría un impacto final de **5,25862%** en el consumo de una hora en nodos críticos, y un **0,84732%** en nodos no críticos, todo esto en dispositivos de gama baja, los cuáles dan el peor resultado.

Sin embargo, hay que recalcar que en el caso de RSA, este posee un alto tiempo de generación automática del par de claves en comparación con ElGamal, pero debido a que dicho cálculo sólo se realiza en una ocasión, este puede ser asumido por los dispositivos.

No obstante, como se ve en este estudio, se puede llegar a consumir bastante tiempo dependiendo del algoritmo utilizado y, aún seleccionando el que menor tiempo consuma manteniendo la seguridad, un 4,559% en nodos críticos es bastante tiempo, sobre todo, en dispositivos de gama baja. Sin embargo, esto es algo necesario para mantener la seguridad y privacidad de los objetos cuando estos la requieran porque, aunque cierta información personal pueda parecer irrelevante, otros datos como la ubicación o las cuentas bancarias pueden suponer un gran peligro en caso de ser suplantadas.

24. UN LENGUAJE ORIENTADO AL USUARIO PARA ESPECIFICAR INTERCONEXIONES ENTRE OBJETOS HETEROGÉNEOS Y UBICUOS EN INTERNET DE LAS COSAS

«Cuando comienzas a intentar resolver un problema, las primeras soluciones que se te vienen a la cabeza son muy complejas y por eso la mayor parte de la gente se queda parada cuando llega a este punto. Pero si sigues, vives con el problema y pelas más capas de la cebolla, llegas a menudo a soluciones muy elegantes y muy simples»
Steve Jobs

«Si buscas resultados distintos, no hagas siempre lo mismo»
Albert Einstein

Uno de los primeros procesos en la etapa de creación de software está en la definición de los casos de uso. En estos, se define mediante palabras que va a hacer el software que se va a desarrollar, es decir, sus requisitos o su tarea. No obstante, como bien es sabido, esto va escrito en lo que nosotros llamamos lenguaje natural. Esto es útil para dejar definida la funcionalidad del sistema y a partir de ahí, ir creándolo. Es decir, es un contrato.

Los casos de uso se encuentran en lo más alto de la pirámide de desarrollo de software, si se sitúa abajo de todo el código fuente desarrollado. Entre ambos, hay una serie de pasos, dependiendo del tipo de arquitectura. Sin embargo, estos no son directos, sino que hay que ir haciendo los diferentes pasos manualmente, aunque algunos de ellos se pueden automatizar o semiautomatizar mediante el uso de Ingeniería Dirigida por Modelos.

Aquí, la parte interesante sería el poder automatizar o semiautomatizar todo el proceso. Luego, para lograr esto, habría que automatizar los primeros pasos, que no son ni más ni menos que la descripción en lenguaje natural del sistema, es decir, los casos de uso.

Con este fin, se ha desarrollado esta iteración, la cual consiste en que los usuarios finales puedan definir mediante el uso de lenguaje natural. Siguiendo una serie de reglas para definir la tarea de la aplicación que desean desarrollar. De esta manera, se podría coger el caso de uso que han descrito y convertirlo directamente en el código fuente que necesitan.

Un lenguaje orientado al usuario para especificar interconexiones entre objetos heterogéneos y ubicuos en Internet de las Cosas

24.1 *DESCRIPCIÓN*

24.1.1 **OBJETIVOS**

Esta sexta iteración se centró en poder ofrecer un mejor sistema de creación de interconexiones entre objetos en el ámbito de Internet de las Cosas de una forma mucho más natural para el usuario respecto al sistema implementado en la segunda iteración y mostrado en el capítulo 20, siendo de este modo la hipótesis de este prototipo. Esta iteración, en vez de ofrecer un DSL, ofrece un lenguaje mucho más cercano al lenguaje natural y cuyos objetivos son:

- Permitir describir casos de uso que sean transformados a la aplicación final.
- Ofrecer un lenguaje cercano al lenguaje natural para crear interconexiones entre objetos.
- Obtener una alta abstracción con dicho lenguaje que ofrezca más facilidad que MOISL.
- Comunicar objetos heterogéneos y ubicuos.

24.1.2 **FUNCIONALIDADES**

Estos objetos requieren del uso de la plataforma Midgar como centro nervioso de las interconexiones de los objetos. Para utilizarla, ha sido necesario modificar la primera capa, la del proceso de definición, para que ofrezca al usuario las utilidades necesarias para definir todo el proceso de una nueva forma más cercana al lenguaje natural.

Para lograr esto, se ha implementado un editor que sea capaz de reconocer palabras muy utilizadas en la definición de procesos con el fin de convertir lo que escriba el usuario a la aplicación final. Entre medias, se hace una conversión y se persiste todo en un modelo formal de MOISL, con el fin de reaprovechar todo lo construido en esa iteración y evitar fallos en la implementación que ya han sido solventados con anterioridad.

Esto ofrece al usuario la posibilidad de crear las interconexiones mediante la descripción de los casos de uso simplemente escribiendo lo que se quiere que realicen, siempre y cuando siga unas pequeñas pautas. Para muchos usuarios, esto es algo más usable que adaptarse o tener que aprender a utilizar un DSL. Además, esto ofrece la posibilidad de trabajar directamente con los casos de uso creados por los usuarios y traducirlos a la aplicación requerida.

24.2 *ARQUITECTURA PROPUESTA*

La arquitectura de este sistema incluye la modificación de la primera capa 3 de la plataforma IoT Midgar. Exactamente sustituye MOISL, que es el DSL que permite interconectar objetos y elegir las circunstancias y condiciones sobre las que ocurrirá todo. En su lugar se introduce MUCSL (Midgar Use Case Specification Language), que como se ha descrito previamente, es un lenguaje más cercano al usuario que permite describir casos de uso y transformarlos en la aplicación final. Esta arquitectura se muestra en la Ilustración 125.

Un lenguaje orientado al usuario para especificar interconexiones entre objetos heterogéneos y ubicuos en Internet de las Cosas

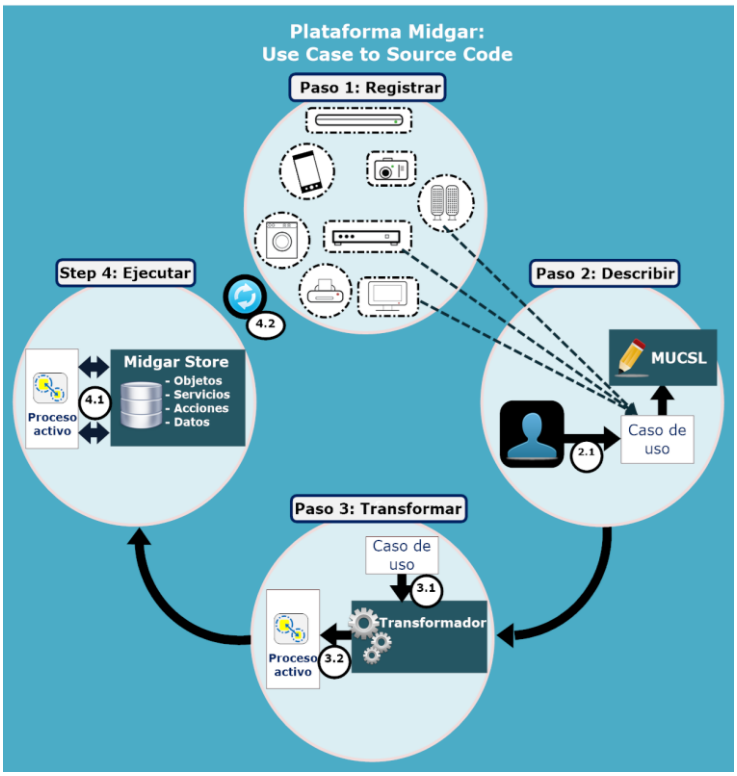


Ilustración 125 Ciclo de vida de la arquitectura de Midgar utilizada en MUCSL

En este caso, solo se ha modificado la capa 1, exactamente lo que corresponde a MOISL para sustituirlo por MUCSL. El resto de capas se mantienen igual que como estaban. En la capa 2 se reutiliza el procesador debido a que el procesador MUCSL transforma todo lo que reconoce en un modelo formal persistido de MOISL, lo que permite reutilizar el procesador y el generador de aplicaciones que crea los demonios y los deja corriendo en la capa 3.

Por otro lado, la capa 4 de Midgar sigue conteniendo los objetos, los cuáles han de registrarse previamente para así permitir ser utilizados en MUCSL.

24.2.1 IMPLEMENTACIÓN

En esta iteración se han reutilizado diferentes cosas de MOISL, siendo el DSL lo único que se ha sustituido por MUCSL. Luego, se ha utilizado la plataforma Midgar. En este apartado se describirán los componentes de MUCSL y cómo funciona cada uno, pues esto involucra solo la primera capa de Midgar, y exactamente solo la modificación del DSL MOISL; siendo el resto idéntico.

Debido a que solo se modifica esa parte, esto será lo que se explique a continuación. No obstante, en el segundo subapartado, se detalla cómo se realiza la transformación del caso de uso a código fuente, a pesar de que esta utiliza el procesador de MOISL.

El resto de la arquitectura se mantiene igual que en MOISL.

Un lenguaje orientado al usuario para especificar interconexiones entre objetos heterogéneos y ubicuos en Internet de las Cosas

24.2.1.1 SINTAXIS Y ESTRUCTURA

MUCSL tiene un léxico simple que consiste en palabras como «*when*», «*if*», «*for*», «*sensorID*» y «*then*» que siguen la arquitectura que se muestra en la Ilustración 126.

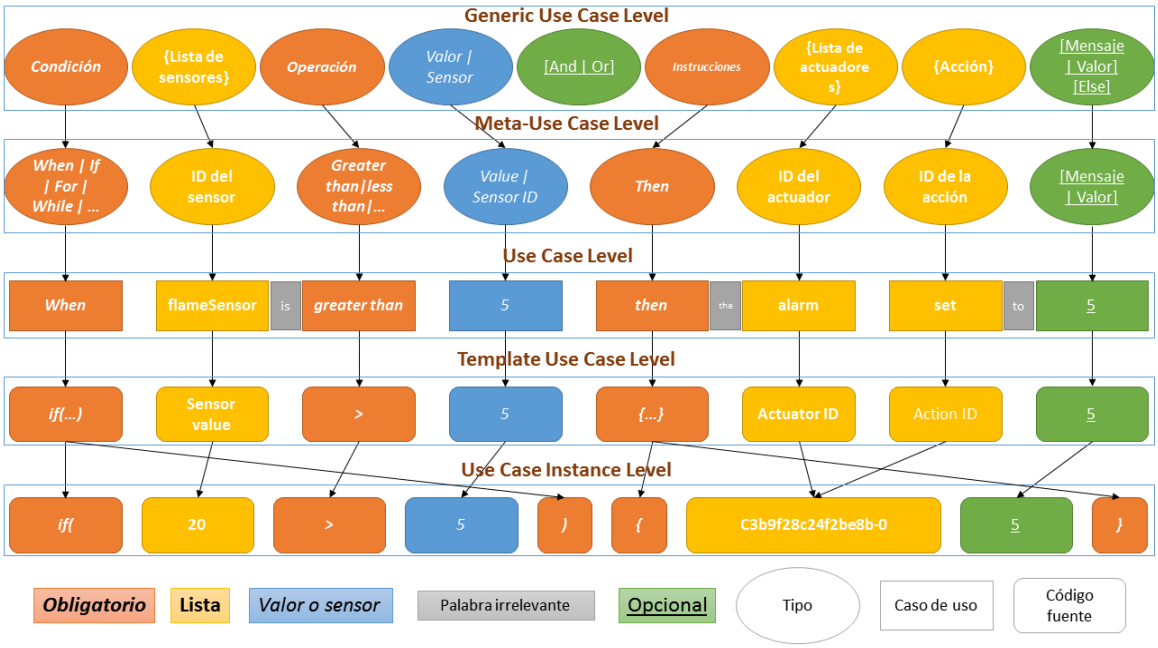


Ilustración 126 Sintaxis y estructura de MUCSL

MUCSL permite al usuario definir la interconexión mediante el uso de cláusulas condicionales al estilo «Si condición entonces acción y sino otra acción» y «Mientras condición entonces acción en caso contrario otra acción», conocidas en inglés como *If-then-else*. Este tipo de sentencias están definidas en la primera capa, conocida como la capa **Generic Use Case**. Esta capa consiste en 6 tipos de palabras, como se muestra en la columna izquierda y central de la Tabla 26: obligatorias, conjunciones, valores, identificadores, opcionales y otras. Esta tabla contiene, además, palabras nuevas (en cursiva) incorporadas tras realizar las pruebas a los participantes, pues se vio que había algunos que utilizaban palabras diferentes, pero con el mismo significado, así como otras palabras basadas en estas últimas. Cabe destacar que no se distingue entre letras mayúsculas y minúsculas ni entre singulares y plurales.

Categoría		Palabra
Obligatoria	Condición	If, for, when, while, meanwhile, as long as
	Operador de comparación	greater than, *more than*, less than, *lower than*, *equal*, equal to, greater or equal than, less or equal than, different from, *set to*, >, <, >=, <=, ==, *!=*
Conjunción		Then
Valor		Texto, número, carácter, booleano, id de sensor
Identificadores		Alfanuméricos, -
Opcional	Conjunción	And, or
	En otro caso	Else, *otherwise*
	Mensaje	Texto, número, carácter, booleano
Otras	Palabras irrelevantes	The, a, an, are, is, be, to, will, should, shall, can, could, would
	Puntuación	«,», «.»

Tabla 26 Palabras clave de MUCSL

Estos grupos de palabras pueden ser compuestos utilizando sentencias simples y naturales que pueden ser interpretadas de forma secuencial. Primero, se tiene la condición para indicar el principio de la sentencia. En segundo lugar, se comparan los valores del sensor o de una lista de sensores con un valor numérico para establecer a verdadero o falso la condición. Esta condición puede ser más compleja si se introduce la palabra «and» o «or», de forma que se permite la inclusión de más comparación y que obligan a que varios se cumplan a la vez o se ofrezcan diferentes alternativas en la que basta con que una se cumpla, respectivamente. La siguiente palabra es la que se utiliza para separar la condición de las instrucciones, la palabra «then». Las instrucciones contienen una lista de actuadores con la acción que el usuario quiere realizar en ese actuador. Después de esto, existe la palabra opcional, que indica una acción alternativa para el actuador y que se representa por la palabra clave «else».

En la segunda capa, **Meta-Use Case**, se encuentran definidas las palabras clave necesarias para definir los casos de uso. Estas palabras clave se muestran en la columna de la derecha de la Tabla 26. Por ejemplo, la condición solo acepta «when» o «if». La lista de sensores necesita de un identificador de sensor o más. La comparación puede ser «greater than», «more than», «less than», «lower than», «equal», «equal to», «greater or equal than», «less or equal than», «different from», «set to», «>», «<», «>=», «<=», «==» o «!=». El valor

puede ser booleano, numérico, un mensaje de texto, un carácter o un identificador de un sensor del que recogerá su valor. Un ejemplo de esto es si se tuviera que encender una luz cuando dos o más fotorresistores de nuestro salón tengan un valor menor que cinco o un valor menor que el de otro fotorresistor situado fuera de la casa, por ejemplo en el jardín. El valor opcional «and» o «or» sirve para concatenar más condiciones. Después viene la palabra «then» para comenzar con las instrucciones que deben de ejecutarse cuando se cumpla la condición. Lo siguiente es la lista de actuadores con sus acciones. Finalmente, está el valor o mensaje opcional o bien el «en caso contrario». El valor o mensaje opcional se debe a que tal vez hay que enviar un dato al actuador o a la acción, pues estos pueden requerir de un parámetro, como es en el caso del vibrador del smartphone que acepta el tiempo de vibración en milisegundos. Por otro lado, el «en caso contrario» o «*else*» se debe a que igual simplemente se quiere ejecutar una acción en caso de que la condición no se cumpla, y así las instrucciones que se quieren ejecutar en ese caso van detrás del «*else/otherwise*».

La tercera capa es la **Use Case**, siendo esta donde el usuario define el caso de uso. En la Ilustración 126 se puede ver que pone «When the B8AC6F48E370 is greater than 5 then the C3b9f28c24f2be8b will be 0 to 5». En este ejemplo, el identificador del objeto es usado en lugar del nombre debido a que queremos asegurarnos que el objeto llamado es ese. Los usuarios pueden ver el ID de sus objetos en la plataforma Midgar, que es la plataforma utilizada como interconectora de los objetos. Después de mapear los IDs de los nombres de los objetos, el caso de uso quedaría tal que así: «When the flameSensor is greater than 5 then the alarm will be set to 5».

La Ilustración 127 representa el paso de descripción utilizando MUCSL y que se corresponde con la tercera capa de la arquitectura. En esta ilustración se puede ver como el usuario escribe un caso de uso en el que utiliza un fotorresistor y el sensor de luz de su smartphone, en los que indica que, en el momento que uno de los dos marque menos de 5 debe de encenderse la luz de su casa. Claramente, esto debe de escribirse utilizando la estructura de MUCSL como se explicó previamente.

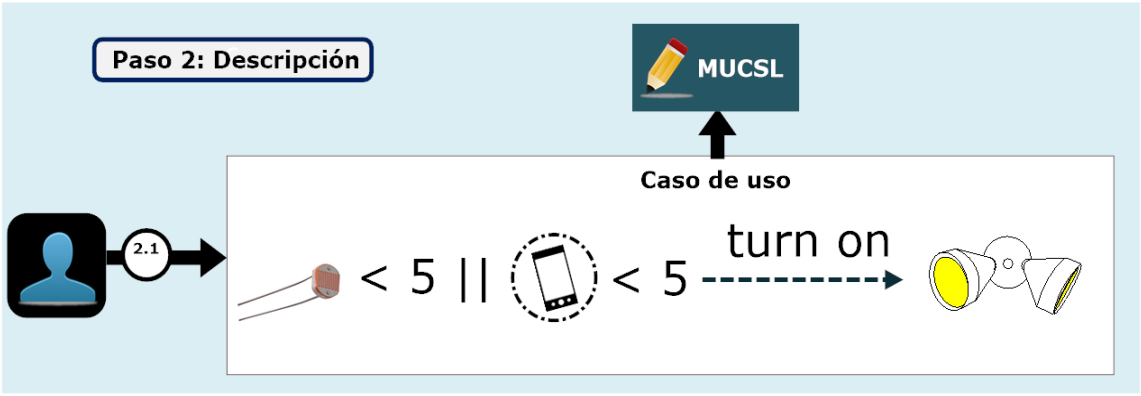


Ilustración 127 Ejemplo de uso de MUCSL

24.2.1.2 MECANISMO DE TRANSFORMACIÓN

En la Ilustración 126 se ve que el sistema cuenta con un total de 5 capas. Previamente se explicaron las tres primeras, que tratan sobre cómo se ha de escribir un caso de uso. Ahora se va a explicar cómo funcionan la cuarta y la quinta capa, las cuáles se corresponden con las capas del procesador de MOISL.

En la cuarta capa, llamada **Template Use Case**, el transformador necesita leer todo el texto de la anterior capa, eliminar las palabras irrelevantes como «the», «is», «will», «to» y así sucesivamente, y obtener los diferentes datos del caso de uso para incorporar estos datos en la plantilla. En la Ilustración 126 se ve como transforma las palabras clave en palabras pertenecientes al GPL, pues por ejemplo, cambia el «when» por un «if».

Finalmente, en la capa **Use Case Instance**, se obtiene el código fuente final. En este caso, se puede ver el caso de uso del usuario transformado a código fuente válido. Por ejemplo, el sensor es «20» debido a que este es el valor de ese sensor en ese momento dado. Mientras, ocurra lo mismo con el valor «5», pero siendo este de otro sensor diferente. En este caso, el ID del actuador ha sido fusionado con el ID de la acción para obtener los datos que se deben de enviar a Midgar para hacer una consulta correcta. En este caso, se ha puesto la acción del actuador, la cual es el número «0», y el «5» corresponde al valor que se le envía como parámetro. Después de esto, se obtiene la aplicación que interconecta los diferentes objetos.

24.2.2 SOFTWARE Y HARDWARE UTILIZADO

En esta sección se muestra el software utilizado para el desarrollo de este prototipo, comenzando por la implementación del sistema:

- La plataforma IoT Midgar:
  - Ruby 2.2.2p95.
  - Rails 4.2.1.
  - Servidor web Thin 1.6.3.
  - Base de datos MySQL 5.5.43.
  - MUCSL, el cual ha sido desarrollado utilizando HTML5, PHP 7 y JavaScript estándar.
  - Para la creación del generador de aplicaciones y la aplicación de conexión con el Arduino al ordenador se optó por utilizar Java 8.

Para las pruebas se utilizó el siguiente equipo hardware:

- Raspberry Pi 2 Model B dedicada con un sistema operativo Raspbian Jessie 4.4.13 v7.
- Dos smartphones Android:
  - Nexus 4 con Android 5.1.1.
  - Motorola con la versión 2.2.2.
- Microcontrolador Arduino Uno SMD basado en el ATmega328.

- Durante los diferentes test se han usado los siguientes sensores y actuadores: termistor TMP36, un altavoz, un servomotor, un motor DC, diferentes LEDs, dos botones, un fotorresistor, un sensor de temperatura y humedad DHT11, y un sensor de llamas KY026.

24.3 *EVALUACIÓN Y DISCUSIÓN*

Esta subsección contiene la metodología utilizada en la evaluación de este prototipo y la evaluación junto a su discusión. La evaluación fue dividida en dos fases. La primera fase es la toma de datos cuantitativos a partir de la evaluación realizada al participante, que consta de tres pruebas. La segunda fase es una encuesta realizada a los participantes tras realizar las pruebas de la primera fase.

24.3.1 **METODOLOGÍA**

El objetivo principal de esta evaluación ha sido el de validar la hipótesis inicial, para la que se deben de cumplir los objetivos presentados. Para lograrlo, se ha creado MUCSL, un lenguaje cercano al natural que permite, cumpliendo una serie de reglas, transformar los casos de uso escritos por los participantes en la aplicación final que interconectará los objetos.

Para validar la hipótesis se han desarrollado dos fases de forma que en la primera se obtengan datos cuantitativos y en la segunda datos cualitativos para evaluarla de forma correcta:

- **Fase 1**: en esta primera fase se ha propuesto tres pruebas a los participantes. Estas tres pruebas han sido tres posibles casos de uso de interconexión de objetos que han debido de escribir utilizando MUCSL. Esta fase se corresponde con la evaluación cuantitativa.
- **Fase 2**: una vez terminada la fase 1, los participantes han tenido que contestar una encuesta, compuesta de 17 declaraciones, utilizando la escala Likert de 5 puntos [781]. La encuesta contiene declaraciones acerca de MUCSL y lo realizado en la primera fase, de forma que los participantes han de decir como de acuerdo están con cada una de las declaraciones. Esto otorga los datos cualitativos a la evaluación.

24.3.1.1 FASE 1

En esta primera fase los participantes han tenido que realizar tres tareas utilizando MUCSL, cada tarea independiente de las otras. Se mandaron tres tareas con el objetivo de que el usuario pudiera familiarizarse, siendo así la primera tarea la más fácil y la segunda y la tercera parecidas.

Durante el transcurso de las pruebas se han tomado datos cuantitativos del uso de los editores, como han sido el tiempo en seguro que tomo a cada participante realizar la aplicación, los centímetros requeridos con el ratón y los clics con el botón derecho e izquierdo del ratón. Para medir estos datos se ha utilizado la herramienta Mousotron [780].

La **primera tarea** ha consistido en crear una aplicación que interconecte un Arduino con un smartphone. El propósito es hacer que esta aplicación utilice el sensor de llamas del Arduino para detectar llamas y, cuando

este marque un valor de 50 o menos, la aplicación envíe una notificación al smartphone con el mensaje «Fuego». El identificador del sensor es «B8AC6F48E370-0» y el del smartphone es «C3b9f28c24f2be8b-0». Una posible solución de la primera tarea se muestra en el Código fuente 26, donde se muestra entre corchetes («[» y «]») posibles palabras opcionales de la frase.

```
If [the] B8AC6F48E370-0 [is] greater than 49 then
C3b9f28c24f2be8b-0 [to] 'fire'
```

Código fuente 26 Solución de la primera tarea

La **segunda tarea** ha sido desarrollar una aplicación que interconecta el Arduino, con su smartphone y una luz. La aplicación debe de utilizar el fotorresistor del Arduino y del smartphone para detectar su se necesita encender la luz. Si el sensor del Arduino baja hasta un valor de 30 o menos y el del smartphone de 20 o menos, la aplicación debe de enviar la orden de encenderse a la luz, en caso contrario la luz deberá de apagarse. Los identificadores del señor del Arduino y del smartphone son «B8AC6F48E370-0» y «C3b9f28c24f2be8b-0» respectivamente. Mientras, los identificadores de las acciones encenderse y apagarse de la luz son «D4az78t31y7ghu8p-0» y «D4az78t31y7ghu8p-1» para cada caso. Una posible solución de la primera tarea se muestra en el Código fuente 27.

```
When [the] B8AC6F48E370-0 [is] equal or less than 30 and
C3b9f28c24f2be8b-0 [is] equal or less than 20 then [the]
C3b9f28c24f2be8b-0 else [the] D4az78t31y7ghu8p-1
```

Código fuente 27 Solución de la segunda tarea

En la **tercera tarea** los participantes han tenido que desarrollar una aplicación que interconecte el Arduino y un ventilador. Así, mediante el uso del sensor de temperatura del Arduino la aplicación será capaz de cambiar la velocidad de movimiento del ventilador. Si el sensor del Arduino alcanza un valor de 25ºC o más, la aplicación añadirá un punto de velocidad al ventilador por cada grado de diferencia. En caso contrario, la aplicación apagará el ventilador. Los identificadores son «B8AC6F48E370-1», «Z7kl94iop22ns89e-0» y «Z7kl94iop22ns89e-1» para el sensor del Arduino y las acciones de más velocidad y apagar del ventilador, respectivamente. Una posible solución de la primera tarea se muestra en el Código fuente 28.

```
For [the] B8AC6F48E370-1 [is] greater or equal than 25 then [the]
Z7kl94iop22ns89e-0 else [the] Z7kl94iop22ns89e-1
```

Código fuente 28 Solución de la tercera tarea

24.3.1.2 FASE 2

En la segunda fase se ha evaluado la parte cualitativa de MUCSL para así obtener la opinión de los participantes y ver su opinión acerca de esta investigación. Para hacer esta encuesta se ha utilizado la escala Likert de 5 puntos como método de evaluación debido a que es un método muy utilizado en el campo de la ingeniería del software para obtener información eficazmente para apoyar la toma de decisiones [795].

Al utilizar la escala Likert de 5 puntos, conteniendo un total de 17 declaraciones, se ofrecen 5 posibles respuestas que son: 1 como *Totalmente en desacuerdo*, 2 como *En desacuerdo*, 3 como *Neutral*, 4 como *De acuerdo*, y 5 como *Totalmente de acuerdo*.

Los participantes realizaron siempre esta encuesta tras finalizar la fase 1, de forma anónima y sin ayuda. Esta encuesta contiene declaraciones sobre MUCSL, sus posibilidades y su posible impacto en Internet de las Cosas y en los *Smart Objects*, ofreciendo una interconexión entre ambos. La Tabla 26 contiene las declaraciones.

#	**Declaraciones**
Q1	El usuario entiende la estructura de los elementos y su papel en el proceso de creación de la aplicación.
Q2	Esta herramienta ofrece una asistencia útil para interconectar objetos heterogéneos.
Q3	La sintaxis es clara, sencilla y natural.
Q4	Esta solución ofrece una forma rápida de desarrollar la tarea indicada.
Q5	Esta solución proporciona ayuda para crear aplicaciones para interconectar objetos.
Q6	La forma de crear aplicaciones mediante el uso de este lenguaje es comprensible.
Q7	El lenguaje no requiere que el usuario tenga conocimientos de programación altos, como en el desarrollo de aplicaciones tradicional.
Q8	La sintaxis incluye suficientes elementos y funcionalidad para que el usuario para crear una amplia gama de interconexiones entre objetos.
Q9	Esta propuesta es una contribución positiva para fomentar el desarrollo de servicios y aplicaciones para Internet de las Cosas.
Q10	Internet de las Cosas y los *Smart Objects* se beneficiarán de esta solución.

#	Declaraciones
Q11	Este lenguaje se podría utilizar para simplificar el proceso de desarrollo clásico de aplicaciones de software en otras áreas (educación, videojuegos, ...).
Q12	El uso del lenguaje disminuye la complejidad de desarrollo de este tipo de aplicaciones.
Q13	Esta sintaxis proporciona una forma fácil e intuitiva para interconectar dispositivos.
Q14	El usuario entiende la sintaxis del lenguaje y su papel en el proceso de creación de la aplicación.
Q15	El usuario comente menos errores usando este lenguaje que si programa.
Q16	El usuario trabaja de una manera más rápida y eficaz mediante el uso de este lenguaje.
Q17	Este lenguaje puede considerarse útil.

Tabla 27 Declaraciones de la encuesta de MUCSL

24.3.2 RESULTADOS

Esta sección contiene los resultados obtenidos en las dos fases junto a su discusión. En la primera subsección se muestran los resultados cuantitativos obtenidos de realizar las tres tareas con MUCSL. Posteriormente en el siguiente apartado se muestran los resultados cualitativos de la encuesta.

24.3.2.1 FASE 1

En esta primera fase se ha tomado el tiempo de cada participante mientras realizaban las tres tareas requeridas. Este tiempo se muestra en la Ilustración 128, la cual contiene el tiempo de cada participante en cada tarea y la media global entre todos los participantes para realizar cada tarea.

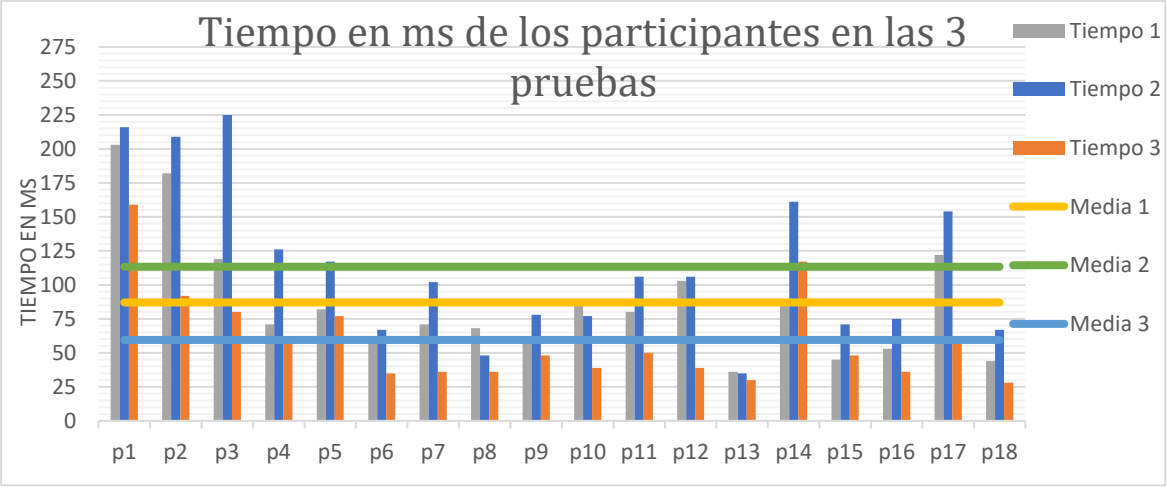


Ilustración 128 Tiempo necesitado por cada participante en cada tarea junto a la media global

Analizando esta gráfica, se pueden sugerir las siguientes interpretaciones:

- Todos los participantes tardaron un poco más al realizar la 2 tarea respecto a la primera, a excepción del P3 y P14, que tardaron bastante más.
- Solo P8, P10 y P13 tardaron menos en la segunda tarea que en la primera.
- La tercera tarea, con una dificultad similar a la segunda, llevó menos que las dos tareas previas, excepto al P14.
- En términos generales, parece ofrecer un rápido aprendizaje debido a la disminución de la media de la 3 tarea respecto a las dos previas.

24.3.2.2 FASE 2

La Tabla 28 muestra las respuestas de cada participante para cada declaración de la encuesta.

	Q 1	**Q 2**	**Q 3**	**Q 4**	**Q 5**	**Q 6**	**Q 7**	**Q 8**	**Q 9**	**Q 10**	**Q 11**	**Q 12**	**Q 13**	**Q 14**	**Q 15**	**Q 16**	**Q 17**
P01	3	5	5	4	5	4	4	5	5	5	5	5	4	5	4	4	5
P02	4	5	4	4	4	4	4	4	5	5	5	4	4	5	5	4	5
P03	3	4	3	4	4	3	5	4	4	4	4	4	5	5	5	5	4
P04	4	3	5	3	3	5	5	3	5	4	4	4	5	5	4	4	5
P05	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
P06	4	4	4	4	4	5	5	4	5	5	5	5	5	4	5	5	5
P07	4	5	5	4	5	5	4	3	5	4	5	3	4	4	5	5	5
P08	5	4	4	4	5	4	3	3	4	4	2	4	4	5	5	4	5
P09	4	5	4	4	3	4	5	4	4	4	3	4	4	4	3	4	3
P10	5	5	4	5	5	4	3	4	5	4	2	4	2	5	4	5	5
P11	4	4	4	4	4	4	4	4	4	4	3	4	4	4	4	4	4
P12	5	5	4	5	5	4	4	5	5	4	5	5	5	4	5	5	5
P13	4	5	5	5	4	5	5	3	5	5	5	5	5	4	4	5	5
P14	5	4	3	4	4	4	2	3	3	3	4	3	4	3	3	3	4
P15	4	5	4	5	5	5	4	4	4	4	2	4	4	4	4	4	4
P16	5	5	5	5	5	5	4	4	4	4	5	5	5	5	5	5	4
P17	5	5	4	5	5	4	4	4	5	5	4	4	5	5	5	4	4
P18	5	5	5	5	5	4	4	4	5	5	5	5	5	5	5	5	5

Tabla 28 Respuestas de los participantes para cada declaración de MUCSL

La Tabla 29 contiene las estadísticas descriptivas globales de la evaluación de la encuesta de MUCSL. Esta tabla muestra el mínimo, el primer cuartil, la mediana o segundo cuartil, el tercer cuartil, el máximo, el rango entre cuartiles y la moda de cada una de las 17 declaraciones de la encuesta.

Un lenguaje orientado al usuario para especificar interconexiones entre objetos heterogéneos y ubicuos en Internet de las Cosas

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17
Mínimo	3	3	3	3	3	3	2	3	3	3	2	3	2	3	3	3	3
Cuartil 1	4	4	4	4	4	4	4	3,25	4	4	3,25	4	4	4	4	4	4
Mediana	4	5	4	4	5	4	4	4	5	4	4,5	4	4,5	5	5	4,5	5
Cuartil 3	5	5	5	5	5	5	5	4	5	5	5	5	5	5	5	5	5
Máximo	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Rango	2	2	2	2	2	2	3	2	2	2	3	2	3	2	2	2	2
Rango inter.	1	1	1	1	1	1	1	0,75	1	1	1,75	1	1	1	1	1	1
Moda	4	5	4	4	5	4	4	4	5	4	5	4	5	5	5	5	5

Tabla 29 Tabla con las estadísticas descriptivas globales de MUCSL

En la Ilustración 129 se pueden ver los datos de cada pregunta representados en un diagrama de cajas y bigotes.

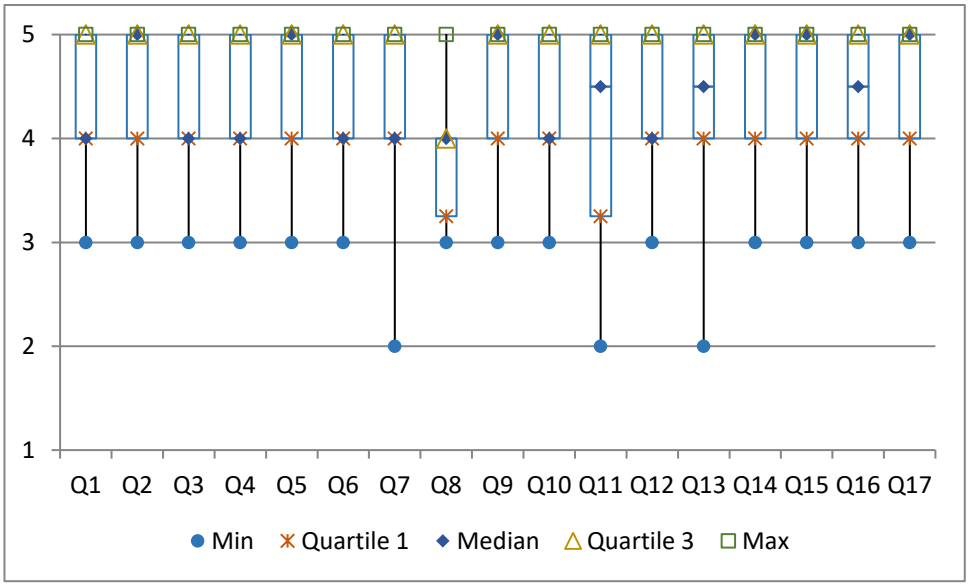


Ilustración 129 Diagrama de cajas y bigotes global para cada declaración de MUCSL

Así, a partir de los datos recogidos y mostrados en la Tabla 29 y la Ilustración 129 se pueden sugerir las siguientes interpretaciones:

- Todas las declaraciones tienen un máximo de 5, lo que indica que al menos 1 persona ha estado totalmente de acuerdo con cada una.
- Todas las declaraciones, exceptuando Q7, Q11 y Q13, tienen un mínimo de 3, lo que indica que en esas declaraciones la peor valoración ha sido «*neutral*».
- Q7, Q11 y Q13 tienen el menor mínimo en «*desacuerdo*».
- Q2, Q5, Q9, Q14, Q15 y Q17 tienen la mediana más alta. De esto podemos deducir que la mayoría de los participantes están totalmente de acuerdo con estas declaraciones.
- Q7, Q11 y Q13 tienen un rango de 3, lo que expresa que hay una gran diversidad de opiniones sobre estas declaraciones. El resto de las declaraciones tienen un rango de 2, lo que indica que los participantes tienen una opinión bastante similar en esas declaraciones.
- De acuerdo con la moda, se puede observar Q2, Q5, Q9, Q11, Q13, Q14, Q15, Q16 y Q17 tienen una moda de 5, lo cual indica que la mayoría de las respuestas elegidas han sido «*Totalmente de acuerdo*». El resto tienen la moda en 4, lo que indica que la respuesta más elegida ha sido «*de acuerdo*».
- Q11 con un rango de 3, un mínimo de 2, una mediana de 4,5, una moda de 5, un rango intercuartílico de 1,75 y un máximo de 5, es la declaración más dudosa, a pesar de que la respuesta más repetida es «*Totalmente de acuerdo*». Por otro lado, mirando Q7, que tiene un rango de 3, un mínimo de 2, una mediana de 4, una moda de 4, un rango intercuartílico de 1 y un máximo de 5, otorgando a Q7 como la declaración peor valorada. Sin embargo, Q13 con un rango de 3, un mínimo de 2, una mediana de 4,5 una moda de 5, un rango intercuartílico de 1 y un máximo de 5 está muy bien valorada, aunque hay algún participante que no está de acuerdo.
- Q8 es una de las declaraciones peor valoradas pues tiene un rango de 2, un mínimo de 3, una mediana de 4, es la única con el cuartil 3 en 4, un rango intercuartílico de 0,75 y una moda de 4. Esto indica que la mayoría de participantes están solo de acuerdo con ella y tienen una opinión muy parecida entre ellos, pues es la declaración con menor rango intercuartílico.

La Tabla 30 muestra las diferentes frecuencias obtenidas de la encuesta para declaración en base a las respuestas elegidas por los participantes. Esta tabla contiene el desglose de cada pregunta para mostrar el número de votos para cada decisión y su porcentaje correspondiente.

Declaración		Totalmente en desacuerdo	En desacuerdo	Neutral	De acuerdo	Totalmente de acuerdo
Q1.	#	0	0	2	8	8
	%	0%	0%	11%	44%	44%
Q2.	#	0	0	1	5	12
	%	0%	0%	6%	28%	67%
Q3.	#	0	0	2	9	7
	%	0%	0%	11%	50%	39%
Q4.	#	0	0	1	9	8
	%	0%	0%	6%	50%	44%
Q5.	#	0	0	2	6	10
	%	0%	0%	11%	33%	56%
Q6.	#	0	0	1	10	7
	%	0%	0%	6%	56%	39%
Q7.	#	0	1	2	9	6
	%	0%	6%	11%	50%	33%
Q8.	#	0	0	5	10	3
	%	0%	0%	28%	56%	17%
Q9.	#	0	0	1	6	11
	%	0%	0%	6%	33%	61%
Q10.	#	0	0	1	10	7
	%	0%	0%	6%	56%	39%
Q11.	#	0	3	2	4	9
	%	0%	17%	11%	22%	50%
Q12.	#	0	0	2	9	7
	%	0%	0%	11%	50%	39%
Q13.	#	0	1	0	8	9
	%	0%	6%	0%	44%	50%
Q14.	#	0	0	1	7	10
	%	0%	0%	6%	39%	56%
Q15.	#	0	0	2	6	10
	%	0%	0%	11%	33%	56%
Q16.	#	0	0	1	8	9
	%	0%	0%	6%	44%	50%
Q17.	#	0	0	1	6	11
	%	0%	0%	6%	33%	61%

Tabla 30 Tabla de frecuencias de las respuestas globales de MUCSL

La Ilustración 130 muestra un gráfico de barras la frecuencia de las respuestas de todos los participantes.

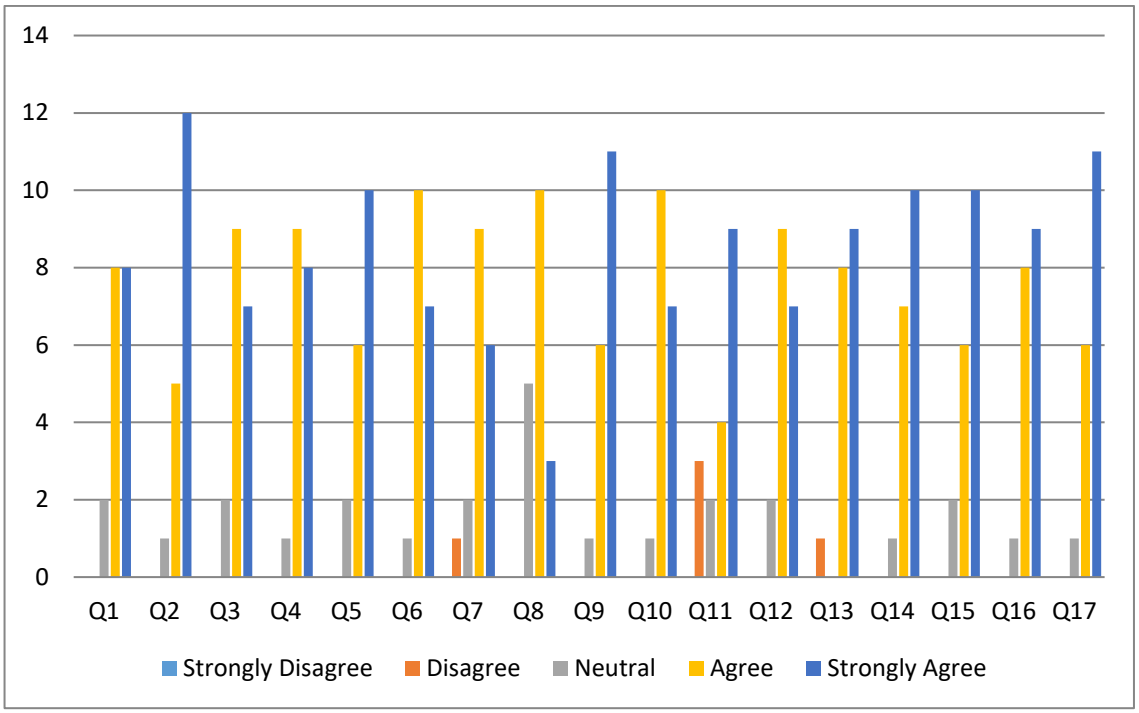


Ilustración 130 Distribución de las respuestas globales de MUCSL

La Ilustración **131** muestra las respuestas de las diferentes declaraciones mediante un gráfico de barras apilado marcando los percentiles.

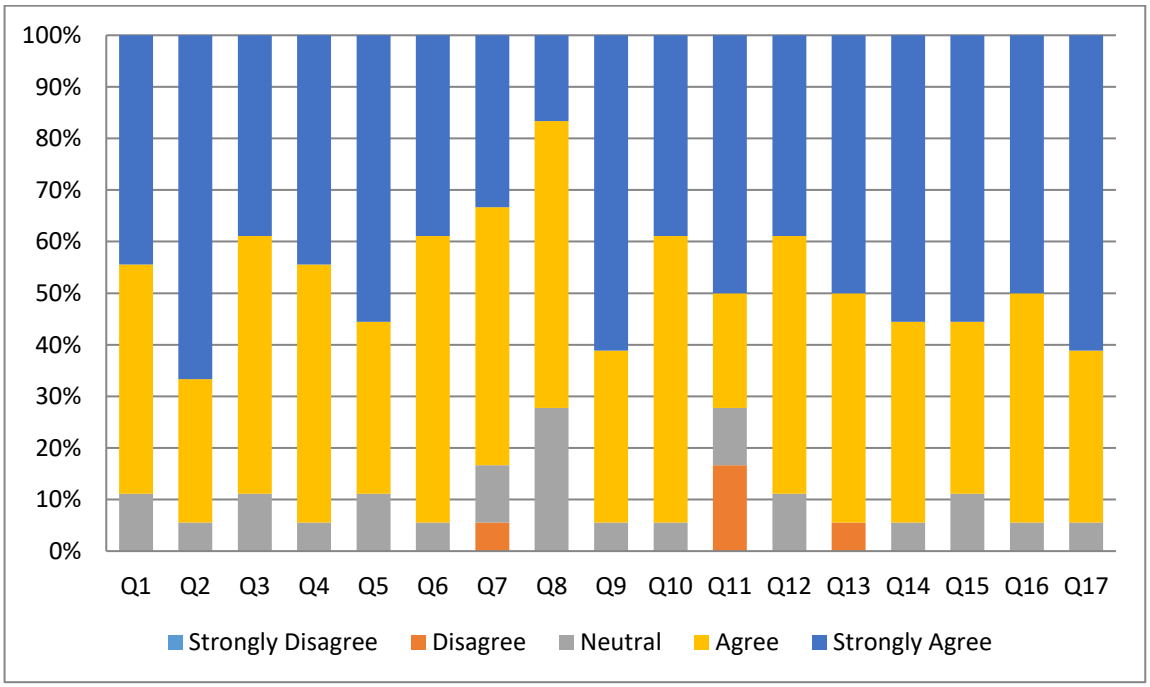


Ilustración 131 **Distribución apilada de las respuestas globales de MUCSL**

Basándonos en los últimos datos mostrados, se pueden sugerir las siguientes interpretaciones:

- Q2 posee un 67% de los votos como «*Totalmente de acuerdo*» y un 28% en «*de acuerdo*», mientras que solo el 6% de los participantes votaron «*Neutral*». En número de votos esto es 12, 5 y 1, respectivamente. Esto indica que los participantes están de acuerdo como mínimo en esta declaración salvo una pequeña minoría, siendo esta la declaración mejor valorada.
- Q9 y Q17 son las siguientes dos declaraciones más valoradas, cambiando un voto «*Totalmente de acuerdo*» a «*de acuerdo*».
- Por otro lado, Q11 es la declaración con peores opiniones y con más respuestas dispares al poseer un 50% de los votos en «*Totalmente de acuerdo*», un 22% en «*de acuerdo*», un 11% en «*neutral*» y un 17% en «*desacuerdo*». No obstante, Q7 es la peor valorada al tener un 33% de los votos en «*Totalmente de acuerdo*», un 50% en «*de acuerdo*», un 11% en «*neutral*» y un 6% en «*desacuerdo*».

24.4 *CONCLUSIONES*

En esta iteración se presenta un novedoso aproximamiento para permitir que los usuarios finales creen dinámicamente la interconexión de objetos en Internet de las Cosas.

Primeramente, la propuesta es una parte integral del desarrollo de procesos para aplicaciones basadas en IoT, las cuales incluyen el registro de objetos en una plataforma, describen sus interconexiones en casos de uso, transformando estos casos de uso en código fuente, y, finalmente, ejecutándolo como una aplicación IoT para interconectar los objetos.

Además, esta aproximación provee a los usuarios finales de un sistema para crear las aplicaciones automáticamente a partir de sus casos de uso, siempre que cumplan unas reglas básicas de MUCSL, que se encarga de traducirlos y crear las aplicaciones interconectoras que funcionan utilizando la plataforma Midgar. Esto permite que los usuarios creen estas aplicaciones sin necesidad de programar.

Los resultados de la evaluación en base al cuestionario realizado con los participantes dicen estar de acuerdo en que esta herramienta ofrece una asistencia útil para interconectar objetos, es una contribución positiva para fomentar el desarrollo de servicios y aplicaciones para Internet de las Cosas y en que el lenguaje puede considerarse útil.

Por otro lado, cabe destacar que habría que mejorar ciertos aspectos en el que algunos participantes no están de acuerdo, como es el caso de Q7, Q8 y Q13, donde hay alguna duda respecto a que los usuarios no requieren conocimientos de programación altos, además de que tal vez la sintaxis requiera más elementos y una mejora de la funcionalidad, así como ser un poco más intuitiva. Luego, habría que intentar mejorar este aspecto en un trabajo futuro para poder permitir un lenguaje mucho más natural que el actual, ofreciendo nuevas opciones de modelo, pero teniendo el cuidado de no complicar el proceso de interconexión, pues lo que se pretende es facilitarlo todo lo posible.

La declaración peor valorada ha sido Q11, que afirma si se podría utilizar en otras áreas. Claramente, esta no es posible en su totalidad debido a que se hizo específicamente para crear la interconexión entre objetos y si se requiriese adaptarla a otras áreas habría que adaptar todo el proceso.

Así, en función de los resultados de la evaluación, se puede decir que este lenguaje ha cumplido los objetivos propuestos que permiten reducir la complejidad de crear este tipo de aplicaciones y acercar el mundo de IoT a los usuarios finales de una forma más cercana a la natural. No obstante, queda mucho trabajo futuro por delante.

25. MIBOT: ASISTENTE PERSONAL AUTOMATIZADO PROGRAMABLE MEDIANTE EL USO DE INTERNET DE LAS COSAS

«Un todo es más que la suma de sus partes»
Aristóteles

«Uno se alegra de resultar útil»
Andrew Martin, en «El hombre bicentenario» de Isaac Asimov

«Los robots cumplirán una función principal en la asistencia a personas de edad avanzada, a las que incluso podrían prestar compañía»
Bill Gates

Con la aparición de diferentes módulos para los objetos y las posibilidades de combinación que surgieron entre los diferentes objetos, se consiguieron crear objetos mucho más inteligentes. Muchos de estos son los conocidos como robots. En algunos casos, estos se usan para ayudar a personas con discapacidades y problemas psicomotrices. Esto hace que surja la necesidad de permitir a estas personas manejarlos y configurarlos.

No obstante, los robots tienen ciertos problemas respecto al resto de *Smart Objects*. En ciertos aspectos son parecidos a los smartphones, pues algunos de estos vienen como un conjunto inmodificable, mientras que otros son similares a los microcontroladores, ya que se pueden cambiar diferentes piezas a gusto del usuario. Sin embargo, el cambio más drástico en su inteligencia, pues algunos funcionan mediante instrucciones predefinidas que hay que ir llamando o se ejecutan bajo ciertos eventos, en otros casos hay que crear estas acciones o movimientos, y en otros casos tienen un sistema de Inteligencia Artificial.

Por ello, para conseguir facilitar el uso de los robots, se ha llevado a cabo esta nueva iteración, siendo una ampliación de MOCSL, y así conseguir que esto pueda ser posible. En esta iteración se investigarán diferentes tipos de entornos con diferentes tipos de robots y diferentes combinaciones de objetos asistentes conectados a Internet para dotarlos y adaptarlos de todo lo necesario para que sean accesibles y usables por personas sin conocimientos de desarrollo.

25.1 *DESCRIPCIÓN*

25.1.1 **OBJETIVOS**

Dado el aumento constante de robots en el mercado, se hace indispensable ampliar MOCSL para incluir la posibilidad de ofrecer un sistema sencillo para programar diferentes tipos de robots y que puedan utilizarlo en escenarios de Internet de las Cosas. Para esto, esta iteración debe de cumplir los siguientes objetivos:

- Desarrollar un Lenguaje de Dominio Específico que permita la creación de aplicaciones que interconecten robots a una plataforma IoT.
- Obtener una alta abstracción para la creación de aplicaciones mediante un DSL web gráfico.
- Encapsular todo el software que el robot requiere para ofrecer diferentes funcionalidades y conectarse a la plataforma destino.

25.1.2 **FUNCIONALIDADES**

Con el fin de cumplir los anteriores objetivos, se ha utilizado la plataforma Midgar y, concretamente, se ha ampliado MOCSL, conociendo esta parte de robots como MiBot. Esto permite utilizar un sistema ya probado con el fin de tratar de ofrecer un sistema mejora que permita crear software para objetos mucho más complejos que vienen ya conectados y con un software, muchas veces, propietario. Este es el caso de los robots Pleo y Lego Mindstorm.

Así, para lograr el objetivo, se ha ampliado el DSL gráfico MOCSL y se ha creado el procesador y las plantillas de cada robot. Para ello, se ha tenido que investigar cómo funciona internamente el software de cada robot y crear una plantilla modificable para permitir utilizar todas las funcionalidades de los robots en base a lo que definan los usuarios, además de contener toda la lógica necesaria para mantener la conexión con la plataforma IoT.

25.2 *ARQUITECTURA PROPUESTA*

La arquitectura del sistema es exactamente igual a la de MOCSL, pues lo que se ha realizado es ampliar MOCSL implementando un nuevo editor que ofreciese la posibilidad de programar para Pleo 2009 y el Lego Mindstorm. Además, se amplió el procesador para poder crear los programas demonio de estos dos objetos.

A continuación, se presentará la implementación de MiBot en Midgar, tanto en su versión textual como en la gráfica. La primera por ser donde se persiste el modelo formal y la segunda acompañada de una descripción del editor gráfico que la apoya.

En este caso, se obviará totalmente la explicación del funcionamiento del procesador debido a que este es similar al ya explicado anteriormente en MOCSL. Lo mismo sucede con la tercera y cuarta capa de Midgar, pues no se han tenido que modificar, siendo así la única diferencia la adición de dos nuevos objetos, el Lego Mindstorm NXT y el Pleo 2009.

25.2.1 IMPLEMENTACIÓN

Con MOCSL, los usuarios pueden crear la lógica necesaria para sus objetos sin necesidad de escribir código. Ahora, con MiBot se permite realizar esto, pero para robots.

Parte de la infraestructura es igual, cambiando solo los DSLs y mejorando los procesadores, pero manteniendo toda la arquitectura idéntica. Por eso mismo, a continuación, se mostrarán los nuevos DSLs, pues tanto las capas como el generador de procesos son iguales a los presentados en MOCSL, claramente, mejorándolos y ampliándolos para que funcionen con MiBot.

25.2.1.1 LENGUAJE DE DOMINIO ESPECÍFICO TEXTUAL

En este apartado se presenta el DSL textual utilizado para crear una transformación intermedia entre el DSL gráfico y la aplicación final y así poder persistir el modelo formal. Este DSL está basado en XML.

La primera parte del DSL es donde se define la red IoT que se va a utilizar, el nombre de la aplicación, el dispositivo que se utilizará y el modelo exacto de este. En el ejemplo del Código fuente 29 se muestra un ejemplo en el que se ha utilizado la red IoT Midgar y el Lego Mindstorm NXT.

```
<application iotNetwork='http://156.35.82.104:3000' name='test'
deviceType='Lego' exactlyDevice='NXT'>
            ...
</application>
```

Código fuente 29 Encabezado del DSL textual de MiBot

La parte intermedia puede llevar dos tipos de nodos, ***sensor*** y ***action***, como se muestra en el Código fuente 30, que contiene todas las opciones del NXT. Estos nodos se pueden repetir 4 y 2 veces respectivamente, siendo los sensores los pines 1, 2, 3 y 4 del NXT y las acciones los pines A, B y C. La diferencia entre ambos radica en que el sensor lleva el pin al que va conectado mientras que las acciones, que son para los movimientos de los motores, están predefinidas para facilitar el uso de los motores, permitiendo elegir entre un motor diferencial que se enchufa al pin A y C o uno normal que va enchufado al pin B.

```
<sensor pin='1'>touch</sensor>
<sensor pin='2' sound</sensor>
<sensor pin='3'>light</sensor>
<sensor pin='4'>ultrasonic</sensor>
<action>Differential:MotorsinAyC</action>
<action>Motor:inB</action>
```

Código fuente 30 Nodos *sensor* y *action* del DSL textual de MiBot

Por último, el Código fuente 31 muestra un ejemplo de la construcción del camino. Aquí, el nodo *path* solo puede existir una vez, pero puede contener tantos nodos *instruction* como se desee, pues estas contienen las instrucciones de forma secuencial que debe de seguir el robot para realizar el camino.

```
<path name ='newPath'>
     <instruction valor='90'>Adelante</instruction>
     <instruction valor='45'>Derecha</instruction>
     <instruction valor='120'>Adelante</instruction>
</path>
```

Código fuente 31 Ejemplo de camino con el DSL textual de MiBot

25.2.1.2 Lenguaje de Dominio Específico Gráfico

La Ilustración 132 muestra una captura de MiBot con la implementación del Lego Mindstorm NXT. Como se ve, MiBot sigue el mismo estilo que MOCSL, pues dispone de las mismas 4 áreas importantes.

Ilustración 132 MiBot Lego Mindstorm NXT

La primera área, que se corresponde con la Ilustración 132ª, contiene el formulario donde se seleccionan los datos de la aplicación, como son la plataforma IoT que utilizará, el nombre de la aplicación, el tipo de dispositivo (Lego Mindstorm o Pleo), y el modelo exacto, NXT en el caso de Lego Mindstorm y 2009 en el caso del Pleo. Una vez el usuario ha seleccionado el modelo exacto, este es mostrado en el *canvas*, zona de color gris claro. En este ejemplo se ha utilizado el Lego Mindstorm NXT.

La Ilustración 132B muestra el área donde el usuario puede seleccionar los pines del Lego Mindstorm NXT para indicar que actuador o sensor irá conectado a ese pin. La Ilustración 132C contiene la selección por medio de dos *checkbox* que sirve para indicar si los motores, en caso de contenerlos, funcionarán como diferencial, que entonces deberán ir en el pin A y Pin C enchufados, o como un solo motor, que entonces habría que enchufarlos en el pin B. Cuando el usuario selecciona uno de estos pines, se muestra un *popup* con los diferentes sensores y actuadores existentes en el Lego seleccionado. Un ejemplo es el mostrado en la Ilustración 133. Este *popup* permite al usuario seleccionar los sensores y actuadores que desee para cada pin. En cambio, si hace clic en el botón «*Cancel*», el usuario cerrará el *popup* y borrará la selección previa, dejando libre ese pin.

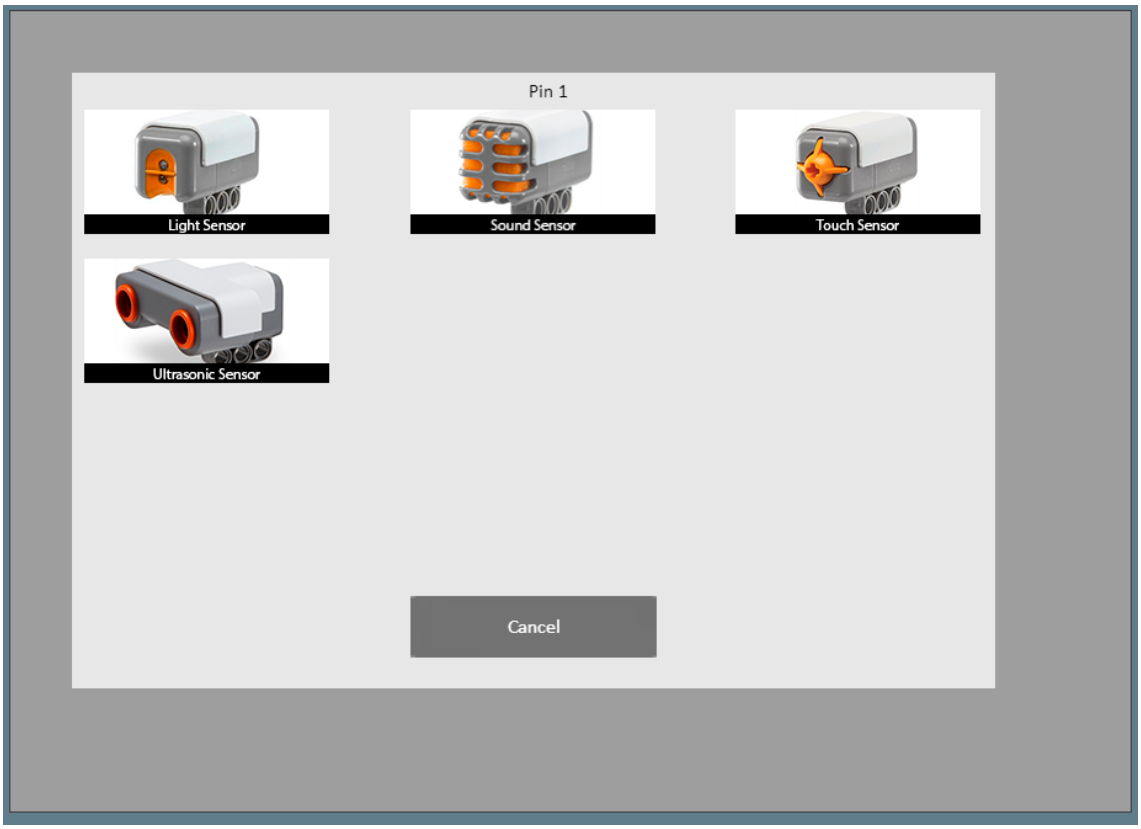


Ilustración 133 Sensores y actuadores del Lego NXT

La última área, que se puede ver en la Ilustración 132D, contiene el creador de caminos. Este permite al usuario crear un camino al Lego y asignárselo mediante un nombre. De esta manera, cuando sea llamado, el Lego hará ese camino. Este creador de caminos permite elegir la dirección en la que andará el Lego y cuantos metros, pudiendo así contener un camino de cualquier longitud.

Tras definir todo, el usuario solo debe de dar al botón «*Generate XML*» con el fin de obtener generar el modelo formal persistido que utiliza la sintaxis textual de este DSL, y así crear la aplicación final.

Por otro lado, si se selecciona el Pleo, la foto que muestra el editor cambia, como se ve en la Ilustración 134. En este caso muestra una foto del Pleo 2009 mostrando en una capa superior la ubicación de todos los sensores junto al nombre del sensor. Estos, por defecto, salen desactivados y por eso se muestra el símbolo de desactivado (⛔). En caso de activar alguno de ellos, el símbolo de ese sensor cambiaría a activado (✅).

Ilustración 134 Sensores activables de Pleo 2009

La diferencia con el Lego Mindstorm es, en el caso del Pleo, el botón del área Ilustración 132C. Este botón cambia la interfaz para mostrar las acciones disponibles a realizar por Pleo. Estas acciones y cambio de interfaz se muestra en la Ilustración 135. Cabe destacar que estas acciones deben de estar almacenadas en la tarjeta Secure Digital (SD) de Pleo para que puedan funcionar.

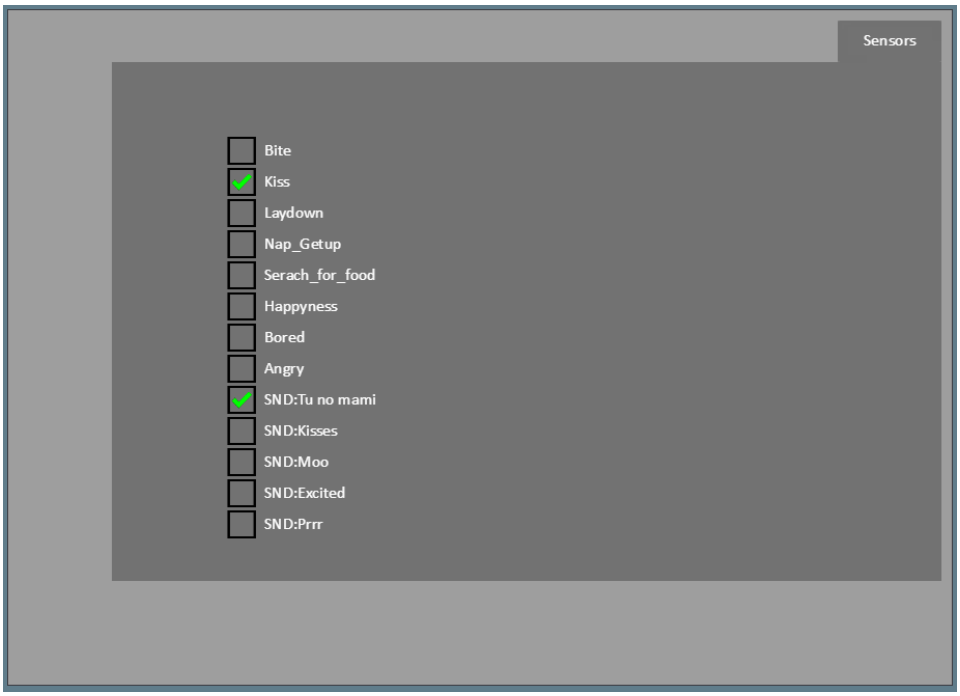


Ilustración 135 Acciones ejecutables de Pleo 2009

El otro aspecto que cambia es el área Ilustración 132D, pues no posee caminos a predefinir debido a que Pleo no ofrece esta opción. A cambio, se ofrece la posibilidad de descargar una de las personalidades de Pleo, que son las mostradas en la Ilustración 136. El problema reside en que Pleo solo permite activar una personalidad o ejecutar acciones de las anteriores, no ambas cosas al mismo tiempo.

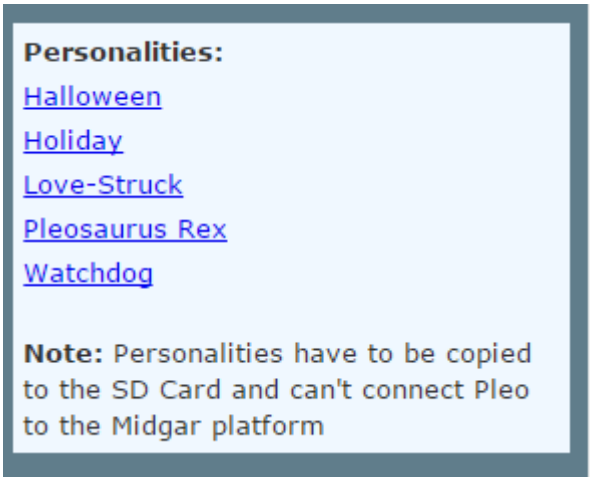


Ilustración 136 **Personalidades de Pleo**

25.2.2 SOFTWARE Y HARDWARE UTILIZADO

- La plataforma IoT Midgar:
  - Ruby 2.2.2p95.
  - Rails 4.2.1.
  - Servidor web Thin 1.6.3.
  - Base de datos MySQL 5.5.43.
  - MUCSL, el cual ha sido desarrollado utilizando HTML5, PHP 5.6 y JavaScript estándar.
  - Para la creación del generador de aplicaciones y la aplicación de conexión con el Arduino al ordenador se optó por utilizar Java 8.

Para las pruebas se utilizó el siguiente equipo hardware:

- Máquina virtual sobre VMware con Ubuntu 14.
- Pleo 2009
- Lego Mindstorm NXT
  - Sensores de luz, sonido, toque y ultrasonidos.

25.3 *CONCLUSIONES*

En esta ampliación de la iteración número 3 se ha ampliado el DSL MOCSL, con MiBot, para que permita la generación de programas para interconectar robots a la plataforma IoT Midgar mediante un sistema similar, pero adaptado a los robots.

Midgar ya poseía una solución para crear la interconexión de los objetos, MOISL. Con MOCSL se podía crear el software necesario para crear *Smart Objects* en smartphones Android y microcontroladores Arduino. Ahora, con MiBot, se demostró que se puede hacer lo mismo para robots, en este caso, para el Pleo 2009 y el Lego Mindstorm NXT. Es decir, los usuarios ahora pueden crear el software que necesitan estos robots para funcionar como los propios usuarios deseen, permitiéndoles además que se registren en la plataforma IoT Midgar. Todo esto sin necesidad de tener conocimientos de desarrollo de software.

«En mi final está mi principio»
Mary, Queen of Scots

«En algún lugar, algo increíble está esperando ser conocido»
Carl Sagan

«Podría parecer que hemos llegado a los límites alcanzables por la tecnología informática, aunque uno debe ser prudente con estas afirmaciones, pues tienden a sonar bastante tontas en cinco años»
John Von Neumann, aproximadamente en 1949

Bloque VI Conclusiones y trabajo futuro

Contenidos de bloque

26. FINAL CONCLUSIONS

'When we think that the day of tomorrow will never come, it has already transformed in yesterday'
Henry Ford

'We crave for new sensations but soon become indifferent to them.
The wonders of yesterday are today common occurrences'
Nicola Tesla

'There is a single light of science, and to brighten it anywhere is to brighten it everywhere'
Isaac Asimov

Once the doctoral thesis was shown together with the prototypes, only remains showing the conclusions about this doctoral thesis, which will answer the next doubts: Will it have accomplished the objectives presented? Will it have verified the hypothesis? Will the hypothesis have been true or false? Finally, what has been the entirety of this doctoral thesis contributions? And their publications? How much valuable has been this doctoral thesis for the Internet of Things?

These are the questions that it has to answer straightaway, because it has been a long and extensive doctoral thesis, working a topic in a very generic way, as is the Internet of Things, mixing it with many other fields, as have been Model-Driven Engineering, Security, Online Social Networks, different Artificial Intelligence subfields, and Big Data. This could have got diverted this doctoral thesis or maybe this could have helped to achieve the objectives, in the case that have not been both.

26.1 *VERIFICATION OF OBJECTIVES*

In chapter 1.3 was shown as a main objective the next:

> To develop a research in the Internet of Things scenario **together** with the use of Model-Driven Engineering to enable users without knowledge of software development to build applications for the Internet of Things in an easy, nimble, and secure way.

Nevertheless, to achieve the objective, it was itemised in various subobjectives, which are commented following together with their verification, evaluation, and accomplishment degree.

1. **Designing *Smart Objects***: A user without knowledge of software development will be able to create so many *Smart Objects* as he wants without the need to have the **technical** knowledge and connect them to the network and with other objects in an easy and quick way.

The first objective was complimented in different iterations did in the prototypes. These have been the corresponding with the chapters of the 'Bloque V'. In these chapters, respectively, it has achieved to create the objects interconnection through a graphic Domain-Specific Language (DSL), creating the objects software, use pictures as sensors, creating the objects interconnections but from the use cases, and creating software for robots.

2. **Security and privacy**: The data that is sent and received by the objects could be intercepted, to obtain information as well to modify the messages and impede an activity, or obtain the control of an object. Therefore, this is a very delicate topic. Because of this, a study must be done about the current security when the objects send messages amongst themselves in order to try to obtain a reliable, secure, and suitable system for the correct network performance, at the same time that the system is transparent or very easy to use for any type of user.

This second objective is very important, especially with what happens nowadays. The objective has been demonstrated in chapter 23, where has been a study with different encryption algorithms to be able to define what are the best algorithms to use in IoT devices, according to the data processing capacity of each device.

3. **Servant objects**: With the appearance of different modules for the objects and the combination possibilities that appear between the different objects, these have allowed creating smarter objects. This is the case of robots. In some cases, these objects are used to help disabled people. In this way, appears the necessity of allowing these people to manage and set up these objects. Because all of this, it will investigate different types of objects to connect them with the Internet and/or Online Social Networks to provide and adapt them with all the necessary to create more accessible and usable objects for any type of people.

This last objective was achieved by the extension of MOCSL in the last iteration, which was shown in chapter 25. In this chapter was shown the extension and how the users can create the software that they want for two robots, the Pleo 2009 and the Lego Mindstorm NXT, allowing defining the sensors to read and the actions to be done.

With this, it is demonstrated the achievement of the objectives arisen that were necessary to fulfil to corroborate the hypothesis of this doctoral thesis, which is therefore verified, and if we remember, the hypothesis was the following:

Applying the Model-Driven Engineering to achieve the creation of interaction human-to-machine and machine-to-machine in an easy and quick way by the automation or **semiautomation** of different processes to develop the applications in a nimble way and reduce the mistakes and costs in the Internet of Things scenarios, at the same time that the user privacy is maintained.

26.2 *CONTRIBUTIONS OF THIS DOCTORAL THESIS*

This subsection lists the novel and original contributions of this thesis:

- An Internet of Things (IoT) platform that supports the heterogeneous objects registry and exchange data between the objects, allowing interconnect between themselves, which does as the brain of the whole ecosystem.
- The use of Model-Driven Engineering (MDE) to facilitate the software development for IoT scenarios.
- One graphic DSL and another textual DSL to create the interconnection between heterogeneous and ubiquitous objects in the Internet of Things.
- A graphic DSL that allows creating the necessary software to define *Smart Objects* as the user wants as well as their interconnection using an IoT platform.
- The Computer Vision integration in the Internet of Things.
- The use of pictures as sensors.
- A security study that shows the best cryptography algorithms in relation to their security/computational cost to create secure messages between *Smart Objects*.

- The possibility of creating the object interconnection from a use case using Natural Processing Language to process it.
- A DSL that allows defining actions for robots to use them in IoT scenarios.

Other contributions derivate from this doctoral thesis, but which do not belong to the main objective of this doctoral thesis, but that belong to the IoT scope and that have been published and contrasted by the scientific community:

- Objects interconnection in IoT using Online Social Networks (OSN).
- A general vision of the IoT's current state, including tools, technologies, lines of investigation, and the related work.
- Creation of an IoT system to emulate the senses of the human body called IntelliSenses.
- A study of the objects existing in IoT scenarios and their differences, as well as a possible classification for the *Smart Objects*.
- A general view about the use of models and MDE, discovering their history, the different approaches, the most relevant implementations, their terminology, their lifecycle, and their characteristics and problems.
- SenseQ: A system to gather information from Online Social Networks and sensors to achieve those users can ask questions to the system. Then, the system will answer the users with answers based on the collected information.
- A general view of the Big Data current state which includes the differences between Big Data, Data Mining, and KDD, the Big Data tools, their importance, their scope of use, the lines of investigation, and the related work.

26.3 *DERIVED PUBLICATIONS*

During the present doctoral thesis, different contributions have been made relating the main line of this doctoral thesis, which have been presented to the scientific community by means of book chapters, conferences, articles (not JCR), and JCR articles. The contributions are the followings:

- Book chapters
    1. [39] C.G. García, J.P. Espada, Using Model-Driven Architecture Principles to Generate Applications based on Interconnecting *Smart Objects* and Sensors, in: V.G. Díaz, J.M.C. Lovelle, B.C.P. García-Bustelo, O.S. Martinez (Eds.), Adv. Appl. Model. Eng., IGI Global, 2014: pp. 73–87.
    2. [43] C.G. García, J.P. Espada, MUSPEL: Generation of Applications to Interconnect Heterogeneous Objects Using Model-Driven Engineering, in: V.G. Díaz, J.M.C. Lovelle, B.C.P. García-Bustelo (Eds.), Handb. Res. Innov. Syst. Softw. Eng., IGI Global, 2015: pp. 365–385.

Final Conclusions

- Conferences
    1. [42] C.G. García, J.P. Espada, E.R.N. Valdez, V.G. Diaz, Midgar: Domain-Specific Language to Generate *Smart Objects* for an Internet of Things Platform, in: 2014 Eighth Int. Conf. Innov. Mob. Internet Serv. Ubiquitous Comput., IEEE, Birmingham, United Kingdom, 2014: pp. 352–357.
    2. [46] D. Meana-Llorián, C. González García, B.C. Pelayo G-Bustelo, J.M. Cueva Lovelle, V.H. Medina García, IntelliSenses: Sintiendo Internet de las Cosas, in: 2016 11th Iber. Conf. Inf. Syst. Technol., IEEE, Gran Canaria, Spain, 2016: pp. 234–239.
    3. [47] D. Meana-Llorián, C.G. García, V. García-Díaz, B.C.P. G-Bustelo, J.M.C. Lovelle, SenseQ: Creating relationships between objects to answer questions of humans by using Social Networks, in: Proc. 3rd Multidiscip. Int. Soc. Networks Conf. Soc. 2016, Data Sci. 2016 - MISNC, SI, DS 2016, ACM Press, New York, New York, USA, 2016: pp. 1–5.
- Articles (no JCR)
    1. [44] C.G. García, J.P. Espada, B.C.P. G-Bustelo, J.M. Cueva Lovelle, Swift vs. Objective-C: A New Programming Language, Int. J. Interact. Multimed. Artif. Intell. 3 (2015) 74–81.
    2. [45] C.G. García, J.P. Espada, B.C.P. G-Bustelo, J.M.C. Lovelle, El futuro de Apple: Swift versus Objective-C, Redes Ing. 6 (2015) 6–16.
    3. [48] C. González García, D. Meana-Llorián, B.C.P. G-Bustelo, J.M.C. Lovelle, A review about *Smart Objects*, Sensors, and Actuators, Int. J. Interact. Multimed. Artif. Intell. 4 (2017) 7–10.
- JCR articles
    1. [40] C.G. García, C.P. García-Bustelo, J.P. Espada, G. Cueva-Fernandez, Midgar: Generation of heterogeneous objects interconnecting applications. A Domain Specific Language proposal for Internet of Things scenarios, Comput. Networks. 64 (2014) 143–158.
    2. [41] G. Cueva-Fernandez, J.P. Espada, V. García-Díaz, C.G. García, N. Garcia-Fernandez, Vitruvius: An expert system for vehicle sensor tracking and managing application generation, J. Netw. Comput. Appl. 42 (2014) 178–188.
    3. [49] Meana-Llorián, D., González García, C., G-Bustelo, B.C.P., Cueva Lovelle, J.M., Garcia-Fernandez, N., 2016. IoFClime: The fuzzy logic and the Internet of Things to control indoor temperature regarding the outdoor ambient conditions. Futur. Gener. Comput. Syst. doi:10.1016/j.future.2016.11.020

26.4 *GENERAL CONCLUSIONS OF THIS DOCTORAL THESIS*

In this doctoral thesis, an Internet of Things web platform was developed, as known as Midgar, which allow interconnecting heterogeneous and ubiquitous objects, besides their creation. To achieve this task, it was applied Model-Driven Engineering.

In that way, with the use of MDE has been achieved the creation of some Domain-Specific Languages, both graphic and textual, which facilitate the labour of creating interconnections and software for the *Smart Objects* to people without software development knowledge. This has allowed the users and participants of the evaluations to obtain the possibility of creating the interconnections and the necessary software without needing to write lines of code and giving a very positive opinion about this system, as has been seen in chapters 20, 21, and 24. Furthermore, to improve also the creation of the lifecycle of the applications, it has been included the improvement of using Natural Language Processing (NLP), allowing for translating the use cases to the final application that will connect the necessary objects, something very relevant not only in the IoT, but in the software engineering too.

On the other hand, it has also been researched in other relevant aspects to improve the Internet of Things. One of these aspects has been the security, exactly, in cryptology. In this aspect, it was thought to choose a technique to protect the communications, mainly, if required a good efficiency. The choice of the correct algorithm was even harder. To resolve this, it has been conducted a study, where it has been investigated different cryptographic techniques and have been made efficiency test to know what algorithm combination could be more secure at the same time that this combination offers a good performance.

Other of the researches, chapter 22, has been the relative with Computer Vision, which has allowed including a computer vision module in the IoT Midgar platform. With this module it has been possible to use pictures, which have been taken with IP cameras, as sensors, allowing an improvement in the work of surveillance giving it an automate or semiautomate danger recognition.

Notwithstanding, although these researches have been a part of the main research of the doctoral thesis, as we have seen in the previous section, there have been a lot of other parallel researches connected with the IoT. Some researches have been general views about the state of the art or proposals as is in the case of *Smart Objects*, sensors, and actuators, IoT, MDE, and Big Data. On the one hand, it has worked with them as have happened with MDE, or because it is a complementary part of the IoT and they needed each other as happens with Big Data, or both cases, as this happens with the *Smart Objects*. On the other hand, these researches have gone looking for new paths of using and the inclusion of the IoT in the daily life, as happens with the mix between the IoT and the OSNs, trying to facilitate the use of the IoT for any user of that OSN. Another possibility could be acquiring information of the OSNs that complements the sensors data of a determinate IoT network. Lastly, other case is IntelliSenses, which has tried to create an IoT system that emulates the human body, being the first stone in a new path oriented to the inclusion of a greater and better intelligence for the IoT by means of the combination of sensors that are centralised in a central artificial intelligence.

As we have seen, this doctoral thesis has been long and extensive due to the research in lots of branches inside the IoT field, being this doctoral thesis generic in the IoT field, and in a continuously searching of improving

the IoT with the combination of other technologies. This has allowed having a more global vision of the IoT capabilities at the expense of a larger study of much fields inside the computer science, giving place to research that may seem unrelated to each other at first glance, as it happens with Computer Vision, Artificial Intelligence, NLP, Big Data, and MDE, but that under the same central point, which is the IoT, they are able to open unimaginable ways.

27. CONCLUSIONES FINALES

«Cuando pensamos que el día de mañana nunca llegará, ya se ha convertido en el ayer»
Henry Ford

«Ansiamos nuevas sensaciones pero enseguida nos volvemos indiferentes a ellas.
Las maravillas del ayer son los sucesos corrientes de hoy»
Nicola Tesla

«La Ciencia es una sola luz, e iluminar con ella cualquier parte es iluminar con ella el mundo entero»
Isaac Asimov

Ya mostrada toda la tesis doctoral junto a sus prototipos, solo quedan las conclusiones acerca de esta tesis, las cuales han de responder las siguientes dudas: ¿habrá cumplido los objetivos presentados? ¿Se habrá verificado la hipótesis correctamente? ¿La hipótesis habrá sido verdadera o falsa? Y finalmente, ¿cuáles han sido la totalidad de las aportaciones de esta tesis? ¿Y sus publicaciones? ¿Cuán valiosa ha sido esta tesis para Internet de las Cosas?

Estas son preguntas que se deben de responder a continuación, pues ha sido una tesis larga y extensa, tratando un tema de manera muy genérica, como es Internet de las Cosas, y mezclándolo con muchos otros campos, como han sido la Ingeniería Dirigida por Modelos, Seguridad: criptología, las Redes Sociales Online, diferentes subcampos de la Inteligencia Artificial y *Big Data*. Esto pudo tanto haber dispersado la tesis como haberla ayudado a cumplir con los objetivos.

✳✳✳✳✳

27.1 *VERIFICACIÓN DE LOS OBJETIVOS*

En el capítulo 1.3 se mostró como objetivo principal que se había planteado a resolver en esta tesis, el siguiente:

> Desarrollar una investigación en el marco de Internet de las Cosas junto con la utilización de la Ingeniería Dirigida por Modelos para permitir a los usuarios sin conocimientos de desarrollo de software construir aplicaciones para IoT de forma fácil, ágil y segura.

Sin embargo, para llevarlo mejor a cabo, se había dividido en varios subobjetivos, de los cuales se comenta a continuación su verificación, evaluación y grado de cumplimiento:

1. **Diseño de objetos inteligentes**: un usuario sin conocimientos en el desarrollo de software podrá crear tantos objetos inteligentes como desee sin necesidad de tener conocimientos técnicos y conectarlos a la red y con otros objetos de una forma fácil y rápida.

El primer objetivo se llevó a cabo en diferentes iteraciones realizadas en los prototipos. Estas han sido las correspondientes con los capítulos pertenecientes al Bloque V. En estos, respectivamente, se ha logrado crear la interconexión de objetos mediante un DSL gráfico, crear el software para los objetos, utilizar fotos como sensores, crear las interconexiones de los objetos a partir de los casos de uso y crear el software para robots.

2. **Seguridad y privacidad**: los datos enviados y recibidos de los objetos podrían ser interceptados, bien para conseguir información o bien para modificarlos y entorpecer una actividad o conseguir el control del objeto. Por lo tanto, este es un tema muy delicado. Por ello, hay que realizar un estudio acerca de la seguridad existentes cuando se envían mensajes entre objetos para intentar conseguir un sistema fiable, seguro y que sea adecuado para el correcto rendimiento de la red, a la vez que sea transparente o muy fácil de usar por cualquier tipo de usuario.

Este segundo objetivo es muy importante, sobre todo con lo que acontece actualmente. El objetivo ha sido demostrado en el capítulo 23, donde se hizo un estudio de diferentes algoritmos de encriptación para así poder definir cuáles serían los mejores a utilizar en los dispositivos IoT, según la capacidad de procesamiento de estos.

3. **Objetos asistentes**: con la aparición de diferentes módulos para los objetos y las posibilidades de combinación que surgieron entre diferentes objetos, se consiguieron crear objetos mucho más inteligentes. Este es el caso de los robots. En algunos casos, estos se usan para ayudar a personas con discapacidades y problemas psicomotrices. Así, surge la necesidad de permitir a estas personas manejarlos y configurarlos. Por esto, se investigarán diferentes formas de facilitar la configuración y el uso de robots con diferentes tipos de objetos para conectarlos a Internet y/o redes sociales para dotarlos y adaptarlos de todo lo necesario para que sean más accesibles y usables por personas cualquier tipo de persona.

Este último objetivo se consiguió mediante la ampliación de MOCSL en la última iteración de esta tesis, mostrada en el capítulo 25. En este capítulo se mostró la ampliación y como podían los usuarios crear el software que deseasen para dos robots, el Pleo 2009 y el Lego Mindstorm NXT, permitiéndoles así definir los sensores a utilizar y las acciones que quieren que los robots realicen.

Con esto, queda demostrado el cumplimiento de los objetivos surgidos y necesarios de cumplir para verificar la hipótesis de esta tesis, que por lo consiguiente queda verificada, y que, si la recordamos, era la siguiente:

Aplicar la Ingeniería Dirigida por Modelos para lograr la creación de la interacción humano-máquina y máquina-máquina de una forma fácil y rápida mediante la automatización o semiautomatización de diferentes procesos para desarrollar las aplicaciones de una manera ágil y reduciendo errores y costes en el marco de Internet de las Cosas, mientras que se mantiene la privacidad del usuario.

27.2 *APORTACIONES DE ESTA TESIS DOCTORAL*

En este subapartado se enumeran las aportaciones novedosas y originadas de esta tesis:

- Una plataforma de Internet de las Cosas que soporta el registro de objetos heterogéneos y su intercambio de datos, permitiendo interconectarlos y hacer de cerebro del ecosistema.
- El uso de Ingeniería Dirigida por Modelos para facilitar el desarrollo de software para entornos IoT.
- Un DSL gráfico y otro textual para crear la interconexión de objetos heterogéneos y ubicuos en Internet de las Cosas.
- Un DSL gráfico que permite crear el software necesario para definir los *Smart Objects* como el usuario desee y permitir su conexión a una plataforma IoT.
- La integración de la Visión por Computador en Internet de las Cosas.
- El uso de fotografías como si fueran sensores.
- Un estudio de seguridad que muestra los mejores algoritmos criptográficos en su relación seguridad/coste computacional para crear mensajes seguros entre *Smart Objects*.

- Creación de la interconexión de los objetos a partir a partir del análisis de lenguaje natural utilizado para definir sus casos de uso.
- Un DSL que permite definir las acciones de robots para ser usados en entornos IoT.

Otras aportaciones derivadas de esta tesis, que no pertenecen a su objetivo principal, pero si al ámbito de esta, que es IoT, de las cuales algunas ya han sido publicadas y contrastadas por la comunidad científica:

- Interconexión de objetos en IoT mediante el uso de redes sociales.
- Visión general del estado general de IoT que incluye sus ámbitos, herramientas, tecnologías, las líneas de investigación y el trabajo relacionado.
- Creación de un sistema IoT para emular los sentidos del cuerpo humano, llamado IntelliSenses.
- Estudio de los tipos de objetos existentes en el ámbito de IoT y sus diferencias, así como de la clasificación de los *Smart Objects*.
- Visión general del estado general del uso de modelos y de MDE, descubriendo su historia, las diferentes aproximaciones, las implementaciones más relevantes, su terminología, su ciclo de vida y sus características y problemática.
- SenseQ: un sistema que recopila información de las redes sociales y de sensores para que los usuarios puedan hacerle preguntas al sistema y este les conteste en base a esa información recopilada.
- Visión general del estado general de *Big Data* que incluye las diferencias entre *Big Data*, minería de datos y KDD, las herramientas utilizadas en *Big Data*, su importancia, su ámbito de uso, las líneas de investigación y el trabajo relacionado.

27.3 *PUBLICACIONES DERIVADAS*

Durante la presente tesis doctoral se han realizado diferentes aportaciones a la línea principal de la tesis, las cuáles han sido presentadas a la comunidad científica mediante su distribución en capítulos de libro, congresos, artículos (no JCR) y artículos JCR. Estas son las siguientes:

- Capítulos de libro
    1. [39] C.G. García, J.P. Espada, Using Model-Driven Architecture Principles to Generate Applications based on Interconnecting *Smart Objects* and Sensors, in: V.G. Díaz, J.M.C. Lovelle, B.C.P. García-Bustelo, O.S. Martinez (Eds.), Adv. Appl. Model. Eng., IGI Global, 2014: pp. 73–87.
    2. [43] C.G. García, J.P. Espada, MUSPEL: Generation of Applications to Interconnect Heterogeneous Objects Using Model-Driven Engineering, in: V.G. Díaz, J.M.C. Lovelle, B.C.P. García-Bustelo (Eds.), Handb. Res. Innov. Syst. Softw. Eng., IGI Global, 2015: pp. 365–385.

- Congresos
    1. [42] C.G. García, J.P. Espada, E.R.N. Valdez, V.G. Diaz, Midgar: Domain-Specific Language to Generate *Smart Objects* for an Internet of Things Platform, in: 2014 Eighth Int. Conf. Innov. Mob. Internet Serv. Ubiquitous Comput., IEEE, Birmingham, United Kingdom, 2014: pp. 352–357.
    2. [46] D. Meana-Llorián, C. González García, B.C. Pelayo G-Bustelo, J.M. Cueva Lovelle, V.H. Medina García, IntelliSenses: Sintiendo Internet de las Cosas, in: 2016 11th Iber. Conf. Inf. Syst. Technol., IEEE, Gran Canaria, Spain, 2016: pp. 234–239.
    3. [47] D. Meana-Llorián, C.G. García, V. García-Díaz, B.C.P. G-Bustelo, J.M.C. Lovelle, SenseQ: Creating relationships between objects to answer questions of humans by using Social Networks, in: Proc. 3rd Multidiscip. Int. Soc. Networks Conf. Soc. 2016, Data Sci. 2016 - MISNC, SI, DS 2016, ACM Press, New York, New York, USA, 2016: pp. 1–5.
- Artículos (no JCR)
    1. [44] C.G. García, J.P. Espada, B.C.P. G-Bustelo, J.M. Cueva Lovelle, Swift vs. Objective-C: A New Programming Language, Int. J. Interact. Multimed. Artif. Intell. 3 (2015) 74–81.
    2. [45] C.G. García, J.P. Espada, B.C.P. G-Bustelo, J.M.C. Lovelle, El futuro de Apple: Swift versus Objective-C, Redes Ing. 6 (2015) 6–16.
    3. [48] C. González García, D. Meana-Llorián, B.C.P. G-Bustelo, J.M.C. Lovelle, A review about *Smart Objects*, Sensors, and Actuators, Int. J. Interact. Multimed. Artif. Intell. 4 (2017) 7–10.
- Artículos JCR
    1. [40] González García, C., García-Bustelo, C.P., Espada, J.P., Cueva-Fernandez, G., 2014. Midgar: Generation of heterogeneous objects interconnecting applications. A Domain Specific Language proposal for Internet of Things scenarios. Comput. Networks 64, 143–158. doi:10.1016/j.comnet.2014.02.010
    2. [41] Cueva-Fernandez, G., Espada, J.P., García-Díaz, V., González García, C., Garcia-Fernandez, N., 2014. Vitruvius: An expert system for vehicle sensor tracking and managing application generation. J. Netw. Comput. Appl. 42, 178–188. doi:10.1016/j.jnca.2014.02.013
    3. [49] Meana-Llorián, D., González García, C., G-Bustelo, B.C.P., Cueva Lovelle, J.M., Garcia-Fernandez, N., 2016. IoFClime: The fuzzy logic and the Internet of Things to control indoor temperature regarding the outdoor ambient conditions. Futur. Gener. Comput. Syst. doi:10.1016/j.future.2016.11.020

27.4 *CONCLUSIONES GENERALES DE ESTA TESIS DOCTORAL*

En esta tesis se ha desarrollo una plataforma web de Internet de las Cosas, Midgar, que permite la interconexión de objetos heterogéneos y ubicuos, así como su creación. Para realizar esta tarea se ha aplicado Ingeniería Dirigida por Modelos.

Así, mediante el uso de MDE, se han conseguido crear varios Lenguajes de Dominio Específico, tanto gráficos como textuales, que faciliten la labor de crear las interconexiones y el software de los *Smart Objects* a los usuarios sin conocimientos de programación. Esto ha permitido que los usuarios y los participantes de las evaluaciones consiguiesen crear las interconexiones y el software sin necesidad de escribir líneas de código y dando una opinión muy positiva sobre este sistema, como bien se ha visto en los capítulos 20, 21 y 24. Además, para mejorar también el ciclo de vida de la creación de aplicaciones se ha incluido la mejora de utilizar Procesamiento de Lenguaje Natural, permitiendo así traducir los casos de uso a la aplicación que conecte los objetos, algo muy relevante no solo en IoT, sino en la ingeniería del software.

Por otro lado, también se ha investigado en otros aspectos relevantes para mejorar Internet de las Cosas. Una de ellas ha sido la Seguridad: criptología. En este aspecto, era difícil elegir una técnica para proteger las comunicaciones, sobre todo, si se requería rendimiento. Todavía más difícil era elegir un algoritmo. Para solucionar esto, se ha llevado a cabo un estudio, en el que se han investigado diferentes técnicas criptográficas y se han realizado pruebas de rendimiento para saber que combinación de ellas podría ser más segura a la vez que permitía un buen rendimiento.

Otra de las investigaciones, capítulo 22, ha sido la relativa a la Visión por Computador, que ha permitido incluir un módulo de Visión por Computador en la plataforma IoT Midgar. Gracias a esto se han podido utilizar las fotografías tomadas por cámaras IP como si fueran sensores, permitiendo así una mejora en el trabajo de vigilancia otorgándole un aspecto automático o semiautomático en el reconocimiento de peligro.

Sin embargo, a pesar de que estas han formado parte de la investigación principal de la tesis, como bien se ha visto en el apartado anterior, ha habido muchas otras investigaciones paralelas relacionadas con IoT. Algunas investigaciones han sido visiones generales del estado del arte o propuestas como es el caso de los Objetos inteligentes, sensores y actuadores, de IoT, de MDE y de Big Data. Estos trabajos han llevado a cabo una investigación rigurosa del tema principal de ellos y han aportado a esta tesis muchos conocimientos relevantes debido a que, o bien se ha trabajado con ellos, como es el caso de MDE, es una parte complementaria de IoT y se necesitan el uno al otro, como sucede con *Big Data*, o ambos casos, como son los *Smart Objects*. En otros casos, estas investigaciones han ido buscando un nuevo camino de uso y de inclusión de IoT en la vida diaria, como sucede con la mezcla entre IoT y las redes sociales, tratando de facilitar así su uso a cualquier usuario de esa red social o bien adquiriendo información de ellas que complementen los datos de los sensores de una determinada red IoT. Por último, en otro caso, en IntelliSenses, se ha tratado de crear un sistema de IoT emulando el cuerpo humano, para así comenzar un camino orientado a la inclusión de una mayor y mejor inteligencia para IoT mediante la combinación de sensores centralizados en una Inteligencia Artificial central.

Conclusiones finales

Como se ha visto, esta tesis ha sido larga y extensa debido a que se han investigado muchas ramas dentro de IoT, siendo genérica en este campo, y se ha buscado continuamente la mejora de IoT mediante la combinación de otras tecnologías. Esto ha permitido tener una visión mucho más global de las capacidades de IoT a costa de un mayor estudio de muchos campos existentes dentro de la informática, dando lugar a investigaciones que pueden parecer poco relacionadas entre sí en un primer vistazo, como es Visión por Computador, Inteligencia Artificial, NLP, *Big Data* y MDE, pero que, bajo un mismo punto central, que es IoT, son capaces de abrir caminos inimaginable.

28. TRABAJO FUTURO

«El Futuro será mejor mañana»
Dan Quayle

«La mejor forma de predecir el futuro es implementarlo»
David Heinemeier Hansson

«Cualquier tecnología lo suficientemente avanzada es indistinguible de la magia»
«La única forma de conocer los límites de lo posible es aventurarse un poco más allá de ellos, en lo imposible»
Arthur C. Clarke

Internet de las Cosas es el presente, pero también es el futuro, quedando así muchas cosas por investigar, cosas que mejorar en él y muchos problemas a resolver. Ideas para mejorar Internet de las Cosas hay muchas y han surgido otras tantas a lo largo de esta tesis. Unas se han llevado a cabo, otras se han quedado en prototipos no introducidos en la tesis y otras han quedado apuntadas en un papel.

Para que no queden en el olvido, se han recopilado las ideas que a título personas se piensa que son las más importantes y relevantes de cara a proseguir el camino de la investigación.

Trabajo futuro

Todos los trabajos futuros o ideas presentadas en este capítulo han surgido a lo largo de esta tesis doctoral, pero se han quedado fuera debido a falta de tiempo o debido a que se salían un poco del marco de esta tesis, pero que igualmente son muy relevantes en el futuro inmediato o bien sirven como vías para seguir investigando en caminos relacionados. Por eso, a continuación, se muestran las ideas de trabajo futuro que se creen que son más relevantes, pero no todas, surgidas durante la tesis.

28.1 *INGENIERÍA DIRIGIDA POR MODELOS*

Juntar MOISL y MOCSL: esto permitiría crear todo, objetos e itnerconexión, en un solo y único paso en vez de en dos separados, ofreciendo una mayor facilidad a los usuarios y tal vez abriendo nuevos caminos en la facilidad de creación de aplicaciones.

28.2 *VISIÓN POR COMPUTADOR*

Reconocimiento óptico de caracteres: OCR da la posibilidad de poder reconocer automáticamente placas o papeles identificativos para así, que podría mejorar el prototipo presentado en el capítulo 22 y que ha utilizado Visión por Computador, permitir un mayor uso de Visión por Computador en IoT.

Reconocimiento de caras: por medio de la Visión por Computador se podría restringir el acceso a determinados sitios no solo por OCR, sino que se podría restringir exactamente en función de la persona exacta por medio de la identificación de la cara de la persona, lo que daría juego a muchas aplicaciones de seguridad.

Reconocimiento de gestos: otro caso similar pero diferente es el del uso e identificación de diferentes gestos para poder enviar órdenes a un sistema por medio de una cámara. Esto permitiría utilizarlas para poder enviar una orden de ejecución simplemente utilizando una cámara. Además, por motivos de seguridad, esta ampliación podría juntarse con las dos anteriores.

Crear un DSL para facilitar el trabajo con temas de Visión por Computador: como se vio en el capítulo correspondiente, trabajar con Visión por Computador es un trabajo arduo. Por medio de un DSL este trabajo podría facilitarse y hacerse mucho más sencillo, permitiendo extraer las características indicadas y entrenar el modelo para identificar otros objetos diferentes.

28.3 *INGENIERÍA DIRIGIDA POR MODELOS E INTELIGENCIA ARTIFICIAL*

Inserción de cajas para permitir el uso de Lógica Difusa en los DSLs: esto permitiría al usuario integrar una parte de Inteligencia Artificial tanto a la interconexión como a los propios objetos «fabricados» para realizar chequeos en base a diferentes sensores y así obtener un único valor. Esto sería una opción para que los propios usuarios creasen de una forma rápida y fácil Inteligencia Artificial para los objetos por medio de la definición de las reglas de Lógica Difusa.

Mejorar el editor de *Smart Objects* utilizando Procesamiento de Lenguaje Natural: mejorar el editor que traduce los casos de uso para interconectar objetos introduciendo un mayor uso de NLP para permitir un uso más fluido del lenguaje natural sin necesidad del uso de unos patrones básicos en su construcción.

28.4 *SMART OBJECTS*

Rendimiento de los *Smart Objects*: mejorar el rendimiento de las aplicaciones en los *Smart Objects* con el fin de que consuman menos batería al utilizar de continuo los sensores, los actuadores y el envío de mensajes.

Inserción de inteligencia mediante un DSL: un DSL al estilo de MOCSL podría permitir la inclusión de inteligencia dentro del objeto para así que el mismo objeto pudiera tomar decisiones previamente a enviar sus datos, como podría ser no enviarlo hasta que no se den ciertas condiciones. Esto permitiría hacer que los objetos consumieran menos datos, procesamiento y batería, además de conseguir que la plataforma IoT estuviese menos saturada al recibir menos mensajes.

Creación de sistemas más complejos incluyendo Inteligencia Artificial: algo que podría dotar a los objetos de mayor inteligencia y un mayor uso, claramente, es la incorporación de Inteligencia Artificial, ya sea por medio de Lógica Difusa para mejorar el procesador de los sensores y crear nuevas variables, de Machine Learning para que vayan adaptándose a sus usuarios, de Visión por Computador para que sean capaces de reconocer objetos y actuar en función del objeto reconocido, o de Procesamiento de Lenguaje Natural para reconocer las órdenes de sus dueños.

Relaciones entre objetos: crear un sistema de relaciones entre objetos para que puedan pedirse automáticamente información necesaria entre objetos cercanos o similares para que puedan dar un mejor servicio del que disponen. Además, este sistema debería de disponer de un sistema de permisos para facilitar la interacción y mantener la privacidad y la seguridad.

28.5 *ESCALABILIDAD Y RENDIMIENTO*

Escalabilidad de las plataformas en Internet de las Cosas: estudios y pruebas sobre diferentes implementaciones de la plataforma IoT para comprobar con cuál se puede ofrecer una mayor escalabilidad. IoT busca conectar todos los objetos del mundo. Cada persona puede disponer, aproximadamente, de decenas de objetos (relojes, móviles, televisiones, ropa, armarios, neveras, etc.) y sólo en Europa hay casi mil millones de personas. Esto exige que la red deberá soportar una cantidad ingente de tráfico, equiparable o incluso superable al que reciben los servidores de Google o Twitter. Por ello, hay que investigar cuál sería la mejor manera de realizar un sistema escalable y resistente para soportar la red IoT. Un ejemplo de ello es Twitter, que tuvo que migrar de Ruby on Rails a Scala debido a estos problemas.

Trabajo futuro

Almacenamiento de datos (SQL vs NoSQL): estudio y pruebas sobre qué tipo de base de datos es la mejor para usar en Internet de las Cosas desde el punto de vista del rendimiento y la escalabilidad. Al igual que ocurre con la propia plataforma como se comentó antes, la base de datos es muy importante, pues si esta es lenta y funciona mal, da igual que la plataforma sea escalable. En la actualidad las bases de datos NoSQL son más rápidas que las SQL. No obstante, puede que tampoco sean la solución debido a la falta de referencias entre los datos y a su lentitud en consultas que requieran relaciones, que, en este caso, es alta.

Estudio de técnicas *Big Data* para procesar los datos de IoT: como se comentó en los dos puntos anteriores, una posible solución a ellos podría ser el uso de *Big Data*. Luego, habría que estudiar esta solución para integrar las diferentes herramientas *Big Data* en IoT de manera que permitiesen una mejora de rendimiento y una mejor velocidad en el procesamiento de datos.

Estudio de los diferentes protocolos: hay muchos protocolos y tecnologías que permiten el envío de datos: REST, *Simple Object Access Protocol* (SOAP), MQTT, WebSockets, CoAP y *Advanced Message Queuing Protocol* (AMPQ), entre otros muchos. La duda que surge aquí es: ¿cuál es el más adecuado para usar en Internet de las Cosas?

28.6 *SEGURIDAD Y PRIVACIDAD*

Mejora en la computación de los algoritmos: esto permitiría que, bajo una arquitectura igual o similar a la presentada en esta tesis, hubiera una mejora considerable de los resultados en tiempo de consumo dependiendo de la viabilidad y los contrastes de las mejoras realizadas, lo que implicaría que se pudieran utilizar algoritmos más seguros en sistemas con menos rendimiento, algo que ahora, como bien se ha demostrado, es difícil.

Registro seguro: introducción de un registro seguro en una red IoT para los objetos que se quieran registrar, permitiendo así mantener su privacidad y seguridad también durante el registro.

Optimización del envío de datos a la plataforma: bajo el sistema actual, los objetos requieren enviar grandes cantidades de información a la plataforma, los cuales podrían ser reducidos. De esta forma se reduciría el tiempo de encriptación y desencriptación de los mensajes, además de consumir menos datos móviles, y por tanto se agilizarían las comunicaciones.

Encriptación *end-to-end*: el sistema propuesto en el capítulo 23 ofrece encriptación del dispositivo al servidor y viceversa, pues es el servidor quien debe de decidir qué hacer. Esto hace que el servidor pueda ver la información privada. Una mejor solución es ofrecer encriptación *end-to-end* de forma que solo el destinatario pueda conocer los datos, no obstante, habría que solucionar como el servidor puede seguir siendo el cerebro de todo el sistema de esta manera.

DSL para configurar un servidor de forma segura: uno de los puntos críticos en la seguridad se encuentra en los servidores, pues si estos no están correctamente configurados, el servidor será inseguro. Por este motivo, se propone crear un DSL que permita configurar todos los aspectos de una forma sencilla y rápida a los usuarios inexpertos para que así pueden tener un servidor configurado correctamente.

Trabajo futuro

Aplicación de técnicas *Big Data*: como se ha visto en su capítulo correspondiente, se está poniendo de modo utilizar técnicas *Big Data* para mantener la seguridad de los sistemas, pues gracias a ellas se pueden procesar muchos más datos más rápido para intentar obtener datos acerca de un posible ataque.

28.7 *REDES SOCIALES*

Comparación y estudio de más Redes Sociales: en uno de los artículos derivados se ha utilizado Twitter como centro de interconexión de objetos de IoT. Sin embargo, esto podría extenderse a otras Redes Sociales como podría ser Facebook. Así, un trabajo futuro sería estudiar esta posibilidad, pues podría ser mejor en el caso de interconectar humanos y objetos.

Red Social de Objetos: crear una red social específica para objetos para que así puedan compartir datos entre ellos y crear de forma automática, como si fueran personas, relaciones entre ellos mismos para dar un mejor servicio para el cual están programados.

Predicción de situaciones: combinar los datos de los sensores junto los datos obtenidos de las Redes Sociales para conseguir predecir posibles situaciones o ser más rápido detectándolas, como pueden ser fuegos, inundaciones, atascos, accidentes, regulación de tráfico, terremotos, enfermedades, problemas sociales, etc.

28.8 *EDUCACIÓN*

Uso de Internet de las Cosas en la educación: IoT podría mejorar el seguimiento de los alumnos para ayudarles a mejorar en ciertos aspectos, como en la escritura, sus emociones en el aula, cómo pueden comportarse en un examen o mejorar en determinadas tareas físicas.

«Mientras me quede algo por hacer, no habré hecho nada»
Julio César

Bloque VII
Anexos

Contenidos de anexo

Anexo I: SIGLAS Y ACRÓNIMOS

«No basta con adquirir sabiduría, es preciso además saber usarla»
Cicerón

«Bip... Bip... Bip...»
Sputnik I

Este anexo contiene todas las siglas y acrónimos utilizados a lo largo de la tesis, junto con su significado original, normalmente en inglés, y la traducción al español, la cual a veces ha sido necesaria traducir sin tener una referencia, pero la cual se ha intentado traducir respetando su significado original. Así, el método utilizado para este anexo es el siguiente:

Sigla o acrónimo *Nombre original en inglés* – Traducción al español

En el caso de que la sigla o acrónimo fuese original del idioma español se utiliza el formato:

Sigla o acrónimo Nombre en español

En el caso de que la sigla o acrónimo sea de lengua inglesa y se mantenga sin traducir se utiliza el formato:

Sigla o acrónimo *Nombre original en inglés*

***aaS**	** as a Service* - * como un Servicio
3DES	Triple DES
AaaS	*Analytics as a Service* – Análisis como un Servicio
ABE	Attribute-Based Encryption – Criptografía Basada en Atributos
ACID	*Atomicity, Consistency, Isolation and Durability* - Atomicidad, Consistencia, Aislamiento y Durabilidad
ACM	*Association for Computing Machinery*
ADM	*Architecture Driven Modernisation*
AEC	Antes de la Era Común
AES	*Advanced Encryption Standard*
AGV	A*utomated Guided Vehicles* - Vehículos Automáticos Guiados
AI	*Artificial Intelligence* – Inteligencia Artificial
AJAX	*Asynchronous JavaScript And XML* - JavaScript asíncrono y XML

Anexo I: siglas y acrónimos

AMPQ	*Advanced Message Queuing Protocol*
ANN	*Artificial Neural Network* - Redes Neuronales Artificiales
API	*Application Programming Interface*
APT	*Automatically Programmed Tools*
ASIMO	*Advanced Step in Innovative Mobility*
ASM	*Assembly / Assembler* - Ensamblador
AUV	*Autonomous Underwater Vehicle* - Vehículo Autónomo Submarino
AWS	*Amazon Web Services*
BaaS	*Backend as a Service* - Backend como un Servicio
BBDD	Bases de Datos
BDaaS	*Big Data as a Service – Big Data* como un Servicio
BES	*Bureau of Environmental Statistics* - Agencia de Estadísticas del Medio Ambiente
BNF	*Backus-Naur Formalism*
BPMN	*Business Process Management and Notation*
CaaS	*Communications as a Service* – Comunicación como un Servicio
CaaS	*Conversation as a Service* – Conversación como un Servicio
CAP	*Consistency, Availability, Partition* - Consistencia, Disponibilidad y tolerancia al Particionado
CASE	*Computer Aided Software Engineering*
CBC	*Cipher Block Chaining*
CCSA	*Cloud Computing and Scientific Applications*
CDC	*Disease Control and Prevention* - Centro de Control y Enfermedades de los Estados Unidos
CEN	*European Committee for Standardization* - Comité Europeo de Normalización
CEO	*Chief Executive Officer* - Director Ejecutivo
CERN	*Conseil Européen pour la Recherche Nucléaire* - Consejo Europeo para la Investigación Nuclear
CIL	*Common Intermediate Language*
CIM	*Computational-Independent Model*
CLI	*Common Language Infraestructure*

Anexo I: siglas y acrónimos

CLOS	*Common Lisp Object System*
CLR	*Common Language Runtime*
CO	*Carbon monoxide* - Monóxido de Carbono
CO_2	*Carbon monoxide 2* - Dióxido de Carbono
COAP	*Constrained Application Protocol*
CORBA	*Common Object Request Broker Architecture®*
CPU	*Central Processing Unit* - Unidad Central de Proceso
CSS	*Cascading Style Sheets*
CSV	*Comma-Separated Values*
CTO	*Chief Technology Officer* – Director de Tecnología
cURL	*Client for URLs* || *Client URL Request Library* || *see URL*
CWM	*Common Warehouse Metamodel™*
DaaS / DBaaS	*Data as a Service* - Datos como un Servicio
DARPA	*Defense Advanced Research Projects Agency* - Agencia de Proyectos de Investigación Avanzados de Defensa de los Estados Unidos de América
DC	*Direct Current* - Corriente Continua
DES	*Data Encryption Standard* – Estándar de Encriptación de Datos
DFS	*Distributed File System* - Sistema de Ficheros Distribuidos
DOE	*Deparment of Energy* - Departamento de Energía
DOF	*Degrees of Freedom* - Grado de Libertad
DOM	*Document Object Model*
DSL	*Domain-Specific Language* – Lenguaje de Dominio Específico
DTLS	*Datagram Transport Layer Security*
EaaS	*Everything as a Service* – Todo como un Servicio
EBNF	*Extended Backus-Naur Formalism*
EC2	*Elastic Computer Cloud*
ECC	*Elliptic curve cryptography* - Criptografía de Curva Elíptica
EDSAC	*Electronic Delay Storage Automatic Calculator*
EMF	*Eclipse Modeling Project*

Anexo I: siglas y acrónimos

EMR	*Elastic MapReduce*
ETA	*Estimated Time of Arrival* – Hora Estimada de Llegada
ETL	*Extract, Transform, Load* - Extracción, Tranformación y Carga
ETSI	*European Telecommunications Standards Institute* - Instituto Europeo de Normas de Telecomunicaciones
FaaS	File as a Service - Fichero como un Servicio
GFT	Google Flu Trends
GIS	*Geographic Information System* – Sistemas de Información geográfica
GMF	*Graphical Modeling Framework*
GNUPG	*GNU Privacy Guard*
GPGPU	*General-Purpose Computing on Graphics Processing Units*
GPL	*General Purpose Language* - Lenguajes de Propósito General
GPS	*Global Positioning System* - Sistema de Posicionamiento Global
GPU	*Graphics Processing Unit* – Unidad de procesamiento gráfico
GRE	*Generic Routing Encapsulation*
H2H	*Human-to-Human* – Humano a Humano
H2M	*Human-to-Machine* – Humano a Máquina
HaaS	*Human as a Service* – Humano como un Servicio
HACE	*Heterogeneous, Autonomous, Complex, Evolving* - Heterogeneidad, Autonomía, Complejidad, Evolución
HDD	*Hard Disk Drives* – Unidad de Disco Duro
HDFS	*Hadoop Distributed File System* - Sistema de Ficheros Distribuidos de Hadoop
HOG	*Histogram of Oriented Gradients*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IA	*Artificial Intelligence* - Inteligencia Artificial
IaaS	*Infrastructure as a Service* - Infraestructura como un Servicio
IaaS	*Integration as a Service* - Integración como un Servicio
IARC	*International Aerial Robotics Competition* - Competición Internacional de Robots Aéreos

Anexo I: siglas y acrónimos

IBM	*International Business Machines Corp.*
ICT	*Information and Communications Technology*
IDC	*International Data Corporation*
IDE	*Integrated Development Environment* - Entorno de Desarrollo Integrado
IDEA	*International Data Encryption Algorithm*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engineering Task Force*
IHH	*IoT-powered in-Home Healthcare*
IIoT	*Industrial Internet of Things* – Internet de las Cosas Industrial
IMDB	*Internet Movie Database*
INCITS	*International Committee for Information Technology Standards*
IoT	*Internet of Things* – Internet de las Cosas
IP	*Internet Protocol* – Protocolo de Internet
IPv6	*Internet Protocol version 6* – Protocolo de Internet versión 6
ISM	*Implementation-Specific Model*
ISO	*International Organization for Standardization* **-** Organización Internacional de Normalización
ISP	*Internet service provider* - Proveedor de Servicios de Internet
JAR	*Java ARchive*
JCR	*Journal Citation Reports*
JDBC	*Java DataBase Connectivity*
JEE	*Java Enterprise Edition*
JSD	*Jackson Systems Development*
JSON	*JavaScript Object Notation*
JSR	*Java Specification Requests*
JVM	*Java Virtual Machine* – Máquina Virtual Java
KBSA	*Knowledge-Based Software Assistant*
KDD	*Knowledge Discovery in Databases* – Descubrimiento de conocimiento en bases de datos

Anexo I: siglas y acrónimos

LAN	*Local Area Network* - Red de Área Local
LBP	*Local Binary Patterns*
LED	*Light-Emitting Diode* – Diodo Emisor de Luz
LINQ	*Language Integrated Query*
LS3	*Legged Squad Support System*
M.E.R.O.DE.	*Model-driven Entity-Relationship Object oriented DEvelopment*
M2M	*Machine-to-Machine* – Máquina a Máquina
MaaS	*Monitoring as a Service* – Monitorización como un Servicio
MB	*Model-Based* - Basado en Modelos
MBE	*Model-Based Engineering* – Ingeniería Basada en Modelos
MBSE	*Model-Based Software/System Engineering* – Ingeniería Software/Sistemas Basada en Modelos
MBSS	*Model-Based Software Synthesis*
MBT	*Model-Based Techniques* – Técnicas Basadas en Modelos
MCSF	*Mobile Client Software Factory*
MD	*Model-Driven* - Dirigido por Modelos
MD5	*Message-Digest Algorithm*
MDA	*Model-Driven Architecture*® - Arquitectura Dirigida por Modelos
MDD	*Model-Driven Development* - Diseño Dirigido por Modelos
MDE	*Model-Driven Engineering*- Ingeniería Dirigido por Modelos
MDOOD	*Model-driven object-oriented development*
MDSD	*Model-Driven Software Development* - Diseño Software Dirigido por Modelos
MGA	*Multigraph Architecture*
MIME	*Multipurpose Internet Mail Extensions* - Extensiones Multipropósito de Correo de Internet
MIPS	*Microprocessor without Interlocked Pipeline Stages*

MIPS	*Model-Integrated Program Synthesis* - Síntesis de Programas Integrados en el Modelo
MIT	*Massachusetts Institute of Technology*
MMS	*Multimedia Messaging Service* – Servicio de Mensajería Multimedia
MOCSL	*Midgar Object Creation Specific Language*
MOF	*Meta-Object Facility*™
MOISL	*Midgar Object Interconnection Specific Language*
MPENK	*Multigraph Programming Environment*
MQTT	*Message Queue Telemetry Transport*
MUCSL	*Midgar Use Case Specification Language*
NASA	*National Aeronautics and Space Administration*
NES	*National Election Study* (de los Estados Unidos de América)
NFC	*Near Field Communication*
NFS	*Network File System* - Sistema de Archivos de Red
NIH	*National Institutes of Health* - Institutos Nacionales de la Salud
NIST	*National Institute of Standards and Technology*
NLP	*Natural Language Processing* – Procesamiento de Lenguaje Natural
NOAA	*US National Oceanic and Atmospheric Administration*
NoSQL	*Not Only SQL*
NSA	*National Security Agency* – Agencia de Seguridad Nacional de los Estados Unidos de América
NSF	*National Science Foundation* - Fundación Nacional de Ciencia
NUMA	*Non-uniform memory access* - Acceso a Memoria No Uniforme
OASIS	*Organization for the Advancement of Structured Information Standards*
OCR	*Optical Character Recognition* – Reconocimiento Óptico de Caracteres

Anexo I: siglas y acrónimos

ODBC	*Open DataBase Connectivity*
ODM	*Ontology Definition Metamodel*
OGC®	*Open Geospatial Consortium*
OGF	*Open Grid Forum*
OLAP	*OnLine Analytical Processing* - Procesamiento Analítico Online
OMG	*Object Management Group®*
ONU	Organización de las Naciones Unidas
OOP	*Object-Oriented Programming* - Programación Orientado a Objetos
OSGi	*Open Services Gateway initiative*
OSN	*Online Social Networks* – Red Social Online
OTAN	Organización del Tratado del Atlántico Norte
OTP	*Open Telecom Platform*
OWL	*Ontology Web Language*
P&G	Procter & Gamble
P2P	*Peer-to-Peer*
PaaS	*Platform as a Service* – Plataforma como un Servicio
PACELC	*Partition-Availability-Consistency-Else-Latency-Consistency* – Partición-Disponibilidad-Consistencia-Sino-Latencia-Consistencia
PGP	*Pretty Good Privacy*
PHP	*PHP Hypertext Preprocessor*
PII	*Personally Identifiable Information* - Información de Identificación Personal
PIM	*Platform-Independent Model* – Modelo Independiente de la Plataforma
PSM	*Platform-Specific Model*
Python LR	*The Python Language Reference*
QoS	*Quality of Service* – Calidad de Servicio
QR Code	*Quick Response Code* – Código QR
QRIO	*Quest for curRIOsity*
QVT	*Query, View & Transformation*

Anexo I: siglas y acrónimos

RACE	*Requirements Analysis and Class Diagram Extraction*
RAE	Real Academia Española
RAID	*Redundant Array of Inexpensive Disks* - Conjunto Redundante de Discos Independientes
RAM	*Random-access memory* - Memoria de Acceso Aleatorio
RC4/RC5	*Rivest Cipher 4 / 5*
RCP	*Rich Client Platform*
RDBMS	Relational Database Management System – Sistema de gestión de bases de datos relacionales
RDF	*Resource Description Framework*
REST	*Representational State Transfer*
RFID	*Radio Frecuency IDentification*
RGB	*Red, Green, Blue* – Rojo, Verde, Azul
RGBD	*Red, Green, Blue, Depth* – Rojo, Verde, Azul, Profundidad
RGB-NIR	*Red, Green, Blue - Near InfraRed* – Rojo, Verde, Azul – Infrarrojo Cercano
RIC	Rango Intercuartílico
ROV	R*emotely Operated Vehicles* - Vehículos Manejados por Control Remoto
RPC	*Remote Procedure Call*
RSA	*Rivest, Shamir, Adleman*
RTES	*Real-Time Embedded Systems* - **S**istemas Embebidos en Tiempo Real
S+S	*Software plus Services*
S3	*Simple Storage Service*
SaaS	*Software as a Service* - Software como un Servicio
SAC	*Social Access Controller*
SCSF	*Smart Client Software Factory*
SD	*Secure Digital* – Seguro Digital
SDK	*Software Development Kit* - Kit de Desarrollo de Software
SDN	*Software Defined Networking* - Redes Definidas por Software
SDSS	*Sloan Digital Sky Survey*
SGML	*Standard Generalized Markup Language*

SHA *Secure Hash Algorithm*

SIFT *Scale-Invariant Feature Transform*

SIoT *Social Internet of Things* - Internet de las Cosas Social

SKA *Square Kilometer Array*

SLA *Service Level Agreement* - Acuerdo de Nivel de Servicio

SNS *Social Network Site* – Sitio de Red Social

SNS *Social Networking Site* – Sitio de Red Social

SOA *Service Oriented Architecture* - Arquitectura Orientada a Servicios

SOAP *Simple Object Access Protocol*

SPARQL *SPARQL Protocol and RDF Query Language*

SQL *Structured Query Language*

SQS *Simple Queue Service*

SSD *Solid-State Drive* – Unidad de estado sólido

SSH *Secure Shell*

SURF *Speeded-Up Robust Features*

SVM *Support Vector Machine* – Máquinas de Vectores de Soporte

SVN *Support Vector Networks* – Redes de Soporte de Vectores

TaaS *Tools as a Service* – Herramientas como un Servicio

TCP *Transmission Control Protocol* - Protocolo de Control de Transmisión

TFM Trabajo Final de Máster

TOPIO *TOSY Ping Pong Playing Robot*

TSVM *Transductive Support Vector Machine*

UAS Unmanned Aircraft System - Sistema de Avión No Tripulado

Anexo I: siglas y acrónimos

UAV	*Unmanned Aerial Vehicle* - Vehículo Aéreo No Tripulado
UDP	*User Datagram Protocol*
UGV	*Unmanned Ground Vehicle* - Vehículo Terrestre No Tripulado
UML	*Unified Modeling Language*® - Lenguaje Unificado de Modelado
UNESCO	*United Nations Educational, Scientific and Cultural Organization* - Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*
VHDL	*Very High Speed Integrated Circuit Hardware Description Language*
VoIP	*Voice over IP* - Voz por IP
VPN	*Virtual Private Network*
VPS	*Virtual Private Server* - Servidor Privado Virtual
W3C	*World Wide Web Consortium*
WAR	*Web Application Archi*ve
WCSF	*Web Client Software Factory*
WoT	*Web of Things* - Web de las Cosas
WSAN	*Wireless Sensor and Actor/Actuator Networks* - Redes de Sensores y Actores/Actuadores
WSN	*Wireless sensor network* – Redes de Sensores
WSSF	*Web Service Software Factory*
WWW	*World Wide Web*
XaaS	*X-as-a-Service*
XMI	*XML Metadata Interchange*®
XML	e*Xtensible Markup Language*
XMPP	*eXtensible Messaging and Presence Protocol*
XOF	*eXtendable-Output Function* – Función de Salida Extensible
XP	*eXtreme Programming*
XSLT	*eXtensible Stylesheet Language Transformations*
YARN	*Yet-Another-Resource-Negotiator*

Anexo II: CONVERSIÓN DE TÉRMINOS INGLÉS-ESPAÑOL

«Mira hacia atrás y ríete de los peligros pasados»
Walter Scott

«Limpia, fija y da esplendor»
Lema de la Real Academia Española

En algunas ocasiones se dio el caso de la no existencia de una traducción directa del término en inglés al idioma de esta tesis doctoral, el español, ya fuera debido a su ausencia de uso en esta especialidad, ciencias de la computación, o a no existir en la RAE. Por esta misma razón, algunas veces ha sido necesario realizar una traducción del término intentado respetar el significado original y la traducción en base a otros términos similares.

En la Tabla 31 se muestran los términos en inglés que no cuentan con una traducción directa junto al término español utilizado, siendo este a veces un anglicismo.

Término en inglés	Término en español
Back-end	*Back-end*
Batch	*Batch*
Big Data	Big Data
Biosequence	Biosecuencia
Bytecode	*Bytecode*
Checkpoint	*Checkpoint*
Chunk	Trozo
Chunkserver	*Chunkserver*
Data pattern processing	Procesamiento de patrones de datos
Discretization	Discretización
Fork	*Fork*
Front-end	*Front-end* Capa de presentación
Fuzzification	*Fuzzificar*
GFS cell - clúster GFS	Clúster GFS

Anexo II: conversión de términos inglés-español

Término en inglés	Término en español
Identation	*Identación*
Information discovery	Descubrimiento de información
Information harvesting	Recolección de información
Interoperability	Interoperabilidad
Inverted Index	*Inverted Index*
Knowledge extraction	Extracción de conocimiento
Livability	Habitabilidad
Log	Log Registro [219]
Man machine interfaces	Interfaces humano-máquina
Map	*Map*
Matching	*Matching*
Mirror	Espejo [220]
Modularity	*Modularidad*
Multitenancy	Multitenencia
Ortogonality	Ortogonalidad
Palletise / Palletize	Poner un palé / pallet
Palletising / Palletizing	*Paletización*
Parallelization	Paralelización
Parser	*Parseador* *Parsear*
Raw data	Datos en bruto
Reduce	*Reduce*
Reusability	*Reusabilidad*
Scalability	*Escalabilidad*

[219] No confundir este tipo de registro, del inglés *Log*, con los registros de las CPUs o el verbo registrar
[220] No confundir con el objeto espejo del español, pues esto es un servidor «idéntico»

Anexo II: conversión de términos inglés-español

Término en inglés	Término en español
Script	*Script*
Serialisation	Serialización
Shadow	Sombra
Shadow master	Maestro «shadow»
Shuffle	*Shuffle*
Sort	Ordenar
Stakeholder	Interesados
Streaming	Streaming
Support Vector Machines	Máquinas de vectores de soporte
Survey	*Survey* [221]
Template	*Template*
Trigger	*Disparador*
Worker	*Worker*

Tabla 31 Traducción de términos inglés-español

[221] No confundir con las «encuestas». Un *survey* es también un tipo de artículo en el cual se realiza un recoplatorio del estado actual de un tema determinado, siendo así un artículo especializado en ese tema.

Anexo III: GLOSARIO

«Tengo que dejar de hablar. Ya te he contado más de lo que sé»
Wolf Logan

«El mayor enemigo del conocimiento no es la ignorancia, sino la ilusión de conocimiento»
Stephen Hawking

Alice y Bob: Alice y Bob son dos nombres que se suelen utilizar en criptología como nombres de sustitución para las personas que intervienen en una conexión, habiendo sido utilizados por primera vez en el artículo de presentación de RSA [376], [388]. Otros nombres utilizados comúnmente junto a su uso son los presentados en el libro de Bruce Schneier [816], a saber:

- Carol, a veces Charlie, como participante en protocolos de 3 usuarios, donde están también Alice y Bob.
- Dave como participante en protocolos de 4 usuarios, sumándose así a Alice, Bob y Carol.
- Eve como espía.
- Mallory como un atacante malicioso.
- Peggy como testeadora.
- Trent como árbitro de confianza.
- Walter como vigilante de Alice y Bob en algunos protocolos.
- Victor como verificador.

Otros nombres y sus usos se pueden ver en la tabla recopilatoria de la Wikipedia [222].

Biosecuencia: palabra compuesta por el prefijo «bio-» y la palabra «secuencia». Es secuencia de datos biológicos.

- «Bio-» [154]:
    - elems. compos. Significa «vida» u «organismo vivo». Biografía, biología. Microbio, aerobia.
    - elems. compos. Significa «biológico, que implica respeto al medio ambiente». Biocombustible, bioagricultura.

[222] https://en.wikipedia.org/wiki/Alice_and_Bob

- «secuencia» [154]:
    - Continuidad, sucesión ordenada.
    - Serie o sucesión de cosas que guardan entre sí cierta relación.

Cinemática: rama de la ingeniería mecánica según [731], rama de la física según [154], que estudia el movimiento sin tener en cuenta las fuerzas que lo causan, estudiando así la trayectoria en función del tiempo.

Complejo de Frankenstein: en las novelas de Isaac Asimov es el miedo de la humanidad a que las máquinas se rebelen contra sus creadores, aludiendo así a la historia titulada *Frankenstein* de la escritora Mary Wollstonecraft Shelley. Sin embargo, según las tres leyes de la robótica de Asimov, el caso de que un robot se revelase contra la humanidad debería de ser imposible de darse.

Cono: los conos son las células foto-receptoras del color que se encuentran en la retina del ojo humano. Estas pueden ser de tres tipos, L (*long*), M (*medium*) y S (*short*), permitiendo distinguir por las longitudes de ondas a las que son sensibles: L-650 nanómetros (luz roja), M-530 nanómetros (luz verde) y S-430 nanómetros (luz azul). La densidad de los tres tipos de conos no es uniforme en el ojo humano, siendo esta relación de: L-11,2, M-5,33 y S-1.

Demonio: siendo este su nombre en sistemas operativos UNIX y en inglés *daemon* (Disk And Execution MONitor), conocido como servicio en sistemas operativos Windows o programa residente en MS-DOS, es un proceso informático que se ejecuta en segunda plano sin que sea controlado por el usuario. Un demonio permite ejecutar tareas de manera continua sin necesidad de la interacción con este tipo de programa [223].

Edge Computing: consiste en trasladar la computacción de uno nodo o más nodos centralizados al extremo de la red, haciendo que esta se encuentra en los dipositivos de los extremos o borde de la red, que son aquellos también que generan la información, siendo estos los manejados por los seres humanos como pueden ser los smartphones, los ordenadores o las tablets [817].

Electromiograma: examen que permite registrar la actividad eléctrica producida por los músculos esqueléticos del cuerpo humano para obtener su estado fisiológico y saber que nervios los activan [224].

Electrooculograma: examen es utilizado en las pruebas del estudio del sueño y que consiste en colocar electrodos cerca de los músculo de los ojos para medir su movimiento [818].

Endpoint: punto de acceso a una aplicación, servicio, proceso o lista en una arquitectura SOA.

[223] https://en.wikipedia.org/wiki/Daemon_(computing)
[224] http://www.bioingenieria.edu.ar/academica/catedras/bioingenieria2/archivos/apuntes/tema%205%20-%20electromiografia.pdf

Anexo III: glosario

Framework: es una infraestructura digital de conceptos, prácticas, tecnologías y criterios que ayudan a poner la base para la organización y desarrollo de software de un tipo de problemática particular y que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. Normalmente, los *frameworks* suelen incluir soporte de programas, bibliotecas y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Gestión del conocimiento: engloba el proceso de capturar datos, desarrollo de competencias, intercambio y uso eficaz de datos para obtener conocimiento en una organización, tratando así de cumplir los objetivos de la organización mediante un mejor uso del conocimiento [819]. Este conocimiento puede estar en bases de datos, documentos, procedimientos, políticas, conocimiento previamente capturado y experiencia de determinados trabajadores [820].

GPGPU: en informática, es el concepto de intentar aprovechar el procesamiento de las GPUs, pues estas están diseñadas para realizar cómputos relacionados con la generación de gráficos 3D en vez de procesamiento general como ocurre con las CPUs [225].

Grados de libertad: los grados de libertad, en inglés *Degrees of Freedom* (DOFs), son el número de articulaciones que un brazo robótico tiene y que permiten juntar sus partes sólidas para así darle movimiento, teniendo típicamente 8 grados de libertad [729].

Inteligencia de enjambre (Swarm-Intelligence): rama de la Inteligencia Artificial que estudia el comportamiento colectivo de sistemas descentralizados, autoorganizados, naturales o artificiales como son los Robots móviles, en donde fue donde se introdujo este concepto por Gerardo Beni y Wang Jin en 1989 [821].

Interoperabilidad: capacidad de comunicar, ejecutar programas o transferir datos a través de diferentes unidades funcionales bajo ciertas especificaciones [368].

Inverted Index: es un índice de datos que almacena un mapeo de contenido, que pueden ser palabras o números, y su posición en la base de datos. Los *Inverted Index* se utilizan para permitir búsquedas más rápidas por texto a costa de incrementar el procesamiento cuando se añade algo a la base de datos.

Man-in-the-Middle: este ataque sirve para vulnerar la seguridad y se da cuando el atacante se encuentra en medio de una conexión entre los dos extremos, permitiéndole así interceptar o alterar las comunicaciones entre las dos partes sin que estas sean conscientes de este ataque [226] [822].

Mecatrónica: es una disciplina que une la ingeniería mecánica, ingeniería electrónica, ingeniería de control e ingeniería informática. La mecatrónica investiga como facilitar el trabajo del ser humano mediante el uso de sistemas de control electrónicos en la industria mecánica. Los sistemas que convergen en esta interdisciplina son los sistemas mecánicos, los sistemas electrónicos y los sistemas programables y de control [823].

[225] http://gpgpu.org/about
[226] https://www.icann.org/news/blog/que-es-un-ataque-de-intermediarios

Anexo III: glosario

Metadatos: datos acerca de los datos [598], [824].

Middleware: la palabra *middleware* surgió en la conferencia de 1968 de la OTAN [22], poniéndolo como ejemplo de ser una parte software que intercambia datos entre diferentes aplicaciones y permitiendo así que diferentes aplicaciones heterogéneas se conecten al *middleware* para obtener o enviar datos por medio de una API.

Órtesis: apoyo o dispositivo externo que permite modificar los aspectos funcionales o estructurales del sistema neuromusculoesquelético. Esto permite ayudar a gente que tiene problemas del aparato locomotor, permitiendo así reemplazar o reforzar las funciones del órgano o miembro que falta [227]. Se diferencian de la prótesis en que esta última sustituye parcial o totalmente, en vez de reemplazarlo.

Paradigma: teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento [154].

RAID: es un sistema de almacenamiento de datos en tiempo real que utiliza diferentes unidades de disco duro o SSD para distribuir y/o hacer copias de los datos, proporcionando, dependiendo de su configuración, mayor integridad, tolerancia a fallos, rendimiento o capacidad. Existen diferentes tipos de RAID como son el 0 para distribuir los datos, el 1 para duplicarlos, el 5 para dividirlos por bloques, y el 10 que es la combinación del 0 y el 1 [825], entre muchos otros tipos.

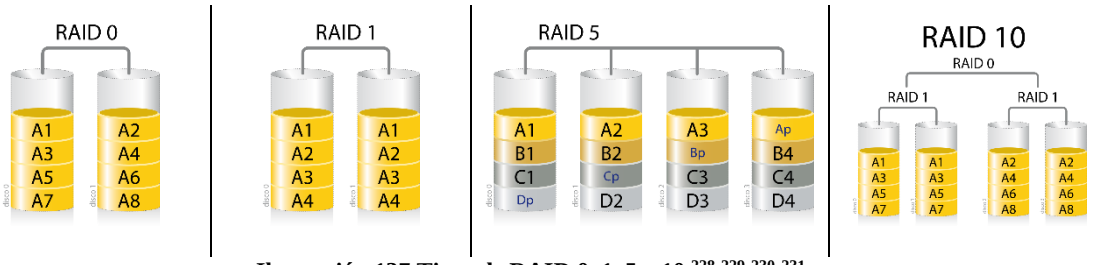


Ilustración 137 Tipos de RAID 0, 1, 5 y 10 [228] [229] [230] [231]

Sistema de ficheros distribuidos: en un sistema de almacenamiento de ficheros distribuidos los datos están distribuidos en diferentes nodos de computación del clúster de ordenadores. Los datos pueden estar también distribuidos a nivel de fichero o de bloque, permitiendo a múltiples nodos interaccionar con diferentes partes de ficheros de gran tamaño simultáneamente [598].

Utility Computing o **The Computing Utility**: es el suministro de recursos computacionales, como pueden ser el almacenamiento o el procesamiento, como un servicio medido similar a los servicios públicos tradicionales, como son el teléfono o la electricidad. Se podría decir que es utilizar los ordenadores bajo el uso de «paga por lo que usas» («pay-per-use»). Esto es lo que se da en los servicios ofrecidos por La Nube (Cloud Computing).

[227] http://www.iso.org/iso/catalogue_detail.htm?csnumber=15800
[228] https://commons.wikimedia.org/wiki/File:Raid0.png
[229] https://commons.wikimedia.org/wiki/File:Raid1.png
[230] https://commons.wikimedia.org/wiki/File:Raid5.png
[231] https://commons.wikimedia.org/wiki/File:Raid10.png

Anexo IV: REFERENCIAS

«He oído que no hay nada tan agradable para un autor
como encontrar sus obras respetuosamente citadas por otros autores doctos»
Benjamin Franklin

[1] E. W. Dijkstra and C. S. Scholten, *Predicate Calculus and Program Semantics*. New York, NY: Springer New York, 1990.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] C. Hao, X. Lei, and Z. Yan, "The application and Implementation research of Smart City in China," in *System Science and Engineering (ICSSE), 2012 International Conference on*, 2012, no. 70172014, pp. 288–292.

[4] K. Ashton, "That 'Internet of Things' thing," *RFiD J.*, vol. 22, no. 7, pp. 97–114, 2009.

[5] H. Gu and D. Wang, "A Content-aware Fridge Based on RFID in Smart Home for Home-Healthcare," in *Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on*, 2009, vol. 2, pp. 987–990.

[6] C. Han, J. M. Jornet, E. Fadel, and I. F. Akyildiz, "A cross-layer communication module for the Internet of Things," *Comput. Networks*, vol. 57, no. 3, pp. 622–633, Feb. 2013.

[7] K. A. Hribernik, Z. Ghrairi, C. Hans, and K. Thoben, "Co-creating the Internet of Things - First Experiences in the Participatory Design of Intelligent Products with Arduino," in *Concurrent Enterprising (ICE), 2011 17th International Conference on*, 2011, pp. 1–9.

[8] E. Mykletun, J. Girao, and D. Westhoff, "Public Key Based Cryptoschemes for Data Concealment in Wireless Sensor Networks," in *2006 IEEE International Conference on Communications*, 2006, vol. 5, no. c, pp. 2288–2295.

[9] G. M. Lee and J. Y. Kim, "Ubiquitous networking application: Energy saving using smart objects in a home," in *2012 International Conference on ICT Convergence (ICTC)*, 2012, pp. 299–300.

[10] L. Tan, "Future internet: The Internet of Things," in *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, 2010, pp. V5-376-V5-380.

[11] A. Broring, P. Maue, C. Malewski, and K. Janowicz, "Semantic mediation on the Sensor Web," 2012, pp. 2910–2913.

[12] The US National Intelligence Council, "Six Technologies with Potential Impacts on US Interests out to 2025," 2008.

[13] UK Intellectual Property Office Informatics Team, "Eight Great Technologies - The Internet of Things: A Patent Overview," Newport, 2014.

[14] UK Government Chief Scientific Adviser, “The Internet of Things: making the most of the Second Digital Revolution,” 2014.

[15] COMMISSION OF THE EUROPEAN COMMUNITIES, “Future networks and the internet Early Challenges regarding the ‘Internet of Things,’” 2008.

[16] European Committee for Standardization, “European Committee for Standardization,” 2016. [Online]. Available: http://www.cen.eu. [Accessed: 12-Feb-2016].

[17] International Organization for Standardization, “International Organization for Standardization,” 2016. [Online]. Available: http://www.iso.org/iso/home.htm. [Accessed: 12-Feb-2016].

[18] European Telecommunications Standards Institute, “European Telecommunications Standards Institute,” 2016. [Online]. Available: http://www.etsi.org. [Accessed: 12-Feb-2016].

[19] L. Da Xu, W. He, and S. Li, “Internet of Things in Industries: A Survey,” *IEEE Trans. Ind. Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[20] Ministry of Internal Affairs and Communications, “u-Japan,” 2007. [Online]. Available: http://www.soumu.go.jp/menu_seisaku/ict/u-japan_en/. [Accessed: 12-Feb-2016].

[21] E. W. Dijkstra, “The humble programmer,” *Commun. ACM*, vol. 15, no. October 1972, pp. 859–866, 1972.

[22] P. Naur and B. Randell, “Software Engineering,” Garmisch, Germany, 1968.

[23] V. García-Díaz, H. Fernández-Fernández, E. Palacios-González, B. C. P. G-Bustelo, O. Sanjuán-Martínez, and J. M. C. Lovelle, “TALISMAN MDE: Mixing MDE principles,” *J. Syst. Softw.*, vol. 83, no. 7, pp. 1179–1191, Jul. 2010.

[24] B. Selic, “MDA manifestations,” *Eur. J. Informatics Prof.*, vol. IX, no. 2, pp. 12–16, 2008.

[25] K. Gama, L. Touseau, and D. Donsez, “Combining heterogeneous service technologies for building an Internet of Things middleware,” *Comput. Commun.*, vol. 35, no. 4, pp. 405–417, Feb. 2012.

[26] G. G. Meyer, K. Främling, and J. Holmström, “Intelligent Products: A survey,” *Comput. Ind.*, vol. 60, no. 3, pp. 137–148, Apr. 2009.

[27] D. Guinard and V. Trifa, “Towards the web of things: Web mashups for embedded devices,” in *2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web*, 2009.

[28] Object Management Group Inc., “MDA Guide Version 1.0.1,” 2003.

[29] S. Kent, “Model Driven Engineering,” *Comput. Comput. Soc.*, vol. 2335, no. 2, pp. 286–298, 2002.

[30] A. Van Deursen, P. Klint, and J. Visser, “Domain-specific languages: an annotated bibliography,” *ACM Sigplan Not.*, vol. 35, no. 6, pp. 26–36, 2000.

[31] C. González García, “MIDGAR: Plataforma para la generación dinámica de aplicaciones distribuidas basadas en la integración de redes de sensores y dispositivos electrónicos IoT,” University of Oviedo, 2013.

[32] H. Kniberg, *Scrum and XP from the trenches*. 2007.

[33] K. Schwaber and J. Sutherland, *The Definitive Guide to Scrum: The Rules of the Game*. 2013.

[34] K. Beck, "Embracing change with extreme programming," *Computer (Long. Beach. Calif).*, vol. 32, no. 10, pp. 70–77, 1999.

[35] K. Beck, *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[36] D. L. O'Leary, *How Greek Science Passed to the Arabs*. Routledge, 2001.

[37] G. Heitink, *Practical theology: History, theory, action domains: Manual for practical theology*. Wm. B. Eerdmans Publishing Co., 1999.

[38] P. Godfrey-Smith, *An introduction to the philosophy of science: Theory and reality*, 1st ed. Chicago: University of Chicago Press, 2003.

[39] C. González García and J. P. Espada, "Using Model-Driven Architecture Principles to Generate Applications based on Interconnecting Smart Objects and Sensors," in *Advances and Applications in Model-Driven Engineering*, V. G. Díaz, J. M. C. Lovelle, B. C. P. García-Bustelo, and O. S. Martinez, Eds. IGI Global, 2014, pp. 73–87.

[40] C. González García, C. P. García-Bustelo, J. P. Espada, and G. Cueva-Fernandez, "Midgar: Generation of heterogeneous objects interconnecting applications. A Domain Specific Language proposal for Internet of Things scenarios," *Comput. Networks*, vol. 64, no. C, pp. 143–158, Feb. 2014.

[41] G. Cueva-Fernandez, J. P. Espada, V. García-Díaz, C. González García, and N. Garcia-Fernandez, "Vitruvius: An expert system for vehicle sensor tracking and managing application generation," *J. Netw. Comput. Appl.*, vol. 42, pp. 178–188, Jun. 2014.

[42] C. González García, J. P. Espada, E. R. N. Valdez, and V. García-Díaz, "Midgar: Domain-Specific Language to Generate Smart Objects for an Internet of Things Platform," in *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2014, pp. 352–357.

[43] C. González García and J. P. Espada, "MUSPEL: Generation of Applications to Interconnect Heterogeneous Objects Using Model-Driven Engineering," in *Handbook of Research on Innovations in Systems and Software Engineering*, V. G. Díaz, J. M. C. Lovelle, and B. C. P. García-Bustelo, Eds. IGI Global, 2015, pp. 365–385.

[44] C. González García, J. P. Espada, B. C. P. G-Bustelo, and J. M. Cueva Lovelle, "Swift vs. Objective-C: A New Programming Language," *Int. J. Interact. Multimed. Artif. Intell.*, vol. 3, no. 3, pp. 74–81, 2015.

[45] C. González García, J. P. Espada, B. C. P. G-Bustelo, and J. M. C. Lovelle, "El futuro de Apple: Swift versus Objective-C," *Redes Ing.*, vol. 6, no. 2, pp. 6–16, 2015.

[46] D. Meana-Llorián, C. González García, B. C. Pelayo G-Bustelo, J. M. Cueva Lovelle, and V. H. Medina García, "IntelliSenses: Sintiendo Internet de las Cosas," in *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*, 2016, pp. 234–239.

[47] D. Meana-Llorián, C. González García, V. García-Díaz, B. C. P. G-Bustelo, and J. M. C. Lovelle, "SenseQ: Creating relationships between objects to answer questions of humans by using Social Networks," in *Proceedings of the The 3rd Multidisciplinary International Social Networks Conference on SocialInformatics 2016, Data Science 2016 - MISNC, SI, DS 2016*, 2016, pp. 1–5.

[48] C. González García, D. Meana-Llorián, B. C. P. G-Bustelo, and J. M. C. Lovelle, "A review about Smart Objects, Sensors, and Actuators," *Int. J. Interact. Multimed. Artif. Intell.*, vol. 4, no. 3, pp. 7–10, 2017.

[49] D. Meana-Llorián, C. González García, B. C. P. G-Bustelo, J. M. Cueva Lovelle, and N. Garcia-Fernandez, "IoFClime: The fuzzy logic and the Internet of Things to control indoor temperature regarding the outdoor ambient conditions," *Futur. Gener. Comput. Syst.*, 2016.

[50] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, "Smart objects as building blocks for the Internet of things," *IEEE Internet Comput.*, vol. 14, no. 1, pp. 44–51, Jan. 2010.

[51] L. Atzori, A. Iera, G. Morabito, and M. Nitti, "The Social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization," *Comput. Networks*, vol. 56, no. 16, pp. 3594–3608, Nov. 2012.

[52] R. Girau, M. Nitti, and L. Atzori, "Implementation of an Experimental Platform for the Social Internet of Things," in *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2013, pp. 500–505.

[53] V. Georgitzikis, O. Akribopoulos, and I. Chatzigiannakis, "Controlling Physical Objects via the Internet using the Arduino Platform over 802.15.4 Networks," in *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 2012, vol. 10, no. 3, pp. 1686–1689.

[54] A. Piras, D. Carboni, and A. Pintus, "A Platform to Collect, Manage and Share Heterogeneous Sensor Data," in *Networked Sensing Systems (INSS)*, 2012, pp. 1–2.

[55] Arduino, "Arduino," 2016. [Online]. Available: https://www.arduino.cc/. [Accessed: 09-Feb-2016].

[56] V. Vujović and M. Maksimović, "Raspberry Pi as a Sensor Web node for home automation," *Comput. Electr. Eng.*, vol. 44, pp. 153–171, 2015.

[57] S. Luo, H. Xia, Y. Gao, J. S. Jin, and R. Athauda, "Smart Fridges with Multimedia Capability for Better Nutrition and Health," in *2008 International Symposium on Ubiquitous Multimedia Computing*, 2008, pp. 39–44.

[58] Fundación Telefónica, "La Sociedad de la Información en España 2014," 2015.

[59] T. Yamanoue, K. Oda, and K. Shimozono, "A M2M System Using Arduino, Android and Wiki Software," in *2012 IIAI International Conference on Advanced Applied Informatics*, 2012, pp. 123–128.

[60] I. F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–114, Aug. 2002.

[61] L. Wang, L. Da Xu, Z. Bi, and Y. Xu, "Data Cleaning for RFID and WSN Integration," *IEEE Trans.*

Ind. Informatics, vol. 10, no. 1, pp. 408–418, Feb. 2013.

[62] D. Uckelmann, M. Harrison, and F. Michahelles, *An Architectural Approach Towards the Future Internet of Things*, no. JANUARY. Springer Berlin Heidelberg, 2011.

[63] C. Perera, C. H. Liu, and S. Jayawardena, "The Emerging Internet of Things Marketplace From an Industrial Perspective: A SuPerera, C., Liu, C.H., Jayawardena, S., 2015. The Emerging Internet of Things Marketplace From an Industrial Perspective: A Survey. IEEE Trans. Emerg. Top. Comput. PP, 13. doi," *IEEE Trans. Emerg. Top. Comput.*, vol. PP, no. 99, p. 13, 2015.

[64] C. Perera, C. H. Liu, S. Jayawardena, and M. Chen, "A Survey on Internet of Things From Industrial Market Perspective," *IEEE Access*, vol. 2, pp. 1660–1679, 2014.

[65] Z. Yan, P. Zhang, and A. V. Vasilakos, "A survey on trust management for Internet of Things," *J. Netw. Comput. Appl.*, vol. 42, pp. 120–134, 2014.

[66] B. M. Howe, Y. Chao, P. Arabshahi, S. Roy, T. McGinnis, and A. Gray, "A Smart Sensor Web for Ocean Observation: Fixed and Mobile Platforms, Integrated Acoustics, Satellites and Predictive Modeling," *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 3, no. 4, pp. 507–521, Dec. 2010.

[67] N. Bulusu, D. Estrin, and L. Girod, "Scalable coordination for wireless sensor networks: self-configuring localization systems," in *Proc. of the 6th International Symposium on Communication Theory and Applications(ISCT A '01), Ambleside, UK*, 2001, no. July, pp. 1–6.

[68] DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION, "TRANSMISSION CONTROL PROTOCOL," 1981.

[69] DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION, "INTERNET PROTOCOL," 1981.

[70] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. S. Wolff, "The past and future history of the Internet," *Commun. ACM*, vol. 40, no. 2, pp. 102–108, Feb. 1997.

[71] E. Savitz, "Gartner: Top 10 Strategic Technology Trends for 2013," *Forbes*, 2013. [Online]. Available: http://www.forbes.com/sites/ericsavitz/2012/10/23/gartner-top-10-strategic-technology-trends-for-2013/#30193fa91d44. [Accessed: 22-Jun-2016].

[72] A. Stanford-Clark and A. Nipper, "MQTT." [Online]. Available: http://mqtt.org/. [Accessed: 09-Feb-2016].

[73] Intel Corporation, "Intel Edison," 2015. [Online]. Available: http://www.intel.com/content/www/us/en/do-it-yourself/edison.html. [Accessed: 09-Feb-2016].

[74] International Telecommunication Union, "Overview of the Internet of things," Geneva, p. 14, 2012.

[75] M. Hasan, E. Hossain, and D. Niyato, "Random access for machine-to-machine communication in LTE-advanced networks: issues and approaches," *IEEE Commun. Mag.*, vol. 51, no. 6, pp. 86–93, Jun. 2013.

[76] E. Borgia, “The Internet of Things vision: Key features, applications and open issues,” *Comput. Commun.*, vol. 54, pp. 1–31, 2014.

[77] LogMeIn, “Xively,” 2013. [Online]. Available: https://xively.com/. [Accessed: 29-Jul-2015].

[78] Carriots, “Carriots,” 2011. [Online]. Available: https://www.carriots.com/. [Accessed: 29-Jul-2015].

[79] Exosite, “Exosite,” 2013. [Online]. Available: http://exosite.com/. [Accessed: 29-Jul-2015].

[80] LORD MicroStrain, “Sensor Cloud.” [Online]. Available: http://www.sensorcloud.com/. [Accessed: 29-Jul-2015].

[81] Etherios, “Etherios,” 2008. [Online]. Available: http://www.etherios.com/. [Accessed: 29-Jul-2015].

[82] ThingWorx™, “ThingWorx,” 2015. [Online]. Available: http://www.thingworx.com/. [Accessed: 29-Jul-2015].

[83] Microsoft, “Azure IoT Suit,” 2015. [Online]. Available: http://www.microsoft.com/en-us/server-cloud/internet-of-things/azure-iot-suite.aspx. [Accessed: 30-Nov-2015].

[84] Amazon, “AWS IoT.” [Online]. Available: https://aws.amazon.com/es/iot/how-it-works/. [Accessed: 27-Nov-2015].

[85] IBM, “IBM Internet of Things,” 2015. [Online]. Available: http://www.ibm.com/analytics/us/en/internet-of-things/. [Accessed: 30-Nov-2015].

[86] Paraimpu SRL, “Paraimpu,” 2012. [Online]. Available: http://paraimpu.crs4.it/. [Accessed: 29-Jul-2015].

[87] Department of Electrical and Electronic Engineering (University of Cagliari), “SIoT,” *http://platform.social-iot.org/*, 2012. [Online]. Available: http://platform.social-iot.org/.

[88] A. Kansal, S. Nath, J. Liu, and W. I. Grosky, “SenseWeb : An Infrastructure for Shared Sensing,” *IEEE Multimed.*, vol. 14, no. 4, pp. 8–13, 2007.

[89] Microsoft, “SenseWeb,” *http://research.microsoft.com/en-us/projects/senseweb/*, 2008. [Online]. Available: http://research.microsoft.com/en-us/projects/senseweb/.

[90] “QuadraSpace,” 2010. [Online]. Available: http://www.quadraspace.org/. [Accessed: 29-Jul-2015].

[91] Sen.se, “Open.Sen.se,” 2015. [Online]. Available: http://open.sen.se/. [Accessed: 29-Jul-2015].

[92] Oak Ridge National Laboratory, “Sensorpedia,” 2009. [Online]. Available: http://www.sensorpedia.com/. [Accessed: 29-Jul-2015].

[93] B. L. Gorman, D. R. Resseguie, and C. Tomkins-Tinch, “Sensorpedia: Information sharing across incompatible sensor systems,” *2009 Int. Symp. Collab. Technol. Syst.*, pp. 448–454, 2009.

[94] EVRYTHNG, “EVRYTHNG,” 2012. [Online]. Available: https://www.evrythng.com/. [Accessed: 29-Jul-2015].

[95] IoBridge, “Thingspeak,” 2013. [Online]. Available: http://www.thingspeak.com. [Accessed: 29-Jul-2015].

[96] Nimbits Inc., “Nimbits,” 2015. [Online]. Available: http://www.nimbits.com. [Accessed: 29-Jul-2015].

[97] CyberVision Inc., “KAA,” 2014. [Online]. Available: http://www.kaaproject.org. [Accessed: 29-Jul-2015].

[98] A. Pintus, D. Carboni, and A. Piras, “The anatomy of a large scale social web for internet enabled objects,” *Proc. Second Int. Work. Web Things - WoT ’11*, p. 1, 2011.

[99] A. Piras, D. Carboni, and A. Pintus, “A platform to collect, manage and share heterogeneous sensor data,” in *2012 Ninth International Conference on Networked Sensing (INSS)*, 2012, pp. 1–2.

[100] A. Pintus, D. Carboni, and A. Piras, “Paraimpu: a platform for a social web of things,” in *International World Wide Web Conference Com- mittee (IW3C2)*, 2012, pp. 401–404.

[101] L. Atzori, A. Iera, and G. Morabito, “SIoT: Giving a Social Structure to the Internet of Things,” *IEEE Commun. Lett.*, vol. 15, no. 11, pp. 1193–1195, Nov. 2011.

[102] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext transfer protocol--HTTP/1.1,” 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2616.txt. [Accessed: 21-Dec-2015].

[103] P. Saint-Andre, “Extensible messaging and presence protocol (XMPP): Core,” 2011. [Online]. Available: http://tools.ietf.org/html/rfc6120. [Accessed: 21-Dec-2015].

[104] A. P. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, “Web Services for the Internet of Things through CoAP and EXI,” in *2011 IEEE International Conference on Communications Workshops (ICC)*, 2011, no. Xml, pp. 1–6.

[105] Z. Shelby, K. Hartke, and C. Bormann, “The constrained application protocol (CoAP),” 2014. [Online]. Available: https://tools.ietf.org/html/rfc7252. [Accessed: 21-Dec-2015].

[106] U.S. Department of Transportation, “Livability,” 2015. [Online]. Available: http://www.transportation.gov/livability/101. [Accessed: 11-Feb-2016].

[107] D. Ding, R. A. Cooper, P. F. Pasquina, and L. Fici-Pasquina, “Sensor technology for smart homes,” *Maturitas*, vol. 69, no. 2, pp. 131–6, Jun. 2011.

[108] P. Lopez, D. Fernandez, A. J. Jara, and A. F. Skarmeta, “Survey of internet of things technologies for clinical environments,” in *Proceedings - 27th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2013*, 2013, pp. 1349–1354.

[109] M. Rothensee, “A high-fidelity simulation of the smart fridge enabling product-based services,” in *3rd IET International Conference on Intelligent Environments (IE 07)*, 2007, vol. 2007, pp. 529–532.

[110] T. Gentry, “Smart homes for people with neurological disability: state of the art,” *NeuroRehabilitation*, vol. 25, no. 3, pp. 209–217, 2009.

[111] M. Chan, E. Campo, D. Estève, and J.-Y. Fourniols, “Smart homes — Current features and future perspectives,” *Maturitas*, vol. 64, no. 2, pp. 90–97, Oct. 2009.

[112] K. Henning, W. Wolfgang, and H. Johannes, “Recommendations for implementing the strategic initiative INDUSTRIE 4.0,” 2013.

[113] E. W. T. Ngai, K. K. L. Moon, F. J. Riggins, and C. Y. Yi, “RFID research: An academic literature review (1995–2005) and future research directions,” *Int. J. Prod. Econ.*, vol. 112, no. 2, pp. 510–520, Apr. 2008.

[114] B. Buxton, V. Hayward, I. Pearson, L. Kärkkäinen, H. Greiner, E. Dyson, J. Ito, A. Chung, K. Kelly, and S. Schillace, “Big data: The next Google,” *Nature*, vol. 455, no. 7209, pp. 8–9, Sep. 2008.

[115] C. Sun, “Application of RFID Technology for Logistics on Internet of Things,” *AASRI Procedia*, vol. 1, pp. 106–111, 2012.

[116] M. C. Domingo, “An overview of the Internet of Things for people with disabilities,” *J. Netw. Comput. Appl.*, vol. 35, no. 2, pp. 584–596, Mar. 2012.

[117] World Health Organization and The World Bank, “World Report on Disability 2011,” Malta, 2011.

[118] Z. Pang and J. Tian, “Ecosystem Analysis in the Design of Open Platform based In-Home Healthcare Terminals towards the Internet-of-Things,” in *15th International Conference on Advanced Communication Technology*, 2013, pp. 529–534.

[119] H. Alemdar and C. Ersoy, “Wireless sensor networks for healthcare: A survey,” *Comput. Networks*, vol. 54, no. 15, pp. 2688–2710, Oct. 2010.

[120] W. Qiuping, Z. Shunbing, and D. Chunquan, “Study On Key Technologies Of Internet Of Things Perceiving Mine,” *Procedia Eng.*, vol. 26, pp. 2326–2333, 2011.

[121] FedEx, “SenseAware: Be in Control,” 2012. [Online]. Available: http://www.senseaware.com/wp-content/uploads/SA-General_Sales_Slick.pdf. [Accessed: 15-Feb-2016].

[122] Cantaloupe Systems, “Seed Cloud,” 2016. [Online]. Available: http://www.cantaloupesys.com/. [Accessed: 15-Feb-2016].

[123] H. Zhou, B. Liu, and D. Wang, “Design and Research of Urban Intelligent Transportation System Based on the Internet of Things,” in *Communications in Computer and Information Science*, vol. 312, no. 70031010, Y. Wang and X. Zhang, Eds. Springer Berlin Heidelberg, 2012, pp. 572–580.

[124] E. Qin, Y. Long, C. Zhang, and L. Huang, “Cloud Computing and the Internet of Things: Technology Innovation in Automobile Service,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8017 LNCS, no. PART 2, 2013, pp. 173–180.

[125] Y. Zhang, B. Chen, and X. Lu, “Intelligent Monitoring System on Refrigerator Trucks Based on the Internet of Things,” in *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*, IEEE, 2012, pp. 201–206.

[126] Newcastle University, Newcastle Council, and Siemens, “Compass4D,” 2015. [Online]. Available: http://www.compass4d.eu/en/cities_02/newcastle/newcastle.htm. [Accessed: 13-Feb-2016].

[127] Department for Business Innovation & Skills, Department for Transport, Technology Strategy Board, The Rt Hon, V. Cable, The Rt Hon Greg Clark MP, and Claire Perry MP, "UK Government fast tracks driverless cars," 2015. [Online]. Available: https://www.gov.uk/government/news/uk-government-fast-tracks-driverless-cars. [Accessed: 13-Feb-2016].

[128] Y. Zhang and J. Yu, "A Study on the Fire IOT Development Strategy," *Procedia Eng.*, vol. 52, pp. 314–319, 2013.

[129] Zhang Ji and Qi Anwen, "The application of internet of things(IOT) in emergency management system in China," in *2010 IEEE International Conference on Technologies for Homeland Security (HST)*, 2010, pp. 139–142.

[130] Amazon Robotics, "Amazon Robotics," 2015. [Online]. Available: https://www.amazonrobotics.com/. [Accessed: 14-Feb-2016].

[131] H. Song, "Internet of things for rural and small town america," in *6th Annual Create West Virginia Training and Education Conference*, 2013, pp. 1–6.

[132] UNESCO, "Protecting Our Heritage and Fostering Creativity," 2015.

[133] N. D. Milder and A. Dane, "Some Thoughts on the Economic Revitalization of Small Town Downtowns," *The Economic Development Journal*, 2013. [Online]. Available: http://www.ecdevjournal.com/en/News/index.aspx?newsId=acaba33d-0e15-4994-b899-eacd1c6d8d21#! [Accessed: 15-Dec-2015].

[134] World Commission on Environment and Development, "Report of the World Commission on Environment and Development: Our Common Future," 1987.

[135] United Nations, "World Urbanization Prospects 2014," New York, May 2014.

[136] A. J. Jara, Y. Sun, H. Song, R. Bie, D. Genooud, and Y. Bocchi, "Internet of Things for Cultural Heritage of Smart Cities and Smart Regions," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, 2015, pp. 668–675.

[137] Y. Sun, Y. Xia, H. Song, and R. Bie, "Internet of Things Services for Small Towns," in *2014 International Conference on Identification, Information and Knowledge in the Internet of Things*, 2014, pp. 92–95.

[138] R. Jesner Clarke, "Smart Cities and the Internet of Everything: The Foundation for Delivering Next-Generation Citizen Services," no. October, Alexandria, pp. 1–18, 2013.

[139] S. Mitchell, N. Villa, M. Stewart-Weeks, and A. Lange, "The Internet of Everything for Cities: Connecting people, Process, Data, and Things to Improve the 'Livability' of Cities and Communities," San Jose, CA, pp. 1–21, 2013.

[140] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, Feb. 2014.

[141] Vienna University of Technology, "European Smart Cities," *http://www.smart-cities.eu*, 2015. [Online]. Available: http://www.smart-cities.eu. [Accessed: 28-Sep-2016].

[142] A. Caragliu, C. Del Bo, and P. Nijkamp, "Smart Cities in Europe," *J. Urban Technol.*, vol. 18, no. 2, pp. 65–82, Apr. 2011.

[143] A. Caragliu, C. del Bo, and P. Nijkamp, "Smart cities in Europe," in *3rd Central European Conference in Regional Science – CERS*, 2009, pp. 45–59.

[144] Shi Lu, "The Smart City's systematic application and implementation in China," in *2011 International Conference on Business Management and Electronic Information*, 2011, vol. 3, pp. 116–120.

[145] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, and D. Pfisterer, "SmartSantander: IoT experimentation over a smart city testbed," *Comput. Networks*, vol. 61, pp. 217–238, Mar. 2014.

[146] SmartSantander, "SmartSantander." [Online]. Available: http://www.smartsantander.eu/. [Accessed: 17-Dec-2015].

[147] SmartAmsterdam, "Smart Amsterdam." [Online]. Available: http://amsterdamsmartcity.com/. [Accessed: 17-Dec-2015].

[148] A. Cenedese, A. Zanella, L. Vangelista, and M. Zorzi, "Padova Smart City: An urban Internet of Things experimentation," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014, pp. 1–6.

[149] P. Casari, A. P. Castellani, A. Cenedese, C. Lora, M. Rossi, L. Schenato, and M. Zorzi, "The 'Wireless Sensor Networks for City-Wide Ambient Intelligence (WISE-WAI)' Project," *Sensors*, vol. 9, no. 6, pp. 4056–4082, May 2009.

[150] M. C. Falvo, R. Lamedica, and A. Ruvio, "An environmental sustainable transport system: A trolley-buses Line for Cosenza city," in *International Symposium on Power Electronics Power Electronics, Electrical Drives, Automation and Motion*, 2012, pp. 1479–1485.

[151] A. Gore, "The Digital Earth: Understanding our planet in the 21st Century," *Aust. Surv.*, vol. 43, no. 2, pp. 89–91, 1998.

[152] S. Wang, "Spatial data mining under Smart Earth," in *2011 IEEE International Conference on Granular Computing*, 2011, pp. 717–722.

[153] K. Aberer, "Smart Earth: From Pervasive Observation to Trusted Information," in *2007 International Conference on Mobile Data Management*, 2007, pp. 3–7.

[154] Real Academia Española, "Real Academia Española," 2016. [Online]. Available: http://dle.rae.es/. [Accessed: 17-Feb-2016].

[155] WordReference.com LLC, "WordReference," 2016. [Online]. Available: http://www.wordreference.com/. [Accessed: 17-Feb-2016].

[156] Alphabet Inc., "Google," 2016. [Online]. Available: http://www.google.es. [Accessed: 17-Feb-2016].

[157] Oxford University Press, "Oxford Dictionaries," 2016. [Online]. Available:

http://www.oxforddictionaries.com/. [Accessed: 18-Feb-2016].

[158] Cambridge University Press, "Cambridge Dictionaries Online," 2016. [Online]. Available: http://dictionary.cambridge.org/. [Accessed: 18-Feb-2016].

[159] D. McFarlane, S. Sarma, J. L. Chirn, C. Y. Wong, and K. Ashton, "Auto ID systems and intelligent manufacturing control," *Eng. Appl. Artif. Intell.*, vol. 16, no. 4, pp. 365–376, Jun. 2003.

[160] C. Y. Wong, D. McFarlane, A. Ahmad Zaharudin, and V. Agarwal, "The intelligent product driven supply chain," in *IEEE International Conference on Systems, Man and Cybernetics*, 2002, vol. vol.4, p. 6.

[161] R. Van Kranenburg, D. Caprio, E. Anzelmo, A. Bassi, S. Dodson, and M. Ratto, "The Internet of Things," in *1st Berlin Symposium on Internet and Society*, 2011, no. October 2015, p. 84.

[162] M. Kärkkäinen, J. Holmström, K. Främling, and K. Artto, "Intelligent products—a step towards a more effective project delivery chain," *Comput. Ind.*, vol. 50, pp. 141–151, 2003.

[163] O. Ventä, *Intelligent products and systems: Technology theme-final report*, no. 635. VTT Technical Research Centre of Finland, 2007.

[164] F. Ramparany and O. Boissier, "Smart devices embedding multi-agent technologies for a pro-active world," in *Proc. Uniquitous Computing Workshop*, 2002.

[165] B. Selic, "The pragmatics of model-driven development," *IEEE Softw.*, vol. 20, no. 5, pp. 19–25, Sep. 2003.

[166] B. Selic, "Models, Software Models and UML," in *UML for Real*, Boston: Kluwer Academic Publishers, 2003, pp. 1–16.

[167] E. Seidewitz, "What models mean," *IEEE Softw.*, vol. 20, no. 5, pp. 26–32, Sep. 2003.

[168] M. H. Morgan and H. L. Warren, *Vitruvius: The Ten Books on Architecture*, 2nd, 2014 ed. HARVARD UNIVERSITY PRESS, 1926.

[169] R. King, *Brunelleschi's Dome: How a Renaissance Ggenius Reinvented Architecture*, 1st ed. Bloomsbury Publishing USA, 2013.

[170] B. Selic, "Model-driven development of real-time software using OMG standards," in *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2003.*, 2003, pp. 4–6.

[171] D. Thomas, "MDA: Revenge of the modelers or UML Utopia?," *IEEE Softw.*, vol. 21, no. 3, pp. 15–17, 2004.

[172] B. Selic, "From Model-Driven Development to Model-Driven Engineering (Keynote Talks)," in *Fifth International Conference on Creating, Connecting and Collaborating through Computing (C5 '07)*, 2007, p. 3.

[173] G. Poels and G. Dedene, "Complexity metrics for formally specified business requirements," in *Proceedings of the Annual Oregon Workshop on Software Metrics*, 1997, pp. 1–11.

Anexo IV: referencias

[174] Architecture Board ORMSC, “Model Driven Architecture (MDA),” 2001.

[175] M. Fowler, “ModelDrivenSoftwareDevelopment,” *14 July 2008*, 2008. [Online]. Available: http://martinfowler.com/bliki/ModelDrivenSoftwareDevelopment.html. [Accessed: 29-Feb-2016].

[176] B. Hailpern and P. Tarr, “Model-driven development: The good, the bad, and the ugly,” *IBM Syst. J.*, vol. 45, no. 3, pp. 451–461, 2006.

[177] S. Sendall and W. Kozaczynski, “Model transformation: the heart and soul of model-driven software development,” *IEEE Softw.*, vol. 20, no. 5, pp. 42–45, Sep. 2003.

[178] J. Bézivin, A. Vallecillo-Moreno, J. García-Molina, and G. Rossi, “MDA® at the Age of Seven: Past, Present and Future,” *Eur. J. Informatics Prof.*, vol. IX, no. June, pp. 4–7, 2008.

[179] S. W. Liddle, D. W. Embley, and S. N. Woodfield, “Cardinality constraints in semantic data models,” *Data Knowl. Eng.*, vol. 11, no. 3, pp. 235–270, 1993.

[180] M. Fowler, “ModelDrivenArchitecture,” 2004. [Online]. Available: http://www.martinfowler.com/bliki/ModelDrivenArchitecture.html. [Accessed: 28-Mar-2016].

[181] Eclipse, “Ecore,” 2010. [Online]. Available: http://wiki.eclipse.org/Ecore. [Accessed: 28-Sep-2016].

[182] M. A. Jackson, “A System Development Method,” *Tools Notions Progr. Constr.*, p. 13, 1982.

[183] G. Dedene and M. Snoeck, “M.E.R.O.DE.: A Model-driven Entity-Relationship Object-oriented DEvelopment method,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 19, no. 3, 1994.

[184] C. Green, D. Luckham, R. Balzer, T. Cheatham, and C. Rich, “Report on a Knowledge-Based Software Assistant,” 1983.

[185] R. Balzer, “A 15 Year Perspective on Automatic Programming,” *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 11, pp. 1257–1268, 1985.

[186] J. A. Zachman, “A framework for information systems architecture,” *IBM Syst. J.*, vol. 26, no. 3, pp. 276–292, 1987.

[187] G. Dedene, “Model-driven object-oriented development by examples (MERO DE.),” in *Object technology’93 conference*, 1993.

[188] M. Snoeck and G. Dedene, “Experiences with object oriented model-driven development,” in *Proceedings Eighth IEEE International Workshop on Software Technology and Engineering Practice incorporating Computer Aided Software Engineering*, 1997, pp. 143–153.

[189] G. Karsai, “Declarative-programming techniques for engineering problems,” Vanderbilt Univ., Nashville, TN (USA), 1988.

[190] B. Abbott, T. Bapty, C. Biegl, A. Ledeczi, G. Karsai, and J. Sztipanovits, “Experiences using model-based techniques for the development of a large parallel instrumentation system,” in *Proc. Conf. Signal Processing Applications and Technology, Cambridge, Mass, pp573-582*, 1992.

[191] D. Harel, “Biting the silver bullet: toward a brighter future for system development,” *Computer (Long. Beach. Calif).*, vol. 25, no. 1, pp. 8–20, Jan. 1992.

[192] M. R. Lowry, “Software Engineering in the Twenty-First Century,” *AI Mag.*, vol. 13, no. 3, pp. 71–87, 1992.

[193] M. R. Lowry, “Methodologies for Knowledge-Based Software Engineering,” 1993.

[194] D. W. Embley, B. D. Kurtz, and S. N. Woodfield, *Object-oriented systems analysis: a model-driven approach*. 1992.

[195] G. Dedene, M. Snoeck, and A. Depuydt, “On Generalisation / Specialisation Hierarchies in M.E.R.O.DE. Object-Oriented Business Modeling,” *Res. Cent. Manag. Informatics (LIRIS), Leuven*, no. 9219, pp. 1–8, 1992.

[196] B. Abbott, T. Bapty, C. Biegl, G. Karsai, and J. Sztipanovits, “Model-based software synthesis,” *IEEE Softw.*, vol. 10, no. 3, pp. 42–52, May 1993.

[197] J. Sztipanovits, G. Karsai, C. Biegl, T. Bapty, A. Ledeczi, and A. Misra, “MULTIGRAPH: an architecture for model-integrated computing,” in *Proceedings of First IEEE International Conference on Engineering of Complex Computer Systems. ICECCS’95*, 1995, pp. 361–368.

[198] G. Dedene and M. Snoeck, “Object-oriented modelling: a new language for consistent business engineering,” 1993.

[199] V. García-Díaz, H. Fernández-Fernández, E. Palacios-González, B. C. P. G-Bustelo, and J. M. C. Lovelle, “Intelligent traceability system of Cabrales cheese using MDA TALISMAN,” in *Proceedings of the 2008 International Conference on Artificial Intelligence*, 2008, vol. 178, no. March 2016, pp. 578–584.

[200] M. Glykas, P. Wilhelmij, and T. Holden, “Object Orientation in Enterprise Modelling and Information System Design,” in *Object Oriented Development, IEE Colloquium on*, 1993, p. 19.

[201] W. Paper, “Guarantee permanent Model/Code consistency: ‘Model driven Engineering’ (MDE) vs ‘Roundtrip engineering’ (RTE),” 2000.

[202] P. Desfray, “MDA – When a major software industry trend meets our toolset, implemented since 1994,” 2001.

[203] J. Greenfield and K. Short, “Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools,” in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications - OOPSLA ’03*, 2004, p. 16.

[204] M. Regio, J. Greenfield, and B. Thuman, “A Software Factory Approach To HL7 Version 3 Solutions,” 2005. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms954602.aspx?f=255&MSPPError=-2147217396. [Accessed: 30-Mar-2016].

[205] B. C. P. G-Bustelo, “TALISMAN: Desarrollo ágil de Software,” University of Oviedo, 2007.

[206] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, “Manifesto for Agile Software Development,” 2001.

[207] Frederick P. Brooks, “No Silver Bullet Essence and Accidents of Software Engineering,” *Computer (Long. Beach. Calif).*, vol. 20, no. 4, pp. 10–19, 1987.

[208] E. Palacios-González, H. Fernández-Fernández, V. García-Díaz, B. C. P. G-Bustelo, J. M. C. Lovelle, and O. S. Martínez, “General purpose MDE tools,” *Int. J. Interact. Multimed. Artif. Intell.*, vol. 1, no. 1, pp. 72–75, 2008.

[209] “Chaos Report,” 1995.

[210] T. A. Bapty and J. Sztipanovits, “Model-based engineering of large-scale real-time systems,” in *Proceedings International Conference and Workshop on Engineering of Computer-Based Systems*, 1997, pp. 467–474.

[211] G. Karsai and J. Sztipanovits, “A Visual Programming Environment for Domain Specific Model-Based Programming,” *IEEE Comput.*, pp. 36–44, 1995.

[212] J. F. F. Sowa and J. A. Zachman, “Extending and formalizing the framework for information systems architecture,” *IBM Syst. J.*, vol. 31, no. 3, pp. 590–616, 1992.

[213] G. Karsai, “Declarative Programming Using Visual Tools,” 1989.

[214] J. D. Ichbiah, B. Krieg-Brueckner, B. A. Wichmann, J. G. P. Barnes, O. Roubine, and J.-C. Heliard, “Rationale for the design of the Ada programming language,” *ACM SIGPLAN Not.*, vol. 14, no. 6b, pp. 1–261, Jun. 1979.

[215] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Prentice Hall, 2008.

[216] N. Wirth, *Modula-2*. Zürich, 1980.

[217] M. Snoeck, “On a Process Algebra Approach for the Construction and Analysis of M.E.R.O.DE.-Based Conceptual Models,” KU Leuven, 1995.

[218] KU Leuven, “MERODE.” [Online]. Available: http://merode.econ.kuleuven.be/publications.aspx.

[219] J. Sutherland, “Business objects in corporate information systems,” *ACM Comput. Surv.*, vol. 27, no. 2, pp. 274–276, Jun. 1995.

[220] V. García-Díaz, “MDCI: Model-Driven Continuous Integration,” University of Oviedo, 2011.

[221] Microsoft Patterns & Practice Group, “Smart Client Software Factory,” 2006. [Online]. Available: http://www.codeplex.com/smartclient/. [Accessed: 30-Mar-2016].

[222] Microsoft Patterns & Practice Group, “Mobile Client Software Factory,” 2006. [Online]. Available: http://mobile.codeplex.com/. [Accessed: 30-Mar-2016].

[223] Microsoft Patterns & Practice Group, “Web Client Software Factory,” 2006. [Online]. Available: http://webclientguidance.codeplex.com/. [Accessed: 30-Mar-2016].

[224] Microsoft Patterns & Practice Group, “Web Service Software Factory,” 2006. [Online]. Available: http://www.codeplex.com/servicefactory/. [Accessed: 30-Mar-2016].

[225] K. Newton, *The Definitive Guide to the Microsoft Enterprise Library*. Berkely, CA, USA: Apress,

2007.

[226] A. W. B. Furtado, A. L. M. Santos, and G. L. Ramalho, “Computer games software factory and edutainment platform for Microsoft .NET,” *IET Softw.*, vol. 1, no. 6, p. 280, 2007.

[227] V. García-Díaz, J. B. Tolosa, B. C. P. G-Bustelo, E. Palacios-González, O. Sanjuán-Martínez, and R. G. Crespo, “TALISMAN MDE Framework: An Architecture for Intelligent Model-Driven Engineering,” in *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, no. November, S. Omatu, M. P. Rocha, J. Bravo, F. Fernández, E. Corchado, A. Bustillo, and J. M. Corchado, Eds. Springer Berlin Heidelberg, 2009, pp. 299–306.

[228] V. García-Díaz, H. Fernández-Fernández, E. Palacios-González, B. C. P. G-Bustelo, O. Sanjuán-Martínez, and J. M. C. Lovelle, “Automated code generation support for BI with MDA TALISMAN,” *Int. J. Interact. Multimed. Artif. Intell.*, vol. 1, no. 2, pp. 87–93, 2009.

[229] Object Management Group Inc., “XML Metadata Interchange®,” 2016. [Online]. Available: http://www.omg.org/spec/XMI/. [Accessed: 26-Feb-2016].

[230] W. E. H. Chehade, A. Radermacher, F. Terrier, B. Selic, and S. Gérard, “A Model-Driven Framework for the Development of Portable Real-Time Embedded Systems,” in *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, 2011, pp. 45–54.

[231] V. García-Díaz, B. C. P. G-Bustelo, and J. M. Cueva Lovelle, “MDCI: Model-driven continuous integration,” *J. Ambient Intell. Smart Environ.*, vol. 4, no. 5, pp. 479–481, 2012.

[232] E. R. Núñez-Valdez, O. Sanjuán-Martínez, B. C. P. G-Bustelo, J. M. C. Lovelle, and G. Infante-Hernandez, “Gade4all: Developing Multi-platform Videogames based on Domain Specific Languages and Model Driven Engineering,” *Int. J. Interact. Multimed. Artif. Intell.*, vol. 2, no. Regular Issue, pp. 33–42, 2013.

[233] T. Stahl, M. Völter, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2013.

[234] Object Management Group Inc., “MDA Guide Rev. 2.0,” 2014.

[235] D. S. Frankel, *Model Driven Architecture Applying Mda*. John Wiley & Sons, 2003.

[236] A. Van Deursen, “Domain-Specific Languages versus Object-Oriented Frameworks: A financial engineering case study,” *Smalltalk Java Ind. Acad.*, pp. 35–39, 1997.

[237] S. Mary and G. David, *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall Englewood Cliffs, 1996.

[238] Object Management Group Inc., “Meta Object Facility™,” 2016. .

[239] Eclipse, “Eclipse Modeling Framework Project,” 2016. [Online]. Available: http://www.eclipse.org/modeling/emf/?project=emf. [Accessed: 26-Feb-2016].

[240] Object Management Group Inc., “Unified Modeling Language®,” *2016*, 2016. [Online]. Available: http://www.omg.org/spec/UML/. [Accessed: 25-Feb-2016].

Anexo IV: referencias

[241] Object Management Group Inc., “Common Object Request Broker Architecture®,” 2016. [Online]. Available: http://www.omg.org/spec/CORBA/. [Accessed: 26-Feb-2016].

[242] World Wide Web Consortium, “XSL Transformations (XSLT),” 2016. [Online]. Available: https://www.w3.org/standards/xml/transformation. [Accessed: 03-Mar-2016].

[243] C. Larman, *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Pearson Education India, 2005.

[244] “Ruby,” 2016. [Online]. Available: https://www.ruby-lang.org. [Accessed: 05-Apr-2016].

[245] D. Flanagan and Y. Matsumoto, *The Ruby Programming Language*, vol. 1. O’Reilly, 2008.

[246] A. P. J. Lauder, “A PRODUCTIVE RESPONSE TO LEGACY SYSTEM PETRIFICATION,” University of Kent at Canterbury, 2001.

[247] A. Watson, “A brief history of MDA,” *Eur. J. Informatics Prof.*, vol. IX, no. 2, pp. 7–11, 2008.

[248] IEEE Architecture Working Group, “1471-2000 - IEEE Recommended Practice for Architectural Description of Software-Intensive Systems,” *IEEE Comput. Soc.*, p. i--23, 2000.

[249] Object Management Group Inc., “Object Management Group®,” 2016. [Online]. Available: http://www.omg.org/. [Accessed: 28-Feb-2016].

[250] Object Management Group Inc., “MDA®,” 2016. [Online]. Available: http://www.omg.org/mda/. [Accessed: 26-Feb-2016].

[251] J. D. Poole, “Model-Driven Architecture: Vision, Standards And Emerging Technologies,” in *Workshop on Metamodeling and Adaptive Object Models, ECOOP01*, 2001, no. April, pp. 1–15.

[252] Object Management Group Inc., “Documents Associated With Meta Objet Facility (MOF) 2.0 Query/View/Transformation, V1.3,” 2016. [Online]. Available: http://www.omg.org/spec/QVT/1.3/. [Accessed: 28-Sep-2016].

[253] Microsoft, “Common Language Runtime (CLR),” *Microsoft MSDN*, 2016. [Online]. Available: https://msdn.microsoft.com/es-es/library/8bs2ecf4%2528v=vs.110%2529.aspx?f=255&MSPPError=-2147217396. [Accessed: 19-Nov-2016].

[254] D. S. Platt, *Introducing Microsoft .Net*, 2nd ed. Redmond, WA, USA: Microsoft Press, 2003.

[255] Object Management Group Inc., “Common Warehouse Metamodel™,” 2016. [Online]. Available: http://www.omg.org/spec/CWM/. [Accessed: 01-Jan-2016].

[256] Object Management Group Inc., “SysML,” 2016. [Online]. Available: http://www.omgsysml.org/. [Accessed: 28-Oct-2016].

[257] Object Management Group Inc., “Ontology Definition Metamodel™ (ODM™),” 2016. [Online]. Available: http://www.omg.org/spec/ODM/1.1/. [Accessed: 28-Oct-2016].

[258] E. International, “C# Language Specification,” 2006.

[259] Microsoft, "C#," 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/kx37x362.aspx. [Accessed: 04-Apr-2016].

[260] F# Software Foundation and F# Community, "F#," 2016. [Online]. Available: http://fsharp.org/. [Accessed: 26-Oct-2016].

[261] F# Software Foundation and F# Community, "The F # 4.0 Language Specification," 2016.

[262] F. Ortin, D. Zapico, J. B. G. Perez-Schofield, and M. Garcia, "Including both static and dynamic typing in the same programming language," *IET Softw.*, vol. 4, no. 4, p. 268, 2010.

[263] F. Ortin and M. García, "Union and intersection types to support both dynamic and static typing," *Inf. Process. Lett.*, vol. 111, no. 6, pp. 278–286, Feb. 2011.

[264] J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley, "The Java Language Specification: Java SE 8 Edition," 2015.

[265] M. Odersky, L. Spoon, and B. Venners, *Programming in Scala: A comprehensive step-by-step guide*, 2nd ed. 2008.

[266] M. Odersky, "The Scala Language Specification Version 2.9." p. 191, 2011.

[267] R. Cope and J. Strachan, "The Groovy Programming Language: Let's Get Groovy," in *JavaOne Sun's 2004 Worldwide Java Developer Conference*, 2004.

[268] Apache Groovy Project, "Groovy," 2016. [Online]. Available: http://www.groovy-lang.org/. [Accessed: 04-Nov-2016].

[269] IronPython Community, "IronPython." [Online]. Available: http://ironpython.net/. [Accessed: 26-Oct-2016].

[270] Python Software Foundation, "Python," 2016. [Online]. Available: https://www.python.org/. [Accessed: 28-Oct-2016].

[271] IBM Corporation, "Assembler language: HLASM Language Reference," 2014. [Online]. Available: http://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.asma400/toc.htm. [Accessed: 26-Oct-2016].

[272] The PHP Group, "PHP Hypertext Preprocessor," 2016. [Online]. Available: http://www.php.net/. [Accessed: 28-Oct-2016].

[273] B. Stroustrup, *The C ++ Programming*. Pearson Education India, 1986.

[274] B. P. Lientz and E. B. Swanson, "Problems in application software maintenance," *Commun. ACM*, vol. 24, no. 11, pp. 763–769, Nov. 1981.

[275] Object Management Group Inc., "Product Success," *2016*, 2016. [Online]. Available: http://www.omg.org/mda/products_success.htm. [Accessed: 24-Feb-2016].

[276] R. C. Gronback and E. Merks, "Model-Driven Software Development Model-Driven Architecture ® at Eclipse," *Eur. J. Informatics Prof.*, vol. IX, no. 2, pp. 35–39, 2008.

Anexo IV: referencias

[277] M. Fowler, "PlatformIndependentMalapropism," 2003. [Online]. Available: http://martinfowler.com/bliki/PlatformIndependentMalapropism.html. [Accessed: 28-Mar-2016].

[278] D. M. Ritchie, "The development of the C language," in *The second ACM SIGPLAN conference on History of programming languages - HOPL-II*, 1993, vol. 28, no. 3, pp. 201–208.

[279] D. Crockford, "JavaScript: The Good Parts," *J. Inf. Process. Manag.*, vol. 44, no. 8, p. 584, 2001.

[280] B. J. Cox, "The objective-C environment: past, present, and future," in *Digest of Papers. COMPCON Spring 88 Thirty-Third IEEE Computer Society International Conference*, 1988, pp. 166–169.

[281] R. Rawlings, "Objective-C: An Object-Oriented language for pragmatists," in *Applications of Object-Oriented Programming, IEE Colloquium on*, 1989, no. November, p. 2/1-2/3.

[282] Apple Inc., "Swift," *https://developer.apple.com/swift/*, 2015. [Online]. Available: https://developer.apple.com/swift/. [Accessed: 30-Nov-2015].

[283] Apple Inc., *The Swift Programming Language*, 16 April 2. Apple Inc., 2014.

[284] TIOBE Software BV, "TIOBE Index," *http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html*, 2014. [Online]. Available: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. [Accessed: 02-Feb-2016].

[285] C. Zapponi, "GitHut," 2014. [Online]. Available: http://githut.info/. [Accessed: 02-Feb-2016].

[286] J. W. Backus, "The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference," in *Proceedings of the International Comference on Information Processing*, 1959, pp. 125–131.

[287] J. W. Backus, J. H. Wegstein, A. van Wijngaarden, M. Woodger, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, and B. Vauquois, "Report on the algorithmic language ALGOL 60," *Commun. ACM*, vol. 3, no. 5, pp. 299–314, May 1960.

[288] A. van Wijngaarden, "Orthogonal Design and Description of a Formal Language." 1965.

[289] B. J. Mailloux, J. E. L. Peck, and C. H. A. Koster, *Report on the Algorithmic Language ALGOL 68*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1969.

[290] H. Bauer, S. Becker, S. L. Graham, E. Satterthwaite, and R. L. Sites, "Algol W: Language Description," 1972.

[291] J. E. Sammet, "The early history of COBOL," in *History of programming languages I*, New York, NY, USA: ACM, 1981, pp. 199–243.

[292] J. E. Sammet and J. Garfunkel, "Summary of Changes in COBOL, 1960-1985," *IEEE Ann. Hist. Comput.*, vol. 7, no. 4, pp. 342–347, Oct. 1985.

[293] Computer and Business Equipment Manufacturers Association, "American National Standard Programming Language FORTRAN," 1978.

[294] Applied Science Division and programming Research Dept., "Fortran - Programmer's Reference Manual." IBM, p. 54, 1956.

[295] N. Wirth, “The Programming Language Pascal,” *Acta Inform.*, vol. 1, no. 1, pp. 35–63, 1971.

[296] O.-J. Dahl and K. Nygaard, “SIMULA: an ALGOL-based simulation language,” *Commun. ACM*, vol. 9, no. 9, pp. 671–678, Sep. 1966.

[297] B. Meyer, “Eiffel: A language and environment for software engineering,” *J. Syst. Softw.*, vol. 8, no. 3, pp. 199–246, Jun. 1988.

[298] A. Goldberg and D. Robson, *Smalltalk-80: the language and its implementation*. Addison-Wesley, 1983.

[299] J. E. Labra Gayo, “Desarrollo modular de procesadores de lenguajes a partir de especificaciones semánticas reutilizables,” University of Oviedo, 2001.

[300] J. McCarthy, “Recursive functions symbolic expressions and their computation by machine, Part I,” *Commun. ACM*, vol. 3, no. 4, pp. 184–195, Apr. 1960.

[301] R. Hickey, “The Clojure Programming Language,” in *Proceedings of the 2008 Symposium on Dynamic Languages*, 2008, p. 1:1--1:1.

[302] J. Armstrong, “Getting Erlang to talk to the outside world,” in *Proceedings of the 2002 ACM SIGPLAN workshop on Erlang - ERLANG ’02*, 2002, pp. 64–72.

[303] J. Armstrong, “Making reliable distributed systems in the presence of software errors,” The Royal Institute of Technology, 2003.

[304] J. Armstrong, “The development of Erlang,” in *Proceedings of the second ACM SIGPLAN international conference on Functional programming - ICFP ’97*, 1997, vol. 16, no. November, pp. 196–203.

[305] S. P. Jones, J. Hughes, L. Augustsson, D. Barton, B. Boutel, W. Burton, J. Fasel, K. Hammond, R. Hinze, P. Hudak, P. Hudak, T. Johnsson, M. Jones, J. Launchbury, E. Meijer, J. Peterson, A. Reid, C. Runciman, and P. Wadler, *Haskell 98: A non-strict, purely functional language*. Cambridge University Press, 1999.

[306] haskell.org, “Haskell,” 2015. [Online]. Available: https://www.haskell.org/. [Accessed: 04-Apr-2016].

[307] D. H. D. Warren, L. M. Pereira, and F. Pereira, “Prolog - the language and its implementation compared with Lisp,” *ACM SIGART Bull.*, no. 64, pp. 109–115, Aug. 1977.

[308] L. Sterling and E. Shapiro, *The art of Prolog: advanced programming techniques*. MIT Press, 1994.

[309] H. W. Lie, “Cascading HTML style sheets -- a proposal,” 1994. .

[310] T. Berners-Lee and D. Connolly, “Hypertext Markup Language (HTML): A Representation of Textual Information and MetaInformation for Retrieval and Interchange,” 1993.

[311] E. F. Codd, “A Relational Model of Data for Large Shared Data Banks,” *Commun. ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[312] A. L. Ambler, M. M. Burnett, and B. A. Zimmerman, “Operational versus definitional: a perspective

on programming paradigms," *Computer (Long. Beach. Calif).*, vol. 25, no. 9, pp. 28–43, Sep. 1992.

[313] E. W. Dijkstra, "Notes on structured programming," in *Structured programming*, Academic Press Ltd. London, UK, 1972, pp. 1–82.

[314] T. Elrad, R. E. Filman, and A. Bader, "Aspect-oriented programming: Introduction," *Commun. ACM*, vol. 44, no. 10, pp. 29–32, Oct. 2001.

[315] D. Ungar and R. B. Smith, "Self: The power of simplicity," in *Conference proceedings on Object-oriented programming systems, languages and applications - OOPSLA '87*, 1987, vol. 12, no. 1987, pp. 227–242.

[316] L. C. Paulson, *ML for the Working Programmer*, 2nd ed. Cambridge University Press, 1996.

[317] G. J. Sussman and G. L. Steele, "SCHEME: An Interpreter for the Extended Lambda Calculus," 1975.

[318] R. Kowalski, "Algorithm = logic + control," *Commun. ACM*, vol. 22, no. 7, pp. 424–436, Jul. 1979.

[319] A. Bowers and P. M. Hill, "An Introduction to Gödel," in *Proceedings of the 4th UK Conference on Logic Programming*, London, United Kingdom, 1993, pp. 299–343.

[320] T. Conway, F. Henderson, and Z. Somogyi, "Code Generation for Mercury," in *In Proceedings of the Twelfth International Conference on Logic Programming*, 1995, pp. 242–256.

[321] Z. Somogyi, F. Henderson, T. Conway, A. Bromage, T. Dowd, D. Jeffery, P. Ross, P. Schachte, and S. Taylor, "Status of the Mercury system," in *In Proceedings of the JICSLP'96 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages*, 1996, pp. 207–218.

[322] Z. Somogyi, F. Henderson, and T. Conway, "The execution algorithm of mercury, an efficient purely declarative logic programming language," *J. Log. Program.*, vol. 29, no. 1–3, pp. 17–64, Oct. 1996.

[323] D. Spinellis, "Notable design patterns for domain-specific languages," *J. Syst. Softw.*, vol. 56, no. 1, pp. 91–99, 2001.

[324] N. Ford and S. Davis, *No Fluff, Just Stuff Anthology*. Pragmatic Bookshelf, 2006.

[325] A. van Deursen and P. Klint, "Little languages: Little maintenance?," *J. Softw. Maint.*, vol. 10, no. 2, pp. 75–92, Apr. 1998.

[326] D. T. Ross, *Origins of the APT language for automatically programmed tools*. New York, NY, USA, 1981.

[327] D. E. Knuth, "Backus Normal Form vs. Backus Naur Form," *Commun. ACM*, vol. 7, no. 12, pp. 735–736, 1964.

[328] Mundo Reader S.L., "Bitbloq," 2010. [Online]. Available: http://bitbloq1.bq.com/. [Accessed: 26-Jul-2016].

[329] Object Management Group Inc., "BPMN," 2016. [Online]. Available: http://www.bpmn.org/. [Accessed: 07-Apr-2016].

[330] International Organization for Standardization, “Information technology - Syntactic metalanguage - Extended BNF,” 1996.

[331] S. I. Feldman, “Make — a program for maintaining computer programs,” *Softw. Pract. Exp.*, vol. 9, no. 4, pp. 255–265, Apr. 1979.

[332] International Organization for Standardization, “ISO 8879:1986 - Information processing -- Text and office systems -- Standard Generalized Markup Language (SGML),” 1986. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=16387. [Accessed: 07-Nov-2016].

[333] IEEE, “IEEE Standard VHDL Language Reference Manual,” 2000.

[334] Antlr and T. Parr, “Antlr,” 2014. [Online]. Available: http://www.antlr.org/. [Accessed: 06-Apr-2016].

[335] T. Parr, *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*, 1st ed. Pragmatic Bookshelf, 2009.

[336] Free Software Foundation Inc., “Bison,” 2014. [Online]. Available: https://www.gnu.org/software/bison/. [Accessed: 06-Apr-2016].

[337] C. Donnelly and R. Stallman, *Bison: The Yacc-compatible Parser Generator*. 2015.

[338] S. C. Johnson, *Yacc: Yet Another Compiler-Compiler*. Bell Laboratories Murray Hill, NJ, 1978.

[339] J. Bentley, “Programming pearls: little languages,” *Commun. ACM*, vol. 29, no. 8, pp. 711–721, Aug. 1986.

[340] “MiniBloq.” [Online]. Available: http://blog.minibloq.org/. [Accessed: 26-Jul-2016].

[341] M. Fowler, “Language Workbenches: The Killer-App for Domain Specific Languages?,” 2005. [Online]. Available: http://www.martinfowler.com/articles/languageWorkbench.html. [Accessed: 05-Apr-2016].

[342] M. Fowler, “DomainSpecificLanguage,” 2008. [Online]. Available: http://martinfowler.com/bliki/DomainSpecificLanguage.html. [Accessed: 08-Apr-2016].

[343] S. Efftinge and M. Völter, “oAW xText: A Framework for Textual DSLs,” in *Proceedings of Workshop on Modeling Symposium at Eclipse Summit*, 2006, p. 118.

[344] S. Cook, G. Jones, S. Kent, and A. C. Wills, *Domain-specific development with visual studio dsl tools*. Pearson Education, 2007.

[345] D. Ghosh, *DSLs in Action*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2010.

[346] F. Dearle, *Groovy for Domain-Specific Languages*. 2010.

[347] K. Czarnecki and U. Eisenecker, *Generative programming: methods, tools, and applications*, 1st ed. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000.

[348] “LaTeX,” 2015. [Online]. Available: https://www.latex-project.org/. [Accessed: 08-Apr-2016].

[349] E. Meijer, B. Beckman, and G. Bierman, “LINQ: Reconciling Objects, Relations and XML in the

.NET," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*, 2006, p. 706.

[350] D. G. Bobrow, K. Kahn, G. Kiczales, L. Masinter, M. Stefik, and F. Zdybel, "CommonLoops: merging Lisp and object-oriented programming," in *Conference proceedings on Object-oriented programming systems, languages and applications - OOPLSA '86*, 1986, vol. 9, pp. 17–29.

[351] C. E. M. Marín, "Modelo específico de dominio para la construcción de Learning Objects independientes de la plataforma," University of Oviedo, 2011.

[352] E. R. Núñez-Valdez, "Sistemas de Recomendación de Contenidos para Libros Inteligentes," University of Oviedo, 2012.

[353] D. S. Kolovos, R. F. Paige, T. Kelly, and F. A. C. Polack, "Requirements for domain-specific languages," in *Proc. of ECOOP Workshop on Domain-Specific Program Development (DSPD)*, 2006, pp. 1–4.

[354] D. C. W.-A. W. Appel and J. L. K.-C. S. Serra, "The Zephyr Abstract Syntax Description Language," in *Conference on Domain-Specific Languages*, 1997.

[355] Fundación Telefónica, "La Sociedad de la Información en España 2015," 2015.

[356] R. Roman, P. Najera, and J. Lopez, "Securing the Internet of Things," *Computer (Long. Beach. Calif).*, vol. 44, no. 9, pp. 51–58, Sep. 2011.

[357] L. Li, "Technology designed to combat fakes in the global supply chain," *Bus. Horiz.*, vol. 56, no. 2, pp. 167–177, Mar. 2013.

[358] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, Sep. 2012.

[359] B. B. Zaidan, A. A. Zaidan, A. K. Al-Frajat, and H. A. Jalab, "On the Differences between Hiding Information and Cryptography Techniques: An Overview," *J. Appl. Sci.*, vol. 10, no. 15, pp. 1650–1655, Dec. 2010.

[360] I. V. S. Manoj, "Cryptography and Steganography," *Int. J. Comput. Appl.*, vol. 1, no. 12, pp. 63–68, Feb. 2010.

[361] M. A. S. Abomhara, "Enhancing Selective Encryption for H . 264 / AVC Using Advanced Encryption Standard," University of Malaya, 2011.

[362] A. J. Raphael and V. Sundaram, "Cryptography and steganography – A survey," *Int. J. Comput. Technol. Appl.*, vol. 2, no. 3, pp. 626–630, 2011.

[363] A. Siper, R. Farley, and C. Lombardo, "The Rise of Steganography," *Proc. Student/Faculty Res. Day, CSIS, Pace Univ.*, pp. 1–7, 2005.

[364] D. Luciano and G. Prichett, "Cryptology: From Caesar Ciphers to Public-Key Cryptosystems," *Coll. Math. J.*, vol. 18, no. 1, p. 2, Jan. 1987.

[365] S. Singh, *The Code Book: How to Make It, Break It, Hack It, Crack It*. Broadway New York, New

York: Delacorte Press, 2002.

[366] F. Gutiérrez Múñoz and A. del Rey Guerrero, “Estudio estadístico de palabras y caracteres en títulos de artículos científico-técnicos en español,” *Rev. española Doc. científica*, vol. 9, no. 2, pp. 121–132, 1986.

[367] Symantec, “Glosario de Seguridad 101,” 2016. [Online]. Available: https://www.symantec.com/es/mx/theme.jsp?themeid=glosario-de-seguridad. [Accessed: 14-Apr-2016].

[368] NIST Big Data Public Working Group: Reference Architecture Subgroup, “NIST Big Data Interoperability Framework: Volume 6, Reference Architecture,” Gaithersburg, MD, Oct. 2015.

[369] M. Dworkin, “Recommendation for Block Cipher Modes of Operation Methods and Techniques,” *Natl. Inst. Stand. Technol. Spec. Publ. 800-38A 2001 ED*, no. December, p. 66, 2001.

[370] W. F. Patrick, “Steganography: Past, Present, Future,” 2001.

[371] L. Da Vinci, *Leonardo’s Notebooks: Writing and Art of the Great Master*, 1st ed. Black Dog & Leventhal, 2013.

[372] G. Landini, “EVIDENCE OF LINGUISTIC STRUCTURE IN THE VOYNICH MANUSCRIPT USING SPECTRAL ANALYSIS,” *Cryptologia*, vol. 25, no. 4, pp. 275–295, Oct. 2010.

[373] G. Kirby, “ZlPF’S LAW,” *UK J. Nav. Sci.*, vol. 10, no. 3, pp. 180–185, 1985.

[374] H. Suo, J. Wan, C. Zou, and J. Liu, “Security in the Internet of Things: A Review,” in *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, 2012, pp. 648–651.

[375] R. Acharya and K. Asha, “Data integrity and intrusion detection in Wireless Sensor Networks,” in *2008 16th IEEE International Conference on Networks*, 2008, pp. 1–5.

[376] G. Paul and S. Maitra, *RC4 stream cipher and its variants*, 1st ed. CRC press, 2011.

[377] R. L. Rivest, “The RC5 encryption algorithm,” in *Fast Software Encryption*, vol. 2, B. Preneel, Ed. Springer Berlin Heidelberg, 1995, pp. 86–96.

[378] R. L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin, “The RC6 Block Cipher,” in *First Advanced Encryption Standard (AES) Conference*, 1998.

[379] D. Selent, “Advanced encryption standard,” 2001.

[380] J. Daemen and V. Rijmen, *The design of Rijndael: AES the Advanced Encryption Standard*. Springer, 2002.

[381] M. E. Smid and D. K. Branstad, “Data Encryption Standard: past and future,” *Proc. IEEE*, vol. 76, no. 5, pp. 550–559, 1988.

[382] H. Kummert, “The PPP Triple-DES Encryption Protocol (3DESE),” 1998. [Online]. Available: http://www.hjp.at/doc/rfc/rfc2420.html.

[383] B. Schneier, “Description of a new variable-length key, 64-bit block cipher (Blowfish),” in *Journal of Chemical Information and Modeling*, vol. 53, no. 9, R. Anderson, Ed. Springer Berlin Heidelberg, 1994, pp. 191–204.

[384] A. Mousa, “Data encryption performance based on Blowfish,” in *47th International Symposium ELMAR, 2005.*, 2005, pp. 131–134.

[385] M. P. Leong, O. Y. H. Cheung, K. H. Tsoi, and P. H. W. Leong, “A bit-serial implementation of the international data encryption algorithm IDEA,” in *Proceedings 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (Cat. No.PR00871)*, 2000, pp. 122–131.

[386] D. Salomon, *Coding for Data and Computer Communications*. New York: Springer-Verlag, 2005.

[387] W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[388] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[389] T. Elgamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

[390] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, “OpenPGP Message Format,” 2007.

[391] The GnuPG Project, “GnuPG,” 2015. [Online]. Available: https://gnupg.org/. [Accessed: 25-Oct-2016].

[392] R. L. Rivest, “The MD4 Message-Digest Algorithm,” 1992.

[393] R. Rivest, “The MD5 Message-Digest Algorithm,” 1992. [Online]. Available: https://tools.ietf.org/html/rfc1321. [Accessed: 05-Aug-2016].

[394] R. L. Rivest, B. Agre, D. V. Bailey, C. Crutchfield, Y. Dodis, K. E. Fleming, A. Khan, J. Krishnamurthy, Y. Lin, L. Reyzin, E. Shen, J. Sukha, D. Sutherland, E. Tromer, and Y. L. Yin, “The MD6 hash function: A proposal to NIST for SHA-3,” Cambridge, MA, 2008.

[395] Donald E. Eastlake and P. E. Jones, “US Secure Hash Algorithm 1 (SHA1),” 2001. .

[396] National Institute of Standards and Technology, “Secure Hash Standard (SHS) Publication,” Gaithersburg, MD, Jul. 2015.

[397] X. Wang, Y. L. Yin, and H. Yu, “Finding Collisions in the Full SHA-1,” in *Advances in Cryptology – CRYPTO 2005*, no. 90304009, V. Shoup, Ed. Santa Barbara, California, USA: Springer Berlin Heidelberg, 2005, pp. 17–36.

[398] M. J. Dworkin, “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,” Gaithersburg, MD, Jul. 2015.

[399] P. N. Mahalle, N. R. Prasad, and R. Prasad, “Threshold Cryptography-based Group Authentication (TCGA) scheme for the Internet of Things (IoT),” in *2014 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems*

(VITAE), 2014, pp. 1–5.

[400] L. Eschenauer and V. D. Gligor, “A key-management scheme for distributed sensor networks,” in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 41–47.

[401] Guanglei Zhao, Xianping Si, Jingcheng Wang, Xiao Long, and Ting Hu, “A novel mutual authentication scheme for Internet of Things,” in *Proceedings of 2011 International Conference on Modelling, Identification and Control*, 2011, no. January, pp. 563–566.

[402] N. YE, Y. Zhu, R. WANG, R. Malekian, and L. Qiao-min, “An Efficient Authentication and Access Control Scheme for Perception Layer of Internet of Things,” *Appl. Math. Inf. Sci.*, vol. 8, no. 4, pp. 1617–1624, Jul. 2014.

[403] Martin Feldhofer, S. Dominikus, and J. Wolkerstorfer, *Strong Authentication for RFID Systems Using the AES Algorithm*, vol. 3156. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[404] B. Calmels, S. Canard, M. Girault, and H. Sibert, “Low-Cost Cryptography for Privacy in RFID Systems,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3928 LNCS, J. Domingo-Ferrer, J. Posegga, and D. Schreckling, Eds. Springer Berlin Heidelberg, 2006, pp. 237–251.

[405] O. Savry, F. Pebay-Peyroula, F. Dehmas, G. Robert, and J. Reverdy, “RFID Noisy Reader How to Prevent from Eavesdropping on the Communication?,” in *Cryptographic Hardware and Embedded Systems - CHES 2007*, P. Paillier and I. Verbauwhede, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 334–345.

[406] G. V. Lioudakis, E. a. Koutsoloukas, N. Dellas, S. Kapellaki, G. N. Prezerakos, D. I. Kaklamani, and I. S. Venieris, “A Proxy for Privacy: the Discreet Box,” in *EUROCON 2007 - The International Conference on “Computer as a Tool,”* 2007, pp. 966–973.

[407] O. Bergmann, S. Gerdes, and C. Bormann, “Simple Keys for Simple Smart Objects,” in *Workshop on Smart Object Security*, 2012.

[408] danah m. Boyd and N. B. Ellison, “Social Network Sites: Definition, History, and Scholarship,” *J. Comput. Commun.*, vol. 13, no. 1, pp. 210–230, Oct. 2007.

[409] M. Blackstock, R. Lea, and A. Friday, “Uniting online social networks with places and things,” in *Proceedings of the Second International Workshop on Web of Things - WoT ’11*, 2011, p. 1.

[410] D. Guinard, M. Fischer, and V. Trifa, “Sharing using social networks in a composable web of things,” in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, 2010, pp. 702–707.

[411] A. Teutle, “Twitter: Network properties analysis,” in *Electronics, Communications and Computer (CONIELECOMP), 2010 20th International Conference on*, 2010, pp. 180–186.

[412] M. Morris, J. Teevan, and K. Panovich, “What Do People Ask Their Social Networks, and Why?: A Survey Study of Status Message Q\&\#38;a Behavior,” *Proc. SIGCHI Conf. Hum. Factors Comput.*

Syst. SE - CHI '10, pp. 1739–1748, 2010.

[413] J. Heidemann, M. Klier, and F. Probst, "Online social networks: A survey of a global phenomenon," *Comput. Networks*, vol. 56, no. 18, pp. 3866–3878, Dec. 2012.

[414] C. Ross, E. S. Orr, M. Sisic, J. M. Arseneault, M. G. Simmering, and R. R. Orr, "Personality and motivations associated with Facebook use," *Comput. Human Behav.*, vol. 25, no. 2, pp. 578–586, Mar. 2009.

[415] C. McCarthy, "Facebook: One Social Graph to Rule Them All?," *CBS News*, 2010. [Online]. Available: http://www.cbsnews.com/news/facebook-one-social-graph-to-rule-them-all/. [Accessed: 15-Sep-2016].

[416] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2014.

[417] D. Doran, S. Gokhale, and A. Dagnino, "Human sensing for smart cities," in *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2013, pp. 1323–1330.

[418] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake shakes Twitter users: real-time event detection by social sensors," in *Proceedings of the 19th international conference on World wide web - WWW '10*, 2010, p. 851.

[419] D. Richter, K. Riemer, and J. vom Brocke, "Internet Social Networking," *Bus. Inf. Syst. Eng.*, vol. 3, no. 2, pp. 89–101, Apr. 2011.

[420] S. Asur and B. A. Huberman, "Predicting the Future with Social Media," in *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2010, vol. 112, pp. 492–499.

[421] L. Garton, C. Haythornthwaite, and B. Wellman, "Studying Online Social Networks," *J. Comput. Commun.*, vol. 3, no. 1, pp. 75–106, Jun. 2006.

[422] L. Atzori, A. Iera, and G. Morabito, "From 'smart objects' to 'social objects': The next evolutionary step of the internet of things," *IEEE Commun. Mag.*, vol. 52, no. 1, pp. 97–105, Jan. 2014.

[423] A. Smith, "22% of online Americans used social networking or Twitter for politics in 2010 campaign," *Report of the Pew Internet Research Center*, 2011. [Online]. Available: http://www.pewinternet.org/2011/01/27/22-of-online-americans-used-social-networking-or-twitter-for-politics-in-2010-campaign/. [Accessed: 11-Sep-2016].

[424] J. Heidemann, "Online Social Networks – Ein sozialer und technischer Überblick," *Informatik-Spektrum*, vol. 33, no. 3, pp. 262–271, Jun. 2010.

[425] K. Chard, S. Caton, O. Rana, and K. Bubendorfer, "Social Cloud: Cloud Computing in Social Networks," in *2010 IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 99–106.

[426] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding online social network usage from a network perspective," in *Proceedings of the 9th ACM SIGCOMM conference*

on Internet measurement conference - IMC '09, 2009, p. 35.

[427] G. Bai, L. Liu, B. Sun, and J. Fang, "A survey of user classification in social networks," in *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2015, vol. 2015–Novem, pp. 1038–1041.

[428] danah M. Boyd and J. Heer, "Profiles as Conversation: Networked Identity Performance on Friendster," in *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, 2006, vol. 3, no. C, p. 59c–59c.

[429] danah michele Boyd, "Friendster and publicly articulated social networking," in *Conference on Human factors and Computing Systems*, 2004, vol. 2, pp. 1279–1282.

[430] C. M. K. Cheung, P.-Y. Chiu, and M. K. O. Lee, "Online social networks: Why do students use facebook?," *Comput. Human Behav.*, vol. 27, no. 4, pp. 1337–1343, Jul. 2011.

[431] S. Perez, "Anti-Facebook Social Network 'Unthink' Launches to Public," *Statew. Agric. L. Use Baseline 2015*, vol. 1, 2015.

[432] D. Gayo-Avello, "Don't turn social media into another 'Literary Digest' poll," *Commun. ACM*, vol. 54, no. 10, pp. 121–128, Oct. 2011.

[433] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?," in *Proceedings of the 19th international conference on World wide web - WWW '10*, 2010, p. 591.

[434] I. Ponce, "MONOGRÁFICO: Redes Sociales," *Observatorio Tecnológico*, 2012. [Online]. Available: http://recursostic.educacion.es/observatorio/web/en/internet/web-20/1043-redes-sociales. [Accessed: 17-Sep-2016].

[435] Lars Backstrom, "Anatomy of Facebook," *Facebook Data Science*, 2011. [Online]. Available: https://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859. [Accessed: 10-Sep-2016].

[436] F. Karinthy, "Chain-Links," in *Everything is Different*, 1929, pp. 1–4.

[437] S. Milgram, "The small world problem," *Psychol. Today*, vol. 2, no. 1, pp. 61–67, 1967.

[438] Oakland University, "The Erdös Number Project," 1996. [Online]. Available: http://wwwp.oakland.edu/enp/. [Accessed: 10-Sep-2016].

[439] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna, "Four Degrees of Separation," in *Proceedings of the 3rd Annual ACM Web Science Conference on - WebSci '12*, 2011, pp. 33–42.

[440] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, "The Anatomy of the Facebook Social Graph," *arXiv.org*, Nov. 2011.

[441] M. Burke, L. Adamic, A. Herdağdelen, and D. Neumann, "Cat People, Dog People," *Research at facebook*, 2016. [Online]. Available: https://research.facebook.com/blog/cat-people-dog-people/. [Accessed: 21-Sep-2016].

[442] J. Choi, S. Park, H. Ko, H.-J. Moon, and J. Lee, "Issues for Applying Instant Messaging to Smart

Home Systems," in *Computational Science and Its Applications – ICCSA 2009*, vol. 5592, no. PART 1, O. Gervasi, D. Taniar, B. Murgante, A. Laganà, Y. Mun, and M. L. Gavrilova, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 649–661.

[443] M. Kranz, L. Roalter, and F. Michahelles, "Things that twitter: social networks and the internet of things," *What can Internet Things do Citiz. Work. Eighth Int. Conf. Pervasive Comput. (Pervasive 2010).*, pp. 1–10, 2010.

[444] P. Anantharam, P. Barnaghi, K. Thirunarayan, and A. Sheth, "Extracting City Traffic Events from Social Streams," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 4, pp. 1–27, Jul. 2015.

[445] A. Cacho, M. Figueredo, A. Cassio, M. V. Araujo, L. Mendes, J. Lucas, H. Farias, J. Coelho, N. Cacho, and C. Prolo, "Social Smart Destination: A Platform to Analyze User Generated Content in Smart Tourism Destinations," in *New Advances in Information Systems and Technologies*, Á. Rocha, M. A. Correia, H. Adeli, P. L. Reis, and M. Mendonça Teixeira, Eds. Cham: Springer International Publishing, 2016, pp. 817–826.

[446] S. Aurell, "Remote Controlling Devices Using Instant Messaging: Building an Intelligent Gateway in Erlang/OTP," in *Proceedings of the 2005 ACM SIGPLAN workshop on Erlang*, 2005, pp. 46–51.

[447] J. Choi and C.-W. Yoo, "Connect with Things through Instant Messaging," in *The Internet of Things*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 276–288.

[448] P. A. Kirschner and A. C. Karpinski, "Facebook® and academic performance," *Comput. Human Behav.*, vol. 26, no. 6, pp. 1237–1245, 2010.

[449] E. C. Tandoc, P. Ferrucci, and M. Duffy, "Facebook use, envy, and depression among college students: Is facebooking depressing?," *Comput. Human Behav.*, vol. 43, pp. 139–146, Feb. 2015.

[450] T. Ryan and S. Xenos, "Who uses Facebook? An investigation into the relationship between the Big Five, shyness, narcissism, loneliness, and Facebook usage," *Comput. Human Behav.*, vol. 27, no. 5, pp. 1658–1664, Sep. 2011.

[451] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A Survey of Research on Cloud Robotics and Automation," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 398–409, Apr. 2015.

[452] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, Randy H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," *Univ. California, Berkeley, Tech. Rep. UCB*, pp. 07–013, 2009.

[453] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Futur. Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, Jun. 2009.

[454] D. Agrawal, S. Das, and A. El Abbadi, "Big Data and Cloud Computing: Current State and Future Opportunities," in *Proceedings of the 14th International Conference on Extending Database Technology - EDBT/ICDT '11*, 2011, p. 530.

[455] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. S. Netto, and R. Buyya, "Big Data computing

and clouds: Trends and future directions," *J. Parallel Distrib. Comput.*, vol. 79–80, pp. 3–15, May 2015.

[456] R. Buyya, S. Pandey, and C. Vecchiola, "Cloudbus Toolkit for Market-Oriented Cloud Computing," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5931 LNCS, Springer Berlin Heidelberg, 2009, pp. 24–44.

[457] S. Pandey and S. Nepal, "Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond," *Futur. Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1774–1776, Sep. 2013.

[458] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing — The business perspective," *Decis. Support Syst.*, vol. 51, no. 1, pp. 176–189, Apr. 2011.

[459] A. Weiss, "Computing in the clouds," *netWorker - Cloud Comput. PC Funct. move onto web*, vol. 11, no. 4, pp. 16–25, Dec. 2007.

[460] Interoute Communications Limited, "What is Cloud Computing?," 2016. [Online]. Available: http://www.interoute.com/cloud-article/what-cloud-computing. [Accessed: 03-Sep-2016].

[461] P. Mell and T. Grance, "The NIST definition of cloud computing," 2011.

[462] G. Pallis, *Cloud Computing: The New Frontier of Internet Computing*, vol. 142. Cham: Springer International Publishing, 2015.

[463] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," in *ACM SIGOPS Operating Systems Review*, 2003, vol. 37, no. 5, pp. 29–43.

[464] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, p. 107, 2008.

[465] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, "The rise of 'big data' on cloud computing: Review and open research issues," *Inf. Syst.*, vol. 47, pp. 98–115, Jan. 2015.

[466] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Futur. Gener. Comput. Syst.*, vol. 29, no. 1, pp. 84–106, 2013.

[467] M. T. Jones, "Anatomy of a Linux hypervisor," *IBM developersWorkds*, 2009. [Online]. Available: http://www.ibm.com/developerworks/library/l-hypervisor/. [Accessed: 08-Sep-2016].

[468] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Sixth symposium on operating system design and implementation*, 2004, pp. 137–150.

[469] B. P. Rimal, E. Choi, and I. Lumb, "A Taxonomy and Survey of Cloud Computing Systems," in *2009 Fifth International Joint Conference on INC, IMS and IDC*, 2009, pp. 44–51.

[470] G. F. Pfister, *In search of clusters*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.

[471] R. Buyya, *High Performance Cluster Computing: Architectures and Systems*, vol. 1. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

[472] S. Bhardwaj, L. Jain, and S. Jain, "Cloud Computing: a Study of Infrastructure As a Service (IaaS),"

Int. J. Eng., vol. 2, no. 1, pp. 60–63, 2010.

[473] W. Dawoud, I. Takouna, and C. Meinel, "Infrastructure as a Service Security: Challenges and Solutions," in *Informatics and Systems (INFOS), 2010 The 7th International Conference on*, 2010, no. January, pp. 1–8.

[474] M. Boucadair and C. Jacquenet, "Software-Defined Networking: A Perspective from within a Service Provider Environment," 2014.

[475] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69, Mar. 2008.

[476] J. Hurwitz, M. Kaufman, and D. Kirsch, *Hybrid Cloud For Dummies*, IBM Limite. John Wiley & Sons, Inc., 2015.

[477] J. Hofstader, "Communications as a Service," *Microsoft MSDN*, 2007. .

[478] Microsoft, "Strategy and Planning for Software plus Services," 2009.

[479] T. S. Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, "What's Inside the Cloud ? An Architectural Map of the Cloud Landscape," *Softw. Eng. Challenges Cloud Comput. 2009. CLOUD '09.*, pp. 23–31, 2009.

[480] M. A. Chauhan, "Foundations for Tools as a Service Workspace: A Reference Architecture," IT University of Copenhagen, 2015.

[481] Raj Bala and A. Chandrasekaran, "Magic Quadrant for Public Cloud Storage Services, Worldwide," 2016. [Online]. Available: http://www.gartner.com/technology/reprints.do?id=1-2IH2LGI&ct=150626&st=sb. [Accessed: 05-Sep-2016].

[482] E. A. Brewer, "Towards robust distributed systems," in *PODC*, 2000, vol. 7.

[483] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, p. 51, Jun. 2002.

[484] S. S. Y. Shim, "Guest Editor's Introduction: The CAP Theorem's Growing Impact," *Computer (Long. Beach. Calif).*, vol. 45, no. 2, pp. 21–22, Feb. 2012.

[485] K. Birman, D. Freedman, Q. Huang, and P. Dowell, "Overcoming CAP with Consistent Soft-State Replication," *Computer (Long. Beach. Calif).*, vol. 45, no. 2, pp. 50–58, Feb. 2012.

[486] D. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story," *Computer (Long. Beach. Calif).*, vol. 45, no. 2, pp. 37–42, Feb. 2012.

[487] E. Brewer, "CAP twelve years later: How the 'rules' have changed," *Computer (Long. Beach. Calif).*, vol. 45, no. 2, pp. 23–29, Feb. 2012.

[488] S. Nadella, "The Partnership of the Future," 2016.

[489] A. M. Turing, "Computing Machinery and Intelligence," *Mind*, vol. 59, pp. 433–460, 1950.

[490] J. Fuegi and J. Francis, “Lovelace & babbage and the creation of the 1843 ‘notes,’” *IEEE Ann. Hist. Comput.*, vol. 25, no. 4, pp. 16–26, Oct. 2003.

[491] J. H. Holland, “Outline for a Logical Theory of Adaptive Systems,” *J. ACM*, vol. 9, no. 3, pp. 297–314, 1962.

[492] M. Melanie, *An introduction to genetic algorithms*, 5th ed. MIT press, 1999.

[493] J. McCarthy, “What is Artificial Intelligence?,” 2001. [Online]. Available: http://lidecc.cs.uns.edu.ar/~grs/InteligenciaArtificial/whatisai.pdf. [Accessed: 26-Apr-2016].

[494] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, “A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence,” Dartmouth, 1955.

[495] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, “A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence,” *AI Mag.*, vol. 27, no. 4, pp. 12–14, 2006.

[496] J. R. Stuart and N. Peter, *Artificial Intelligence A Modern Approach*, 2nd ed. New Jersey: Prentice Hall, 2003.

[497] P. Simon, *Too Big to Ignore: The Business Case for Big Data*, 1st ed. Wiley, 2013.

[498] M. Wernick, Y. Yang, J. Brankov, G. Yourganov, and S. Strother, “Machine Learning in Medical Imaging,” *IEEE Signal Process. Mag.*, vol. 27, no. 4, pp. 25–38, Jul. 2010.

[499] A. McCallum, K. Nigam, J. Rennie, and K. Seymore, “A Machine Learning Approach to Building Domain-Specific Search Engines,” *Int. Jt. Conf. Artif. Intell.*, vol. 16, pp. 662–667, 1999.

[500] P. Viola and M. J. Jones, “Robust Real-Time Face Detection,” *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, May 2004.

[501] C. G. Keller, T. Dang, H. Fritz, A. Joos, C. Rabe, and D. M. Gavrila, “Active Pedestrian Safety by Automatic Braking and Evasive Steering,” *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1292–1304, Dec. 2011.

[502] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. 2011.

[503] T. Menzies and Y. Hu, “Data mining for very busy people,” *Computer (Long. Beach. Calif).*, vol. 36, no. 11, pp. 22–29, Nov. 2003.

[504] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2012.

[505] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., vol. 27, no. 2. New York, NY: Springer New York, 2009.

[506] N. Lange, C. M. Bishop, and B. D. Ripley, “Neural Networks for Pattern Recognition,” *J. Am. Stat. Assoc.*, vol. 92, no. 440, p. 1642, Dec. 1997.

[507] X. Zhu, “Semi-Supervised Learning Literature Survey,” *Sci. York*, pp. 1–59, 2007.

[508] J. A. Hertz, A. S. Krogh, R. G. Palmer, and A. S. Weigend, *Introduction to the Theory of Neural*

Computation, vol. I, no. June. Basic Books, 1991.

[509] A. Gammerman, V. Vovk, and V. Vapnik, "Learning by transduction," in *Fourteenth conference on Uncertainty in artificial intelligence*, 1998, no. l, pp. 148–155.

[510] T. Joachims, "Transductive Inference for Text Classification Using Support Vector Machines," in *ICML '99 Proceedings of the Sixteenth International Conference on Machine Learning*, 1999, vol. 99, pp. 200–209.

[511] T. Joachims, "Transductive Support Vector Machines," 2006, pp. 105–118.

[512] R. Caruana, "Multitask Learning," Carnegie Mellon University, 1997.

[513] J. R. Quinlan, "Induction of Decision Trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[514] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, Jun. 1993.

[515] S.-C. Wang, "Artificial Neural Network," in *Interdisciplinary Computing in Java Programming*, Boston, MA: Springer US, 2003, pp. 81–100.

[516] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[517] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[518] J. Schmidhuber, "Deep Learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[519] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*, 1992, pp. 144–152.

[520] C. Cortes and V. Vapnik, "Support-Vector Networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[521] K.-R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik, "Predicting time series with support vector machines," in *Artificial Neural Networks—ICANN'97*, vol. 1327, no. X, 1997, pp. 999–1004.

[522] M. S. Aldenderfer and R. K. Blashfield, *Cluster Analysis*. Sage Publications, Inc., 1984.

[523] R. Duda, P. Hart, and D. Stork, *Pattern Classification and Scene Analysis*. John Wiley & Sons Inc, 1973.

[524] V. Estivill-Castro, "Why so many clustering algorithms," *ACM SIGKDD Explor. Newsl.*, vol. 4, no. 1, pp. 65–75, Jun. 2002.

[525] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The KDD process for extracting useful knowledge from volumes of data," *Commun. ACM*, vol. 39, no. 11, pp. 27–34, Nov. 1996.

[526] L. Greiner, "What is Data Analysis and Data Mining?," 2011. [Online]. Available:

http://www.dbta.com/Editorial/Trends-and-Applications/What-is-Data-Analysis-and-Data-Mining-73503.aspx. [Accessed: 26-May-2016].

[527] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian Network Classifiers," *Mach. Learn.*, vol. 29, no. 2/3, pp. 131–163, 1997.

[528] C. Wang, N. Komodakis, and N. Paragios, "Markov Random Field modeling, inference & learning in computer vision & image understanding: A survey," *Comput. Vis. Image Underst.*, vol. 117, no. 11, pp. 1610–1627, Nov. 2013.

[529] R. Muhammad Anwer, D. Vázquez, and A. M. López, "Opponent Colors for Human Detection," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6669 LNCS, 2011, pp. 363–370.

[530] T. B. Moeslund and E. Granum, "A Survey of Computer Vision-Based Human Motion Capture," *Comput. Vis. Image Underst.*, vol. 81, no. 3, pp. 231–268, Mar. 2001.

[531] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, vol. 1, pp. 886–893.

[532] Dong-Chen He and Li Wang, "Texture Unit, Texture Spectrum And Texture Analysis," in *12th Canadian Symposium on Remote Sensing Geoscience and Remote Sensing Symposium*, 1989, vol. 5, pp. 2769–2772.

[533] Dong-chen He and Li Wang, "Texture Unit, Texture Spectrum, And Texture Analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 28, no. 4, pp. 509–512, Jul. 1990.

[534] L. Wang and D.-C. He, "Texture classification using texture spectrum," *Pattern Recognit.*, vol. 23, no. 8, pp. 905–910, Jan. 1990.

[535] X. Wang, T. X. Han, and S. Yan, "An HOG-LBP human detector with partial occlusion handling," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, no. Iccv, pp. 32–39.

[536] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, vol. 2, no. [8, pp. 1150–1157 vol.2.

[537] R. Funayama, H. Yanagihara, L. Van Gool, T. Tuytelaars, and H. Bay, "Robust interest point detector and descriptor," US 8,165,401 B2, 2006.

[538] S. H. Walker and D. B. Duncan, "Estimation of the Probability of an Event as a Function of Several Independent Variables," *Biometrika*, vol. 54, no. 1/2, p. 167, Jun. 1967.

[539] D. R. Cox, "The Regression Analysis of Binary Sequences," *J. R. Stat. Soc.*, vol. 20, no. 2, pp. 215–242, 1958.

[540] Y. Freund and R. E. Schapire, "A decision theoretic generalization of on-line learning and an application to boosting," *Comput. Syst. Sci.*, vol. 57, pp. 119–139, 1997.

[541] M. Begum and F. Karray, "Visual Attention for Robotic Cognition: A Survey," *IEEE Trans. Auton.*

Ment. Dev., vol. 3, no. 1, pp. 92–105, Mar. 2011.

[542] Y. Liu and Q. Dai, "A survey of computer vision applied in Aerial robotic Vehicles," in *2010 International Conference on Optics, Photonics and Energy Engineering (OPEE)*, 2010, vol. 1, no. 201, pp. 277–280.

[543] M. Mubashir, L. Shao, and L. Seed, "A survey on fall detection: Principles and approaches," *Neurocomputing*, vol. 100, pp. 144–152, Jan. 2013.

[544] H. Foroughi, A. Naseri, A. Saberi, and H. Sadoghi Yazdi, "An eigenspace-based approach for human fall detection using Integrated Time Motion Image and Neural Network," in *2008 9th International Conference on Signal Processing*, 2008, pp. 1499–1503.

[545] Zhengming Fu, E. Culurciello, P. Lichtsteiner, and T. Delbruck, "Fall detection using an address-event temporal contrast vision sensor," in *2008 IEEE International Symposium on Circuits and Systems*, 2008, pp. 424–427.

[546] H. Foroughi, B. S. Aski, and H. Pourreza, "Intelligent video surveillance for monitoring fall detection of elderly in home environments," in *2008 11th International Conference on Computer and Information Technology*, 2008, no. Iccit, pp. 219–224.

[547] B. Jansen and R. Deklerck, "Context aware inactivity recognition for visual fall detection," in *2006 Pervasive Health Conference and Workshops*, 2006, pp. 1–4.

[548] L. A. Zadeh, "Fuzzy logic and approximate reasoning," *Synthese*, vol. 30, no. 3–4, pp. 407–428, 1975.

[549] L. a. Zadeh, "Fuzzy sets," *Inf. Control*, vol. 8, no. 3, pp. 338–353, Jun. 1965.

[550] P. Grant, "A new approach to diabetic control: Fuzzy logic and insulin pump technology," *Med. Eng. Phys.*, vol. 29, no. 7, pp. 824–827, Sep. 2007.

[551] E. Portmann, A. Andrushevich, R. Kistler, and A. Klapproth, "Prometheus - Fuzzy information retrieval for semantic homes and environments," in *3rd International Conference on Human System Interaction*, 2010, pp. 757–762.

[552] O. G. Duarte, "Sistemas de lógica difusa. Fundamentos," *Rev. Ing. e Invetigación*, no. 42, pp. 22–30, 1999.

[553] W. A. Kwong and K. M. Passino, "Dynamically focused fuzzy learning control," *IEEE Trans. Syst. Man Cybern. Part B*, vol. 26, no. 1, pp. 53–74, 1996.

[554] D. F. Larios, J. Barbancho, F. J. Molina, and C. León, "LIS: Localization based on an intelligent distributed fuzzy system applied to a WSN," *Ad Hoc Networks*, vol. 10, no. 3, pp. 604–622, 2012.

[555] I. Chamodrakas and D. Martakos, "A utility-based fuzzy TOPSIS method for energy efficient network selection in heterogeneous wireless networks," *Appl. Soft Comput.*, vol. 11, no. 4, pp. 3734–3743, Jun. 2011.

[556] S. Bagchi, "A fuzzy algorithm for dynamically adaptive multimedia streaming," *ACM Trans. Multimed. Comput. Commun. Appl.*, vol. 7, no. 2, pp. 1–26, Feb. 2011.

[557] G. Cueva-Fernandez, J. Pascual Espada, V. García-Díaz, and R. Gonzalez-Crespo, "Fuzzy decision method to improve the information exchange in a vehicle sensor tracking system," *Appl. Soft Comput.*, vol. 35, pp. 1–9, Oct. 2015.

[558] G. Cueva-Fernandez, J. P. Espada, V. García-Díaz, R. G. Crespo, and N. Garcia-Fernandez, "Fuzzy system to adapt web voice interfaces dynamically in a vehicle sensor tracking application definition," *Soft Comput.*, pp. 1–14, May 2015.

[559] B. Z. Manaris and W. D. Dominick, "NALIGE: a user interface management system for the development of natural language interfaces," *Int. J. Man. Mach. Stud.*, vol. 38, no. 6, pp. 891–921, Jun. 1993.

[560] B. Z. Manaris and B. M. Slator, "Interactive Natural Language Processing: Building on Success," *Computer (Long. Beach. Calif).*, vol. 29, no. 7, pp. 28–32, Jul. 1996.

[561] E. M. Sibarani, M. Nadial, E. Panggabean, and S. Meryana, "A Study of Parsing Process on Natural Language Processing in Bahasa Indonesia," in *2013 IEEE 16th International Conference on Computational Science and Engineering*, 2013, pp. 309–316.

[562] B. Manaris, "Natural Language Processing: A Human-Computer Interaction Perspective," in *Advances in Computers*, vol. 47, no. C, 1998, pp. 1–66.

[563] C. Friedman, T. C. Rindflesch, and M. Corn, "Natural language processing: State of the art and prospects for significant progress, a workshop sponsored by the National Library of Medicine," *J. Biomed. Inform.*, vol. 46, no. 5, pp. 765–773, Oct. 2013.

[564] K. W. Church and L. F. Rau, "Commercial applications of natural language processing," *Commun. ACM*, vol. 38, no. 11, pp. 71–79, 1995.

[565] F. Colace, M. De Santo, L. Greco, and P. Napoletano, "Text classification using a few labeled examples," *Comput. Human Behav.*, vol. 30, pp. 689–697, 2014.

[566] P. Nesi, G. Pantaleo, and G. Sanesi, "A hadoop based platform for natural language processing of web pages and documents," *J. Vis. Lang. Comput.*, vol. 31, pp. 130–138, Dec. 2015.

[567] R. Miikkulainen, *Subsymbolic natural language processing: An integrated model of scripts, lexicon, and memory*. MIT press, 1993.

[568] A. Akmajian, R. A. Demer, A. K. Farmer, and R. M. Harnish, *Linguistics: An introduction to language and communication*. MIT press, 2001.

[569] J. Allen, *Natural language understanding*, 2nd ed. Pearson, 1995.

[570] D. S. Carrell, D. Cronkite, R. E. Palmer, K. Saunders, D. E. Gross, E. T. Masters, T. R. Hylan, and M. Von Korff, "Using natural language processing to identify problem usage of prescription opioids," *Int. J. Med. Inform.*, vol. 84, no. 12, pp. 1057–1064, Dec. 2015.

[571] S. Alansary, M. Nagi, and N. Adly, "A suite of tools for Arabic natural language processing: A UNL approach," in *2013 1st International Conference on Communications, Signal Processing, and their Applications (ICCSPA)*, 2013, pp. 1–6.

[572] P. Drews and P. Fromm, “A natural language processing approach for mobile service robot control,” in *Proceedings of the IECON’97 23rd International Conference on Industrial Electronics, Control, and Instrumentation (Cat. No.97CH36066)*, 1997, vol. 3, pp. 1275–1277.

[573] F. Siasar djahantighi, M. Norouzifard, S. H. Davarpanah, and M. H. Shenassa, “Using natural language processing in order to create SQL queries,” in *2008 International Conference on Computer and Communication Engineering*, 2008, pp. 600–604.

[574] O. Keszocze, M. Soeken, E. Kuksa, and R. Drechsler, “Lips: An IDE for model driven engineering based on natural language processing,” in *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, 2013, pp. 31–38.

[575] M. Ibrahim and R. Ahmad, “Class Diagram Extraction from Textual Requirements Using Natural Language Processing (NLP) Techniques,” in *2010 Second International Conference on Computer Research and Development*, 2010, pp. 200–204.

[576] A. Tripathy and S. K. Rath, “Application of Natural Language Processing in Object Oriented Software Development,” in *2014 International Conference on Recent Trends in Information Technology*, 2014, pp. 1–7.

[577] A. McAfee and E. Brynjolfsson, “Big data: The Management Revolution,” *Harv. Bus. Rev.*, vol. 90, no. 10, 2012.

[578] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, “Knowledge Discovery in Databases: An Overview,” *AI Mag.*, vol. 13, no. 3, pp. 57–70, 1992.

[579] J.-P. Dijcks, “Oracle: Big data for the enterprise,” Redwood Shores, CA, U.S.A., 2013.

[580] S. Lavalle, E. Lesser, R. Shockley, M. S. Hopkins, and N. Kruschwitz, “Big Data, Analytics and the Path From Insights to Value,” *Winter 2011*, vol. 52, no. 2, pp. 21–31, 2011.

[581] H. Chen, R. H. L. Chiang, and V. C. Storey, “Business Intelligence and Analytics: From Big Data To Big Impact,” *MIS Q.*, vol. 36, no. 4, pp. 1165–1188, 2012.

[582] L. Rokach and O. Maimom, *Data mining with decision trees: theory and applications*. 2007.

[583] W. Fan and A. Bifet, “Mining Big Data: Current Status, and Forecast to the Future,” *ACM SIGKDD Explor. Newsl.*, vol. 14, no. 2, p. 1, Apr. 2013.

[584] E. Letouzé, “Big Data for Development: Challenges & Opportunities,” *Glob. Pulse is a United Nations Initiat.*, no. May, p. 47, 2012.

[585] Cloud Security Alliance, “Top Ten Big Data Security and Privacy Challenges,” 2012.

[586] The Apache Software Foundation, “Apache™ Hadoop®,” 2016. [Online]. Available: http://hadoop.apache.org/. [Accessed: 16-Jun-2016].

[587] J. Ahrens, B. Hendrickson, G. Long, S. Miller, R. Ross, and D. Williams, “Data-Intensive Science in the US DOE: Case Studies and Future Challenges,” *Comput. Sci. Eng.*, vol. 13, no. 6, pp. 14–24, Nov. 2011.

[588] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition, and productivity," 2011.

[589] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, *Advances in knowledge discovery and data mining*. The MIT Press, 1996.

[590] G. Bell, T. Hey, and A. Szalay, "Beyond the Data Deluge," *Science (80-.).*, vol. 323, no. 5919, pp. 1297–1298, 2009.

[591] J. Mervis, "Agencies Rally to Tackle Big Data," *Science (80-.).*, vol. 336, no. 6077, pp. 22–22, Apr. 2012.

[592] G. Bello-Orgaz, J. J. Jung, and D. Camacho, "Social big data: Recent achievements and new challenges," *Inf. Fusion*, vol. 28, pp. 45–59, Mar. 2016.

[593] K. Kambatla, G. Kollias, V. Kumar, and A. Grama, "Trends in big data analytics," *J. Parallel Distrib. Comput.*, vol. 74, no. 7, pp. 2561–2573, Jul. 2014.

[594] Intel IT Center, "Big Data Analytics. Intel's IT Manager Survey on How Organizations Are Using Big Data," 2012.

[595] HowieT, "The Big Bang: How the Big Data Explosion Is Changing the World," *Microsoft Blog*, 2013. [Online]. Available: https://blogs.msdn.microsoft.com/microsoftenterpriseinsight/2013/04/15/the-big-bang-how-the-big-data-explosion-is-changing-the-world/. [Accessed: 24-May-2016].

[596] C. L. Philip Chen and C. Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data," *Inf. Sci. (Ny).*, vol. 275, pp. 314–347, 2014.

[597] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[598] NIST Big Data Public Working Group: Definitions and Taxonomies Subgroup, "NIST Big Data Interoperability Framework: Volume 1, Definitions," Gaithersburg, MD, Oct. 2015.

[599] E. Savitz, "Gartner: 10 Critical Tech Trends For The Next Five Years," *Forbes*, 2012. [Online]. Available: http://www.forbes.com/sites/ericsavitz/2012/10/22/gartner-10-critical-tech-trends-for-the-next-five-years/. [Accessed: 23-Jun-2016].

[600] A. Labrinidis and H. V. Jagadish, "Challenges and opportunities with big data," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 2032–2033, Aug. 2012.

[601] P. T. Metaxas, E. Mustafaraj, and D. Gayo-Avello, "How (Not) to Predict Elections," in *2011 IEEE Third Int'l Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third Int'l Conference on Social Computing*, 2011, pp. 165–171.

[602] D. Lazer, R. Kennedy, G. King, and A. Vespignani, "The Parable of Google Flu: Traps in Big Data Analysis," *Science (80-.).*, vol. 343, no. 6176, pp. 1203–1205, Mar. 2014.

[603] B. Wildavsky, "Tilting at Billboards," *The New Republic*, no. August, pp. 19–20, 1990.

[604] T. Kalil, "Big Data is a Big Deal," 2012. [Online]. Available:

https://www.whitehouse.gov/blog/2012/03/29/big-data-big-deal. [Accessed: 09-May-2016].

[605] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *AI Mag.*, vol. 17, no. 3, pp. 37–54, Aug. 1996.

[606] W. H. Inmon, *Building the data warehouse*, 3rd ed. New York, NY, USA: John Wiley & Sons Inc, 2002.

[607] D. Moody and M. A. . Kortink, "From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design," in *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2000)*, 2000, pp. 1–12.

[608] E. F. Codd, S. B. Codd, and C. T. Salley, "Providing OLAP (on-line Analytical Processing) to User-analysts: An IT Mandate," *Codd Date*, vol. 32, pp. 3–5, 1993.

[609] J. H. Friedman, "Data Mining and Statistics: What's the connection?," in *roceedings of the 29th Symposium on the Interface Between Computer Science and Statistics*, 1997, pp. 3–9.

[610] G. Piatetsky-Shapiro, "From Data Mining to Big Data and Beyond," *http://insideanalysis.com/*, 2012. [Online]. Available: http://insideanalysis.com/2012/04/data-mining-and-beyond/. [Accessed: 20-Jun-2016].

[611] S. H. Ha and S. C. Park, "Application of data mining tools to hotel data mart on the Intranet for database marketing," *Expert Syst. Appl.*, vol. 15, no. 1, pp. 1–31, Jul. 1998.

[612] NIST Big Data Public Working Group: Reference Architecture Subgroup, "NIST Big Data Interoperability Framework: Volume 5, Architectures White Paper Survey," Gaithersburg, MD, Oct. 2015.

[613] Microsoft, "Data Mining Algorithms (Analysis Services - Data Mining)," *Microsoft MSDN*, 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms175595.aspx. [Accessed: 30-Jun-2016].

[614] D. J. Hand, *Discrimination and classification*, vol. 1. J. Wiley, 1981.

[615] G. Piatetsky-Shapiro, "Knowledge Discovery in Real Databases: A Report on the IJCAI-89 Workshop," *AI Mag.*, vol. 11, no. 4, pp. 68–70, 1990.

[616] U. Fayyad, D. Haussler, and P. Stolorz, "KDD for Science Data Analysis: Issues and Examples," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996, pp. 50–56.

[617] U. M. Fayyd, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery: an overview," in *Advances in knowledge discovery and data mining*, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. Menlo Park, CA, USA: Morgan Kaufmann, 1996, pp. 1–34.

[618] Microsoft, *Data mining*. 1998.

[619] Microsoft, "Discretization Methods (Data Mining)," *Microsoft MSDN*, 2016. [Online]. Available:

https://msdn.microsoft.com/en-us/library/ms174512.aspx. [Accessed: 30-Jun-2016].

[620] U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-value attributes for classification learning," *Int. Jt. Conf. Artif. Intell.*, pp. 1022–1027, 1993.

[621] J. R. Mashey, "Big Data and the next wave of infraStress," in *Computer Science Division Seminar, University of California, Berkeley*, 1997.

[622] S. M. Weiss and N. Indurkhya, *Predictive data mining: a practical guide*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann, 1997.

[623] F. Diebold, "On the Origin (s) and Development of the Term Big Data," Philadelphia, PA, USA, 12–037, 2012.

[624] T. Hey, S. Tansley, and K. Tolle, *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Redmond, Washington, 2009.

[625] M. Chen, S. Mao, and Y. Liu, "Big Data: A Survey," *Mob. Networks Appl.*, vol. 19, no. 2, pp. 171–209, Apr. 2014.

[626] J. Dutcher, "What Is Big Data?," 2014. [Online]. Available: https://datascience.berkeley.edu/what-is-big-data/. [Accessed: 25-May-2016].

[627] J. S. Ward and A. Barker, "Undefined By Data: A Survey of Big Data Definitions," *arXiv.org*, p. 2, Sep. 2013.

[628] D. Boyd and K. Crawford, "Critical Questions for Big Data: Provocations for a cultural, technological, and scholarly phenomenon," *Information, Commun. Soc.*, vol. 15, no. 5, pp. 662–679, 2012.

[629] C. Pettey and L. Goasduff, "Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data," *Gartner*, 2011. [Online]. Available: http://www.gartner.com/newsroom/id/1731916. [Accessed: 21-Jun-2016].

[630] Gartner Inc., "IT Glossary: Big Data," *Gartner*, 2016. [Online]. Available: http://www.gartner.com/it-glossary/information-governance. [Accessed: 23-Jun-2016].

[631] B. J. Gantz and D. Reinsel, "Extracting Value from Chaos," 2011.

[632] NIST Big Data Public Working Group: Technology Roadmap Subgroup, "NIST Big Data Interoperability Framework: Volume 7, Standards Roadmap," Gaithersburg, MD, Oct. 2015.

[633] D. Laney, "3D Data Management: Controlling Data Volume, Velocity, and Variety," *META Gr. Res. Note*, vol. 6, p. 70, 2001.

[634] D. Goldston, "Big data: Data wrangling," *Nature*, vol. 455, no. 7209, pp. 15–15, Sep. 2008.

[635] NIST Big Data Public Working Group: Security and Privacy Subgroup, "NIST Big Data Interoperability Framework: Volume 4, Security and Privacy," Gaithersburg, MD, 2015.

[636] IBM, "Big data at the speed of business." [Online]. Available: http://www-01.ibm.com/software/data/bigdata/. [Accessed: 23-May-2016].

[637] C. Doctorow, “Big data: Welcome to the petacentre,” *Nature*, vol. 455, no. 7209, pp. 16–21, Sep. 2008.

[638] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, “Finding a needle in Haystack: Facebook’s photo storage,” in *Design*, 2010, no. October, pp. 1–8.

[639] M. Trewe, “How carriers gather, track and sell your private data,” *The American Genius*, 2012.

[640] A. Sharp, “Dispatch from the Denver debate,” 2012. [Online]. Available: https://blog.twitter.com/2012/dispatch-from-the-denver-debate. [Accessed: 04-May-2016].

[641] Twitter Inc., “Twitter: Company,” 2016. [Online]. Available: https://about.twitter.com/es/company. [Accessed: 09-Jun-2016].

[642] N. Sawant and H. Shah, “Big Data Application Architecture Q&A A Problem - Solution Approach,” Intergovernmental Panel on Climate Change, Ed. Cambridge: Cambridge University Press, 2013, p. 172.

[643] World Data Group, “The World Bank,” 2016. [Online]. Available: http://data.worldbank.org/indicator/. [Accessed: 09-Jun-2016].

[644] YouTube, “YouTube: Statistics,” 2015. [Online]. Available: https://www.youtube.com/yt/press/en/statistics.html. [Accessed: 09-Jun-2016].

[645] Google, “Google Photos: One year, 200 million users, and a whole lot of selfies,” *Google Blog*, 2016. [Online]. Available: https://googleblog.blogspot.com.es/2016/05/google-photos-one-year-200-million.html. [Accessed: 01-Jun-2016].

[646] F. Michel, “How many public photos are uploaded to Flickr every day, month, year?,” 2015. [Online]. Available: https://www.flickr.com/photos/franckmichel/6855169886/. [Accessed: 04-May-2016].

[647] Facebook, “Newsroom,” 2016. [Online]. Available: https://newsroom.fb.com/company-info/. [Accessed: 09-Jun-2016].

[648] Cisco, “Cisco Visual Networking Index: Forecast and Methodology, 2015–2020,” 2016. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html. [Accessed: 09-Jun-2016].

[649] P. E. Dewdney, P. J. Hall, R. T. Schilizzi, and T. J. L. W. Lazio, “The Square Kilometre Array,” *Proc. IEEE*, vol. 97, no. 8, pp. 1482–1496, Aug. 2009.

[650] IEEE Computer Society and ACM, “SC08 International Conference for High Performance Computing,” 2008. [Online]. Available: http://sc08.supercomputing.org/. [Accessed: 23-Jun-2016].

[651] Astrophysical Research Consortium and The Sloan Digital Sky Survey, “SDSS,” 2016. [Online]. Available: http://www.sdss.org/. [Accessed: 23-Jun-2016].

[652] D. Gayo-Avello, “No, you cannot predict elections with twitter,” *Internet Comput. IEEE*, vol. 16, no. 6, pp. 91–94, 2012.

[653] NIST Big Data Public Working Group: Definitions and Taxonomies Subgroup, “NIST Big Data

Interoperability Framework: Volume 2, Big Data Taxonomies," Gaithersburg, MD, Oct. 2015.

[654] L. Garber, "Using In-Memory Analytics to Quickly Crunch Big Data," *Computer (Long. Beach. Calif).*, vol. 45, no. 10, pp. 16–18, Oct. 2012.

[655] N. Marz and J. Warren, *Big Data: Principles and best practices of scalable real-time data systems*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2015.

[656] R. Agerri, X. Artola, Z. Beloki, G. Rigau, and A. Soroa, "Big data for Natural Language Processing: A streaming approach," *Knowledge-Based Syst.*, vol. 79, pp. 36–42, May 2015.

[657] M. K. McKusick and S. Quinlan, "GFS: Evolution on Fast-forward," *Queue*, vol. 7, no. 7, p. 10, Aug. 2009.

[658] J. Talbot, R. M. Yoo, and C. Kozyrakis, "Phoenix++: Modular MapReduce for Shared-Memory Systems," in *MapReduce '11 Proceedings of the second international workshop on MapReduce and its applications*, 2011, pp. 9–16.

[659] The Apache Software Foundation, "Hadoop Releases." [Online]. Available: https://archive.apache.org/dist/hadoop/core/. [Accessed: 01-Jul-2016].

[660] Amazon Web Services Inc., "Amazon Elastic MapReduce," 2016. [Online]. Available: https://aws.amazon.com/elasticmapreduce/. [Accessed: 01-Jan-2016].

[661] Microsoft Azure, "HDInsight," 2016. [Online]. Available: https://azure.microsoft.com/en-us/services/hdinsight/. [Accessed: 04-Jul-2016].

[662] Infochimps, "Infochimps," 2016. [Online]. Available: http://www.infochimps.com/infochimps-cloud/overview/. [Accessed: 04-Jul-2016].

[663] IBM, "IBM Big Insights." [Online]. Available: http://www.ibm.com/analytics/us/en/technology/biginsights/. [Accessed: 04-Jul-2016].

[664] Amazon Web Services Inc., "Amazon S3," 2016. [Online]. Available: https://aws.amazon.com/es/s3/. [Accessed: 04-Jul-2016].

[665] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed Data-Parallel Programs from Sequential Building Building Blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 - EuroSys '07*, 2007, p. 59.

[666] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, and J. Currey, "DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language," in *OSDI'08 Eighth Symposium on Operating Systems Design and Implementation*, 2008, pp. 1–14.

[667] Microsoft, "LINQ," *Microsoft MSDN*, 2016. [Online]. Available: https://msdn.microsoft.com/es-es/library/bb397926.aspx. [Accessed: 05-Jul-2016].

[668] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, "SCOPE: easy and efficient parallel processing of massive data sets," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1265–1276, Aug. 2008.

[669] Cloudera Inc., “Cloudera,” 2016. [Online]. Available: http://www.cloudera.com/. [Accessed: 05-Jul-2016].

[670] N. Leavitt, “Will NoSQL Databases Live Up to Their Promise?,” *Computer (Long. Beach. Calif).*, vol. 43, no. 2, pp. 12–14, Feb. 2010.

[671] M. Stonebraker, “SQL databases v. NoSQL databases,” *Commun. ACM*, vol. 53, no. 4, p. 10, Apr. 2010.

[672] Jing Han, Haihong E, Guan Le, and Jian Du, “Survey on NoSQL database,” in *2011 6th International Conference on Pervasive Computing and Applications*, 2011, pp. 363–366.

[673] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon’s Highly Available Key-value Store,” in *SOSP ’07 Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, 2007, vol. 41, no. 6, p. 205.

[674] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A Distributed Storage System for Structured Data,” *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 1–26, Jun. 2008.

[675] A. Lakshman and P. Malik, “Cassandra- A Decentralized Structured Storage System,” in *ACM SIGOPS Operating Systems Review*, 2010, vol. 44, no. 2, pp. 35–40.

[676] A. Mostosi, “The Big-Data Ecosystem Table,” 2016. .

[677] A. Thusoo, J. Sen Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, “Hive - a warehousing solution over a map-reduce framework,” *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1626–1629, 2009.

[678] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, “Pig latin: A Not-So-Foreign Language for Data Processing,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD ’08*, 2008, p. 1099.

[679] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, R. Benjaminn, S. Srinavasan, and U. Srivastava, “Building a High-Level Dataflow System on top of Map-Reduce: The Pig Experience,” *Proc. VLDB EndowmentVldb ’09*, pp. 1–12, 2009.

[680] Apache Software Foundation, “Apache Impala,” 2016. [Online]. Available: http://impala.io/. [Accessed: 07-Jul-2016].

[681] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, “Interpreting the Data: Parallel Analysis with Sawzall,” *Sci. Program.*, vol. 13, no. 4, pp. 277–298, 2005.

[682] Apache Software Foundation, “Apache Sqoop,” 2016. [Online]. Available: http://sqoop.apache.org/. [Accessed: 07-Jul-2016].

[683] Apache Software Foundation, “Apache Flume,” 2016. .

[684] Apache Software Foundation, “Chukwa,” 2016. [Online]. Available: http://chukwa.apache.org/.

[Accessed: 07-Jul-2016].

[685] A. Rabkin and R. Katz, “Chukwa: A System for Reliable Large-Scale Log Collection,” in *LISA ’10: 24th Large Installation System Administration Conference*, 2010, pp. 163–177.

[686] Apache Software Foundation, “Apache Oozie,” 2016. [Online]. Available: http://oozie.apache.org/. [Accessed: 07-Jul-2016].

[687] Apache Software Foundation, “Hue,” 2016. [Online]. Available: http://gethue.com/. [Accessed: 07-Jul-2016].

[688] Apache Software Foundation, “Apache Ambari,” 2016. [Online]. Available: http://ambari.apache.org/. [Accessed: 07-Jul-2016].

[689] Apache Software Foundation, “Apache Zookeeper,” 2016. [Online]. Available: https://zookeeper.apache.org/. [Accessed: 07-Jul-2016].

[690] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, “ZooKeeper: Wait-free Coordination for Internet-scale Systems,” in *USENIX Annual Technical Conference*, 2010, vol. 8, pp. 11–11.

[691] Apache Software Foundation, “Apache Mahout,” 2016. [Online]. Available: http://mahout.apache.org/. [Accessed: 07-Jul-2016].

[692] G. Malewicz, M. H. Austern, A. J. . Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, “Pregel: A System for Large-Scale Graph Processing Grzegorz,” in *Proceedings of the 2010 international conference on Management of data - SIGMOD ’10*, 2010, p. 135.

[693] Z. Richardson, “Goldenorb,” 2011. [Online]. Available: https://github.com/jzachr/goldenorb. [Accessed: 06-Jul-2016].

[694] “Apache Giraph,” 2011. [Online]. Available: http://giraph.apache.org/. [Accessed: 06-Jul-2016].

[695] A. Suresh, “Phoebus,” 2010. [Online]. Available: https://github.com/xslogic/phoebus. [Accessed: 06-Jul-2016].

[696] Apache Software Foundation, “Apache Avro,” 2016. [Online]. Available: http://avro.apache.org/. [Accessed: 07-Jul-2016].

[697] Apache Software Foundation, “Apache Spark,” 2016. [Online]. Available: http://spark.apache.org/. [Accessed: 07-Jul-2016].

[698] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster Computing with Working Sets,” in *HotCloud’10 Proceedings of the 2nd USENIX conference on Hot topics in Cloud Computing*, 2010, p. 10.

[699] Apache Software Foundation, “Apache Tez,” 2016. [Online]. Available: http://tez.apache.org/. [Accessed: 07-Jul-2016].

[700] Apache Software Foundation, “Apache Storm,” 2016. [Online]. Available: http://storm.apache.org/. [Accessed: 07-Jul-2016].

[701] Apache Software Foundation, “Apache Phoenix,” 2016. [Online]. Available:

https://phoenix.apache.org/. [Accessed: 07-Jul-2016].

[702] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed Stream Computing Platform," in *2010 IEEE International Conference on Data Mining Workshops*, 2010, pp. 170–177.

[703] R Development Core Team, "R: A Language and Environment for Statistical Computing," *R Found. Stat. Comput.*, 2011.

[704] G. Piatetsky-Shapiro, "R, Python Duel As Top Analytics, Data Science software – KDnuggets 2016 Software Poll Results," 2016. [Online]. Available: http://www.kdnuggets.com/2016/06/r-python-top-analytics-data-mining-data-science-software.html. [Accessed: 07-Jul-2016].

[705] The Open Web Application Security Project, *A Guide to Building Secure Web Applications and Web Services*, 2nd ed. 2005.

[706] Cloud Security Alliance, "Expanded Top Ten Big Data Security and Privacy Challenges," 2013.

[707] K. Vamsee, "Solid State Drive vs. Hard Disk Drive: Price and Performance Study," 2011.

[708] NIST Big Data Public Working Group: Use Cases and Requirements Subgroup, "NIST Big Data Interoperability Framework: Volume 3, Use Cases and General Requirements," Gaithersburg, MD, Oct. 2015.

[709] E. Gilbert and K. Karahalios, "Widespread Worry and the Stock Market," in *Proceedings of the 4th International AAAI Conference on Weblogs and Social Media*, 2010, pp. 58–65.

[710] J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *J. Comput. Sci.*, vol. 2, no. 1, pp. 1–8, Mar. 2011.

[711] S. Ansolabehere and E. Hersh, "Validation: What Big Data Reveal About Survey Misreporting and the Real Electorate," *Polit. Anal.*, vol. 20, no. 4, pp. 437–459, Oct. 2012.

[712] Y. Shimshoni, N. Efron, and Y. Matias, "On the Predictability of Search Trends," *Google Res. Blog*, Oct. 2009.

[713] P. M. Polgreen, Y. Chen, D. M. Pennock, and F. D. Nelson, "Using Internet Searches for Influenza Surveillance," *Clin. Infect. Dis.*, vol. 47, no. 11, pp. 1443–1448, Dec. 2008.

[714] J. Ginsberg, M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant, "Detecting influenza epidemics using search engine query data," *Nature*, vol. 457, no. 7232, pp. 1012–1014, Feb. 2009.

[715] V. Lampos, T. De Bie, and N. Cristianini, "Flu Detector - Tracking Epidemics on Twitter," in *Machine Learning and Knowledge Discovery in Databases*, J. L. Balcázar, F. Bonchi, A. Gionis, and M. Sebag, Eds. Springer Berlin Heidelberg, 2010, pp. 599–602.

[716] M. E. J. Newman and R. M. Ziff, "A fast Monte Carlo algorithm for site or bond percolation," *Phys. Rev. E*, vol. 64, no. 1, p. 16706, Jan. 2001.

[717] P. Boldi, M. Rosa, and S. Vigna, "HyperANF: approximating the neighbourhood function of very large graphs on a budget," in *Proceedings of the 20th international conference on World wide web* -

WWW '11, 2011, p. 625.

[718] M. J. Matarić, "Situated robotics," *Encycl. Cogn. Sci.*, pp. 25–30, 2002.

[719] M. Hillman, "Rehabilitation robotics from past to present - a historical perspective," in *The Eighth International Conference on Rehabilitation Robotics*, 2003, p. 4.

[720] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Gálvez-López, K. Häussermann, R. Janssen, J. M. M. Montiel, A. Perzylo, B. Schießle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "RoboEarth," *IEEE Robot. Autom. Mag.*, vol. 18, no. 2, pp. 69–82, Jun. 2011.

[721] I. Dutta, A. D. Bogobowicz, and J. J. Gu, "Collective robotics - a survey of control and communication techniques," in *2004 International Conference on Intelligent Mechatronics and Automation, 2004. Proceedings.*, 2004, no. August, pp. 505–510.

[722] E. Garcia, M. A. Jimenez, P. Gonzalez De Santos, and M. Armada, "The evolution of robotics research," *IEEE Robot. Autom. Mag.*, vol. 14, no. 1, pp. 90–103, Mar. 2007.

[723] W. Ju and L. Takayama, "Should robots or people do these jobs? A survey of robotics experts and non-experts about which jobs robots should do," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 2452–2459.

[724] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses," *AI Mag.*, vol. 29, no. 1, p. 9, 2008.

[725] E. Guizzo, "Three Engineers, Hundreds of Robots, One Warehouse," *IEEE Spectr.*, vol. 45, no. 7, pp. 26–34, Jul. 2008.

[726] E. Guizzo, "Cloud Robotics: Connected to the Cloud, Robots Get Smarter," *IEEE Spectrum*, 2011. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/robotics-software/cloud-robotics. [Accessed: 20-Aug-2016].

[727] G. Hu, W. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *IEEE Netw.*, vol. 26, no. 3, pp. 21–28, May 2012.

[728] K. Watanabe and K. Izumi, "A survey of robotic control systems constructed by using evolutionary computations," in *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, 1999, vol. 2, pp. 758–763.

[729] V. Patidar and R. Tiwari, "Survey of robotic arm and parameters," in *2016 International Conference on Computer Communication and Informatics (ICCCI)*, 2016, pp. 1–6.

[730] Z. Lu, A. Chauhan, F. Silva, and L. S. Lopes, "A brief survey of commercial robotic arms for research on manipulation," in *2012 IEEE Symposium on Robotics and Applications (ISRA)*, 2012, pp. 986–991.

[731] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed. Pearson Education International, 2004.

[732] P. Dario, E. Guglielmelli, B. Allotta, and M. C. Carrozza, "Robotics for medical applications," *IEEE*

Robot. Autom. Mag., vol. 3, no. 3, pp. 44–56, 1996.

[733] E. Nakano, “First approach to the development of the patient care robot,” in *Proc. 11th International Symposium of Industrial Robots*, 1981, pp. 87–94.

[734] Y. Kim and A. M. Cook, *Manipulation and Mobility Aids*. London, U.K: Chapman and Hall, 1985.

[735] M. Topping, “The development of Handy 1, a robotic aid to independence for the severely disabled,” in *Mechatronic Aids for the Disabled, IEE Colloquium on*, 1995, vol. 1995, pp. 2–2.

[736] J. C. Rosier, J. A. van Woerden, L. W. van der Kolk, H. H. Kwee, J. J. Duimel, J. J. Smits, A. A. Tuinhof de Moed, G. Honderd, and P. M. Bruyn, “The MANUS wheelchair-mounted manipulator: system design and implementation,” in *Proceedings of the 20th International Symposium on Industrial Robots*, 1989, pp. 443–450.

[737] H. F. M. Van der Loos, S. J. Michalowski, and L. J. Lefter, “Design of an Omnidirectional Mobile Robot as a Manipulation Aid for the Severely Disabled,” in *Interactive Robotic Aids*, New York, New York, USA, 1986, pp. 61–63.

[738] W. S. Harwin, A. Ginige, and R. D. Jackson, “A Potential Application in Early Education and a Possible Role for a Vision System in a Workstation Based Robotic Aid for Physically Disabled Persons,” in *Interactive Robotic Aids--One Option for Independent Living: An International Perspective*, R. Foulds, Ed. 1986, pp. 18–23.

[739] A. Meade, “Dexter--A finger-spelling hand for the deaf-blind,” in *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, 1987, vol. 4, pp. 1192–1195.

[740] L. Pedersen, D. Kortenkamp, D. Wettergreen, and I. Nourbakhsh, “A Survey of Space Robotics,” *7th Int. Symp. Artif. Intelligent, Robot. Autom. Sp.*, p. 8, 2003.

[741] J. Yuh, “Design and Control of Autonomous Underwater Robots: A Survey,” *Auton. Robots*, vol. 8, no. 1, pp. 7–24, 2000.

[742] T. S. Tadele, T. de Vries, and S. Stramigioli, “The Safety of Domestic Robotics: A Survey of Various Safety-Related Publications,” *IEEE Robot. Autom. Mag.*, vol. 21, no. 3, pp. 134–142, Sep. 2014.

[743] T. Geng, B. Porr, and F. Wörgötter, “Coupling of neural computation with physical computation for stable dynamic biped walking control,” 2005.

[744] C. Orlowski, “Legged Squad Support System (LS3),” *Defense Advanced Research Projects Agency*, 2012. [Online]. Available: http://www.darpa.mil/program/legged-squad-support-system. [Accessed: 29-Aug-2016].

[745] P. Gonzalez de Santos, M. A. Armada, and M. A. Jimenez, “Ship building with ROWER,” *IEEE Robot. Autom. Mag.*, vol. 7, no. 4, pp. 35–43, Dec. 2000.

[746] R. Molfino, M. Armada, F. Cepolina, and M. Zoppi, “Roboclimber the 3 ton spider,” *Ind. Robot An Int. J.*, vol. 32, no. 2, pp. 163–170, Apr. 2005.

[747] M. Gavrilović, M. Marić, and M. Vukobratović, “An approach to the synthesis of lower-extremity

control," in *External control of human extremities: Proc. Internat. Sympos. Yugoslav Committee for Electronics and Automation, Belgrade*, 1967, pp. 206–211.

[748] M. Vukobratović and D. Juricic, "Contribution to the Synthesis of Biped Gait," *IEEE Trans. Biomed. Eng.*, vol. BME-16, no. 1, pp. 1–6, Jan. 1969.

[749] M. Vukobratović, A. A. Frank, and D. Juricic, "On the Stability of Biped Locomotion," *IEEE Trans. Biomed. Eng.*, vol. BME-17, no. 1, pp. 25–36, Jan. 1970.

[750] A. A. Frank and M. Vukobratović, "On the synthesis of biped locomotion machines," in *8th International Conference on Medical and Biological Enginereing*, 1969.

[751] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. Autom.*, vol. 2, no. 1, pp. 14–23, Jun. 1986.

[752] B. P. Gerkey and M. J. Matarić, "Sold!: auction methods for multirobot coordination," *IEEE Trans. Robot. Autom.*, vol. 18, no. 5, pp. 758–768, Oct. 2002.

[753] L. E. Parker, "ALLIANCE: an architecture for fault tolerant multirobot cooperation," *IEEE Trans. Robot. Autom.*, vol. 14, no. 2, pp. 220–240, Apr. 1998.

[754] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss, "Self Organizing Robots Based on Cell Structures - CKBOT," in *IEEE International Workshop on Intelligent Robots*, 1988, pp. 145–150.

[755] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos, "Cloud Robotics: Current Status and Open Issues," *IEEE Access*, no. August, pp. 1–1, 2016.

[756] IEEE Society of Robotics and Automation's Technical Committee, "Networked Robots," 2015. [Online]. Available: http://www-users.cs.umn.edu/~isler/tc/. [Accessed: 29-Aug-2016].

[757] J. Kuffner, "Cloud-enabled robots," in *IEEE-RAS international conference on humanoid robotics*, 2010.

[758] I. Asimov, *Runaround*. USA: Street & Smith, 1942.

[759] I. Asimov, *Robots and Empire*, 1st ed. USA, 1985.

[760] RoboEarth, "What Is RoboEarth?," 2013. [Online]. Available: http://roboearth.org/. [Accessed: 20-Aug-2016].

[761] Ching-Hu Lu and Li-Chen Fu, "Robust Location-Aware Activity Recognition Using Wireless Sensor Network in an Attentive Home," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 4, pp. 598–609, Oct. 2009.

[762] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, and S. Kawatsuma, "Emergency response to the nuclear accident at the Fukushima Daiichi Nuclear Power Plants using mobile rescue robots," *J. F. Robot.*, vol. 30, no. 1, pp. 44–63, Jan. 2013.

[763] M. S. Alam, I. A. Jamil, K. Mahmud, and N. Islam, "Design and implementation of a RF controlled robotic environmental survey assistant system," in *16th Int'l Conf. Computer and Information Technology*, 2014, no. March, pp. 438–442.

[764] K. Pfeiffer, M. Bengel, and A. Bubeck, “Offshore robotics - Survey, implementation, outlook,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 241–246.

[765] W. F. Lages and V. M. de Oliveira, “A survey of applied robotics for the power industry in Brazil,” in *2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI)*, 2012, pp. 78–82.

[766] M. F. M. Campos, G. A. S. Pereira, S. R. C. Vale, A. Q. Bracarense, G. A. Pinheiro, and M. P. Oliveira, “A robot for installation and removal of aircraft warning spheres on aerial power transmission lines,” *IEEE Trans. Power Deliv.*, vol. 18, no. 4, pp. 1581–1582, Oct. 2003.

[767] C. Mello Jr., E. M. Gonçalves, E. Estrada, G. Oliveira, H. Souto Jr., R. Almeida, S. Botelho, T. Santos, and V. Oliveira, “TATUBOT – Robotic System for Inspection of Undergrounded Cable System,” in *2008 IEEE Latin American Robotic Symposium*, 2008, pp. 170–175.

[768] O. Vermesan and P. Friess, *Internet of things: converging technologies for smart environments and integrated ecosystems*. River Publishers, 2013.

[769] E. Palacios-González, H. Fernández-Fernández, V. García-Díaz, B. C. P. G-Bustelo, and J. M. C. Lovelle, “A review of intelligent software development tools,” *Proc. 2008 Int. Conf. Artif. Intell. ICAI 2008 Proc. 2008 Int. Conf. Mach. Learn. Model. Technol. Appl.*, no. April 2016, pp. 585–590, 2008.

[770] M. Voelter, “A Catalog of Patterns for Program Generation,” in *EuroPLoP*, 2003, pp. 285–320.

[771] The US National Intelligence Council, “Disruptive Civil Technologies: Six Technologies with Potential Impacts on US Interests out to 2025,” Apr, 2008.

[772] J. Bohn, V. Coroamă, M. Langheinrich, F. Mattern, and M. Rohs, “Living in a World of Smart Everyday Objects—Social, Economic, and Ethical Implications,” *Hum. Ecol. Risk Assess. An Int. J.*, vol. 10, no. 5, pp. 763–785, 2004.

[773] Twitter Inc., “Twitter,” 2016. [Online]. Available: https://twitter.com/. [Accessed: 27-Apr-2016].

[774] Facebook, “Facebook,” 2016. [Online]. Available: https://www.facebook.com/. [Accessed: 26-Apr-2016].

[775] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[776] A. Pintus, D. Carboni, A. Piras, A. Giordano, and I. Elettrica, “Building the Web of Things with WS-BPEL and Visual Tags Web of Things using Service-oriented Architecture standards,” in *The Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010)*, 2010, no. c, pp. 357–360.

[777] D. Guinard, V. Trifa, T. Pham, and O. Liechti, “Towards physical mashups in the Web of Things,” in *2009 Sixth International Conference on Networked Sensing Systems (INSS)*, 2009, pp. 1–4.

[778] Eclipse, “Ecore Tools,” 2014. [Online]. Available: https://wiki.eclipse.org/Ecore_Tools. [Accessed: 28-Sep-2016].

Anexo IV: referencias

[779] D. Ho, "Notepad++," 2016. [Online]. Available: http://notepad-plus-plus.org/. [Accessed: 25-Jul-2016].

[780] Blacksun Software, "Mousotron," 2016. [Online]. Available: http://www.blacksunsoftware.com/mousotron.html. [Accessed: 25-Jul-2016].

[781] R. Likert, "A technique for the measurement of attitudes," *Arch. Psychol.*, vol. 22, pp. 1–55, 1932.

[782] B. A. Kitchenham and S. L. Pfleeger, "Principles of survey research part 2," *ACM SIGSOFT Softw. Eng. Notes*, vol. 27, no. 1, pp. 18–20, Jan. 2002.

[783] R. M. Groves, F. J. Fowler, M. P. Couper, J. M. Lepkowski, E. Singer, and R. Tourangeau, *Survey Methodology*. Wiley-Interscience, 2004.

[784] B. Kaucic and T. Asic, "Improving introductory programming with Scratch?," in *2011 Proceedings of the 34th International Convention MIPRO*, 2011, pp. 1095–1100.

[785] Besttoolbars, "AppsGeyser," 2011. [Online]. Available: http://www.appsgeyser.com. [Accessed: 26-Jul-2016].

[786] iBuildApp Inc., "iBuildApp," 2011. [Online]. Available: http://ibuildapp.com/. [Accessed: 26-Jul-2016].

[787] Indigo Rose Software, "Andromo," 2016. [Online]. Available: http://www.andromo.com. [Accessed: 26-Jul-2016].

[788] Paperlit S.R.L., "AppsBuilder," 2015. [Online]. Available: http://www.apps-builder.com. [Accessed: 26-Jul-2016].

[789] Infinite Monkeys, "Infinite Monkeys," 2016. [Online]. Available: http://www.infinitemonkeys.mobi. [Accessed: 26-Jul-2016].

[790] Appy Pie, "Appypie," 2013. [Online]. Available: http://www.appypie.com. [Accessed: 26-Jul-2016].

[791] Como Ltd., "Como," 2015. [Online]. Available: http://my.como.com. [Accessed: 26-Jul-2016].

[792] AppMachine, "AppMachine," 2011. [Online]. Available: http://www.appmachine.com. [Accessed: 26-Jul-2016].

[793] Lifelong Kindergarten, "Scratch," 2015. [Online]. Available: https://scratch.mit.edu/. [Accessed: 05-Dec-2016].

[794] S. Garner, "Learning to program from scratch," in *Advanced Learning Technologies, 2009. ICALT 2009. Ninth IEEE International Conference on*, 2009, pp. 451–452.

[795] M. Kasunic, *Designing an effective survey*. Pittsburgh, 2005.

[796] B. Schneier, "Schneier on Security: Cryptanalysis of SHA-1," *Schneier.com*, 2005. [Online]. Available: https://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html. [Accessed: 09-Aug-2016].

[797] M. Stevens, "Fast collision attack on MD5," *IACR ePrint Arch. Rep.*, vol. 104, p. 17, 2006.

[798] F. Mendel, T. Nad, and M. Schläffer, "Improving Local Collisions: New Attacks on Reduced SHA-256," in *Advances in Cryptology – EUROCRYPT 2013*, T. Johansson and P. Q. Nguyen, Eds. Athens, Greece: Springer Berlin Heidelberg, 2013, pp. 262–278.

[799] R. L. Rivest, "OFFICIAL COMMENT: MD6," 2009. [Online]. Available: http://groups.csail.mit.edu/cis/md6/OFFICIAL_COMMENT_MD6_2009-07-01.txt. [Accessed: 07-Aug-2016].

[800] N. Sklavos and O. Koufopavlou, "On the hardware implementations of the SHA-2 (256, 384, 512) hash functions," in *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, 2003, vol. 5, pp. 153–156.

[801] National Institute of Standards and Technology, "FIPS PUB 180-2 (with Change Notice 1)," 2008.

[802] M. Eichlseder, F. Mendel, and M. Schläffer, "Branching Heuristics in Differential Collision Search with Applications to SHA-512," in *Fast Software Encryption*, vol. 8540, C. Cid and C. Rechberger, Eds. London, UK: Springer Berlin Heidelberg, 2015, pp. 473–488.

[803] J. Guo, S. Ling, C. Rechberger, and H. Wang, "Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6477 LNCS, 2010, pp. 56–75.

[804] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *Advances in Cryptology – EUROCRYPT 2005*, R. Cramer, Ed. Aarhus, Denmark: Springer Berlin Heidelberg, 2005, pp. 19–35.

[805] M. Stevens, A. Lenstra, and B. de Weger, "Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities," in *Advances in Cryptology – Eurocrypt 2007*, vol. 4515, no. Lecture Notes in Computer Science, M. Naor, Ed. Barcelona, Spain: Springer Berlin Heidelberg, 2007, pp. 1–22.

[806] X. Wang, H. Yu, and Y. L. Yin, "Efficient Collision Search Attacks on SHA-0," in *Advances in Cryptology – CRYPTO 2005*, no. 90304009, V. Shoup, Ed. Santa Barbara, California, USA: Springer Berlin Heidelberg, 2005, pp. 1–16.

[807] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang, "Preimages for Step-Reduced SHA-2," in *Asiacrypt*, vol. 5912, M. Matsui, Ed. Springer Berlin Heidelberg, 2009, pp. 578–597.

[808] S. K. Sanadhya and P. Sarkar, "New Collision Attacks against Up to 24-Step SHA-2," in *Progress in Cryptology - INDOCRYPT 2008*, vol. 5365 LNCS, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 91–103.

[809] D. Khovratovich, C. Rechberger, and A. Savelieva, "Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family," in *Fast Software Encryption*, A. Canteaut, Ed. Springer Berlin Heidelberg, 2012, pp. 244–263.

[810] M. Lamberger and F. Mendel, "Higher-Order Differential Attack on Reduced SHA-256," *IACR Cryptol. ePrint Arch.*, vol. 37, pp. 1–16, 2011.

[811] C. Dobraunig, M. Eichlseder, and F. Mendel, “Analysis of SHA-512/224 and SHA-512/256,” in *Asiacrypt 2015*, T. Iwata and J. H. Cheon, Eds. Auckland, New Zealand: Springer Berlin Heidelberg, 2015, pp. 612–630.

[812] F. Foerster, M. Smeja, and J. Fahrenberg, “Detection of posture and motion by accelerometry: a validation study in ambulatory monitoring,” *Comput. Human Behav.*, vol. 15, no. 5, pp. 571–583, Sep. 1999.

[813] C. J. James and S. Kumar, “Detection of posture and motion by accelerometer sensors,” *2011 3rd Int. Conf. Electron. Comput. Technol.*, vol. 4, pp. 369–371, Apr. 2011.

[814] J. Mantyjarvi, J. Himberg, and T. Seppanen, “Recognizing human motion with multiple acceleration sensors,” *2001 IEEE Int. Conf. Syst. Man Cybern. e-Systems e-Man Cybern. Cybersp.*, vol. 2, pp. 747–752, 2001.

[815] M. Agrawal and P. Mishra, “A comparative survey on symmetric key encryption techniques,” *Intern. J. Comput. Sci. Eng.*, vol. 4, no. 5, pp. 877–882, 2012.

[816] B. Schneier, *Applied cryptography: Protocols, algorithms, and source code in C*, 2nd ed. Wiley, 1996.

[817] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, “Edge-centric Computing: Vision and Challenges,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015.

[818] A. Bulling, J. A. Ward, H. Gellersen, and G. Tröster, “Robust Recognition of Reading Activity in Transit Using Wearable Electrooculography,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5013 LNCS, 2008, pp. 19–37.

[819] T. H. Davenport, “Saving IT’s Soul: Human-Centered Information Management,” 1994. [Online]. Available: https://hbr.org/1994/03/saving-its-soul-human-centered-information-management. [Accessed: 16-Aug-2016].

[820] M. E. D. Koenig, “What is KM? Knowledge Management Explained,” *KMWorld*, 2012. [Online]. Available: http://www.kmworld.com/Articles/Editorial/What-Is-.../What-is-KM-Knowledge-Management-Explained-82405.aspx. [Accessed: 16-Aug-2016].

[821] G. Beni and J. Wang, “Swarm Intelligence in Cellular Robotic Systems,” in *Robots and Biological Systems: Towards a New Bionics?*, P. Dario, G. Sandini, and P. Aebischer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 703–712.

[822] N. Asokan, V. Niemi, and K. Nyberg, “Man-in-the-Middle in Tunnelled Authentication Protocols,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3364 LNCS, B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, Eds. Springer Berlin Heidelberg, 2005, pp. 28–41.

[823] P. A. R. Rojas, “Mecatrónica - Revolución para el siglo XXI,” *Met. Actual*, pp. 46–53.

[824] L. E. Welicki, “Meta-Especificación y Catalogación de Patrones de Software con Lenguajes de

Dominio Específico y Modelos de Objetos Adaptativos: Una Vía para la Gestión del Conocimiento en la Ingeniería del Software," Universidad Pontificia de Salamanca, 2006.

[825] Intel Corporation, "RAID 0,1,5,10,Matriz RAID,Listo para RAID para la tecnología de almacenamiento rápido Intel®," 2016. [Online]. Available: http://www.intel.la/content/www/xl/es/support/boards-and-kits/000005867.html. [Accessed: 02-Oct-2016].

«La frase más emocionante que se escucha en la ciencia, la que anuncia nuevos descubrimientos,
no es "¡Eureka!" sino "Fue divertido"»
Isaac Asimov

«Tener un final feliz depende, por supuesto,
de donde quieras que acabe tu historia»
Orson Welles