UNIVERSIDAD DE OVIEDO

ESCUELA DE INGENIERÍA INFORMÁTICA

PROYECTO FIN DE MASTER

"SARP: Plataforma de aplicaciones de Realidad Aumentada en colaboración con Objetos Inteligentes"

DIRECTORES: Vicente García Díaz, Cristina Pelayo García Bustelo

AUTOR: Sergio González Amigo

Vº Bº del Director del Proyecto

RESUMEN

La Realidad Aumentada (RA) suplementa la realidad añadiendo objetos virtuales a la realidad captada por las personas. Mediante un dispositivo, el usuario puede obtener la información digital asociada a los objetos reales, ya sea calculando la posición de este o analizando determinados patrones en las imágenes capturadas por el dispositivo y el usuario. Este dispositivo puede ser un teléfono inteligente o Smartphone, que dispone de las herramientas necesarias para realizar esos cálculos, como la cámara, el GPS, WIFI, la brújula o el acelerómetro y que ha sido el principal responsable de hacer esta tecnología accesible para la mayoría del público debido a su gran popularidad.

Un aspecto interesante de esta tecnología es que simplifica considerablemente la selección de la información en la que el usuario está interesado. Si por ejemplo un usuario desea obtener el número de teléfono o la dirección de una web de un anuncio que se encontraba en la calle, lo puede obtener simplemente apuntando con su Smartphone hacia el anuncio en cuestión. Del mismo modo, un turista podría obtener información de un monumento haciendo que el edificio aparezca en la pantalla de su dispositivo móvil.

Esta característica podría ser utilizada en otra tecnología cada vez más popular, Internet de las Cosas (Internet of Things o IoT). Con un número cada vez mayor de los principales dispositivos que conforman IoT, los Objetos Inteligentes (o Smart Objects), acceder a la información que estos proporcionan puede resultar una tarea compleja. La RA podría en este caso solucionar este problema proporcionando interfaces de usuario gráficas a los dispositivos, disponibles cuando el usuario apunta con su dispositivo hacia los objetos inteligentes.

Sin embargo, las actuales plataformas para crear contenido de RA presentan un problema o limitación que impide elaborar aplicaciones de RA que interactúen correctamente con objetos inteligentes. Este problema se corresponde con que las plataformas no permiten definir el comportamiento de los objetos virtuales (representantes de los objetos reales en las aplicaciones de RA). Esto supone que la interacción con el usuario se limita a mostrarle

información, reduciendo las acciones que este pueda realizar en la aplicación. Esto es suficiente en algunas ocasiones, pero en el caso de los objetos inteligentes se perdería una de las mayores ventajas de estos, que es controlarlos a través de una red de forma remota.

Por ese motivo, en esta investigación se presenta la plataforma SARP, una propuesta basada en Ingeniería Dirigida por Modelos (Model Driven Engineering o MDE) que permite crear aplicaciones de RA y definir el comportamiento de los objetos virtuales mediante los editores textual o gráfico propuestos, combinando ambas tecnologías de un modo sencillo y permitiendo a cualquiera, aunque no posea conocimientos de programación, crear aplicaciones de este tipo.

PALABRAS CLAVE

Realidad Aumentada, Objeto virtual, Internet de las cosas, Objeto inteligente, Teléfono inteligente, Ingeniería Dirigida por Modelos.

ABSTRACT

Augmented Reality (AR) supplements real adding virtual objects to the reality captured by persons. A user can obtain information related to virtual objects through a device, either calculating the position or analyzing patterns in images captured by device and user. This device can be a Smartphone, which has the needed tools to make these calculations, like the camera, GPS, WIFI, compass or accelerometer and also, it has been the main responsible to make this technology available for most of the public due to its great popularity.

An interesting aspect of this technology is that simplifies the user the selection of information that is interested in. For example, if a user wishes to obtain the phone number or a web from a street advert, he just can get it by pointing with his smartphone to such advert. In the same way, a tourist could obtain information of a monument making the building display on his screen smartphone.

This feature could be used in other technology that is increasing its popularity, Internet of Things (IoT). More devices which conforms this technology, smart objects, are available every day, and select the information provided by each of this objects can be an important complex task. AR could solve, in this case, this problem providing graphical user interfaces to these devices, available when users points with their AR devices to smart objects.

However, current platforms to create AR content have an important gap or limitation between AR and IoT that impede created applications to interact correctly with smart objects. This gap corresponds with the limitation of platforms to define the behavior of virtual objects (representatives of the real ones). It means that interaction with users is limited to show them information, reducing their actions made it on applications. This is enough in some situations, but representing smart objects, the main advantage of these would be lost, that is control them remotely through a network.

For this reason, in this research, SARP platform is presented, an approach based on Model Driven Engineering (MDE) which allows to create AR apps and define the behavior of virtual objects with either textual or graphical editors, combining both technologies in a sim-

ple way, allowing anyone, even if he or she haven't programing skills, to create this kind of applications.

KEYWORDS

Augmented Reality, Virtual Object, Internet of Things, Smart Object, Smartphone, Model Driven Engineering.

ÍNDICE

Introducción 14

Motivación 18

Contribución 21

Posibles ámbitos 22

Estado de la situación actual 24

1. Ingeniería dirigida por modelos 24
2. Arquitectura dirigida por modelos 25
   - 2.1. Modelo de computación independiente (CIM) 26
   - 2.2. Modelo de plataforma independiente (PIM) 26
   - 2.3. Modelo de plataforma específica (PSM) 26
3. Realidad aumentada 27
4. Smart Objects 29
5. MDE aplicado en RA y IoT 30
6. Frameworks y plataformas de RA disponibles 31
   - 6.1. Guide-Me 31
   - 6.2. Wikitude SDK 33
   - 6.3. Zapcode 36
   - 6.4. Reaility editor 38

Propuesta: Plataforma SARP 42

7. Combinando Realidad aumentada y Objetos inteligentes 42
8. Arquitectura de SARP 44

9. Componentes 45
9.1. Editor textual (plugin para Eclipse) 46
9.2. Editor gráfico (web) 46
9.3. Servidor 46
9.4. Objetos inteligentes 46
9.5. Dispositivo de RA 47
10. Ingeniería dirigida por modelos en SARP 48
11. Lenguaje de Dominio Específico 50
11.1. Ejemplo de RA y Objetos inteligentes 52
12. Editor textual 57
13. Editor gráfico (aplicación web) 58
14. Descripción de las capas / procesos 63
14.1. Capa de edición o “Edition layer” 63
14.2. Capa de persistencia o “Persistence Layer” 64
14.3. Capa de comunicación o “Comunication Layer” 65
14.4. Capa de procesamiento o “Processor Layer” 66
14.5. Prototipo Android (Aplicación “exploradora”) 68
14.6. Prototipos de objetos inteligentes 69
Evaluación 71
15. Bombilla inteligente 71
16. Reproductor de Música 72
17. Cerradura inteligente 73
18. Interacción entre objetos virtuales 74
Conclusiones y trabajo futuro 75
Referencias 77

Índice de Figuras

Figura 1 Ejemplo proporcionado por el OMG de la distribución en capas en MDA 26
Figura 2 Imagen representativa de Internet of Things y algunos objetos inteligentes mostrada en "http://www.techpolicydaily.com/" 30
Figura 3 Metamodelos utilizados en GuideMe 32
Figura 4 Ejemplo de sistema de presentación de diapositivas con GuideMe 33
Figura 5 Imagen inicial utilizada en Wikitude Studio como patrón de reconocimiento 34
Figura 6 Componentes de ejemplo en una aplicación de Wikitude Studio 34
Figura 7 Aplicación cliente Android de Wikitude visualizando la aplicación de ejemplo elaborada con Wikitude Studio 35
Figura 8 Editor gráfico de la plataforma Zapcode 36
Figura 9 Componentes de la escena 1 de la aplicación de ejemplo elaborada con Zapcode 37
Figura 10 Aplicación cliente de Android de Zapcode visualizando la aplicación elaborada con el editor gráfico de la plataforma 38
Figura 11 La cerradura electrónica, actuando como objeto inteligente, permite desbloquear la puerta mediante una orden remota 39
Figura 12 El usuario introduce el código de seguridad para desbloquear la puerta a través de una interfaz de RA 39
Figura 13 Modelo de capas propuesto en la plataforma 45
Figura 14 Componentes principales de la plataforma 48
Figura 15 Metamodelo utilizado en SARP 49
Figura 16 Componentes de la aplicación "Music Player" 53
Figura 17 Diagrama de estados de la aplicación "Music Player" 54
Figura 18 Entorno de desarrollo para el DSL textual de SARP 58
Figura 19 Editor gráfico (aplicación y propiedades de los objetos virtuales) 59
Figura 20 Mapa de Google mostrado en la aplicación web para seleccionar la posición física del objeto virtual 60
Figura 21 Editor de los componentes del Objeto Virtual 61
Figura 22 Editor gráfico del comportamiento de un objeto virtual en SARP 62
Figura 23 Dialogo mostrado para definir una transición 62
Figura 24 Comunicación entre objetos inteligentes y el servidor 65
Figura 25 Ejemplo de control de estados en la capa de procesamiento en la aplicación "Music Player" 67
Figura 26 Aplicación Explorer mostrando las aplicaciones disponibles en SARP 68
Figura 27 Interfaz de usuario mostrada en la aplicación Music player 68
Figura 28 Diagrama de estados de la bombilla inteligente 71

Figura 29 Aplicación mostrada en el dispositivo de RA 72
Figura 30 Reproductor de Música mostrado desde el dispositivo de RA 73
Figura 31 Diagrama de estados para la cerradura inteligente 73
Figura 32 Cerradura inteligente antes (izquierda) y después (derecha) de que el usuario introduzca correctamente la contraseña (5-7-9) 74
Figura 33 Diagrama de estados mostrando la interacción entre objetos virtuales 74

INTRODUCCIÓN

Internet de las cosas (Internet of Things) hace referencia a cualquier objeto que a través de distintos mecanismos sea accesible desde diferentes redes (Atzori, Iera, and Morabito 2010). Esto permite obtener información de dispositivos independientemente de su ubicación física y tomar decisiones basadas en esa información, así como modificar los dispositivos para que realicen una acción determinada (Pascual Espada 2012). Esta tecnología está actualmente incrementando su popularidad debido en parte, al número de objetos inteligentes disponibles en red, siendo los smartphones unos de los más comunes (Thompson 2005), y a las aplicaciones de esta tecnología en distintas áreas como transporte y logística, asistencia médica, entornos inteligentes o dirigidas a un uso social entre otros (Atzori, Iera, and Morabito 2010).

Una de las ventajas de disponer de objetos a través de una red es, que además de disponer de su interfaz "física", se pueden manipular a través de otras aplicaciones o de otros objetos inteligentes. Esto proporciona nuevas formas de interacción, permitiendo, por ejemplo, interactuar con un objeto físico a través de una interfaz de usuario de una aplicación. Estos objetos inteligentes se comunican con su entorno y con otros objetos inteligentes, por lo que una de sus mayores ventajas es poder interactuar con otros sistemas, sin embargo, también es su objetivo el de proporcionar información a los usuarios interesados en ella. Disponiendo actualmente de un número cada vez mayor de estos objetos y siendo también más común la interacción entre una persona y un objeto inteligente, resulta interesante proporcionar una interfaz de usuario común y sencilla a esta tecnología.

El modelo de ciudades inteligentes (Schaffers et al. 2011) es un ejemplo en el que la interacción entre objetos inteligentes y personas puede ser muy habitual, especialmente si se mantiene la tendencia actual de adaptar las ciudades a este concepto. Las ciudades inteligentes basan su funcionamiento principalmente en el Internet de las cosas, y por tanto, en el uso de una gran variedad de dispositivos que permiten obtener información de la misma (Zanella et al. 2014). Esta información puede ser de gran utilidad desde un sistema centralizado que analice los datos y permita detectar cualquier inconveniente y decidir cuál es la solución más eficiente o adecuada, así como conocer el estado general de la ciudad

(Oliveira and Campolargo 2013). Guías turísticas interactivas, información sobre el transporte público en tiempo real, estado de los aparcamientos o servicios de negocios, como restaurantes, son algunos de los datos disponibles en una ciudad inteligente (Schaffers et al. 2011). Sin embargo, mostrar y hacer accesible toda esta información a los usuarios puede ser una tarea compleja (Oliveira y Campolargo 2013).

El primer inconveniente aparece en el momento en el que el usuario desea seleccionar aquellos datos en los que está interesado (Heun, Kasahara, and Maes 2013). Si una persona quisiera consultar el menú de un restaurante, o si dispone de mesas libres, a través de internet, en el mejor de los casos, el usuario contaría con una aplicación con los datos de la ciudad, donde debería seleccionar en primer lugar los restaurantes y, a continuación, el restaurante especifico en el que está interesado (no solo el nombre, ya que si cuenta con varios locales, deberá seleccionar el que se encuentra en el lugar de interés). El segundo problema, relacionado con el primero, se presenta si no hubiera disponible una aplicación con toda la información unificada. En este caso, el usuario debería encontrar en primer lugar el medio en el que encontrar esa información (página web, una aplicación etc.) lo que supone además una variedad de interfaces que podrían requerir un periodo de aprendizaje o adaptación para el usuario (Dubois et al. 2013). Otro problema añadido, sería que el usuario desconozca el hecho de que dispone de esa información, por lo que sería necesario proporcionar de un mecanismo único y sencillo que permitiese "descubrir" la información disponible (Schmalstieg and Wagner 2007).

Por lo tanto, teniendo en cuenta los requisitos buscados y los problemas presentados, aparece la necesidad de encontrar una tecnología que permita agilizar los procesos intermedios comentados, *seleccionar los datos de un modo eficiente* y *disponer de una interfaz común a cualquier dispositivo*. Es en este aspecto donde tiene sentido combinar la Realidad Aumentada (RA) con los objetos inteligentes. Superponiendo información digital en la realidad captada por un usuario, la RA permite solucionar el problema de identificar los objetos de un modo más eficiente y más natural (visualizando el objeto en sí con cualquier dispositivo que lo permita, como un Smartphone) (Azuma et al. 2001). En comparación con el ejemplo del restaurante, identificarlo consistiría en apuntar con un teléfono inteligente hacía el local. También cumple con el segundo requisito, un sistema común a cualquier objeto, mostrando la misma mecánica de funcionamiento (apuntar con el dispositivo de RA hacia el objeto) (Hölleler et al. 1999). Por otro lado, proporciona una posible solución para "descubrir" objetos inteligentes disponibles, ya que estos se pueden encontrar si la posición geográfica se encuentra en el sistema (Bartie and Mackaness 2006). Del mismo modo, la RA también encuentra ventajas en utilizarse de forma complementaria a IoT

puesto que al utilizarse de forma conjunta, los objetos virtuales se corresponden con objetos físicos, mostrando información más actualizada, más interactiva y posiblemente, más relevante, proporcionando al usuario un nivel más de interacción.

Otro aspecto a tener en cuenta es la cantidad de investigaciones orientadas a demostrar y aplicar las ventajas de la RA en diferentes áreas. Como ejemplo de aplicaciones colaborativas, (Bauer et al. 2001) proponen un sistema que permite a varios usuarios compartir la misma "experiencia" de realidad aumentada, añadiendo información al mundo virtual y asociando esta información a los objetos reales o a localizaciones. En (Ralph Dörner et al. 2003) muestran un Sistema orientado al turismo que funciona con reconocimiento de voz para explorar marcadores en la ciudad de Edimburgo. En la industria, (Friedrich, Jahn, and Schmidt 2002) muestran ARVIKA, basado en RA para facilitar los procesos de producción implicados en las industrias de automoción y aeroespacial. La RA también puede ser utilizada para mejorar la seguridad de algunos sistemas, como los vehículos. En (Stefan Müller-Schneiders. 2012) muestran un sistema en el que la información de las señales de tráfico se muestra resaltadas y superpuestas en el parabrisas del conductor como un sistema de ayuda, permitiendo que este se pueda centrar más en la conducción. En (Shelton and Hedley 2002) presentan un sistema para mejorar el proceso de aprendizaje de los alumnos sobre las relaciones entre el sol y la tierra, demostrando sus aplicaciones en eLearning. En (Dubois et al. 2013), muestran una plataforma para crear prototipos de aplicaciones basadas en sistemas de interacción mixtos, en los que objetos físicos intervienen en la interacción con objetos virtuales, y otros como entretenimiento, simulación como ayuda en los entrenamientos de ciertas habilidades y muchos otros en los que la interfaz proporcionada por la RA aporta numerosas ventajas (Van Krevelen and Poelman 2010; Azuma et al. 2001).

Además de las diferentes investigaciones, también están mostrando interés en la RA algunas de las empresas más influyentes en tecnología. Las Google Glases, de Google, o su reciente adquisición de la empresa de RA, Magic Leap, Microsoft y sus HoloLens o la adquisición de Metaio por parte de Apple.

También cabe destacar el "sencillo" proceso de adaptación a esta tecnología, pudiendo hacer uso de varias alternativas, como la posición física de los dispositivos si se utiliza geolocalización, o etiquetas RFID, códigos QR, o patrones en el caso de reconocimiento mediante imágenes.

Sin embargo, pese a las ventajas de la RA mostradas, las actuales plataformas enfocadas a crear contenido basado en esta tecnología *no permiten controlar el comportamiento de los objetos virtuales*, lo cual sería interesante para las aplicaciones de RA en general, e impres-

cindible para aquellas en combinación con los objetos inteligentes. Si una supuesta aplicación de RA mostrara información sobre los monumentos de una ciudad en una ruta turística, cada objeto virtual representaría un monumento en concreto. En las plataformas actuales, cada objeto virtual se compone de elementos de información estática, como texto o imágenes. En este caso, mostrar información estática como su historia o datos de interés es suficiente, puesto que la interacción entre el usuario y el monumento (como una unidad) no tendría sentido.

Sin embargo, en otra hipotética aplicación de RA, en la que se presentasen diferentes elementos domésticos conectados a una red en una casa, mostrar este tipo de información no sería suficiente. El motivo es que el estado de estos objetos puede variar dependiendo de muchos factores, como la interacción con las personas o con otros objetos inteligentes, y si la información no varía de acuerdo a esos estados, se perdería la correspondencia entre lo virtual y lo real, requisito principal en RA (MILGRAM y KISHINO 1994). Además, en este caso, lo más interesante sería poder interactuar con ellos a través de los objetos virtuales que los representan (Heun, Hobin, y Maes 2013), como encender la televisión, la luz, el reproductor de música o abrir las persianas. Más importante aún sería poder definir qué aspectos de esos objetos se controlan y cómo, por ejemplo, apagar automáticamente el reproductor de música si se enciende la televisión. Para conseguirlo, sería necesario proporcionar un método que permita a los usuarios definir cómo interactúan los objetos virtuales con los usuarios y con los objetos inteligentes. De este modo, controlar el comportamiento en los objetos virtuales resulta el aspecto más importante en la combinación entre RA e IoT. Por este motivo, este será el principal problema a resolver en este trabajo.

MOTIVACIÓN

Las actuales plataformas dedicadas a la generación de contenido de RA (sin necesidad de programar) permiten en su gran mayoría generar contenido de tipo estático (imágenes, videos, animaciones, etc.), y aquellas que permiten añadir interacción, lo hacen a un nivel muy básico, como realizar un determinado conjunto de acciones cuando se pulsa un botón, abrir una web, enviar un email, etc o, en algunos casos, condicionar esta interacción a diferentes eventos, como la posición geográfica en la que se encuentra cuando se detecta un patrón de imágenes. Esto es posiblemente debido a que hasta hace relativamente poco, las herramientas relacionadas con la RA se enfocaban en abstraer los complejos procesos implicados en esta (reconocimiento de imágenes, situar los objetos virtuales en la posición física correcta, etc.).

Algunas de estas herramientas, o frameworks, como ARToolkit (Kato y Billinghurst 1999), Metaio, Vuforia o Wikitude (Madden 2011), proporcionan un Kit de Desarrollo Software (Software Development Kit o SDK) con el que los desarrolladores pueden elaborar aplicaciones de RA para distintos dispositivos, como los Smartphones. Con estos frameworks, los desarrolladores pueden elaborar prácticamente cualquier contenido, implementando las aplicaciones de RA como cualquier otra aplicación, como el navegador para coches desarrollado con Wikitude, que muestra las indicaciones sobre el escenario real, permitiendo al conductor visualizar la carretera mientras observa las indicaciones[1].

Sin embargo, el uso de la tecnología de RA es solo un intermediario en lo que más importa en este tipo de aplicaciones: el contenido, y cómo mostrarlo (Poupyrev et al. 2002). Debido a que son los desarrolladores los encargados de adaptarlo a esta tecnología, resulta muy complicado adaptar una gran cantidad de contenido y responder a la demanda de los usuarios (Belimpasakis et al. 2010). La solución a este problema está en permitir a los usuarios crear su propio contenido y compartirlo con los demás. Pero para ello es necesario que la creación de este sea posible sin necesidad de programar, puesto que la mayoría de los usuarios no poseen estos conocimientos.

[1] https://www.wikitude.com/showcase/wikitude-navigation/

Como solución a este problema, se desarrollaron varias soluciones que permiten diseñar elementos o componentes mostrados en las escenas de RA y asociarlos con distintos identificadores (como posiciones o patrones de imágenes). El editor web de Wikitude, por ejemplo, permite seleccionar componentes como imágenes o textos y situarlos en una posición en la pantalla del dispositivo (de forma gráfica, arrastrando y soltándolos con el ratón) cuando una imagen es reconocida. Otras plataformas proponen diferentes alternativas con el objetivo de añadir más interacción en las escenas de RA, como Zapcode, que permite añadir interacción definiendo escenas, modificando el contenido dependiendo de los intereses de los usuarios, o algunas acciones como enviar un correo o enlazar una página web.

Sin embargo, aunque las soluciones adoptadas presentan numerosas ventajas y algunas ya muestran características orientadas a controlar la interacción entre el usuario y el contenido de las aplicaciones, en general, las interacciones y cómo se comportan los objetos virtuales ante estas, es muy limitado, impidiendo a los usuarios crear contenido personalizado a las necesidades de cada uno, haciendo muy complicado conseguir una aplicación que realice una tarea compleja para un determinado caso de uso.

Un ejemplo de esta limitación podría ser el de un restaurante que quisiera permitir a los clientes reservar una mesa mediante RA. Para ello, en primer lugar sería necesario que la aplicación consultase el sistema para saber si la mesa está libre y, dependiendo de si lo está o no, el objeto virtual que representa la mesa, debería permitir reservarla o no. En este caso, la información mostrada en la aplicación de RA depende del estado de los objetos de la escena (las mesas) por lo que mostrar información estática no resultaría adecuado (si una mesa estuviera reservada, el cliente no podría comprobarlo desde la aplicación). Por otro lado, no solo se tiene que mostrar la información actualizada, sino que un cambio en los objetos virtuales debería suponer una modificación en el estado del sistema (el usuario hace la reserva).

La principal ventaja de los objetos inteligentes es que además de mostrar de forma digital la información referente a ellos mismos, sea posible controlarlos a través de una red. Si los objetos virtuales representan simplemente la información de los objetos inteligentes, se pierde la principal ventaja de estos. Para conseguir una mayor correspondencia entre ambos objetos, virtuales e inteligentes, es necesario proveer de mecanismos más complejos que permitan controlar el comportamiento de los primeros.

Definir el comportamiento de un objeto virtual en general no es una tarea sencilla, y está asociada en la mayoría de los casos a la programación. Pero, como se ha explicado, es ne-

cesario que sea posible crear las aplicaciones sin estos conocimientos. Para conseguir esto, aparece la Ingeniería dirigida por modelos (Model Driven Engineering o MDE) como la solución más apropiada. MDE permite entre otras cosas que usuarios no expertos definan las especificaciones de una aplicación manipulando componentes con los que ya están familiarizados (Van Deursen et al. 2000), obteniendo el sistema funcional sin necesidad de programar. Además, aplicar MDE en los sistemas de interacción mixtos ya ha demostrado diferentes ventajas (Dubois et al. 2013). Concretamente, en su investigación indican cinco beneficios que describen como: identificación de las entidades, generación de múltiples representaciones, construcción, mantenimiento y manipulación de los gráficos que representan los modelos/aplicaciones, esquematización del proceso de desarrollo y soporte para la integración de decisiones en el diseño.

CONTRIBUCIÓN

La propuesta de este trabajo se presenta como posible solución al problema de crear contenido para aplicaciones de realidad aumentada que interactúe con objetos inteligentes. Mediante la elaboración de una plataforma basada en la ingeniería dirigida por modelos, se plantean los siguientes objetivos:

- Diseñar un modelo (o metamodelo) correspondiente con los elementos que intervienen en el dominio planteado. Este metamodelo deberá unificar tanto los objetos virtuales como los inteligentes, y a su vez, permitir definir el comportamiento de los mismos en las aplicaciones creadas.
- Proporcionar herramientas que, basadas en el metamodelo definido, faciliten a los usuarios la elaboración de las aplicaciones. Estas herramientas se corresponden con dos lenguajes de dominio específico (Specific Domain Language o DSL), uno textual y uno gráfico.
- Integrar de forma transparente los objetos inteligentes en las aplicaciones. La creación de las aplicaciones en la plataforma deberá ser independiente de si en ellas aparecen objetos inteligentes o no.

POSIBLES ÁMBITOS

Se destacan los siguientes ámbitos en los que las aplicaciones creadas con el sistema propuesto aportarían contribuciones relevantes:

- **Educación**: Uno de los problemas presentes en integrar la RA en la educación es la dificultad para elaborar contenido o tareas para los alumnos basadas en esta tecnología (Kerawalla et al. 2006). Con la plataforma propuesta es posible crear este tipo de contenido, puesto que basándose en la interacción con el usuario, la aplicación le puede proporcionar un feedback que les indique a los alumnos si comprenden correctamente los conceptos. Mostrar un cuestionario con varias posibles respuestas sería un ejemplo. Dependiendo de la respuesta marcada por el usuario, la aplicación mostraría una imagen indicando si es correcta o no.
- **Smart cities**: En el apartado de introducción se presentaban los posibles inconvenientes para seleccionar determinada información en una ciudad inteligente. Esta plataforma presenta una posible solución a esos problemas. Para ello, los propios usuarios pueden añadir contenido relacionado con la información de la ciudad. De esta manera, los demás usuarios pueden consultarla apuntando con sus dispositivos hacia los objetos reales, como una estación de autobús que muestre cuando llega el siguiente.
- **Smart homes**: Similar al problema de las ciudades inteligentes, seleccionar el dispositivo que se desea controlar en una Smart Home (Casa inteligente) puede resultar más sencillo mediante una interfaz basada en RA. En lugar de consultar en un smartphone todos los dispositivos disponibles en la Smart Home y a continuación seleccionar el de interés, con la RA este proceso se reduce a apuntar hacia ese dispositivo con el smartphone.
- **Juegos**: Juegos habituales en RA como la búsqueda del tesoro podrían elaborarse de un modo más interactivo en combinación con los objetos inteligentes. En este caso, el tesoro podría situarse en una caja fuerte a la que solo se puede acceder cuando el jugador ha resuelto las pistas.

- **Industria**: Una de las áreas donde más se están utilizando ambas tecnologías es la industria. Permitiendo definir y visualizar diferentes procesos industriales, esta plataforma podría proporcionar facilidades en este aspecto (Pentenrieder et al. 2007).

ESTADO DE LA SITUACIÓN ACTUAL

1. *INGENIERÍA DIRIGIDA POR MODELOS*

La Ingeniería Dirigida por Modelos (Model-Driven Engineering) o MDE (Kent 2002) surge como una posible solución a los problemas asociados al desarrollo de software, especialmente aquel orientado al ámbito empresarial debido principalmente a una mayor complejidad de este. Problemas estableciendo los requisitos del sistema, en la planificación o en la estimación de los costes son algunos de los más comunes durante el desarrollo de proyectos de software.

Para solucionar estos problemas se plantea la automatización de algunos de los procesos implicados en el desarrollo del software, así como la reutilización de componentes ya desarrollados. Reutilizar componentes supone aumentar considerablemente el rendimiento al reducir el tiempo de desarrollo, además eleva la calidad del software desarrollado puesto que los componentes utilizados ya han sido probados, siendo independientes del sistema, lo que supone también una mejora en el mantenimiento de este (García-Díaz et al. 2012).

Para conseguir la automatización, MDA se basa en el uso de modelos (Seidewitz 2003). Los modelos permiten elevar el nivel de abstracción sobre los lenguajes de programación, como Java o C#. Esta mayor abstracción supone una mayor productividad, al permitir separar los conceptos de análisis y diseño del sistema, de las especificaciones de la implementación. El uso de modelos permite también describir de forma más precisa los requisitos del sistema. Para ello se parte en primer lugar del dominio concreto del problema. Los conceptos principales de este dominio se describen mediante el uso de un meta-modelo.

Los meta-modelos definen su estructura, construcciones, propiedades y conectores mediante una *sintaxis abstracta*, dependiente del dominio y que no varía. Mediante una *sintaxis concreta* se especifica la notación del lenguaje que los usuarios de este utilizarán. El meta-modelo permite delimitar los conocimientos del dominio formalmente, que es la

característica principal que permite realizar las transformaciones de forma automática partiendo de la especificación del sistema.

Una vez definidos el dominio y el o los meta-modelos, se puede definir el DSL. La principal ventaja de los DSLs es que su semántica está basada en los conceptos que componen el dominio del problema, por lo que los usuarios están familiarizados con los términos de este, adaptándose fácilmente y pudiendo usarlo aunque no se trate de usuarios expertos (un programador, por ejemplo) (Van Deursen and Klint 2002).

Los DSLs pueden ser tanto textuales, como es habitual en los lenguajes de modelado, o gráficos, que realizan las definiciones de forma gráfica. Para este proyecto se han elaborado ambas puesto que cada una aporta diferentes propiedades y ventajas.

2. ARQUITECTURA DIRIGIDA POR MODELOS

La arquitectura dirigida por modelos (Model Driven Engineering o MDA) surge en marzo de 2000 por el grupo Object Management Group (OMG) con el objetivo de proporcionar portabilidad, interoperabilidad y reusabilidad a través de la separación arquitectónica de conceptos (Soley et al. 2000).

Esta iniciativa surge a consecuencia de las complicaciones asociadas al desarrollo de un software cada vez más complejo y con más características, y al que se le requiere también menos errores. Aplicando estos conceptos al proceso de diseño y desarrollo de software se pretende incrementar la productividad y la reutilización de sistemas (Pelayo García-Bustelo 2007).

MDA propone separar la definición de la implementación en cada plataforma específica, de modo que con una especificación se pueda implementar el sistema en varias plataformas o tecnologías. El paso de la especificación a las diferentes plataformas se realiza basándose en dos conceptos básicos de MDA, los modelos y las transformaciones. Los modelos se definirían en el nivel de abstracción más alto y es dependiente del dominio concreto del sistema. Las transformaciones, por otro lado, se aplican a las capas de abstracción definidas por MDA, desde la más alta, hasta el código fuente ejecutable. La transformación entre estas capas debería realizarse de un modo automático.

Las capas que permiten definir las diferentes partes que componen el sistema son, los requisitos del sistema (modelo de computación independiente), la estructura del sistema, pero independientemente de las plataformas especificas (modelo de plataforma indepen-

diente) y diseño y detalles de las plataformas específicas que usa el sistema (modelo de plataforma específica).

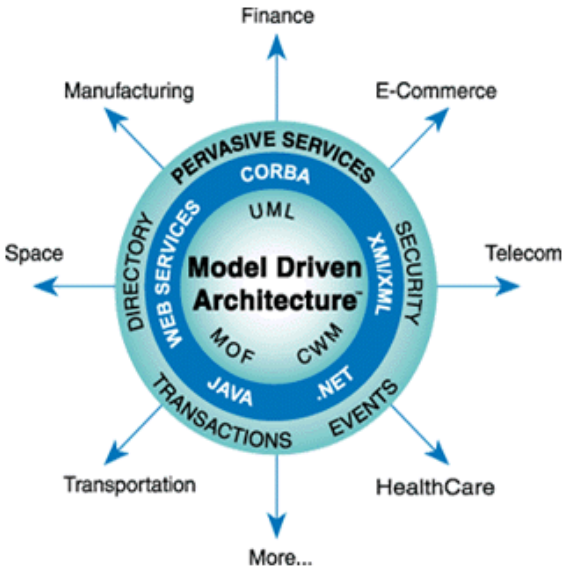


Figura 1 Ejemplo proporcionado por el OMG de la distribución en capas en MDA

2.1. MODELO DE COMPUTACIÓN INDEPENDIENTE (CIM)

Esta capa está orientada a los expertos del dominio, aquellos que conocen los requisitos del sistema y el modo en el que este debe funcionar. Estos expertos no tienen por qué tener conocimientos sobre los detalles de cómo el sistema debe funcionar para cumplir con estos requisitos, es decir su diseño e implementación. El CIM permite precisamente eso, definir los requisitos del sistema ocultando los detalles del diseño y la implementación.

2.2. MODELO DE PLATAFORMA INDEPENDIENTE (PIM)

En esta capa, la siguiente a CIM, se describen las operaciones del sistema, pero sin especificar los detalles de cada plataforma específica. En esta capa se suelen utilizar especificaciones de modelado como: The Unified Modeling Language (UML) o Meta-Object Facility (MOF). Las principales ventajas de esta capa son que mejora la productividad al generar de forma automática la parte del sistema dependiente de las plataformas específicas, y por este motivo, mejora también la portabilidad del sistema al ser independiente los modelos de la tecnología utilizada.

2.3. MODELO DE PLATAFORMA ESPECÍFICA (PSM)

La PSM contiene los detalles de cada plataforma específica en la que se ejecutará el sistema. Mientras que el PIM describe el sistema independientemente de la plataforma, el PSM especifica cómo el sistema utiliza cada plataforma. El PSM contiene la información necesaria para generar el código de la aplicación. En esta capa se utilizan lo que en MDA se cono-

ce como puentes. Estos puentes permiten conectar sistemas con distintas tecnologías, proporcionando interoperabilidad de forma automática puesto que, como los modelos, en el PSM, los puentes también se crean a partir de los PIM.

3. *REALIDAD AUMENTADA*

La realidad aumentada comienza en 1960, con el primer prototipo creado por Ivan Sutherland y sus estudiantes de Harvard, en el que se utilizaba un HDM (Head-mounted display) para mostrar gráficos en 3D. El principal objetivo era el de mejorar la interacción entre las personas y el mundo real, de modo que, mediante el uso de objetos virtuales, los usuarios puedan realizar tareas "reales". Para ello, el dispositivo permitía al usuario visualizar objetos virtuales superpuestos en la "realidad", proporcionando información al usuario con una interacción más sencilla. Posteriormente se siguieron realizando investigaciones en esta área con una gran variedad de aplicaciones (Azuma et al. 2001).

Aunque las definiciones de RA no limitan la experiencia ofrecida por esta tecnología a la superposición de imágenes, el proceso habitual que se utiliza para elaborar una escena de RA, tal y como describieron (Hölleler et al. 1999), es analizar lo que el usuario está observando en ese momento, comprobar qué objetos virtuales pueden situarse en lo que está observando y calcular como mostrarlo en combinación con los objetos reales. Cada objeto virtual tiene la información que lo define tanto para posicionarlo (ya sea la posición geográfica de este, mediante un sistema de localización, o mediante un patrón visual o imagen que permita identificarlo mediante el análisis de imágenes), como para representarlo (una imagen, sonido, texto…). Estos objetos pueden estar en cualquier lugar almacenados mientras sean accesibles para el dispositivo que elabora la RA. Sin embargo, como se muestra en (Belimpasakis et al. 2010) basar la accesibilidad de estos objetos virtuales en servicios web, permite al usuario disponer de un mayor contenido y más actualizado.

Uno de los aspectos necesarios para realizar este tipo de experiencia es, por tanto, seguir el movimiento del usuario, para poder situar los objetos virtuales correctamente donde se corresponden. Clasificando la tecnología disponible para seguir el movimiento de los usuarios encontramos las siguientes categorías (Van Krevelen and Poelman 2010):

- Basado en la posición geográfica del usuario: En esta categoría se utilizan técnicas como mecánica, ultrasonidos, magnetismo, sistemas de posicionamiento global, radio o inerciales.

- Basados en la óptica: Analizando las imágenes capturadas por una cámara se detecta la geometría de la escena para estimar la posición del usuario. En esta categoría se pueden encontrar varias técnicas como marcadores, comparación de modelos, leds…

El avance en la tecnología necesaria para utilizar la RA tuvo gran importancia en el desarrollo de librerías y frameworks, (Didier et al. 2012; Piekarski and Thomas 2001). Las aplicaciones se podían utilizar en dispositivos más accesibles (más baratos, más pequeños y más potentes), haciendo que más usuarios pudieran utilizarlas y aumentando por tanto la demanda de herramientas para elaborar estas aplicaciones. (Van Krevelen and Poelman 2010) muestran una detallada descripción de estos dispositivos utilizados en RA. Dispositivos de audición, dispositivos de visualización como pantallas mostrando un video en tiempo real "modificado" (Peter Shirley et al. 2008), dispositivos basados en "lentes" que superponen imágenes a las imágenes "reales" (Cakmakci, O et al. 2006), cascos (Raskar et al. 1998), proyectando imágenes sobre los objetos reales (Rekimoto et al. 1998) o basadas en el tacto, de modo que el dispositivo de interacción reaccione dependiendo de los movimientos del usuario.

Actualmente se encuentran disponibles multitud de plataformas y frameworks para el desarrollo de aplicaciones de realidad aumentada, tanto para dispositivos específicos como soluciones que incluyen soporte para múltiples plataformas. Debido a la dificultad técnica que surge de la propia tecnología, se desarrollaron en primer lugar frameworks que permitían abstraer las operaciones básicas que conforman la realidad aumentada, como el tratamiento de imágenes para el reconocimiento de patrones, librerías para modelar o renderizar objetos virtuales que dependen de parámetros como la posición (en espacios tridimensionales), o detección de la posición de los dispositivos/usuarios mediante el uso de sensores como GPS (Global Position System o Sistema de Posicionamiento Global), WI-FI (Wireless Fidelity o Fidelidad Inalámbrica) o brújula.

Uno de los ejemplos más claros del uso de estos frameworks fueron las aplicaciones de realidad aumentada para smartphones (como Layar o Wikitude), con los que haciendo uso de los sensores del dispositivo, la cámara y la pantalla, el usuario puede obtener información de su entorno simplemente apuntando con su Smartphone a la zona en cuestión (Piekarski y Thomas 2001; Owen, Tang, y Xiao 2003; Magnenat-Thalmann y Thalmann 1999).

Con el aumento del uso de estas aplicaciones, estos frameworks evolucionaron a plataformas orientadas directamente al desarrollo de aplicaciones en diferentes dispositivos, aña-

diendo una capa más de abstracción, permitiendo centrarse más en la lógica y el dominio del problema de las aplicaciones. Algunos de estos frameworks son ARMedia, Metaio, Vuforia, o ArToolkit. En este punto, algunas plataformas comienzan a proporcionar soluciones que permiten elaborar aplicaciones de realidad aumentada sin que sean necesarios conocimientos de programación. Algunas de estas plataformas son LayAR, Wikitude, Aurasma, Zapcode, o GuideMe (los primeros comerciales y GuideMe orientado a la investigación). Sin embargo, como se muestra en el apartado de Frameworks y plataformas disponibles, la mayoría de las aplicaciones creadas por estas plataformas se limitan prácticamente a mostrar contenido estático asociado a ciertos marcadores como posiciones o patrones de imágenes.

4. SMART OBJECTS

(McFarlane et al. 2003) definen los Objetos Inteligentes ("Smart objects" o "intelligent products") como "*una representación basada en la información y propiedades físicas de un objeto la cual dispone de una identificación única de este, con la capacidad de comunicarse de forma efectiva con su entorno, almacenar información por sí mismo, implementar un lenguaje para mostrar sus características, requisitos de producción etc. Y es capaz de participar o tomar decisiones relevantes para el funcionamiento de sí mismo*".

Dependiendo del nivel o capacidad de un objeto para procesar y gestionar este tipo de información, se distinguen tres niveles de inteligencia (Meyer et al., 2009):

- Información sobre sí mismo. Se corresponde con la capacidad del objeto para leer u obtener su propia información. Este es el nivel más básico puesto que el objetivo de este tipo de objetos es permitir el acceso a su información de forma digital.
- Envío voluntario de información. Este nivel se corresponde con la capacidad del objeto para enviar información cuando detecta ciertos eventos. En este caso, el dispositivo notifica la información a los dispositivos asociados, al contrario que en el primer nivel, donde es necesario leer la información para obtenerla.
- Procesamiento y decisiones propias. En este nivel se encuentran aquellos objetos que, procesando la información de la que disponen, son capaces de tomar decisiones y controlar su propio comportamiento.

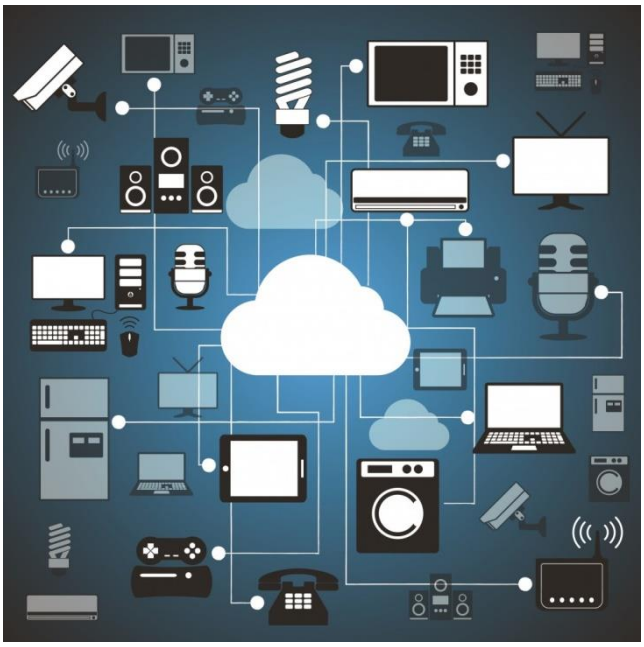


Figura 2 Imagen representativa de Internet of Things y algunos objetos inteligentes mostrada en "http://www.techpolicydaily.com/"

Actualmente el número de este tipo de dispositivos es muy elevado, y es muy común utilizar habitualmente algunos de ellos, como Smart TVs (Televisiones Inteligentes), Smartphones o electrodomésticos, como los mostrados en la Figura 2.

Un problema a tener en cuenta en esta tecnología es la diversidad de objetos disponibles así como las tecnologías y protocolos para conectarse entre ellos (Pascual Espada 2012), (González García 2013). Sin embargo debido a que el objetivo de utilizar objetos inteligentes en esta investigación es la de demostrar las ventajas de combinar RA y objetos inteligentes, no se usarán en el prototipo plataformas como la propuesta en (González García 2013), aunque si se tiene en cuenta una posible integración con este tipo de plataformas en la sección de trabajo futuro y conclusiones.

5. *MDE APLICADO EN RA Y IOT*

Se han mostrado algunas de las ventajas de aplicar MDE o iniciativas de este como MDA a proyectos software en general, lo que ya supone un motivo suficiente como para aplicarlo en el caso de esta plataforma en concreto, sin embargo, hay dos motivos principales por los que se ha aplicado MDE en este caso. Uno de los motivos ha sido que, como se ha comentado en el apartado de IoT y RA, las tecnologías en ambos casos ya son lo suficientemente maduras como para que los usuarios las utilicen de forma habitual y asequible, lo que quiere decir que el componente más necesario es el contenido de estas, y un modo de

conseguir esto es que sean los propios usuarios los que puedan crearlo de forma sencilla, en este caso, mediante el uso de un DSL. El otro motivo, y contribución principal de esta investigación, es que ese mismo contenido creado por los usuarios consista en la combinación de componentes de realidad aumentada como son los objetos virtuales y componentes de IoT, funcionando de forma conjunta.

6. *FRAMEWORKS Y PLATAFORMAS DE RA DISPONIBLES*

6.1. GUIDE-ME

Este framework es interesante ya que se trata de un trabajo de investigación basado en el uso de la ingeniería dirigida por modelos aplicada a los sistemas de interacción mixtos (Dubois et al. 2012). Se trata de un framework con un nivel de abstracción bastante alto en la que para describir el sistema se definen varios modelos basados en tres metamodelos. El primero de estos metamodelos es ASUR, que define la interacción del usuario con los elementos del sistema, independientemente de la tecnología o implementaciones utilizadas.

Para definir esta interacción, el metamodelo cuenta con cuatro tipos de entidades, "adaptadores de entrada y salida" (como pantallas o sensores), "sistema", que representa las entidades digitales del sistema (como una aplicación), usuarios y "objetos reales". Las entidades interactúan entre sí intercambiando información a través de los canales de interacción, también definidos en el metamodelo. El metamodelo también cuenta con más mecanismos para definir con más detalle el sistema, como el emisor de la acción, el receptor o el tipo de medio que se utiliza para la comunicación (luz, contacto físico, infrarrojos, aire…).

El segundo metamodelo, ASUR-IL, permite definir precisamente la arquitectura del software que dará soporte a la implementación concreta de la aplicación. En este metamodelo se utilizan dos tipos de interacciones, "entity sub-assembly", que representa los componentes dependientes del sistema y utilizan el patrón MVC (Model-View-Controller o Modelo-Vista-Controlador) para representar los elementos funcionales del sistema y "adapter sub-assembly", que representa los componentes independientes del sistema y sigue el modelo Arch, diferenciando dos tipos de componentes, "device" y "API" (Application Program Interface o Interfaz de Programación de Aplicaciones).

El tercer metamodelo, basado en la plataforma WComp, permite describir los componentes de software utilizados en la implementación y que producen la aplicación "ejecutable".

Este metamodelo se define a través de "contenedores", que describen los aspectos no funcionales del componente (como instancias, gestión de memoria o conectores) y los "diseñadores", que describen los aspectos funcionales del componente.

En este framework, las transformaciones entre metamodelos se pueden realizar de dos formas (ambas automáticas). Mediante un "mapeo" simple entre los conceptos de los modelos o haciendo uso de una ontología. Para ambos casos el framework define dos metamodelos "puente", de ASUR a ASUR-IL y de ASUR-IL a WComp.

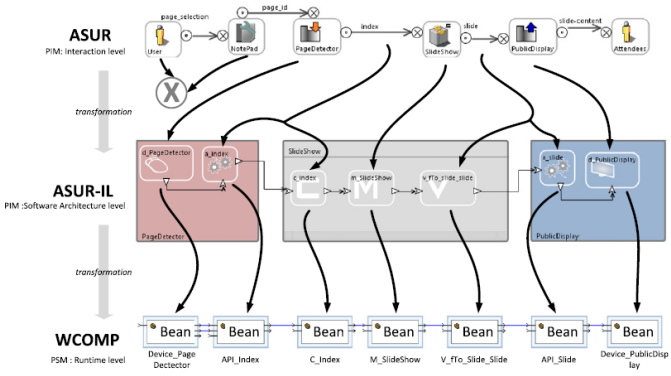


Figura 3 Metamodelos utilizados en GuideMe

Para hacer uso de este framework se ha diseñado una implementación en un Eclipse específico elaborado con EMF (Eclipse Modeling Framework) y GMF (Graphical Modeling Framework), que cuenta con las herramientas necesarias para desarrollar los modelos específicos (tanto de forma gráfica, como textual, ya que la plataforma cuenta con un lenguaje de dominio específico).

La Figura 3 muestra un ejemplo proporcionado por GuideMe, un sistema que permite realizar una presentación de diapositivas de forma interactiva. En el sistema se definen (metamodelo ASUR), el usuario que realiza la presentación y los espectadores (usuarios), el bloc de notas (objeto real), el detector de páginas y el sensor táctil para controlar las animaciones de la presentación (adaptadores de entrada), las diapositivas (objeto digital) y las pantallas privada, para el usuario que realiza la presentación y pública, para los espectadores (adaptadores de salida). Mediante una transformación automática a partir de este modelo, se obtiene el segundo modelo (ASUR-IL), como se muestra en la Figura 1, obteniendo los componentes de la arquitectura del sistema. Como dispositivos, el sensor detector de páginas y el detector para controlar las animaciones. Los componentes dependientes del sistema (entitiy sub-assembly), se corresponden con el control de las diapositivas. Las pantallas pública y privada se corresponden con los dispositivos de salida. Una segunda transformación, obtiene el modelo que describe los componentes software (WComp),

que se corresponden con los controladores de los componentes (cámara, pantallas, diapositivas o sensor táctil).

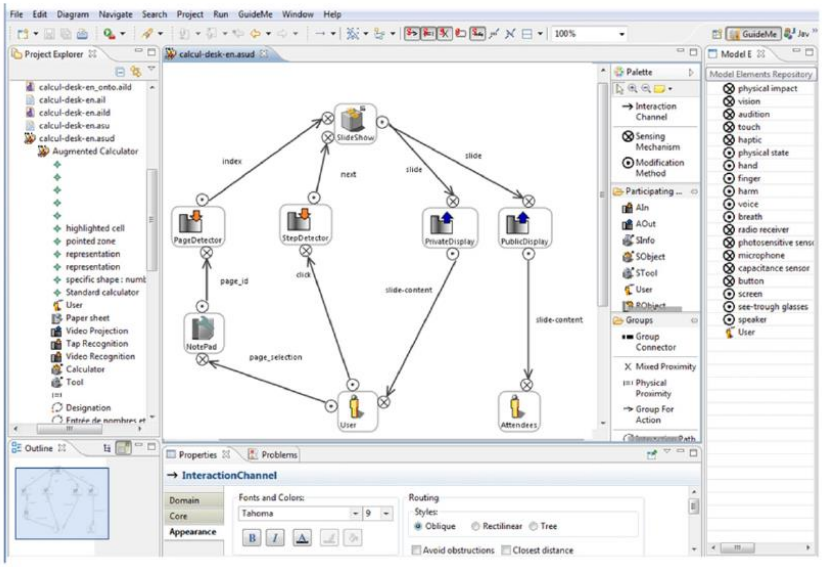


Figura 4 Ejemplo de sistema de presentación de diapositivas con GuideMe

La solución propuesta por esta plataforma proporciona una gran flexibilidad, puesto que permite definir el modelo principal independientemente de la implementación a utilizar, sin embargo, cabe destacar que este framework está orientado a la participación en el diseño de las aplicaciones por usuarios no expertos y aunque, como demuestran, las decisiones tomadas por los mismos se reflejan notablemente durante el desarrollo (el modelo de plataforma específica se basa en componentes de WComp), la implementación de las aplicaciones queda delegada en expertos del sistema.

También se debe considerar que este framework está orientado a sistemas de interacción mixtos, en los que el usuario interactúa con los objetos virtuales a través de objetos físicos, mientras que en RA (considerada como un tipo de MIS)(MILGRAM and KISHINO 1994) son los objetos virtuales los que poseen un mayor protagonismo, especialmente en este trabajo, en el que el usuario interacciona con los objetos físicos (objetos inteligentes), a través de los objetos virtuales. Esto supone que posiblemente ambos trabajos no pertenezcan a la misma categoría, sin embargo, se ha considerado oportuno describir este sistema puesto que es uno de los que más se acerca al objetivo de este trabajo.

6.2. WIKITUDE SDK

Esta plataforma (Madden 2011; Bruna Gómez 2012), proporciona tanto un SDK (Software Development Kit o Kit de Desarrollo Software) para desarrollar aplicaciones en distintos sistemas, como un editor gráfico basado en componentes comunes en este tipo de aplica-

ciones. Con el SDK, tanto la lógica como la presentación se definen utilizando estándares web como HTML5 (HyperText Markup Language o Lenguaje de Marcas de HiperTexto), Javascript y CSS (Cascading Style Sheets u Hoja de Estilo en Cascada), minimizando el código dependiente de cada plataforma.

La plataforma soporta reconocimiento de imágenes y geolocalización, generando eventos cuando se reconoce un patrón visual o cuando el usuario se sitúa en la posición correcta. También permite el uso de varios componentes para elaborar la interfaz de usuario, como imágenes, videos, sonidos, HTML (con CSS y JavaScript), modelos gráficos tridimensionales o texto. El editor gráfico permite utilizar el reconocimiento de imágenes, pero no la geolocalización. Para ello, solicita una imagen inicial que se utiliza como patrón de reconocimiento.

Figura 5 Imagen inicial utilizada en Wikitude Studio como patrón de reconocimiento

A continuación, el usuario selecciona los componentes que quiere mostrar cuando esta imagen es reconocida, mostrando en la pantalla (o dispositivo de reproducción) esos componentes. Los componentes disponibles son, texto, imágenes, botones, HTML, modelos en 3D (3 Dimensions o 3 Dimensiones) y videos.

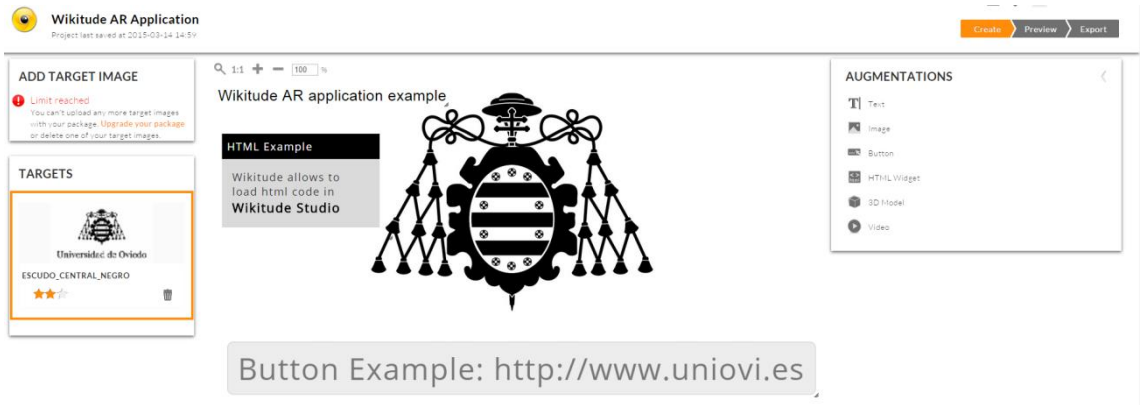


Figura 6 Componentes de ejemplo en una aplicación de Wikitude Studio

La plataforma permite previsualizar la aplicación antes de exportarla y hacerla pública. Utilizando uno de los clientes de wikitude, por ejemplo el de Android, se accede a esta versión de la aplicación en el apartado "Desarrollador", iniciando sesión con la cuenta del sitio web. De este modo no se requiere más configuración en el cliente ya que este se descarga del servidor los ficheros necesarios para iniciar la aplicación. Para publicar la aplicación es necesaria una licencia de pago de Wikitude, que permite acceder a la aplicación publicada desde la aplicación oficial.

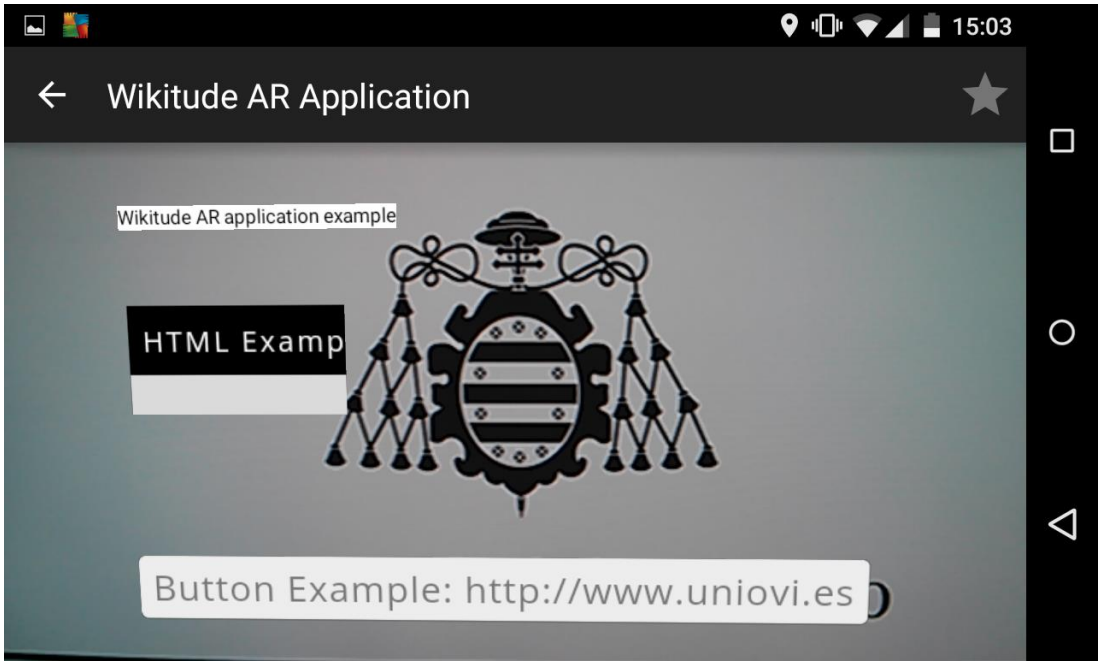


Figura 7 Aplicación cliente Android de Wikitude visualizando la aplicación de ejemplo elaborada con Wikitude Studio

La principal ventaja de esta plataforma, con respecto al editor gráfico, es que asegura que se pueda realizar una aplicación de realidad aumentada sin conocimientos de programación. Sin embargo, a cambio, se pierde bastante flexibilidad en el tipo de aplicaciones que se pueden elaborar (especialmente si se tiene en cuenta el gran número de opciones que proporciona el SDK), reduciéndose principalmente a aplicaciones estáticas que muestran información en los "puntos" establecidas por el usuario. Además, como se muestra en la Figura 7, el renderizado del componente web presenta algunos problemas (no se muestra correctamente, y el código JavaScript no se ejecuta, como si lo hace el editor). De este modo, el editor muestra varios apartados en los que se podrían realizar diferentes "mejoras", sin embargo, como se ha comentado, el SDK proporciona numerosas características y funcionalidades que han contribuido a que se utilice como soporte para elaborar el prototipo en esta investigación.

6.3. ZAPCODE

Zapcode, también orientado a un uso comercial, permite elaborar contenidos de realidad aumentada a través de su aplicación web. Se basa en el reconocimiento de imágenes, utilizando como identificador una variante creada por ellos de los códigos QR.

Con su editor gráfico, permite crear aplicaciones (públicas) basadas en escenas. Las escenas permiten que la aplicación pueda mostrar varias interfaces para un punto de interés, utilizando animaciones para pasar de una a otra. La plataforma permite también asociar acciones a los textos y los botones, como cambiar de escena, enlazar a otra página o llamar a un número de teléfono. El hecho de utilizar estas escenas y acciones, proporciona a esta plataforma más flexibilidad que otros editores gráficos manteniendo la facilidad de uso, ya que el uso de estas añade más interacción, acercándose más al uso de las aplicaciones "habituales". También permite, en su versión gratuita, añadir elementos gráficos como imágenes, videos, sonidos, álbumes de fotos (con animaciones), botones o texto. La plataforma soporta más tipos de gráficos como modelos o animaciones en 3D, aunque para utilizarlos es necesario adquirir la licencia de pago.

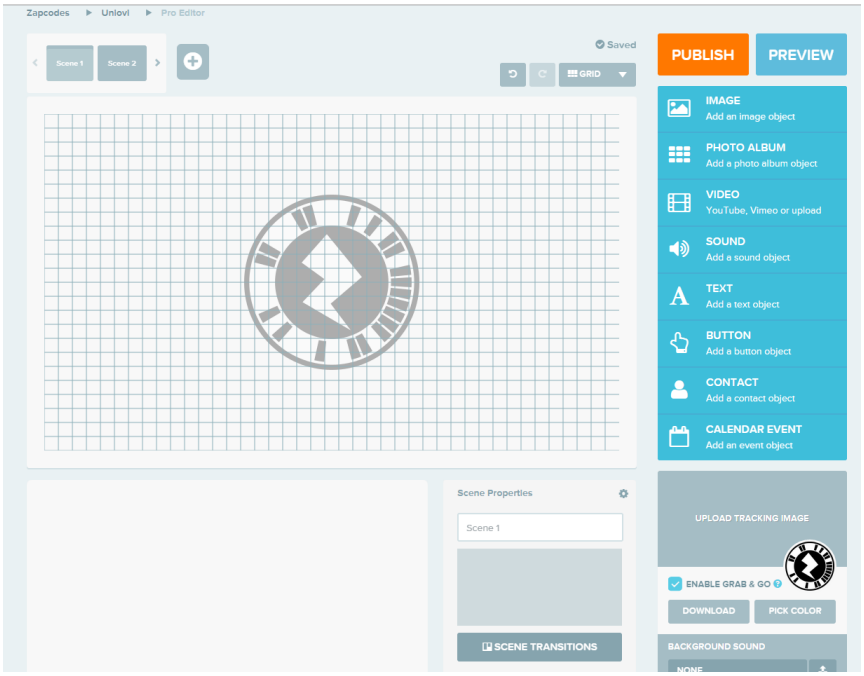


Figura 8 Editor gráfico de la plataforma Zapcode

Para elaborar la aplicación, se seleccionan los componentes que aparecen en cada escena. El editor permite modificar las propiedades de los componentes. Se presentan cuatro tipos de propiedades: las propiedades del componente, como el color del texto, su alineación, o su tipo de fuente, las acciones, como cambiar de escena o enlazar con un sitio web, transiciones, que permite definir una animación con el cambio de escena y apariencia, como el tamaño, color o tipo de bordes La escena inicial se muestra cuando el código de identificación es detectado por la aplicación cliente.

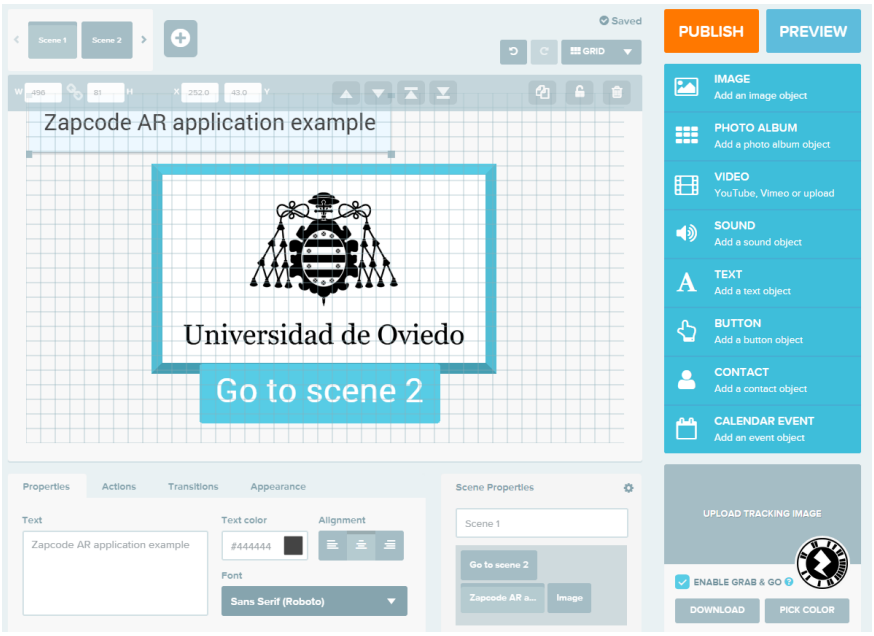


Figura 9 Componentes de la escena 1 de la aplicación de ejemplo elaborada con Zapcode

Una vez elaborada la aplicación, se puede previsualizar a través de su cliente (iOS o Android) escaneando con su aplicación el código proporcionado por la plataforma (el código se puede modificar para adaptarlo a una imagen colaborativa, aunque debe cumplir las especificaciones de la plataforma). La aplicación se puede publicar de forma gratuita, desde el editor, de modo que cualquier cliente que escanee el código obtiene la aplicación publicada en la plataforma.

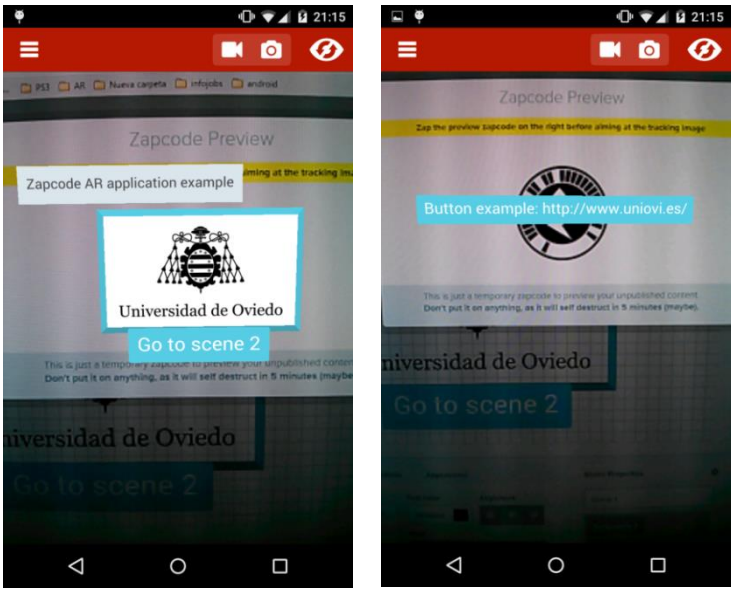


Figura 10 Aplicación cliente de Android de Zapcode visualizando la aplicación elaborada con el editor gráfico de la plataforma

6.4. REALITY EDITOR

(Heun, Hobin, and Maes 2013) muestran un sistema para proporcionar interfaces graficas mediante RA a los objetos inteligentes. Aunque la plataforma, compuesta por un dispositivo de RA (como un Smartphone) y varias herramientas para elaborar las interfaces de los objetos, no está disponible de forma pública, se ha considerado necesario describir brevemente su funcionamiento, así como la principal diferencia entre su enfoque y el de esta propuesta. Como demostración del uso de su plataforma, enseñan varios prototipos en los que, mediante una tablet controlan el comportamiento y las conexiones entre varios objetos inteligentes. Un ejemplo es una puerta con una cerradura (sin llave) en la que, para abrirla, el usuario debe orientar su Smartphone hacía la puerta e introducir el código de seguridad a través de la pantalla en la que se muestra un teclado numérico.

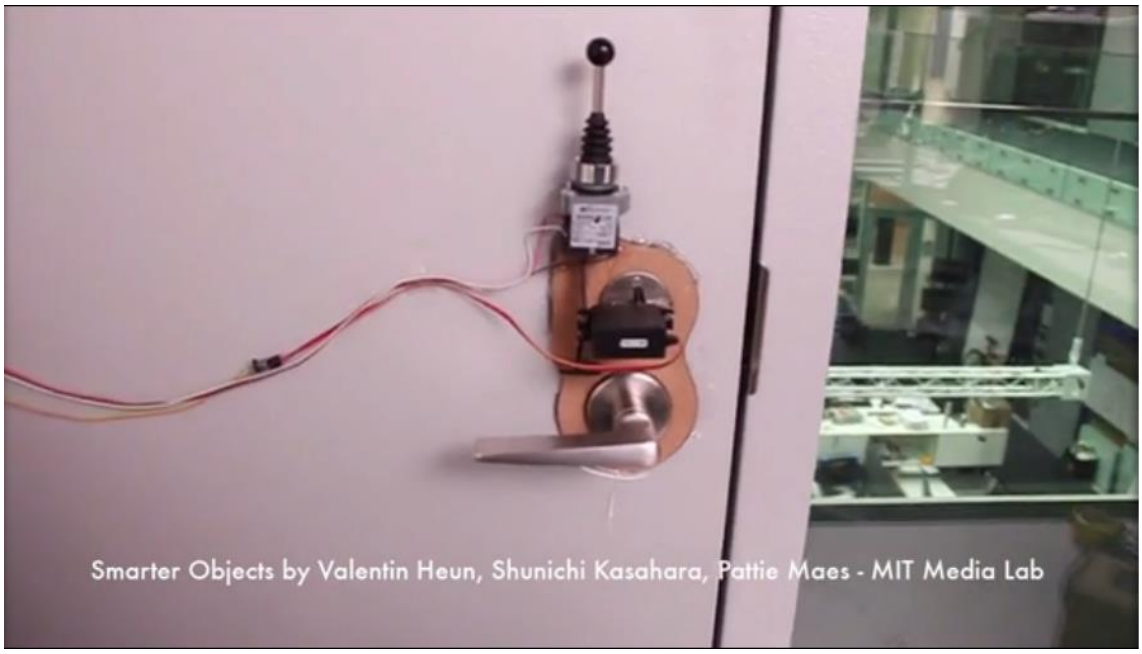
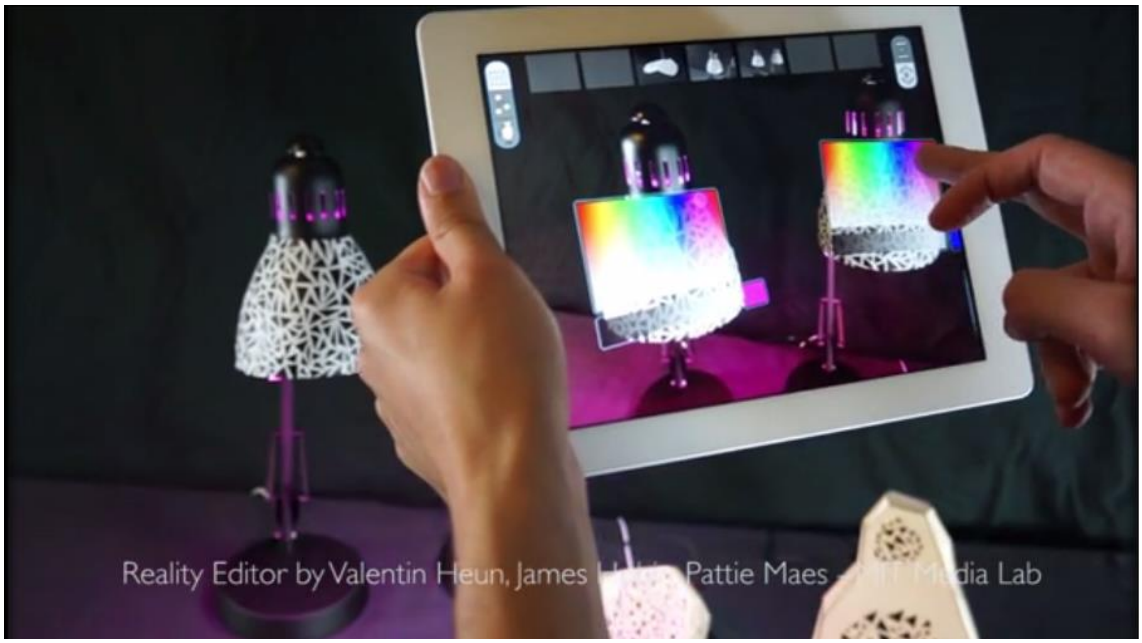


Figura 11 La cerradura electrónica, actuando como objeto inteligente, permite desbloquear la puerta mediante una orden remota.

Figura 12 El usuario introduce el código de seguridad para desbloquear la puerta a través de una interfaz de RA

En otro ejemplo muestra un sistema que combina objetos inteligentes y realidad aumentada permitiendo asociar unos objetos inteligentes con otros y manipularlos desde la interfaz de usuario. En el ejemplo, se asocia una lámpara con una radio (desde la aplicación), cuando se enciende la lámpara, la radio se enciende también.

En el mismo ejemplo, mediante la interfaz gráfica, el usuario puede cambiar el color de la lámpara en un selector de colores que se muestra en la pantalla cuando esta se detecta desde la cámara.



También explican cómo la principal ventaja de combinar la RA y los objetos inteligentes es que evita que el usuario tenga que memorizar las correspondencias entre los objetos y las interfaces que controlan esos objetos. El motivo es que haciendo uso de la RA el usuario simplemente debe "visualizar" el objeto que desea manipular para poder utilizar su interfaz "virtual".

La principal diferencia entre esta plataforma, Reality Editor y SARP, es que mientras la primera está orientada a que el usuario modifique el comportamiento de los objetos inteligentes, SARP permite describir el comportamiento de los objetos virtuales. Esto supone que, en Reality editor, una vez creada la implementación para un objeto inteligente concreto, la RA es utilizada para modificar el comportamiento "predefinido", mientras que en SARP ese comportamiento se especifica en el momento de crear la aplicación. En este caso, SARP no se plantea como una mejora, sino como una propuesta diferente enfocada a la creación de contenido para aplicaciones de RA.

Otras propuestas, como SmartX Virtuality muestra una introducción a un sistema que soporta esta combinación entre RA y IoT pero orientado al soporte de dispositivos con menos recursos, usando el bluetooth como principal tecnología para la comunicación objetos-servidor y un bajo consumo de la memoria en los dispositivos (Singh et al. 2015). (Lee et al. 2007) también muestran un sistema basado en realidad aumentada que permite interactuar con componentes conectados a una red doméstica mediante gestos, como una televisión, un ordenador o una videocámara. Para ello hace uso del protocolo UPnP (Universal Plug and Play) permitiendo al usuario controlar los dispositivos de un modo similar a un control remoto.

PROPUESTA: PLATAFORMA SARP

Se han presentado los problemas actuales para crear contenido que aproveche las ventajas de la realidad aumentada y los objetos inteligentes, así como las ventajas de combinar estas dos tecnologías. Como posible solución a estos problemas se propone SARP, una plataforma basada en MDE que aprovecha estas ventajas permitiendo a los usuarios elaborar aplicaciones combinando RA y objetos inteligentes de un modo unificado y sencillo.

7. COMBINANDO REALIDAD AUMENTADA Y OBJETOS INTELIGENTES

La plataforma propuesta no pretende sustituir las soluciones actuales, que permiten crear aplicaciones de RA de un modo efectivo, sino que el objetivo es ampliar la funcionalidad añadiendo el soporte necesario para integrar de la forma más natural posible los objetos inteligentes en las aplicaciones de RA.

Por ese motivo, los procesos utilizados en SARP para crear una aplicación son muy similares a los de la mayoría de las plataformas analizadas, con un proceso nuevo encargado de gestionar esta integración. Cada proceso se corresponde con una capa en la plataforma, formada en total por cuatro de estas. Tres de estas capas se implementan en un servidor que centraliza gran parte de las operaciones: la *Capa de Edición* o *Edition Layer*, la *Capa Persistencia* o *Persistence Layer* y la *Capa de Comunicación* o *Comunication Layer*, encargadas de la creación y almacenamiento de las aplicaciones de los usuarios, así como de comunicar los dispositivos cuando son requeridas, y que se explican en detalle en el apartado *Descripción de las capas* / procesos. Del mismo modo, el servidor sigue una Arquitectura Orientada a Servicios (Service Oriented Architecture o SOA) en la que los dispositivos de RA invocan esos servicios.

La cuarta capa es la encargada de gestionar el comportamiento de los objetos virtuales, así como su interacción con el usuario y con los objetos inteligentes que pudieran representar. Esta capa, además, no se encuentra en el servidor, sino que se ejecuta en cada dispositivo cliente en las aplicaciones creadas en SARP.

Esta capa recibe el nombre de *Capa de Procesamiento* o *Processor Layer,* y es la responsable de añadir el nuevo proceso de integración con los objetos inteligentes. En esta capa el comportamiento de los objetos virtuales se controla definiendo los *estados* en los que un objeto puede encontrarse. Aunque esta solución no es la más completa desde la perspectiva de la interacción con los objetos inteligentes, puesto que se omite parte de la información tanto de lectura como de escritura en estos, se ha elegido este método para controlar el comportamiento de los objetos por dos motivos: El primero es que la definición de los posibles estados en un sistema es el medio principal para describir su comportamiento en ingeniería del software (Dumas y Hofstede 2001) y además se puede representar formalmente por medio de diagramas de estado en UML (Unified Modeling Language) (Harel y Naamad 1996). Por otro lado, aunque esto limita la interacción con los dispositivos (suponiendo las persianas de una casa inteligente, estas podrían abrirse o cerrarse, pero no abrirlas exactamente el 39%, por ejemplo), esto no impide demostrar la interacción entre las personas y los objetos inteligentes por medio de la realidad aumentada, que es el objetivo principal, y ha permitido simplificar notablemente la implementación del prototipo propuesto para la plataforma.

De este modo, los estados se representan mediante valores numéricos naturales (0..n) y se definen mediante dos conceptos, *transiciones* y *condiciones,* que se almacenan en la capa de persistencia en el momento de crear la aplicación. Una transición se define especificando el estado anterior en el que el objeto debe encontrarse para poder cambiar al estado nuevo. Las condiciones definen las restricciones para cambiar al nuevo estado. En SARP se permiten tres tipos de condiciones, *eventos asociados al objeto*, como por ejemplo, que el usuario visualice el objeto, *eventos asociados a componentes de la interfaz de usuario del objeto*, por ejemplo, el usuario pulsa uno de los botones del objeto, y *estados de otros objetos*. Un estado puede definir 0 o más condiciones y uno o más estados predecesores por condición. En cada estado, se puede definir además, si el objeto es visible o no para el usuario. Del mismo modo que los estados, las transiciones y condiciones se almacenan como valores numéricos. Las transiciones con un valor: el estado anterior, y las condiciones con dos: el identificador del componente condicionante y el estado de ese componente (ej. el id de una imagen y 1 representando que el usuario la ha visualizado), aunque en el momento de crearlas, el usuario no utiliza estos valores numéricos, sino palabras reservadas del DSL que los representan (como "*selected*").

La capa de procesamiento aporta una gran flexibilidad a las aplicaciones creadas en la plataforma, integrando los objetos inteligentes y combinando ambas tecnologías de un modo sencillo. Independientemente de si los objetos virtuales están asociados o no a un objeto

inteligente, el usuario visualiza el mismo tipo de interfaz. Sin embargo, cuando un objeto virtual se corresponde con un objeto inteligente, ambos compartirán el estado en el que se encuentran de forma sincronizada. De este modo, adaptar un objeto inteligente a la plataforma se reduce a permitir recibir el estado del objeto virtual y actuar dependiendo de ese estado del objeto.

8. *ARQUITECTURA DE SARP*

La plataforma está formada por cuatro capas en las que se realizan los diferentes procesos que intervienen desde la elaboración de las aplicaciones de RA, hasta su ejecución en los dispositivos clientes. Cada una de las capas interactúa con las demás, describiendo el funcionamiento de la plataforma del siguiente modo:

En primer lugar, el usuario inicia el proceso en la capa de edición, ya sea mediante el editor textual o el gráfico. Aquí el usuario define su nueva aplicación. Esto es, los elementos virtuales que aparecen en ella y su aspecto, así como su funcionalidad/comportamiento o posición física que les corresponda. A continuación, esta información es serializada en formato JSON y se envía a la capa de persistencia, donde es almacenada en la base de datos una vez que se ha procesado. En este punto la aplicación ya estaría creada y disponible en la plataforma.

A continuación, un usuario con su dispositivo, solicitaría esta aplicación. Para ello consulta la lista de aplicaciones disponibles en la plataforma mediante una aplicación nativa que se comunica con el servidor. El usuario selecciona una y el servidor le envía los datos necesarios para cargarla en la capa de procesamiento. La comunicación entre el dispositivo cliente y el servidor se realiza a través de la capa de comunicación, quien contiene los servicios encargados de gestionar la información de la plataforma.

Esta capa tiene una gran importancia durante el uso de una aplicación, puesto que debe servir diferentes datos en los momentos adecuados. En primer lugar, los objetos virtuales que están asociados a una posición geográfica se cargan en la aplicación por demanda (lazy load), esto quiere decir que no todos los objetos virtuales se envían al inicio de la aplicación, sino que, a medida que el usuario se desplaza físicamente por un lugar, la capa de procesamiento se encarga de enviar la posición del usuario al servidor, y la capa de comunicación decide qué objetos virtuales enviarle, dependiendo de la distancia al usuario (esta distancia se puede modificar desde la aplicación de RA). Por otro lado también se encarga de un aspecto muy importante en la plataforma: establecer la conexión entre un

objeto inteligente y su representante virtual en la aplicación. Para ello, cuando un usuario interactúa en la aplicación de RA con un objeto virtual que se corresponde con uno inteligente, la capa de procesamiento solicita a la capa de comunicación los datos necesarios para establecer una conexión con el objeto en sí. La capa de comunicación consulta la BB.DD y si el objeto está registrado en la plataforma, y disponible para aceptar conexiones, le envía los datos a la capa de procesamiento, que establece una conexión con el objeto inteligente, ahora independientemente del servidor.

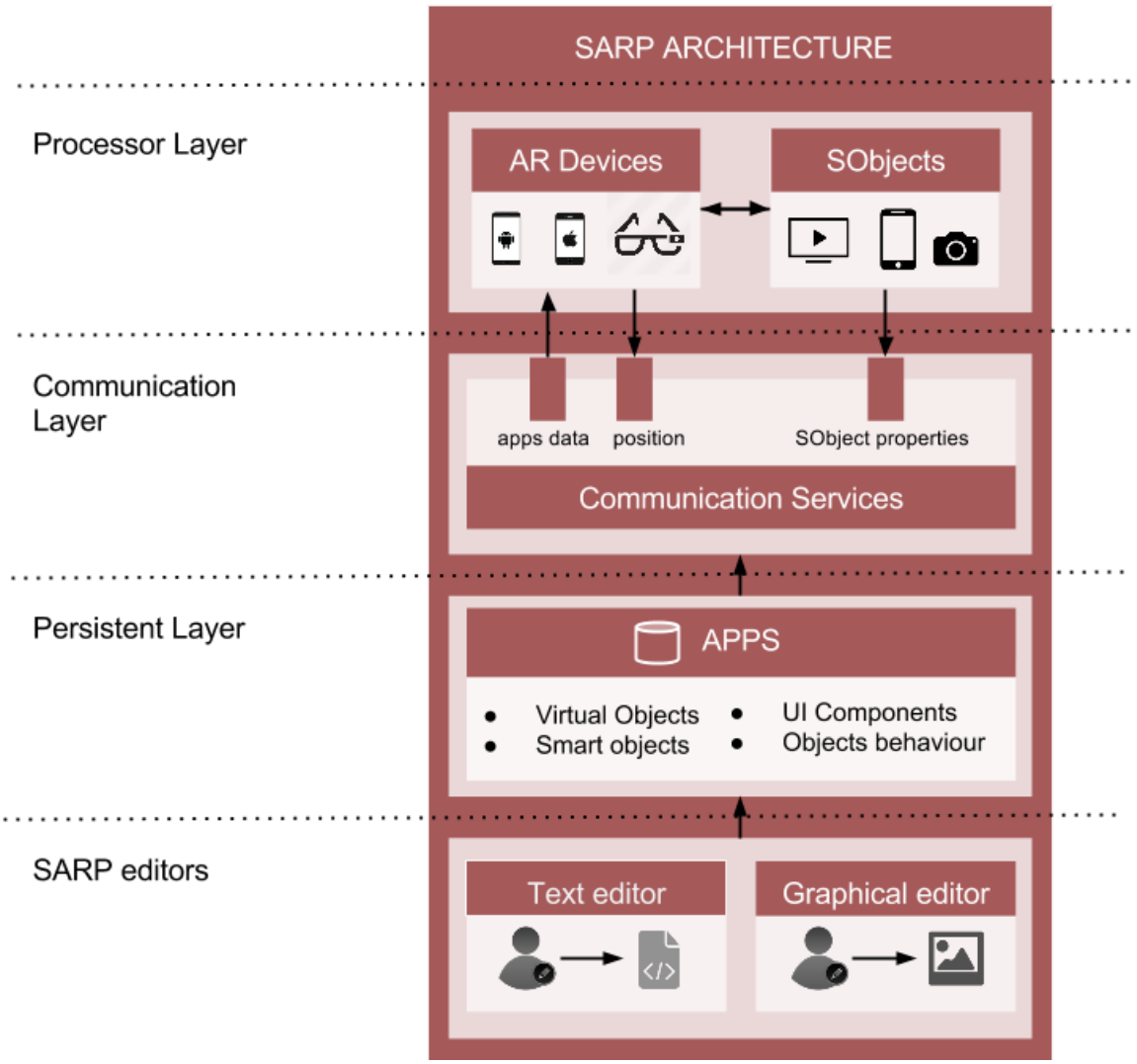


Figura 13 Modelo de capas propuesto en la plataforma

9. *COMPONENTES*

Cada una de las capas comentadas está formada, a su vez, por un conjunto de componentes que intervienen en el funcionamiento de la plataforma. Para comprender de forma general

la función de cada uno de estos, se describen a continuación sus principales características:

9.1. EDITOR TEXTUAL (PLUGIN PARA ECLIPSE)

El usuario dispone en la plataforma de dos editores para elaborar la aplicación, el primero de ellos se corresponde con el editor textual. Desarrollado como un plugin para eclipse y basado en el DSL de la plataforma, este editor facilita al usuario la creación de aplicaciones mediante una sintaxis resaltada, sugerencias de código y detección de errores durante la edición. Una vez descrita la aplicación, el plugin se encarga de preparar la información que la define y la envía al servidor para que la procese y sea almacenada en la base de datos.

9.2. EDITOR GRÁFICO (WEB)

El otro editor, el gráfico, se corresponde con una aplicación web desde la que el usuario define la aplicación de forma más visual, mediante formularios y componentes web que le guían y ayudan durante el proceso de elaboración. Aunque este editor se encuentra en el servidor, no se almacena directamente la información, si no que se utilizan los servicios de la plataforma, del mismo modo que el plugin de eclipse. De esta manera se consigue una mayor independencia entre el editor y la plataforma, por lo que sería sencillo trasladar el editor a un servidor apartado de los servicios.

9.3. SERVIDOR

El servidor es la pieza central de la plataforma. En él se encuentra la base de datos, los servicios, y el editor gráfico (web). Desde el servidor se gestionan las comunicaciones y se procesan los datos para crear las aplicaciones. Los servicios están disponibles a través de una interfaz basada en servicios web fullRest, que utiliza el formato JSON. Estos servicios son los encargados de procesar y almacenar los datos cuando se crea una aplicación, enviar esos mismos datos a los dispositivos de RA cuando lo solicitan, gestionar los objetos inteligentes registrados en la plataforma y de su comunicación con las aplicaciones de RA.

9.4. OBJETOS INTELIGENTES

Como se ha explicado, la plataforma propuesta no proporciona mecanismos demasiado complejos para soportar o registrar objetos inteligentes, sin embargo, se propone un protocolo de comunicación para demostrar la interacción con estos. De este modo, actualmente, para que un objeto inteligente esté disponible en la plataforma, este tiene que cumplir tres condiciones. La primera es que implemente una interfaz de servidor de web sockets (ws). La segunda es que, a través del servicio de registro en la plataforma, indique los datos necesarios para establecer una conexión de ws (es decir, su dirección IP, el puerto de escucha y su dirección MAC como identificador). La tercera es que el servidor ws imple-

mente al menos un método que reciba un valor numérico, correspondiente con el estado de un objeto virtual sincronizado con este, y otro método que envíe el estado del objeto inteligente si este varía, a los dispositivos que hubieran establecido una conexión. Para simplificar la adaptación de los objetos inteligentes a estos requisitos, se proporciona una librería en Java que gestiona la conexión con el servidor e implementa la interfaz de servidor de web sockets, también obtiene automáticamente los datos necesarios para enviarlos al servidor.

9.5. DISPOSITIVO DE RA

Para adaptar los dispositivos de RA a la plataforma, se ha utilizado el SDK de Wikitude. Este SDK permite elaborar aplicaciones de RA para distintas plataformas, como Android, iOS o algunas Smart Glasses, como Google Glass, Vuzix Smart Glass o Epson Moverio. La implementación de las aplicaciones creadas con este SDK se basa en dos partes. Una parte se corresponde con código nativo de cada plataforma, dedicado a las operaciones básicas de RA (acceso a la cámara, red, brújula, componentes gráficos…), y que es independiente del funcionamiento de la aplicación. El funcionamiento en sí, y el contenido de la aplicación es la segunda parte. La lógica de la aplicación y el contenido, se describen en este SDK mediante el uso de tecnologías web (HTML, CSS y JavaScript) y que, por tanto, es común a todas las implementaciones de cada plataforma. Este aspecto ha sido muy importante puesto que ha permitido que los usuarios de RA deban instalar únicamente una aplicación nativa en su dispositivo y desde esa misma aplicación, pueden acceder a todas las aplicaciones de RA de SARP. Esta aplicación se denomina en la plataforma como “Exploradora”. En esta aplicación, el usuario consulta la lista de aplicaciones en SARP. Una vez que selecciona una, el servidor le envía la información que la describe. Esta información es procesada en la capa de procesamiento, y posteriormente se muestra al usuario.

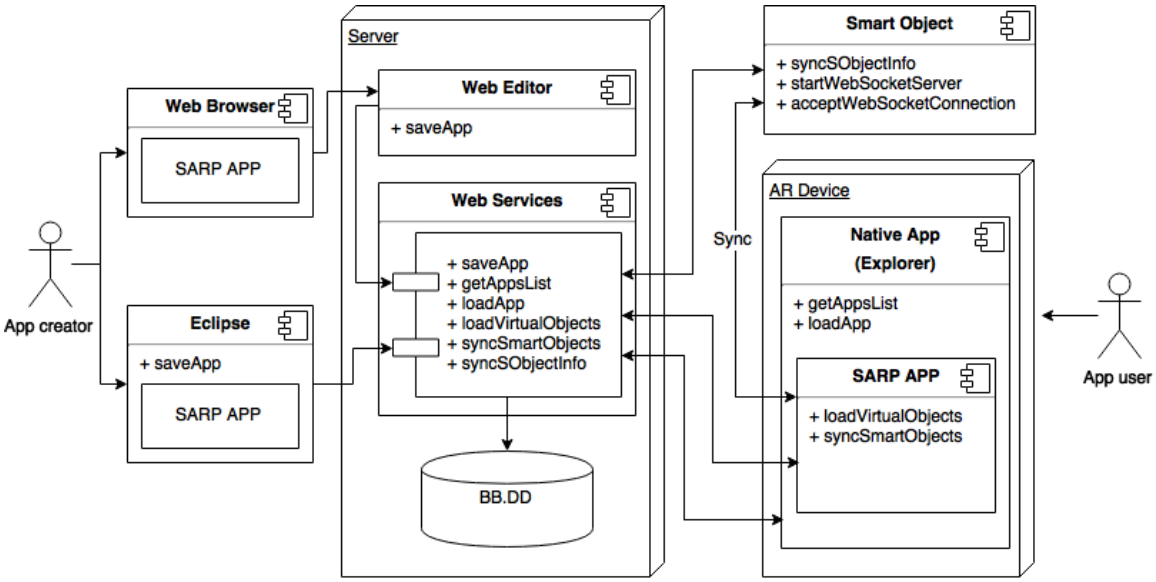


Figura 14 Componentes principales de la plataforma

10. INGENIERÍA DIRIGIDA POR MODELOS EN SARP

A continuación se describen los componentes del metamodelo en el que se basa la plataforma para definir los modelos del dominio del problema: creación de aplicaciones de realidad aumentada con soporte para objetos inteligentes.

En el metamodelo se pueden diferenciar tres partes relacionadas con el dominio. La primera es relativa a los conceptos implicados en las aplicaciones de RA. Debido a que en la plataforma se hace uso de un SDK para crear las aplicaciones, la definición de esta parte del metamodelo queda prácticamente delimitada por el mismo, lo cual tiene sentido si se tiene en cuenta que el objetivo de un SDK es modelar software para un determinado dominio, en este caso, el SDK de wikitude que permite crear aplicaciones de RA.

La segunda parte es la relacionada con los objetos inteligentes en la que se definen sus propiedades y relaciones con los componentes de la aplicación. No se han añadido aspectos de IoT más complejos al metamodelo debido a que el objetivo de este trabajo es introducir las ventajas de la combinación de ambas tecnologías, y no proporcionar un sistema completamente funcional, aunque si se propone como trabajo futuro.

La tercera parte describe los componentes que permiten controlar el comportamiento de los objetos virtuales, que como se ha explicado y justificado, se basa en la definición de los estados de los objetos. Esta parte representa por tanto el aspecto más importante en el metamodelo, correspondiéndose con la contribución principal de la plataforma.

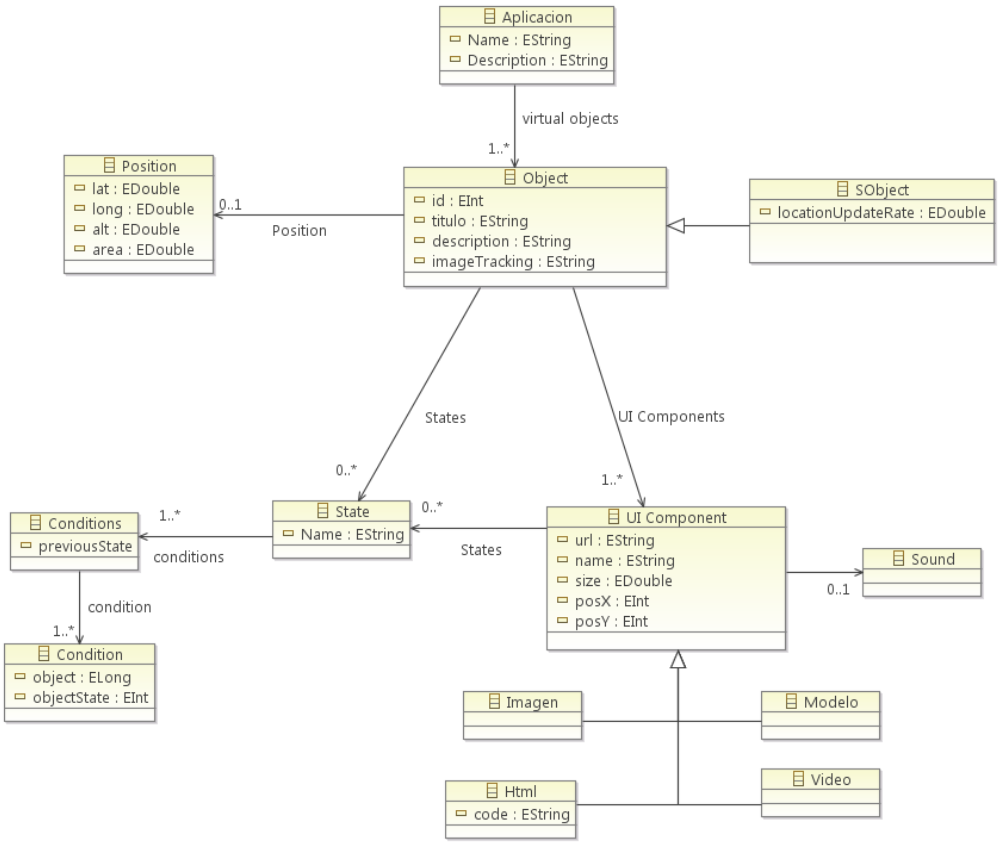


Figura 15 Metamodelo utilizado en SARP

En la Figura 15 se muestra el metamodelo utilizado por la plataforma con los tres aspectos principales antes comentados. "Object" se corresponde con los objetos virtuales, y representa la información que el usuario quiere mostrar en la aplicación. Esta información se relaciona con el entorno real de dos posibles formas: añadiendo un identificador a un objeto real (como el código QR que aparece en multitud de artículos de compra, como una prenda de vestir) o mediante una imagen de rastreo (ambas se indica mediante una URL). La segunda opción es utilizar una posición física (coordenadas geográficas), que se especifican en "Position" a través de los atributos latitud y longitud. Para realizar la asociación con un objeto inteligente, además de cualquiera de los dos métodos anteriores, se deberá especificar su identificador (habitualmente su dirección MAC).

"UI Component" representa un componente de la interfaz de usuario mostrada en la aplicación por cada objeto virtual, y puede ser de cuatro tipos (imagen, html, modelo o video). Los cuatro componentes se especifican indicando la dirección en la que se encuentra el recurso en sí, y en el caso del HTML se puede especificar directamente el código que lo define.

"State" se corresponde con los estados que definen el comportamiento del objeto virtual. Además se puede especificar si el objeto se muestra o no en ese estado a través del atributo "visible". Aunque el usuario define los estados del objeto, los componentes de la interfaz de usuario ("UI Component") también se representan mediante estados, y el usuario los puede utilizar para añadir condiciones basadas en la interacción con estos. En este caso, estos estados se limitan a "seleccionado" o "no seleccionado". A su vez, cada estado está definido por una o más condiciones, descritas por los estados predecesores válidos para cambiar al nuevo estado, y el estado del objeto virtual o componente del objeto como restricción para ese cambio. Con los estados añadidos al modelo, el usuario es capaz de definir de un modo elaborado el comportamiento de los objetos virtuales. De este modo, suponiendo la aplicación hipotética descrita en la introducción, en la que el requisito era apagar el reproductor musical si se encendía la televisión, se podría especificar del siguiente modo:

```
Reproductor musical->State( 0 / apagado) -> Conditions(previousState: 1 / encendido)
->condition( idTelevesión, 1 / encendida)
```

Esto podría leerse como: El reproductor de música cambia al estado 0 (apagado), si su estado anterior era 1 (encendido) y si el objeto idTelevesión (la televisión) cambia al estado 1 (encendido). Esto podría aplicarse del mismo modo a objetos virtuales "simples". Por ejemplo, un objeto virtual podría representar una pregunta con múltiples respuestas, haciendo uso de cuatro componentes de tipo texto (una pregunta y tres posibles respuestas), el estado del objeto varía en este caso dependiendo de la respuesta que seleccione el usuario. Si el objeto cambiase al estado 3, y esa fuese la respuesta correcta, podría indicarse el atributo "visible" a false y ocultar el cuestionario. A la vez, un segundo objeto podría cambiar al estado "visible" si el cuestionario cambia al estado 3, mostrando una imagen indicando que el usuario ha respondido correctamente.

11. LENGUAJE DE DOMINIO ESPECÍFICO

A partir del metamodelo se realizaron las conversiones a dos sintaxis concretas, correspondientes con los editores textual y gráfico. Los componentes del lenguaje se diseñaron con el objetivo de resultar del modo más natural posible a los usuarios de la plataforma, de modo que los elementos que intervienen en una experiencia de RA se correspondan de forma directa con componentes o nodos del lenguaje.

De esta manera, el primer nodo que aparece en una aplicación elaborada en la plataforma es "Application", en el que se indica el nombre y descripción de la misma. Información que será mostrada a todos los usuarios cuando acceden desde su dispositivo de RA.

```
application: "Aplicacion de prueba"
description: "Descripcion de la aplicacion"
```

Los siguientes componentes definen los elementos que se muestran en la aplicación así como su comportamiento. En una aplicación de realidad aumentada, cualquier "cosa" puede identificarse con un objeto virtual e interactuar con el usuario, motivo por el cual se ha escogido "**Object**" como nombre para designar cualquier elemento que intervenga en la aplicación. Como atributos, se pueden especificar aquellos que permiten definir cómo se muestra la información al usuario, esto es, el **nombre** y la **descripción**, el **tamaño** del mismo (con las medidas que se usan en el SDK de wikitude, SDUs, correspondiéndose 1SDU con el tamaño que una persona visualiza cuando observa un objeto de 10 metros de altura a 10 metros de distancia del objeto), por último, se indican los modos en los que el usuario puede acceder a la información del objeto. La primera opción es mediante el uso de coordenadas geográficas, de modo que el objeto se sitúe en una **posición** física. La segunda es indicando una **imagen de rastreo** que permita identificar un patrón en un objeto real y asociarlo con el objeto virtual.

El siguiente fragmento de código se corresponde con parte de una aplicación desarrollada en SARP que muestra el logo de la Escuela de Ingeniería Informática de Oviedo sobre el propio edificio.

```
Object: "Escuela de Ingenería Informática de Oviedo (EIIO)"
       description: "Representative image"
       position: (43.355164, -5.851229)
       size: 10
```

A continuación se especifican los componentes de la interfaz gráfica que forman dicho objeto, denominado "**component**". Este es uno de los aspectos delimitados por el SDK y que se corresponde con cinco tipos de componentes, **texto** simple, **imágenes**, **páginas web**, **videos**, o **modelos** en 3 dimensiones. Sin embargo, como se explica en el apartado del editor textual, el SDK permite utilizar los elementos HTML como componentes "nativos", lo cual permite elaborar prácticamente cualquier tipo de aplicación.

Los atributos de los componentes son comunes entre sí, exceptuando HTML, que dispone de una propiedad más que permite añadir código HTML directamente en el componente.

Este atributo se denomina "**code**". El resto de atributos comunes se corresponden con **size** que indica el tamaño del componente gráfico, también en SDUs, **position**, que especifica la posición relativa al objeto, y **url** que se corresponde con la dirección del recurso a mostrar.

Siguiendo con el ejemplo anterior, el siguiente código representa un componente de tipo "imagen"

```
component:"image_component", IMAGE
  size: 10
  position: 0,0
  url: "https://ingenieriainformatica.uniovi.es/TemaIngenieriaInformatica/
              images/uniovi/logo.png"
```

El último componente es "**state**" que representa el comportamiento de cada objeto virtual en la aplicación. En cada estado se puede indicar un nombre, a modo de identificador para el propio usuario, un estado del objeto (visible o no) y un posible estado previo que representa la transición de un estado a otro. A su vez, cada estado se define por un conjunto de condiciones, denominadas en el lenguaje como "conditions". Cada condición está formada por el estado del objeto en el que se debe encontrar para que la condición sea válida (ya sea el propio objeto o cualquiera que intervenga en la aplicación) y el objeto en sí (para identificarlo).

```
state:"changeVisibility", false
            condition: "image component", state:"selected"
```

El ejemplo anterior añade un estado al objeto "Escuela de Ingenería Informática de Oviedo (EIIO)" que cambia la visibilidad del objeto a "false" cuando el componente "image component" es pulsado, permitiendo visualizar el edificio. En el caso de los componentes, la plataforma asigna automáticamente dos estados (selected y deselected), que se corresponden con pulsaciones de los usuarios sobre los propios componentes.

11.1. EJEMPLO DE RA Y OBJETOS INTELIGENTES

El ejemplo anterior muestra los componentes básicos del DSL y una sencilla aplicación. A continuación, se muestra un ejemplo en el que se combinan objetos virtuales e inteligentes, principal característica de la plataforma.

El objetivo de la aplicación es proporcionar una interfaz virtual a un reproductor de música. Con esa interfaz, se controla la reproducción de una canción, siendo los posibles estados, "parado", "reproduciendo" o "pausado". Para ello, la aplicación cuenta con un objeto

virtual cuya interfaz de usuario está formada por tres componentes gráficos de tipo imagen, uno por cada estado del objeto, que representan los tres botones habituales en un reproductor.

El código correspondiente a los componentes gráficos es el siguiente:

```
application: "Music Player"
description: "Remote control for music player with AR"

SObject: "music_player", "00:11:22:33:44:55"
      description: "Music player controls"
      image: "http://ui-cloud.com/res/qoozon/Music-player-PSD
                /previews/music_player.jpg"
      size: 3

      component:"play", IMAGE
            size: 1
            position: 0,0
            url: " https://www.iconfinder.com/icons/2488
                        /fwd_play_icon#size=128"

      component:"pause", IMAGE
            size: 1
            position: 2,0
            url: " https://www.iconfinder.com/icons/2487
                      /pause_player_icon#size=128"

      component:"stop", IMAGE
            size: 1
            position: 1,1
            url: " https://www.iconfinder.com/icons/2491
                      /player_stop_icon#size=128"
```

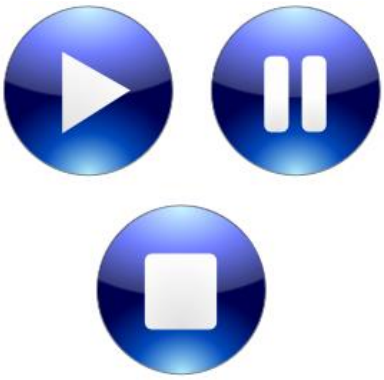


Figura 16 Componentes de la aplicación "Music Player"

La Figura 16 muestra la interfaz que el usuario visualizaría cuando este apunta con su dispositivo de RA hacia el reproductor de música real. Como se puede observar, el código necesario para asociar el objeto virtual con el inteligente simplemente es usar el componente “SObject”, con los mismos atributos que “Object”, a excepción del identificador del

mismo, que se corresponde con la dirección mac del dispositivo, el cual se puede consultar en la plataforma. Como se muestra en el apartado del editor gráfico, en este editor no es necesario consultar la dirección del dispositivo ya que la web carga automáticamente los dispositivos registrados, mostrando un nombre y una descripción en un formulario web. En el caso del editor textual, el usuario deberá consultar los identificadores en una web habilitada para ello.

A continuación se describe el comportamiento de la aplicación, que dependerá de los botones que el usuario pulse y el momento en el que los pulse. El siguiente diagrama muestra de forma gráfica los estados y transiciones que intervienen en la aplicación.

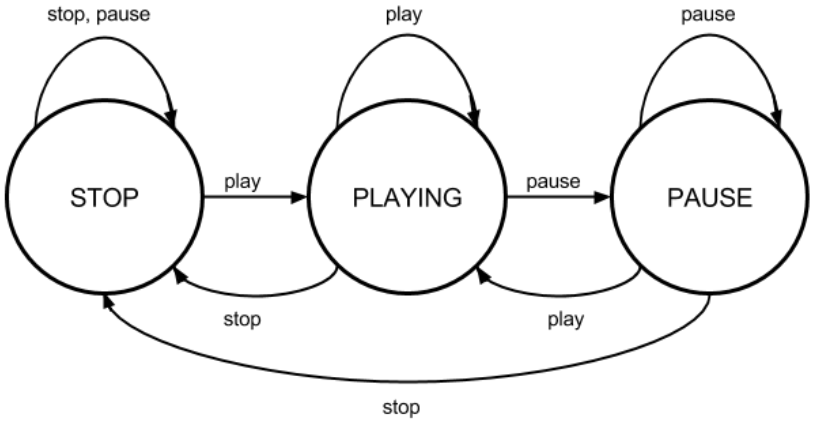


Figura 17 Diagrama de estados de la aplicación "Music Player"

En este caso, se ha elegido el funcionamiento habitual para un reproductor de música, pero el usuario podría definir cualquier otro tipo de comportamiento. Por ejemplo, si se está reproduciendo música y el usuario pulsa “play”, el reproductor podría cambiar al estado “pause”. El código con el que se correspondería este diagrama en SARP sería el siguiente:

```
state:"stop", true
      previousState: "playing"
            condition: "play", state:"selected"
      previousState: "pause"
            condition: "pause", state:"selected"
      previousState: "stop"
            condition: "stop", state:"selected"

state:"playing", true
      previousState: "stop"
            condition: "stop", state:"selected"
      previousState: "pause"
            condition: "pause", state:"selected"
      previousState: "playing"
```

```
            condition: "play", state:"selected"

state:"pause", true
      previousState: "playing"
            condition: "play" state:”selected”
      previousState: "pause"
            condition: "pause" state:”selected”
```

El código mostrado se corresponde con los estados del diagrama de la Figura 17. Como se puede observar, en el código se incluyen los casos en los que el estado y el componente pulsado coinciden, es decir, se mantiene el estado (por ejemplo, se está reproduciendo una canción y el usuario pulsa play). La plataforma soporta esta descripción ya que es necesario en casos de ambigüedad, sin embargo, para simplificar la elaboración de la aplicación, los casos en los que no se especifica, la plataforma se comporta ignorando esos cambios. De este modo, el código anterior se podría simplificar al siguiente:

```
state:"stop", true
      previousState: "playing"
            condition: "play", state:"selected"
      previousState: "pause"
            condition: "pause", state:"selected"

state:"playing", true
      previousState: "stop"
            condition: "stop", state:"selected"
      previousState: "pause"
            condition: "pause", state:"selected"

state:"pause", true
      previousState: "playing"
            condition: "play" state:”selected”
```

De este modo, la aplicación ya estaría completa, y cada cambio de estado en la aplicación ejecutada en un dispositivo se sincronizaría con el reproductor de música. Para que el reproductor se sincronice con el dispositivo, es necesario registrarlo en la plataforma. En este aspecto es donde sería interesante un sistema para crear aplicaciones de objetos inteligentes compatibles con SARP, ya que es necesario programar el dispositivo para que se comporte de acuerdo a los estados sincronizados con la plataforma. Como se ha comentado, esa posibilidad se plantea como trabajo futuro, sin embargo, para facilitar la elaboración de aplicaciones, la plataforma proporciona una librería en Java que permite registrar el dispositivo y gestionar los eventos de un modo prácticamente transparente para el usuario.

```
public class MusicPlayer implements MListener{

private enum STATES{
		STOP,
		PLAYING,
		PAUSE
	}

	@Override
	public void action(Object o) {
		switch (STATES.values()[(Integer)o]) {
		case STOP:
			Player.stop();
			break;
		case PLAYING:
			Player.play();
			break;
		case PAUSE:
			Player.pause();
			break;
		default:
			break;
		}
	}




	/**
	 * @param args
	 */
	public static void main(String[] args) {
		MusicPlayer musicPlayer = new MusicPlayer();
		SARP sarp = new SARP("nombre", "descripcion", musicPlayer);
		try {
			sarp.register();
			sarp.start();
		} catch (IOException e) {
			e.printStackTrace();
		}

	}
}
```

El código mostrado en Java se corresponde con la aplicación que se instalaría en el reproductor. En primer lugar, se registra el dispositivo con un nombre y una descripción en la plataforma mediante la clase SARP, que actúa como "fachada" de la librería de la plataforma (si el dispositivo ya estaba registrado, simplemente se actualizan las propiedades). A continuación se inicia el servidor de websockets, que abre un canal de comunicación para los dispositivos clientes que soliciten una conexión websocket. La librería se encarga de gestionar las conexiones con los dispositivos y con el servidor. Una vez sincronizado con

un dispositivo, cuando este cambia de estado, se notifica a la clase Music Player (que implementa la interfaz "listener") el nuevo estado en el método "action". Una vez que recibe el estado, la gestión del dispositivo se define del modo habitual, independiente de la comunicación. Se ha omitido la implementación de la gestión del reproductor en sí puesto que no es relevante para la explicación de la comunicación con la plataforma. El uso de esta librería en Java es un método "provisional" para conectar los objetos con la plataforma, pero permite demostrar en que consiste el protocolo de comunicación. Como se puede observar, basar la arquitectura y el comportamiento de los objetos en estados, permite simplificar considerablemente la comunicación. Por supuesto, sería necesario un método más orientado a la interoperabilidad, que permitiera la comunicación independientemente del lenguaje del dispositivo (Atzori, Iera, and Morabito 2010), pero el proceso en sí prácticamente no se modificaría.

12. EDITOR TEXTUAL

La conversión de la sintaxis abstracta a la concreta se realizó utilizando Eclipse Modeling Framework. Este framework permite describir de un modo sencillo tanto la sintaxis del DSL (mediante reglas y expresiones regulares) como los artefactos o código generado cuando se procesa el programa escrito en el DSL, para lo cual el framework proporciona el lenguaje Xtext. Además, una vez diseñado e implementado el DSL, el framework proporciona un entorno de desarrollo para el mismo, a modo de un plugin para Eclipse.

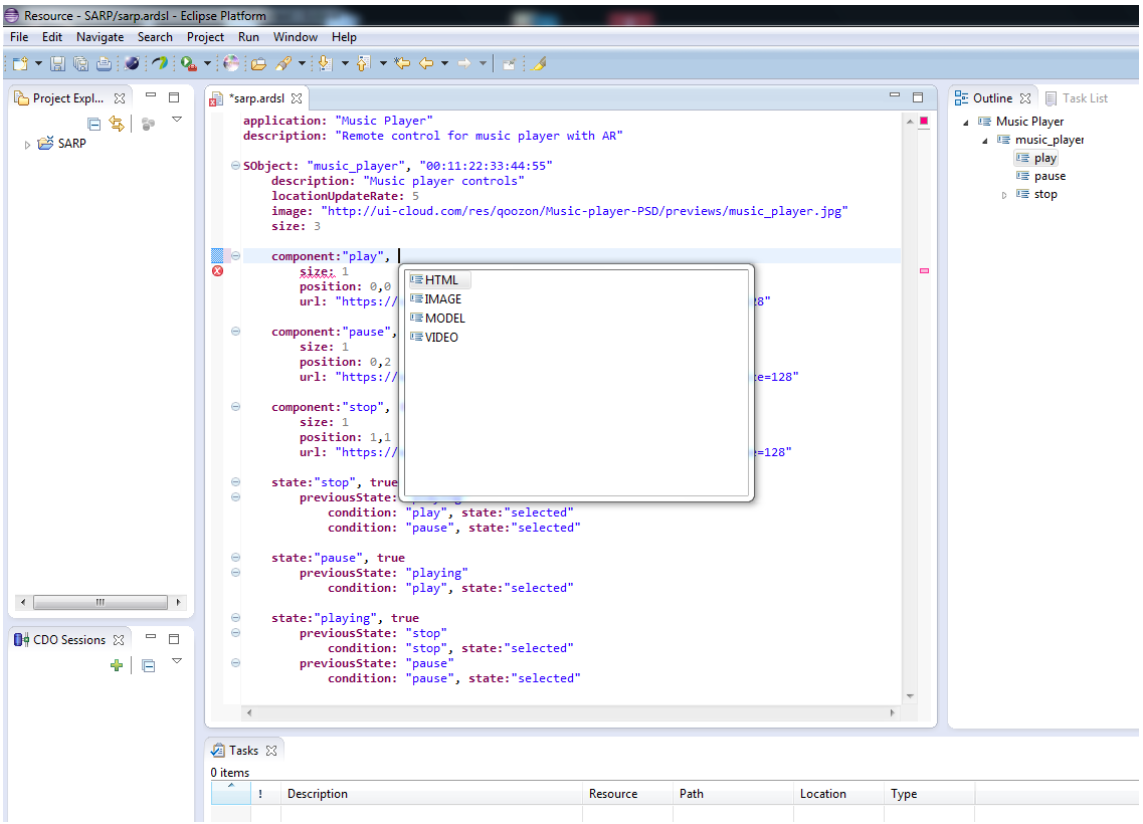


Figura 18 Entorno de desarrollo para el DSL textual de SARP

En la Figura 18 se muestra el plugin de eclipse elaborado para el editor textual de la plataforma. El plugin proporciona todas las características y ventajas disponibles habitualmente en Eclipse. Como se puede ver en la imagen, en la parte izquierda del editor se muestra una ventana con un explorador que permite gestionar los ficheros del proyecto. La ventana de la derecha muestra de forma resumida los elementos contenidos en la aplicación del DSL. La ventana central se corresponde con el editor de código. Esta ventana, además de mostrar el código con la sintaxis del DSL resaltada, detecta errores en el código de la aplicación y ofrece sugerencias mediante la función autocompletar de los componentes del DSL disponibles en la parte de código que se está escribiendo, lo cual facilita considerablemente esta labor en el editor textual.

13. *EDITOR GRÁFICO (APLICACIÓN WEB)*

El editor textual proporciona un medio simplificado para crear las aplicaciones en la plataforma. Sin embargo, sigue tratándose de un lenguaje de programación. Para alguien que no esté acostumbrado a usar un lenguaje de programación, esto puede resultar una tarea complicada. Por ese motivo, además del editor textual, se ha elaborado un editor gráfico, partiendo también de la sintaxis abstracta. El editor gráfico se corresponde con una apli-

cación web. A continuación se explican los apartados de los que se compone la aplicación web con el mismo ejemplo de la aplicación del reproductor de música para facilitar la comparación entre ambos editores.

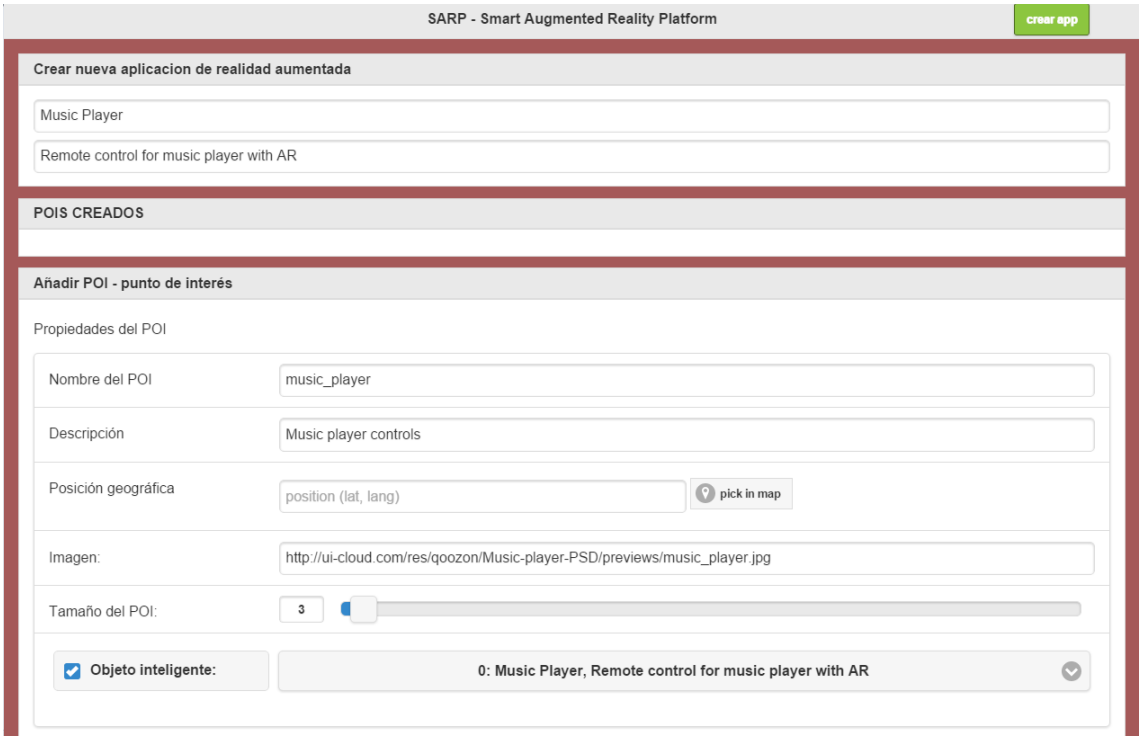


Figura 19 Editor gráfico (aplicación y propiedades de los objetos virtuales)

La Figura 19 muestra la parte correspondiente a la descripción de la aplicación, las propiedades de un objeto virtual y los objetos virtuales creados. En este apartado se muestran también los objetos creados (Objetos creados en la web). Como se puede observar, en este apartado aparecen dos utilidades que facilitan considerablemente la creación de la aplicación. La primera es que se puede seleccionar una ubicación física en un mapa de Google.

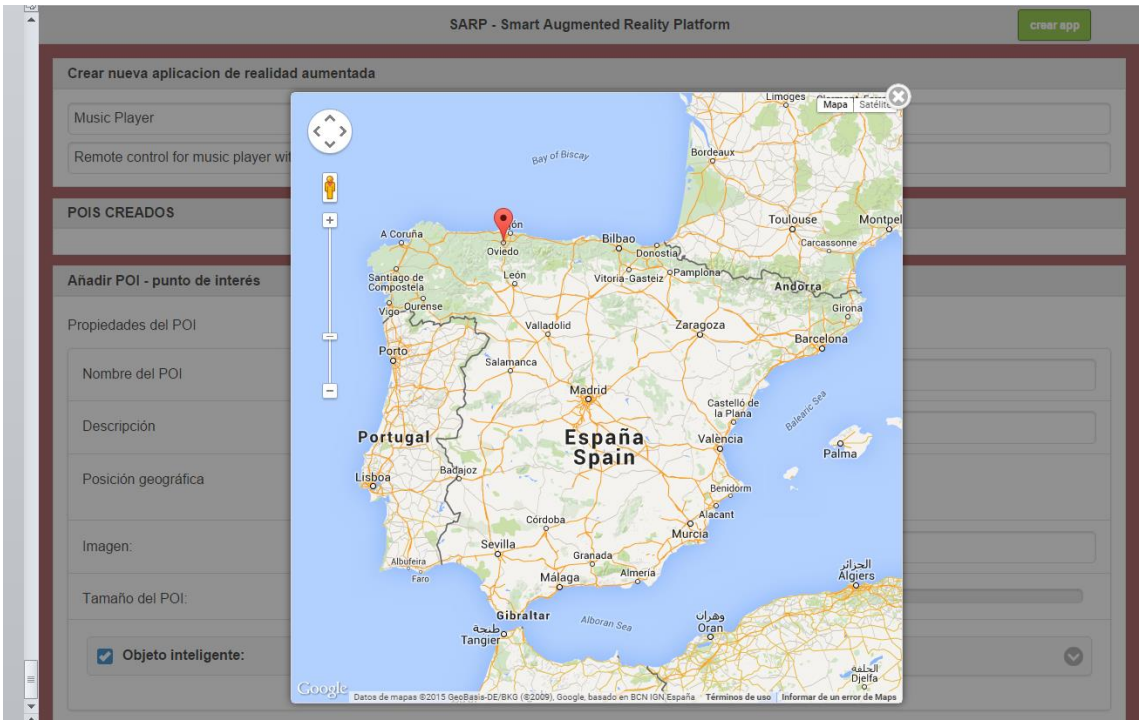


Figura 20 Mapa de Google mostrado en la aplicación web para seleccionar la posición física del objeto virtual

La segunda, mostrada en la Figura 19, es el formulario con el que se asocia el objeto inteligente. En el editor textual, el identificador utilizado debía de ser el que se corresponde con la base de datos, es decir, la dirección mac del dispositivo. Para ello, el usuario debe consultar en la plataforma el identificador del objeto inteligente que desea asociar. Esto supone dos inconvenientes. El primero es los posibles errores asociados a utilizar un identificador poco intuitivo, pero necesario para la correcta identificación de los mismos. El segundo es un problema en el propio código editado, ya que si el usuario no recuerda el objeto inteligente que había seleccionado, deberá consultarlo de nuevo en la plataforma.

En la web, mediante una petición a unos de los servicios de la plataforma, se cargan dinámicamente los dispositivos disponibles junto a sus nombres y descripciones. En este caso, la plataforma se encarga de la gestión de los identificadores y el usuario simplemente debe seleccionar el objeto que quiere utilizar.

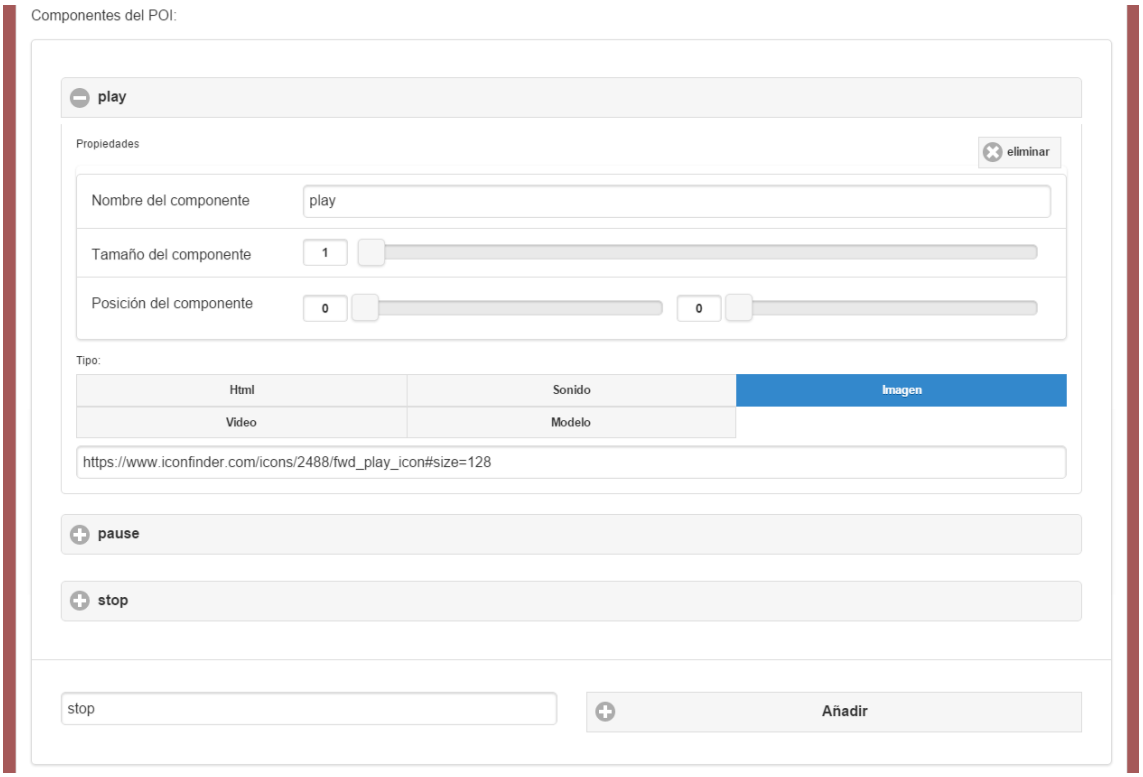


Figura 21 Editor de los componentes del Objeto Virtual

La Figura 21 se corresponde con el apartado del editor de componentes de los objetos virtuales. Las opciones disponibles son las mismas que en el editor textual, pero en este caso se establecen algunas restricciones que ayudan a evitar errores en la edición de los componentes. Por ejemplo, el tamaño del componente no puede ser mayor que el del objeto virtual. Del mismo modo, no se puede establecer una posición relativa del componente mayor que el tamaño del objeto (si la medida del objeto es 3, y la del componente es 1, su mayor desplazamiento en el eje x será 2). También se muestra la opción de editar código HTML si se elige este tipo de componente.

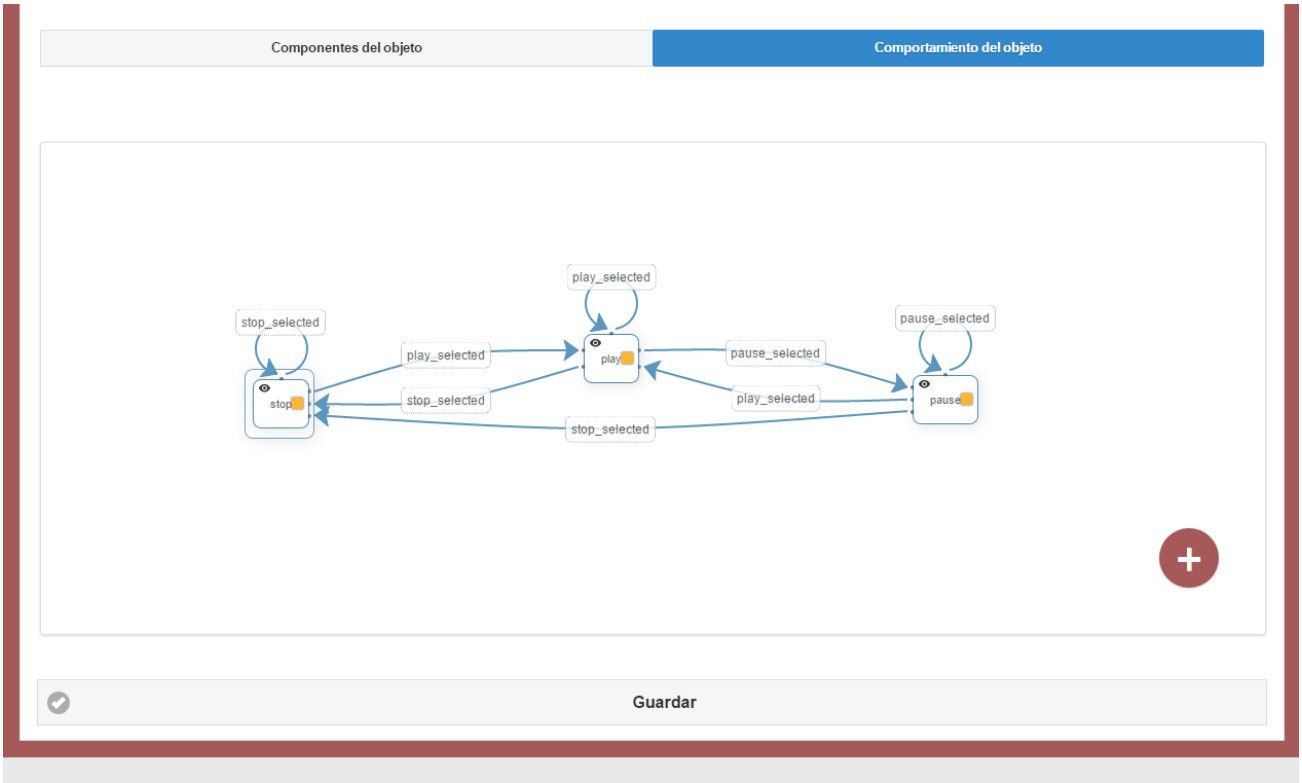


Figura 22 Editor gráfico del comportamiento de un objeto virtual en SARP

La edición del comportamiento de un objeto virtual en el editor gráfico es mucho más visual que en el textual, mostrando un editor de diagramas de estado. El editor permite arrastrar los componentes que representan los estados, tanto para modificar su posición, como para definir las transiciones. Las transiciones también se definen de forma guiada. Cuando el usuario define una transición, se muestra un dialogo indicando los componentes disponibles y sus condiciones/estados asociados.

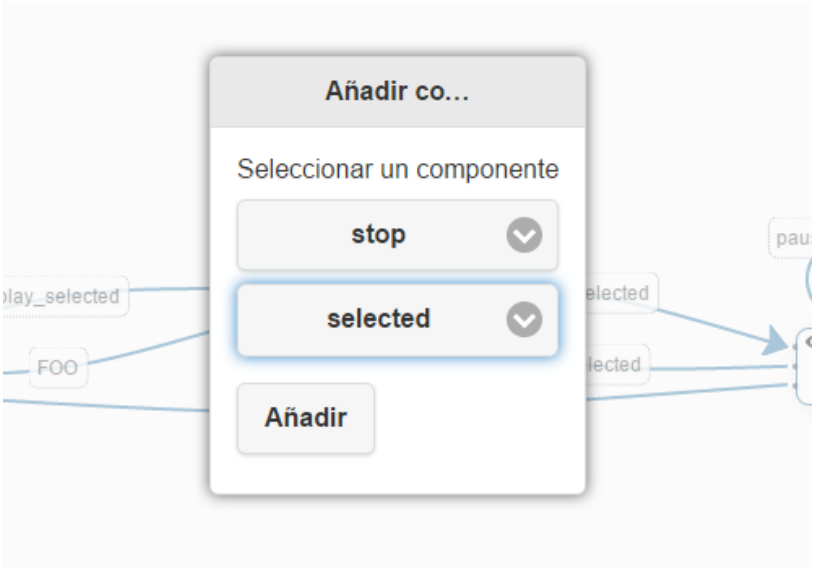


Figura 23 Dialogo mostrado para definir una transición

14. DESCRIPCIÓN DE LAS CAPAS / PROCESOS

A continuación se describe en mayor detalle el funcionamiento de cada una de las capas de la plataforma y cómo interactúan con las demás. Se muestran también los protocolos utilizados en la comunicación entre los componentes y los principales datos almacenados en la plataforma, necesarios para gestionar las aplicaciones creadas por los usuarios. La descripción se realizará en el mismo orden en el que intervendrían las capas desde la creación de la aplicación hasta su uso en un dispositivo cliente. Comenzando por la capa de edición en la que el usuario describe la aplicación, seguida por la capa de persistencia, en la que se procesan y almacenan los datos de esta, a continuación la capa de comunicación, que envía esos datos a un dispositivo cliente cuando la aplicación es requerida y, por último, la capa de procesamiento, situada en los dispositivos de RA y que gestiona el funcionamiento de los objetos virtuales.

14.1. CAPA DE EDICIÓN O "EDITION LAYER"

Una vez que se ha definido la aplicación, los datos que la describen se envían a la capa de persistencia, donde se almacenan en la base de datos. Para ello, los datos que la definen son serializados en formato JSON y enviados a esta capa a través de un servicio web, desde el plugin de eclipse o desde la aplicación web (el que elija el usuario).

```
{
"name":"Music player",
"description":" Remote control for music player with AR",
"objects":[
    "title": "Music player",
    "description": "music player remote controls",
    "sobject": "F8:DD:8F:81:A3:EE",
    "geopos":{
        "latitud": "43.368179268119",
        "longitud": "-5.8650577068329",
        "altitud": "0"
    },
    "graphicUI": [{
        "url":
"https://cdn1.iconfinder.com/data/icons/realistiK-new/128x128/actions/player_play.png",
    "idComponent": "1",
     "type": "0,          //Image
     "size": "1",    //Size of component in SDUs
     "position":{
        "x": "0",
        "y": "0"
     }
```

```
"stateManager": "
[{
    "name": "playing",
    "conditions": [[{
          "componentId":  "0",          //stop button
          "component": "2",
          "state": "selected",
          "idPreviousState": "2"
          },
        {
        "componentId": "1",       //play button
          "component": "2",
          "state": "selected",
          "idPreviousState": "1"
         },
         {
          "componentId":  "2",         //pause button
          "component": "2",
          "state": "selected",
          "idPreviousState": "0"
         }
```

}]]	, "visible":true], “state”:1]

Tabla 1Fragmento de la aplicación "Music Player" serializada en formato JSON.

Como se puede observar en el ejemplo de la Tabla 1, donde se muestra un fragmento que contiene la aplicación de ejemplo “Music Player”, la información mostrada es la misma que la especificada por el usuario en cualquiera de ambos editores.

Dependiendo del editor utilizado por el usuario, este JSON se forma de dos modos distintos. En el caso del plugin de eclipse, en primer lugar se transforma el DSL a la aplicación en Java antes descrita, que representa el mismo modelo pero en este lenguaje. El objetivo de esta aplicación es precisamente enviar la información a la capa de persistencia, a través de la capa de comunicación. Para ello, la aplicación en Java se encarga de serializar la información del modelo de datos y enviarla haciendo uso de los servicios web disponibles en la capa de comunicación.

En el caso de la aplicación web, el envío se realiza desde el propio editor, a través del JavaScript que contiene la mayor parte de la lógica del mismo. En este caso no se realiza una transformación, sino que el modelo de datos se obtiene mientras el usuario define su aplicación. Una vez que la ha creado, la información es serializada, obteniendo el mismo JSON que en el plugin de eclipse, y se envía a la capa de persistencia a través de los servicios web.

14.2. CAPA DE PERSISTENCIA O “PERSISTENCE LAYER”

Esta capa es la encargada de almacenar todos los datos que permiten recuperar las aplicaciones creadas para ser enviadas a los usuarios cuando las solicitan. Para ello, el esquema de la base de datos sigue un modelo entidad-relación, que permite representar de forma directa las entidades identificadas en el metamodelo de la plataforma, mostrado en la Figura 15. De este modo, cada entidad del metamodelo se corresponde con una tabla en la BBDD, en la que cada columna representa los atributos de dicha entidad. Del mismo modo, la capa de persistencia se encarga de gestionar la integridad referencial, respetando las relaciones establecidas en el metamodelo. Esta capa solo es accesible a través de la capa de comunicación, que se encarga de realizar las consultas oportunas, como la solicitud por demanda de objetos virtuales.

14.3. CAPA DE COMUNICACIÓN O "COMUNICATION LAYER"

14.3.1. Interfaz de comunicaciones

El envío de la información desde la capa de persistencia hasta los dispositivos se realiza por tanto en esta capa, la capa de comunicación. La petición de los datos, desde el cliente, se hace a través de los servicios web fulREST disponibles en esta capa. Se ha elegido este tipo de interfaz, y no otros como SOAP debido a las ventajas que la primera aporta cuando las comunicaciones se realizan entre multitud de dispositivos con cantidades de información "poco" elevadas (Guinard 2009).

14.3.2. Sincronización entre objetos virtuales y objetos inteligentes

La tarea de esta capa también es gestionar la sincronización entre los objetos virtuales y los objetos inteligentes. La sincronización entre objetos inteligentes gestionados desde un servidor es un problema considerable y habitual en IoT (Zhang and Zhu 2011), debido especialmente a la carga de trabajo derivada en el servidor. Por ese motivo, y tratándose de un trabajo de investigación, se ha considerado oportuno hacer uso de una tecnología que aún situándose en sus inicios, aporta características que la hacen muy interesante para este tipo de arquitecturas. Esta tecnología son los *web-sockets,* y en el caso de esta plataforma, ha permitido reducir la gestión de la comunicación entre objetos inteligentes y los dispositivos clientes a un simple envío de la dirección de los primeros a los segundos y partir de ahí, la comunicación se gestiona entre ambos, sin necesidad de interacción con el servidor. Para ello, el protocolo de comunicación entre el objeto inteligente, el servidor y el dispositivo cliente es el siguiente:

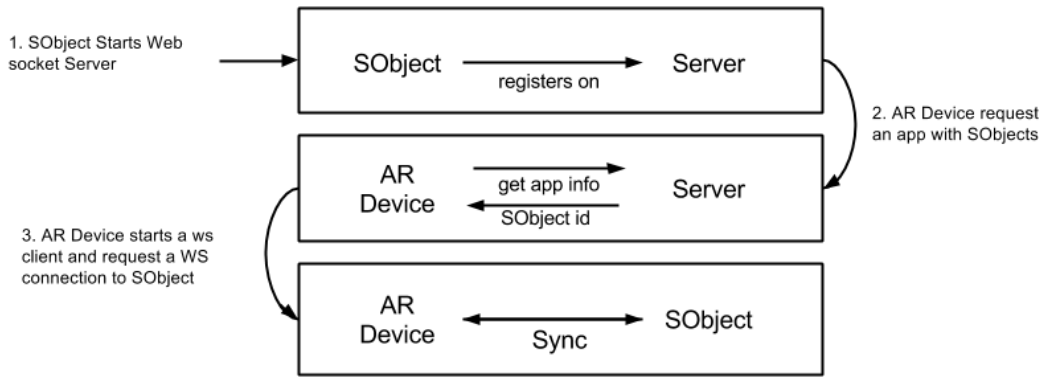


Figura 24 Comunicación entre objetos inteligentes y el servidor

Como se puede observar, cuando el objeto inteligente se registra en SARP, le envía varios parámetros que lo definen, entre ellos su propia dirección. A continuación, si un dispositi-

vo solicita una aplicación que hace uso de objetos inteligentes, el servidor le envía la aplicación y la información relativa a los objetos que intervienen en la aplicación. El cliente solicita establecer una conexión web-socket con cada objeto inteligente y, si están disponibles, aceptan la conexión y mantienen sincronizados sus estados con los objetos virtuales de la aplicación. Esta solicitud es posible ya que cada objeto inteligente registrado en SARP es un servidor de web-sockets.

Por otro lado, la comunicación en sí entre los dispositivos es mínima. Cuando uno de los dispositivos cambia de estado y lo comunica a los demás, la información enviada es simplemente el valor numérico del nuevo estado, sin formato, y que es suficiente puesto no es necesario intercambiar más información para sincronizar los estados de los objetos. Esto es posible gracias a la gestión realizada por los Web Sockets. En el caso de los dispositivos que reciben este valor, lo reciben independientemente de la plataforma y el lenguaje en el que se ejecute, y en el caso de los dispositivos que lo envían, si ocurre un error durante el envío, este es notificado debidamente indicando los posibles motivos.

14.4. CAPA DE PROCESAMIENTO O "PROCESSOR LAYER"

Como se ha explicado, esta capa está compuesta por los objetos virtuales que componen la aplicación. Esta capa no se encuentra en el servidor, sino que está implementada en la aplicación "exploradora" en cada dispositivo cliente, con los datos de la aplicación enviados a través de la api fulREST. En primer lugar se procesan los componentes de la interfaz gráfica de cada objeto virtual, así como su posición o imagen de rastreo. A continuación, se procesan los datos relativos a esta capa. Estos datos son los estados, definidos por sus transiciones y condiciones, que la capa procesa como un *observer* por cada objeto virtual y componente de la interfaz, de modo que si uno de los objetos cambia de estado, aquellos subscritos a los eventos del mismo puedan actuar de acuerdo a ese cambio.

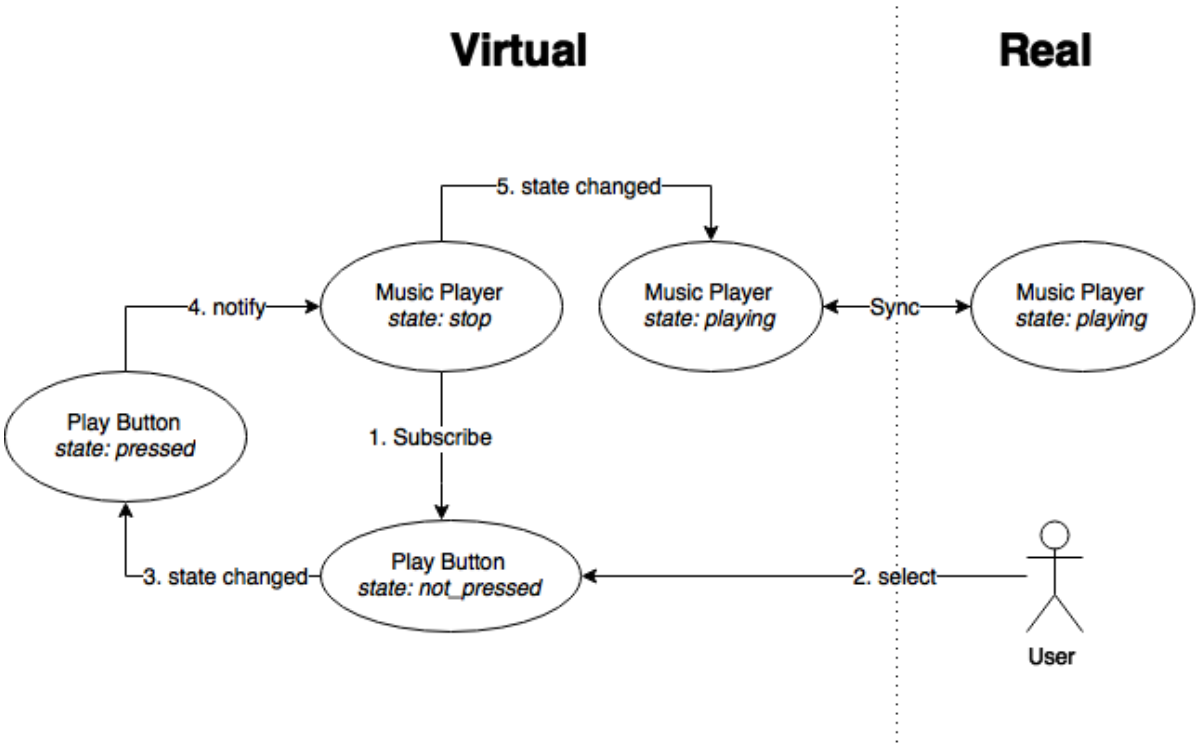


Figura 25 Ejemplo de control de estados en la capa de procesamiento en la aplicación "Music Player"

La Figura 25 muestra una parte del proceso realizado en esta capa para gestionar los estados definidos en la aplicación de ejemplo “Music Player”. Como se puede observar, el objeto virtual que representa el reproductor de musica (real), comienza en el estado “stop”. A continuación, siguiendo las especificaciones, este estado se subscribe a los eventos, o cambios de estado de uno de sus propios componentes, el botón “play”. En esta aplicación, este proceso se realizaría del mismo modo para los botones “pause” y “stop”. Una vez que se han establecido las relaciones entre los componentes y sus estados (definidos por las condiciones especificadas por el usuario), la aplicación se encuentra inicializada. En este punto, siguiendo con el diagrama mostrado, el usuario pulsa el botón “play”, que cambia su estado a “pressed” y comunica este cambio a todos sus “subscriptores”, en este caso, el objeto “Music Player”. A continuación, este último recibe el nuevo estado del componente, que comprueba si en el estado actual (stop), y el nuevo estado del componente (pressed), debería cambiar a otro estado. Una de las definiciones del estado “playing” era precisamente esta (estado previo “stop” y botón “play” seleccionado), por lo que se realiza el cambio al nuevo estado. Por último, el objeto “Music Player” está asociado con el reproductor de música real, por lo que sincroniza su estado con este, que comienza a reproducir música.

14.5. PROTOTIPO ANDROID (APLICACIÓN "EXPLORADORA")

Como prototipo de dispositivo de RA, se implementó la aplicación nativa "exploradora" en la plataforma Android (versión 2.1 en adelante). En esta aplicación, la primera pantalla que se muestra al usuario es la lista de aplicaciones disponibles en SARP, mostrando el nombre y una descripción.

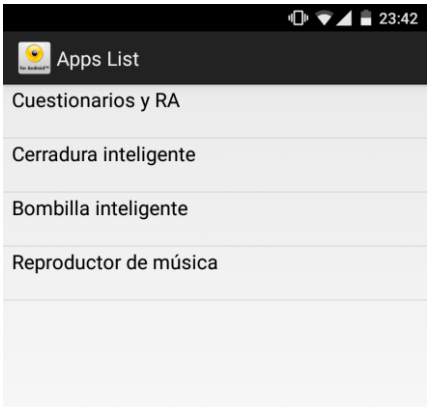


Figura 26 Aplicación Explorer mostrando las aplicaciones disponibles en SARP

A continuación, el usuario selecciona una y esta se abre pasando a una segunda pantalla, en la que se muestran las imágenes capturadas por la cámara. Desde ese momento, el usuario ya puede utilizar la aplicación e interactuar con los posibles objetos que se definieran en ella.

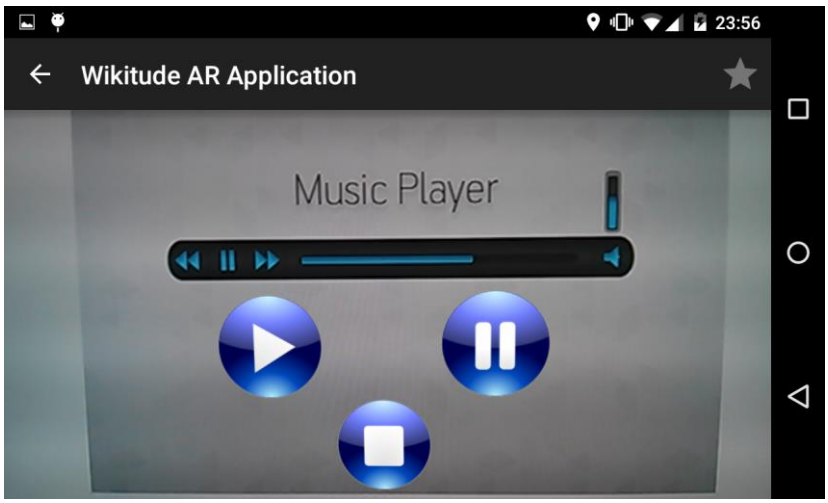


Figura 27 Interfaz de usuario mostrada en la aplicación Music player

Pulsando el botón "atrás", el usuario vuelve a la lista de aplicaciones pudiendo cambiarla en cualquier momento.

14.6. PROTOTIPOS DE OBJETOS INTELIGENTES

Para representar los objetos inteligentes, se elaboraron varias aplicaciones en Java que simulan el comportamiento de posibles objetos. Estas aplicaciones se instalaron en tres Smartphones (Nexus 4, htc wildfire y htc magic), y un ordenador portátil (Windows 8). Los objetos simulados son los siguientes:

Reproductor de Música

Instalado en el portátil, esta aplicación simula un reproductor de música físico, reproduciendo música en los altavoces del ordenador. Utilizando la librería de gestión "MediaPlayer", de Java, se implementaron las acciones básicas del reproductor, como reproducir, pausar o detener. Mientras que la lógica se implementó con la librería proporcionada por SARP (el código se corresponde con el mostrado en el apartado del DSL) y que por tanto depende de las especificaciones de los usuarios cuando crean las aplicaciones. Del mismo modo, esta librería es la encargada de comunicarse con la plataforma para registrar el reproductor en esta y que esté disponible desde cualquier dispositivo de RA.

Bombilla inteligente

Para simular el funcionamiento de una bombilla inteligente, se utilizó el flash de la cámara de los Smartphones. Para ello, se implementó una aplicación en Android en el que se procesan dos posibles estados: encendido y apagado, que se corresponden con los posibles estados del accesorio de la cámara. Del mismo modo que el reproductor, para registrarlo en la plataforma y gestionar la lógica, se utilizó la librería en Java de la plataforma. De este modo, cuándo y cómo se encienda o apague la luz depende de las aplicaciones creadas.

Cerradura inteligente

Un ejemplo bastante ilustrativo de combinar AR e IoT es el de la puerta con una cerradura que se abre cuando el usuario introduce la contraseña correctamente en un teclado que aparece en el dispositivo, mostrado en la sección Estado de la situación actual. Aunque en este caso la cerradura en sí y el mecanismo no se muestra, como demostración de la capacidad de la plataforma, se implementó una aplicación que simula esta cerradura. En esta aplicación, los posibles estados son "abierto" y "cerrado". En este ejemplo se aprovechan claramente las ventajas de la solución utilizada en la plataforma, basada en estados. En este caso, el objeto inteligente implementa acciones muy simples y concretas: abrir la cerradura y cerrarla. Cómo se abra o cierre depende de las aplicaciones elaboradas. De este modo, una aplicación podría mostrar simplemente un botón y si el usuario lo pulsa, se envía el estado "abierto" a la cerradura y esta se abre. Sin embargo, también podría crearse una aplicación que mostrase un teclado numérico en el que el usuario debe introducir la

contraseña correctamente si quiere abrir la cerradura. De esta forma, la lógica de los objetos inteligentes queda delegada en las implementaciones de cada una de las aplicaciones creadas en SARP por los usuarios.

EVALUACIÓN

Como ejemplos de aplicaciones creadas en esta plataforma se muestran cuatro diferentes. Tres de ellas hacen uso de los objetos inteligentes "simulados" que se han mostrado, y el objetivo de la cuarta es demostrar la interacción entre objetos virtuales. Debido a la arquitectura propuesta, basada en estados, la configuración de estos se realiza la primera vez que se conectan con la plataforma, y a partir de ese momento, se define su comportamiento desde las aplicaciones creadas con los editores.

15. *BOMBILLA INTELIGENTE*

En esta aplicación se simula el control de una bombilla inteligente mediante RA. Para ello se define en primer lugar el objeto virtual que la representa. La "interfaz" física de una bombilla es por lo general un interruptor, por lo que esa imagen será la que se utilice en la aplicación () como interacción entre el usuario y el objeto inteligente. El comportamiento en este caso es muy sencillo, y puede describirse con el siguiente diagrama:

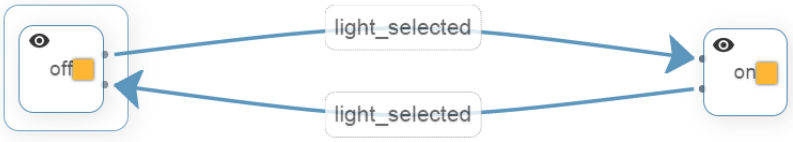


Figura 28 Diagrama de estados de la bombilla inteligente

En este escenario, el usuario enciende la luz con su Smartphone, por lo que el uso de patrones de imágenes no es una buena opción para asociar el objeto virtual con el real, ya que la escena podría empezar en un ambiente con poca luz. En su lugar, la bombilla se ha asociado con su posición geográfica. De este modo, el usuario puede encender la luz incluso si este no puede visualizar la propia "bombilla", para lo cual simplemente debería hacer una búsqueda rápida describiendo una circunferencia mientras apunta con su Smartphone hacia el lugar. Cuando este detecte que el usuario está apuntando hacia el objeto inteligen-

te, mostrará el interruptor y lo podrá pulsar para encender la luz. Si el usuario vuelve a pulsar el mismo interruptor, la luz se apagará.

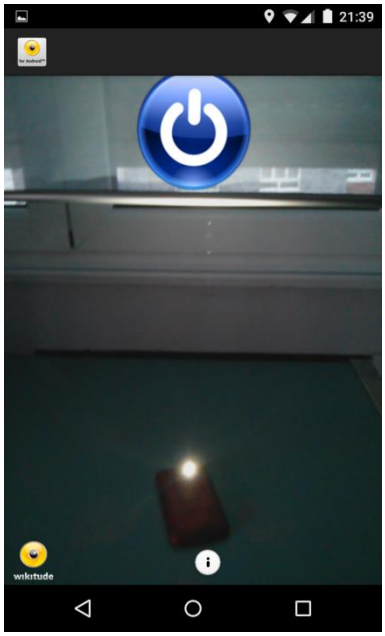


Figura 29 Aplicación mostrada en el dispositivo de RA

16. *REPRODUCTOR DE MÚSICA*

Esta es la aplicación utilizada como ejemplo para describir los editores gráfico y textual, así como el DSL. Se trata de un reproductor de música que el usuario puede controlar a través de una interfaz gráfica de RA. El diagrama de estados que representa el comportamiento del objeto virtual es el mismo que el mostrado en la Figura 22. Cuando el usuario apunta hacia el reproductor (en este caso simulado con un Smartphone), los tres botones habituales para controlar la reproducción de música aparecen en la pantalla del dispositivo.

Figura 30 Reproductor de Música mostrado desde el dispositivo de RA

17. CERRADURA INTELIGENTE

Esta aplicación es la creada para hacer uso de la cerradura inteligente simulada. Como se ha comentado, aunque el objeto inteligente está programado para soportar dos estados (abierto y cerrado), fue posible controlar su comportamiento de modo que el usuario deba introducir una contraseña para cambiar al estado abierto. El diagrama utilizado para ello es el siguiente:

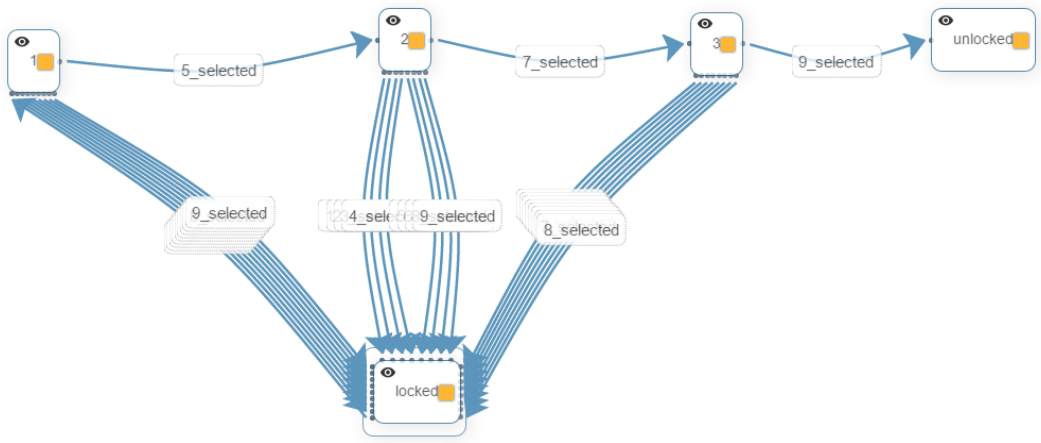


Figura 31 Diagrama de estados para la cerradura inteligente

Cuando el usuario apunta hacia la "cerradura", aparece un teclado numérico en la pantalla, donde debe introducir correctamente la contraseña (5-7-9 en el ejemplo) para desblo-

quearla. Como se puede observar en la Figura 31, este es un ejemplo en el que es necesario especificar todas las posibles condiciones para cada estado. De este modo, la cerradura cambiará a "abierta" solo si el usuario introduce correctamente la contraseña en el orden especificado.

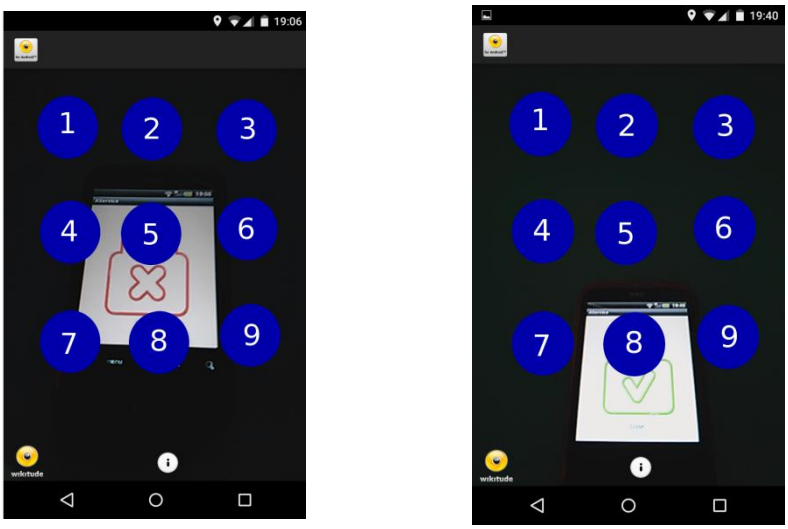


Figura 32 Cerradura inteligente antes (izquierda) y después (derecha) de que el usuario introduzca correctamente la contraseña (5-7-9)

18. *INTERACCIÓN ENTRE OBJETOS VIRTUALES*

En la cuarta aplicación elaborada, se muestra un sencillo ejemplo que demuestra cómo interactúan los objetos inteligentes en la plataforma. En esta aplicación, la bombilla inteligente se apaga automáticamente cuando el reproductor comienza a reproducir música. Para ello, se hace uso del estado "play" del reproductor asignándolo como una de las condiciones para el estado "off" de la bombilla inteligente. El diagrama utilizado para establecer este comportamiento entre ambos objetos es el siguiente:

Figura 33 Diagrama de estados mostrando la interacción entre objetos virtuales

CONCLUSIONES Y TRABAJO FUTURO

En esta investigación se ha presentado una plataforma basada en MDE que permite crear aplicaciones en las que la Realidad Aumentada e Internet of Things se complementan ofreciendo a los usuarios una experiencia unificada, permitiendo controlar los objetos inteligentes a través de interfaces gráficas basadas en RA. Estas aplicaciones se elaboran sin que sean necesarios conocimientos de programación. Para ello, se han desarrollado dos editores (textual y gráfico), basados en el DSL diseñado para representar las escenas de RA, con el que los usuarios pueden crear sus propias aplicaciones. La arquitectura propuesta para esta plataforma es lo suficientemente flexible para que los usuarios puedan definir diferentes comportamientos para los objetos inteligentes con la misma configuración inicial. De este modo, los objetos virtuales y los inteligentes interactúan de forma intuitiva, respondiendo correctamente a las acciones de los usuarios. Aunque el objetivo inicial se podría considerar alcanzado, se plantean los siguientes aspectos que podrían extender o mejorar las características de la propuesta presentada:

- **Integración con plataformas de IoT**: Como se ha comentado, SARP no proporciona mecanismos suficientemente complejos para configurar los objetos inteligentes de modo que sean compatibles con la plataforma sin ser necesaria la programación. La integración con una de las plataformas de este tipo disponibles, podría ser una posible solución.
- **Procesamiento dinámico de datos:** La comunicación entre objetos virtuales e inteligentes, se basa en este caso en la sincronización entre los estados de ambos, como se ha demostrado, esto proporciona suficiente flexibilidad para interactuar con los objetos inteligentes, sin embargo, hay casos en los que podrían no estar claros los posibles estados de un objeto inteligente, o disponer de un número muy elevado de estos (por ejemplo sensores que reciben muchos valores). Una posible solución sería permitir definir los estados desde la propia plataforma, de modo que al definir el comportamiento de los objetos virtuales, estuvieran disponibles el

tipo de valores de un objeto inteligente y, en base a eso, describir los estados referentes a la aplicación.

- **Evaluación de usabilidad:** En el apartado de evaluación se realizó un análisis sobre la funcionalidad de la plataforma, pero sería necesario elaborar un estudio con usuarios que permitiese evaluar la facilidad de uso de las herramientas proporcionadas y si efectivamente pueden ser utilizadas por usuarios no expertos.
- **Estudio de rendimiento con web sockets:** En la plataforma se introdujo el uso de esta plataforma debido a que las características que muestra deberían ser favorables en la comunicación entre objetos. Sin embargo, sería necesario realizar un estudio sobre el rendimiento y la escalabilidad que proporciona con un mayor número de dispositivos.
- **Seguridad:** Actualmente las aplicaciones publicadas en SARP están disponibles para todos los usuarios. Añadir control de usuarios y características de seguridad permitiría restringir el acceso a las aplicaciones a un determinado grupo de usuarios y manipular información más sensible desde la plataforma.
- **Dispositivos de RA:** Aunque para demostrar la funcionalidad se ha utilizado un dispositivo (Android) de RA, SARP está diseñado para ser multiplataforma, por lo que podrían añadirse más dispositivos de RA de un modo relativamente sencillo.

REFERENCIAS

Atzori, Luigi, Antonio Iera, y Giacomo Morabito. 2010. «The Internet of Things: A survey». *Computer Networks* 54 (15): 2787-2805. doi:10.1016/j.comnet.2010.05.010.

Azuma, R., Y. Baillot, R. Behringer, S. Feiner, S. Julier, y Blair MacIntyre. 2001. «Recent advances in augmented reality». *IEEE Computer Graphics and Applications* 21 (6): 34-47. doi:10.1109/38.963459.

Bartie, Phil J., y William A. Mackaness. 2006. «Development of a Speech-Based Augmented Reality System to Support Exploration of Cityscape». *Transactions in GIS* 10 (1): 63-86. doi:10.1111/j.1467-9671.2006.00244.x.

Bauer, M., B. Bruegge, Gudrun Klinker, A. MacWilliams, T. Reicher, S. Riss, Christian Sandor, y M. Wagner. 2001. «Design of a component-based augmented reality framework». En , 45-54. doi:10.1109/ISAR.2001.970514.

Belimpasakis, P., Yu You, y P. Selonen. 2010a. «Enabling Rapid Creation of Content for Consumption in Mobile Augmented Reality». En *2010 Fourth International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST)*, 1-6. doi:10.1109/NGMAST.2010.13.

———. 2010b. «Enabling Rapid Creation of Content for Consumption in Mobile Augmented Reality». En , 1-6. doi:10.1109/NGMAST.2010.13.

Bruna Gómez, Marcos. 2012. «Tecnologías de geolocalización mediante realidad aumentada. Layar y Wikitude», septiembre. https://riunet.upv.es/handle/10251/17029.

Didier, J., M. Chouiten, M. Mallem, y S. Otmane. 2012. «ARCS: A framework with extended software integration capabilities to build Augmented Reality applications». En , 60-67. doi:10.1109/SEARIS.2012.6231170.

Dubois, Emmanuel, Dominique L. Scapin, Syrine Charfi, y Christophe Bortolaso. 2013. «Usability Recommendations for Mixed Interactive Systems: Extraction and Integration in a Design Process». En *Human Factors in Augmented Reality Environments*, editado por Weidong Huang, Leila Alem, y Mark A. Livingston, 181-99. Springer New York.

Dumas, Marlon, y Arthur H. M. ter Hofstede. 2001. «UML Activity Diagrams as a Workflow Specification Language». En *UML 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, editado por Martin Gogolla y Cris Kobryn, 76-90. Lecture Notes in Computer Science 2185. Springer Berlin Heidelberg. http://link.springer.com/chapter/10.1007/3-540-45441-1_7.

«Exploring MARS: developing indoor and outdoor user interfaces to a mobile augmented reality system». 2014. Accedido junio 23. http://www.sciencedirect.com/science/article/pii/S009784939900103X.

Friedrich, Wolfgang, D. Jahn, y L. Schmidt. 2002. «ARVIKA-Augmented Reality for Development, Production and Service.» En , 2002:3-4. Citeseer.

García-Díaz, Vicente, B. Cristina Pelayo García-Bustelo, y Juan Manuel Cueva Lovelle. 2012. «MDCI: Model-driven continuous integration». *Journal of Ambient Intelligence and Smart Environments* 4 (5): 479-81. doi:10.3233/AIS-2012-0173.

González García, Cristian. 2013. «MIDG AR: Plataforma para la generación dinámica de aplicaciones distribuidas basadas en la integración de redes de sensores y dispositivos electrónicos IoT», julio.

Guinard, Dominique. 2009. «Towards the web of things: Web mashups for embedded devices». En *In MEM 2009 in Proceedings of WWW 2009. ACM*.

Harel, David, y Amnon Naamad. 1996. «The STATEMATE Semantics of Statecharts». *ACM Trans. Softw. Eng. Methodol.* 5 (4): 293-333. doi:10.1145/235321.235322.

Heun, Valentin, James Hobin, y Pattie Maes. 2013. «Reality Editor: Programming Smarter Objects». En *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, 307-10. UbiComp '13 Adjunct. New York, NY, USA: ACM. doi:10.1145/2494091.2494185.

Heun, Valentin, Shunichi Kasahara, y Pattie Maes. 2013. «Smarter objects: using AR technology to program physical objects and their interactions». En , 961-66. ACM.

Kato, Hirokazu, y Mark Billinghurst. 1999. «Marker tracking and hmd calibration for a video-based augmented reality conferencing system». En *Augmented Reality, 1999.(IWAR'99) Proceedings. 2nd IEEE and ACM International Workshop on*, 85-94. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=803809.

Kent, Stuart. 2002. «Model Driven Engineering». En , editado por Michael Butler, Luigia Petre, y Kaisa Sere, 286-98. Lecture Notes in Computer Science. Springer Berlin Heidelberg.

Kerawalla, Lucinda, Rosemary Luckin, Simon Seljeflot, y Adrian Woolard. 2006. «"Making It Real": Exploring the Potential of Augmented Reality for Teaching Primary School Science». *Virtual Reality* 10 (3-4): 163-74. doi:10.1007/s10055-006-0036-4.

Lee, Jangho, Jee-In Kim, Jiyong Kim, y Ji-Young Kwak. 2007. «A Unified Remote Console Based on Augmented Reality in a Home Network Environment». En , 1-2. doi:10.1109/ICCE.2007.341516.

Madden, Lester. 2011. *Professional Augmented Reality Browsers for Smartphones: Programming for Junaio, Layar and Wikitude*. John Wiley & Sons.

Magnenat-Thalmann, Nadia, y Daniel Thalmann. 1999. «Virtual reality software and technology». *Encyclopedia of Computer Science and Technology, Marcel Dekker* 41 (VRLAB-ARTICLE-2007-019). http://infoscience.epfl.ch/record/101398/files/Magnenat_Thalmann_Thalmann_Ency_99.pdf.

McFarlane, Duncan, Sanjay Sarma, Jin Lung Chirn, C. Y Wong, y Kevin Ashton. 2003. «Auto ID systems and intelligent manufacturing control». *Engineering Applications of Artificial Intelligence*, Intelligent Manufacturing, 16 (4): 365-76. doi:10.1016/S0952-1976(03)00077-0.

MILGRAM, Paul, y Fumio KISHINO. 1994. «A Taxonomy of Mixed Reality Visual Displays». diciembre. http://search.ieice.org/bin/summary.php?id=e77-d_12_1321.

Oliveira, Álvaro, y Margarida Campolargo. 2013. *From Smart Cities to Human Smart Cities*. Lisboa.

Owen, Charles, Arthur Tang, y Fan Xiao. 2003. «ImageTclAR: A blended script and compiled code development system for augmented reality». En *Proceedings of the International Workshop on Software Technology for Augmented Reality Systems*, 537-44. Citeseer. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.5081&rep=rep1&type=pdf.

Pascual Espada, Jordán. 2012. «Diseño de objetos virtuales colaborativos orientados a servicios en el marco de Internet de las cosas», julio.

Pelayo García-Bustelo, Begoña Cristina. 2007. «TALISMAN: desarrollo ágil de Software con Arquitecturas Dirigidas por Modelos». julio. http://www.tdx.cat/handle/10803/35683.

Pentenrieder, Katharina, Christian Bade, Fabian Doil, y Peter Meier. 2007. «Augmented Reality-based factory planning-an application tailored to industrial needs». En

Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on, 31-42. IEEE. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4538822.

Piekarski, Wayne Piekarski, y Bruce H. Thomas. 2001. «Tinmith-evo5–An Architecture for Supporting Mobile Augmented Reality Environments». En , 177-177. IEEE Computer Society.

Poupyrev, I., D.S. Tan, M. Billinghurst, H. Kato, H. Regenbrecht, y N. Tetsutani. 2002. «Developing a generic augmented-reality interface». *Computer* 35 (3): 44-50. doi:10.1109/2.989929.

Schaffers, Hans, Nicos Komninos, Marc Pallot, Brigitte Trousse, Michael Nilsson, y Alvaro Oliveira. 2011. «Smart Cities and the Future Internet: Towards Cooperation Frameworks for Open Innovation». En , editado por John Domingue, Alex Galis, Anastasius Gavras, Theodore Zahariadis, Dave Lambert, Frances Cleary, Petros Daras, et al., 431-46. Lecture Notes in Computer Science. Springer Berlin Heidelberg.

Schmalstieg, D., y Daniel Wagner. 2007. «Experiences with Handheld Augmented Reality». En *6th IEEE and ACM International Symposium on Mixed and Augmented Reality, 2007. ISMAR 2007*, 3-18. doi:10.1109/ISMAR.2007.4538819.

Seidewitz, Ed. 2003. «What Models Mean». *IEEE Software*.

Shelton, B.E., y N.R. Hedley. 2002. «Using augmented reality for teaching Earth-Sun relationships to undergraduate geography students». En , 8 pp. - . doi:10.1109/ART.2002.1106948.

Singh, Ajay Kumar, Souvik Roy, Simrat Sodhi, Daksh Goel, y Shaifali Madan Arora. s. f. «SmartX Virtuality: A Smarter way to Interact Virtually with Physical Objects».

Soley, Richard, y others. 2000. «Model driven architecture». *OMG white paper* 308: 308.

Stefan M¨uller-Schneiders. s. f. «Augmented reality in advanced driver assistance systems».

Thompson, C.W. 2005. «Smart devices and soft controllers». *IEEE Internet Computing* 9 (1): 82-85. doi:10.1109/MIC.2005.22.

Van Deursen, Arie, y Paul Klint. 2002. «Domain-specific language design requires feature descriptions». *CIT. Journal of computing and information technology* 10 (1): 1-17.

Van Deursen, Arie, Paul Klint, y Joost Visser. 2000. «Domain-Specific Languages: An Annotated Bibliography.» *Sigplan Notices* 35 (6): 26-36.

Van Krevelen, D. W. F., y R. Poelman. 2010. «A survey of augmented reality technologies, applications and limitations». *International Journal of Virtual Reality* 9 (2): 1.

Zanella, Andrea, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, y Michele Zorzi. 2014. «Internet of Things for Smart Cities». *IEEE Internet of Things Journal* 1 (1): 22-32. doi:10.1109/JIOT.2014.2306328.

Zhang, Handong, y Lin Zhu. 2011. «Internet of Things: Key technology, architecture and challenging problems». En *2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, 4:507-12. doi:10.1109/CSAE.2011.5952899.