Received June 11, 2020, accepted July 12, 2020, date of publication July 28, 2020, date of current version August 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3012503

Midgar: Creation of a Graphic Domain-Specific Language to Generate Smart Objects for Internet of Things Scenarios Using Model-Driven Engineering

CRISTIAN GONZÁLEZ GARCÍA[1], **DANIEL MEANA-LLORIÁN**[1], **VICENTE GARCÍA-DÍAZ**[1], **ANDRÉS CAMILO JIMÉNEZ**[2], **AND JOHN PETEARSON ANZOLA**[2]

[1]Department of Computer Science, Sciences Building, University of Oviedo, 33007 Oviedo, Spain

[2]Department of Electronics and Systems, Fundación Universitaria Los Libertadores, Bogotá 111221, Colombia

Corresponding author: John Petearson Anzola (jpanzolaa@libertadores.edu.co)

This work was supported in part by the Ingeniería Dirigida por Modelos MDERG Research Group, University of Oviedo, the Research Project Ingeniería Dirigida Por Modelos MDERG financed by P.R., Proyecto Plan Regional, under Contract FC-15-GRUPIN14-084, in part by the Severo Ochoa Program financed by Asturias Government, under Contract PA-17-PF-BP16074, and in part by the International Collaboration from the Department of Electronics and Systems of the Fundación Universitaria Los Libertadores, under Project ING-009-19.

ABSTRACT Currently, we have around us many Smart Objects. With the use of these objects, we can obtain benefits in our daily lives, as well as recommendations and help when we travel. Alternatively, we may increase and improve our industrial processes through the automation of certain tasks. Notwithstanding, we need to use specific software or to develop our own applications. Nevertheless, the main problem arises when we need to develop our own application because we need to save money, or in other cases, the existing applications are not adapted to us. In this case, it is possible that we need to learn new things, the money will then be spent, and such a process is likely to involve problems related to the Software Crisis. So, the main motivation is to create an environment which can reuse the previous knowledge and help people without knowledge about programming to create Smart Objects. Then, the research question of this paper is the following: Could we enable the creation of Smart Objects in an easy and efficient way for people who do not have programming skills? As a possible solution, we have developed a graphic Domain-Specific Language using the Midgar platform. In order to validate our proposal, we make an evaluation split into different phases; the first one consisted in measuring data obtained from participants when they were performing a specific task, and the second one consisted of a survey to collect their opinions about our proposal. Moreover, we also did a comparison of the measured data between two graphical editors and two different participant profiles according to their knowledge about Smart Objects.

INDEX TERMS Internet of Things, smart objects, model-driven engineering, domain-specific language.

I. INTRODUCTION

The Internet of Things (IoT) is the interconnection of heterogeneous and ubiquitous objects between themselves [1]–[4]. People's daily life has a lot of objects with Internet connection like smartphones [5], tablets, Smart TVs, microcontrollers [6], [7], Smart Tags [3], computers, laptops, cars, or cheaper sensors, with improved wireless connections [8], and other things to make IoT interconnections between different objects in their homes. With these things, people have heterogeneous objects because these are of different types and people have ubiquitous objects because objects are installed in different places and some of them can be moved around the world. If we think about this, we could already think that we have the Internet of Things.

With the IoT, we can create an enormous network to interconnect objects and facilitate our daily life. Some examples: to improve the tracking of deliveries and objects situation, to improve the factory production, or the security in the industry [9]–[11]; to prevent natural disasters, to automate

The associate editor coordinating the review of this manuscript and approving it for publication was Shancang Li.

actions in farms, or to obtain data about the ecosystem in order to protect the fauna and flora according to certain situations, which is known as Smart Earth [12]–[14]; the improvement of Smart Cities to help citizens in their daily life, for instance, to park, to avoid traffic jams, to automate traffic lights, to change the public transport in real-time or so on [15]–[17]; to improve people's life with Smart Homes with the use of a Smart Fridge to help ill people or facilitating the garden maintenance [18]–[20] or to save money with the heating [21]. However, people need to create an application or buy it. The ideal world would be one in which each person could put their objects in the IoT with no problems, with only a minimum knowledge about the IoT and the object that they use. Thus, we could have a lot of sensors and data to create a huge IoT network for our City or our World.

Nevertheless, people depend on important and big companies, which are developing new IoT systems. The products of one company depend on the object, thus maybe you must use specific software and hardware. For example, LG, Samsung, Telefónica, and other companies offer packs for creating an IoT network in your own home. Nonetheless, these solutions usually only allow interconnecting objects with other objects of the same brand. For instance, in almost all cases, if you use an object of a particular company, you only can use the mobile application that this company provides to interact with its objects. Another example is when you use a server and you only can interconnect with these server objects of the same server brand. This breaks the heterogeneity of the Internet of Things because the objects of one company do not use the same language or message system to talk to each other. In other cases, companies use their IoT products in their own language. This problem obliges us to use only the same product brand to create an ecosystem. In this case, it is possible that people cannot use the better sensor of another brand to create an IoT system for his company or house because then, people should need to change all the system or they should use both systems at the same time instead of only have one. Maybe, they should use both because, for example, each system has the better sensor in different areas. This problem impedes the evolution of the IoT because it limits the evolution according to the progress of companies. In this case, if companies improve their systems, their sensors, their IoT servers, their products, thereafter the IoT will evolve. However, currently, all the evolution depends on the software and hardware that companies provide to us. Notwithstanding, some open initiatives, like Arduino, or a cheap system like Raspberry Pi, allow us to create our IoT things but in these cases, we have to program all the logic, which requires programming skills.

Therefore, our hypotheses are the next:

- It is possible to use Model-Driven Engineering to create a Domain-Specific Language (DSL) that enables people without experience in the development of applications for Smart Objects to create Smart Objects although they have not programming skills.
- Our proposal is more efficient than other alternatives according to the time, the clicks, and the movement of the mouse required creating Smart Objects.

To solve our two hypotheses, a possible solution is to allow creating and interconnecting different objects between themselves. We proposed a solution based on a graphic DSL called *Midgar Object Interconnection Specific Language* (MOISL) to solve the interconnection problem amongst heterogeneous and ubiquitous objects in [22], using the Midgar IoT platform. With this solution, people without development knowledge can interconnect the objects of an IoT network. However, this is only a partial solution to the problem that we have presented because it does not solve the problem of generating applications. In our first proposal, people needed to have a registered application in the IoT platform, which has to interpret specific messages. In this article, we propose a possible solution to create the necessary software for the objects. We propose a graphic DSL called *Midgar Object Creation Specific Language* (MOCSL). In this proposal, we want to offer a graphic DSL to facilitate the object creation to people without development knowledge. Then, the users of this proposal can be people without knowledge about programming but with the knowledge or attitude to learn about the IoT or in the "Do it yourself" (DIY) movement, or a skilled technical team which needs to develop and create the needed software for an IoT network, maybe from a company or government. In this way, people need a smartphone and know their smartphone brand and/or an Arduino and know how to connect sensors and actuators to their Arduino. In the Arduino case, people do not need to know how to create the source code to use the temperature sensor, the flame sensor, or the servomotor. People only need to be able to connect sensors and actuators in the Arduino. In both cases, they will have an IoT server to upload the data. It can be public or private.

Thus, we have researched to offer this abstraction in order to facilitate the interconnection between the real world and the virtual world with no developed source code. With this proposal, we want to allow a quicker and easier way for the object creation to improve the implementation of the IoT in our daily life, in our industry, our home, our education, our security, or in many other situations. Then, the aim of this proposal is to allowed people with knowledge about this domain and a little knowledge about computing to create applications for the IoT easily. So, the contribution is to create a tool to allow people to create Smart Objects and demonstrate that it is possible to reuse specific parts of the software to solve it.

The remainder of this paper is structured as follows: In section II we explain the state of the art where we discuss the Internet of Things, Smart Objects, Model-Driven Engineering, Domain-Specific Languages, the more important current IoT platforms, and the related work on applications that generate the software that the objects need. In section III we explain the proposal. We present the idea about the generation of objects for the Internet of Things platform and its architecture. Section IV contains the evaluation of our proposal and the discussion of the obtained results. Section V has our conclusions about this research. Finally, in section VI, we describe some possible options as the future work of this proposal.

II. STATE OF THE ART

In the Internet of Things, we have different objects, which can be smart or not [23]. In both cases, these objects are very heterogeneous and different. Then, the creation of the software that an object needs is a hard task: people need to know how to program that object and people need time.

A possible solution to create an easier system to be managed and the software that these objects need is applying Model-Driven Engineering to create a Domain-Specific Language. Nevertheless, firstly, we need to know how the IoT platforms are at this moment and how other and similar applications allow doing this task.

A. THE INTERNET OF THINGS

In recent years, we could see the growth of the interaction amongst heterogeneous and ubiquitous objects with each other. It is known as the Internet of Things (IoT) [1], [3], [20], [22], [24], [25]. This was announced by the National Intelligence Council of the United States in [26] and by the United Nations [1]. The origin was the necessity of supply chains and the identification of objects [27], people, and animals through the use of RFID intelligent tags [1], [4], [28]. Then, with the IoT, we can identify the different objects of the world to interact the real world with the virtual world. It offers to us a lot of possibilities as we can see in many types of research about Smart Earth [16], Smart Home [2], [21], [29], [30], Smart Cities [16], and Smart Towns [31], [32]. All of them facilitate the life of humans with different automation and notifications. Nevertheless, the Internet of Things requires integrated intelligence, connectivity, and interaction [3]. This is why we have different Internet of Things platforms to interconnect our objects.

However, we have a problem when we work with the IoT: We must develop the software to make Smart Objects function, and this can require some hours and knowledge about software development. After that, we have to do the interconnection of the different objects between themselves. However, this second part was solved previously in [22].

B. SMART OBJECTS

Smart Objects, also known as Intelligent Products, are physical elements with different properties, which are identifiable during their useful life, interacting with the environment and other objects, and acting intelligently according to different conditions, as we can see in [23]. They can obtain different environmental data like humidity, location, temperature, gravity, acceleration, and so on. Besides, they have various actuators to do different things when they obtain a certain datum from another object, the environment, or through an intelligent network like an IoT platform. For example, they could vibrate, show a notification, make a call, or send messages if it is a smartphone, tablets or similar. Other possibilities are when they could turn on/turn off the sound, accelerate/decelerate an engine or a fan, and anything that you can create with a microcontroller like an Arduino.

Nevertheless, users must create the software to obtain the object data, create the interconnection amongst different Smart Objects, or use a specific software with specific hardware although in this case, users lose the possibility of interconnecting heterogeneous objects.

C. MODEL-DRIVEN ENGINEERING

Software engineering continually offers new methods and tools to improve the software development process. Although the use of modelling standards such as the Unified Modeling Language (UML) [33] is a commonly accepted practice, in recent years it is gaining strength the use of a relatively new approach for the development called Model-Driven Engineering (MDE) [34], based entirely on the use of models as first-class artefacts for development. With that, models like those provided by the UML standard are gaining importance. Thus, the Model-Driven Architecture (MDA) [35] standard has appeared as a standardized way of conducting MDE from UML models and metamodels. The use of models, with common underlying technologies, allows an increase of the portability, reusability, and interoperability of all software that is built. Thus, technologies such as those included in the Eclipse Modeling Project [36] have arisen offering a complete toolset for working with models to build any type of software.

For this proposal, we used MDE to study the problem and obtain an abstraction of it. With this abstraction, we have obtained a Metamodel and a complete view about the problem. Based on this metamodel, we can create a Domain-Specific Language to solve our hypotheses and create automatically any definition using this DSL.

D. DOMAIN-SPECIFIC LANGUAGES

From the use of standardized models as a basis for the development of software, the Domain-Specific Languages (DSLs) [37] have also gained in popularity. Thus, very powerful tools such as Xtext [38] are currently been widely used. Xtext, for example, is a framework that focuses on the creation of DSLs using models as the core technology. The main idea is to define small programming languages tailored to a specific domain of knowledge, avoiding technological concepts and focusing on keywords and constructs related to the specific domain being dealt with [39]. Thus, common General-Purpose Languages (GPLs) such as C++, Java, C#, Objective-C, or Swift use many keywords (e.g., static, protected, double, import, etc.) that have nothing to do with the final purpose of the applications that are being defined [40]. A DSLs can be designed to only use specific domain concepts, allowing that experts in a field can use it to create solutions, even without the high technical knowledge that GPLs require [41]. DSLs can be interpreted or compiled like any other language, but most often, the code is benefited from the reuse of model-based technologies and with standardized tools based on a common root metamodel (Meta-Object Facility^{TM} (MOF) [42] / Ecore) [43]. Thus, programs are translated into well-established languages to be interpreted by their virtual machines and compilers.

Then, we have created a graphic DSL (MOCSL) in order to allow the object creation for people without development knowledge or people who need to create objects in an easy and fast way. We have designed MOCSL using Ecore [44] as a Meta-metamodel and we have created the DSL using JavaScript.

E. SMART OBJECTS

We have different Internet of Things platforms to interconnect our objects. We can classify the IoT platforms into four groups [24]:

- Business platforms: Xively [45] (now it is a part of Cloud IoT), Exosite [46], SensorCloud [47], Etherios [48], ThingWorx [49], Altair SmartWorks (previously Carriots) [50], Azure IoT Suit [51], Amazon Web Services [52], and IBM Internet of Things [53].
- Research platforms: Midgar [22], [54], [55], Paraimpu [6], [56], [57], QuadraSpace [58], SenseWeb [59]–[61], and SIoT [62].
- Platforms in beta state: Sensorpedia [63], [64], Evrythng [65], and Open.Sen.se [66].
- Open Source platforms: ThingSpeak [67], Nimbits [68], and Kaa [69].

All these IoT platforms are great and all have different important qualities to improve the IoT. Nonetheless, these IoT platforms require a minimum of knowledge about development because all of them need that users create the object software, the interconnection between the object and the platform, or both. Sometimes, the IoT platforms help users with an Application Programming Interface (API) to facilitate the interconnection with the IoT platform and/or to read sensor data in different Smart Object or microcontrollers like Android smartphones, Arduino microcontrollers, Raspberry Pi microcomputers, and so on. In this case, the IoT platforms reduce the time and the users' effort to develop applications but users still need knowledge about software development and still need to develop the program.

One problem in the IoT is the necessity of developing applications to interconnect Smart Objects between themselves. Some IoT platforms help to interconnect Smart Objects through a DSL or forms. According to this problem, we have proposed a possible solution with MOISL to improve this problem in [22] and proposed the idea in [24].

Another problem is when users want to use a Smart Object or a microcontroller, but users do not have knowledge on how to develop an application. Then, users have some possibilities. They could learn to program, but, in this case, they need a lot of time or they could buy a specific application for their own Smart Object. For this problem, a possible solution is to use similar software used for the previous problem but the IoT platforms do not have this option. However, the Arduino community offers software to facilitate this task but with some restrictions. We will explain more about this in the next section. This is why we propose a new DSL for the Midgar Platform to facilitate the generation of Smart Objects and microcontroller for IoT scenarios in this article.

F. RELATED WORK: APPLICATIONS THAT GENERATE SOFTWARE TO MANAGE OBJECTS

One important issue with the Internet of Things is the need to know how to develop applications to interconnect different objects. Typically, such objects are based on the use of different types of smartphones or microcontrollers. Thus, we need to develop a different kind of applications using different programming languages, libraries, and platforms. Then, to interconnect heterogeneous objects we need to know how to develop applications with a large number of technologies. One of the main difficulties is the software development phase because it is a complex, difficult and error-prone task [70]. However, we can find some solutions to facilitate this task for people without development knowledge. They even include the possibility of working with data through the flow modification inside each of the objects. In our work, we did not include this functionality because we have already included this functionality in MOISL [22] and our aim now is the object creation for the Internet of Things. In this section, we are going to mention some of the most important solutions.

1) SOLUTIONS FOR SMARTPHONES

We have focused on various graphic editors to create applications for Android since it is the most extended mobile operating system by far. Thus, there exist important tools that could be used like AppsGeyser [71], iBuildApp [72], and Andromo [73]. However, there are also editors that include support for other additional platforms such as AppsBuilder [74] and Infinite Monkeys [75] for iOS and HTML, Appypie [76], which has bought Infinite Monkeys, for Blackberry and Windows Phone, Como (currently called Swiftic) [77] for iOS, or AppMachine [78] for Windows Phone and with Web Services (eXtensible Markup Language (XML) and JavaScript Object Notation (JSON)) in the pro plan and only to download the data and interpret them.

All these graphic web editors share with our proposal different features and allow to create solutions with a drag & drop system using different blocks to add the necessary functionalities. Others even provide the use of default templates with a minimum set of features to make the creation of applications easier.

2) SOLUTIONS FOR MICROCONTROLLERS

MiniBloq [79] is one of the most representative graphic editors for Multiplot and Arduino platforms. It is an open-source tool based on a desktop application generator that proposes a jigsaw puzzle type system to design the application life cycle.

In addition, it generates the source code in real-time to be uploaded to the microcontroller, although it does not allow modifying it.

However, with MiniBloq, blocks/pieces movements may be confusing. For example, if we want to add a piece to improve another more basic piece, we need to add this piece from the piece menu instead of the general menu. However, if we want to start the next instruction, we must use the general menu. There is also the opposite case and it is possible to find the same functionality under different pieces with a different icon. MiniBloq provides less freedom to use the pins because these depend on the piece type. It also includes by default the header files "mbq.h" and "PingIRReceiver.h", even if we do not need their modules. In addition, to repeat a task we need to insert a loop because all the source code that the editor generates is in the "setup method" instead of the "loop method", according to the normal use of language, which is based on Wiring [80]. This may be incorrect because in Arduino the "setup method" is for establishing the different variables and pins and the "loop method" is for creating our repetitive task or main task.

Besides, this graphic editor has a special piece to insert source code, pieces to create variables, and pieces that have math operations. Notwithstanding, it lacks pieces for creating a flow like "if" or "while" or logic pieces like "true" or "false", or conditions in an easy way.

Thus, Minibloq clearly facilitates the generation of applications, but if we need a critical application for the industrial area, we could have some difficulties for designing it.

There are other graphic editors to generate applications for microcontrollers. For example, ArduBlock [81], is a plugin for the default Arduino development environment. It allows building generic applications by just dragging and dropping blocks. Some other alternatives with a similar set of features include Scratch for Arduino [82], [83], Modkit [84], Atmel Studio [85] which is very similar to Visual Studio and you have to know how to program, or XOD [86], that is a powerful environment in its early stages which proposes a language to design applications not only for the Arduino platform but for other microcontrollers.

3) ARCHITECTURAL MODELS

Internet of Things Architecture (IoT-A) was a proposal about an architectural model and a definition of a set of building blocks for the IoT [87]. The reason for this project was to improve the interoperability and creating standards and guidelines to protocols, algorithms, and interfaces for IoT systems. Besides, it assesses existing IoT protocols and develops modelling tools and a description language for interactions between devices in the IoT. However, the project finished on November 30th, 2013. In the official webpage, they do not have results, and the project website is down.

On the other hand, this paper tries to solve the creation of Smart Objects to a specific IoT network, in our case Midgar, using MDE and the creation of a DSL to facilitates the development.

III. CASE STUDY: MIDGAR

Midgar is an Internet of Things platform developed to research the different problems of the IoT. Currently, Midgar offers support to create the interconnection between heterogeneous and ubiquitous objects using the DSL called MOISL [22], [55] in a secure way [54]. MOISL is a DSL to facilitate the creation of the interconnection between Smart Objects in IoT scenarios. Then, MOISL allows users to specify how each object interacts with each other. For instance, if the Midgar platform has a Smartphone and Arduino microcontroller registered, using MOISL, a user can specify the conditions and actions which they want to do with those objects. For example, when the smartphone sends a message with the location and the Midgar platform detects that the smartphone is very close to the house, it can send a message to the Arduino to turn on the heating or the air conditioning according to the temperature of the house. However, in that first iteration, the logic of the objects had to be developed by users.

In this paper, we try to find a solution for the object generation in order to facilitate the creation of Smart Objects for almost any people without the knowledge of programming. Then, with this new DSL, called MOCSL, we facilitate the creation of the objects and their interaction with an IoT platform because users only have to select the sensor and actuators that they want to use. After that, using MOISL; they can create the interaction between their objects. In this section, we will describe the Midgar Platform architecture, which includes the new graphic DSL MOCSL.

A. GENERATION OF APPLICATIONS FOR SMART OBJECTS

With *Midgar Object Creation Specific Language* (MOCSL), users can create the necessary logic for their objects without writing source code using a drag & drop system. Users only need to choose the sensors and actuators that they want to use on their smartphone or Arduino using the DSL MOCSL, which has been developed using HTML5. After it, MOCSL generates a native application with everything that has been defined by the user and that objects need to be a Smart Object. This native application can be the source code for their smartphone or all the C and Java code for their Arduino. On the one hand, users only need to select the sensors and actuators of their smartphones that they want to use and permit other devices to use. On the other hand, users need to assemble the required sensors and actuators on their real microcontroller and define in MOCSL the sensors and actuators that they have assembled and they want to use. Afterwards, they obtain the native application to connect the microcontroller to the computer USB port. This application has two parts: The Java application of the PC and the C application to the Arduino. The C application has to be uploaded to the Arduino using the Arduino Integrated Development Environment (IDE). Besides, it is running constantly in the Arduino to do the task which had been defined by the user: read sensors and/or run the actuators. It contains all the source code with the functionality that the users have defined and that is necessary to run the sensors and actuators that they have selected, including the needed libraries. This application is sending the data constantly to the USB port. Then, the Java application, which is running in the PC, is reading the USB port to obtain or send data from/to the Arduino. Furthermore, the Java application is sending the data to the IoT Midgar server. The main purpose of the Java application is to be an intermediary between the Arduino and the server. Then, with this intermediary, we can improve the capacity of the Java application to use Big Data or manage the data with more potential in the PC. In case you want to avoid this part, you would need to buy and insert a Global System for Mobile communications (GSM) or Ethernet shield to the Arduino. In the case of smartphones, they obtain a native application that can send the information to the IoT platform and the libraries and source code that needs to parse the messages and access to the sensors and actuators that the user had defined. Both applications support the Midgar functionality to send to this IoT platform the messages with all the information about sensors and actuators through the Internet using the web services of the platform. These web services have been developed using a Representational State Transfer (REST) architecture. All the data from and to Smart Objects are sent using the REST service of Midgar, which is working on the HTTP protocol which we have improved with cryptography security in the previous work [54], a protocol using XML) files: From registering a new Smart Object to send and receive data from the IoT server. Source Code 1 shows an XML sent from an Android Smartphone to register it as a Smart Object in Midgar, as well as its services.

```
<register>
    <device>
        <idDevice>e6644a22aae2cec6</idDevice>
        <descriptionDevice>Nexus
4</descriptionDevice>
    </device>
    <service>
        <idService>e6644a22aae2cec6-
0</idService>

<description>Accelerometer</description>
        <sendType>float</sendType>
    </service>
    <action>
        <idAction>e6644a22aae2cec6-
1</idAction>
        <nameAction>Vibration</nameAction>

<descriptionAction>Vibration</descriptionActio
n>
        <hasMessage>int</hasMessage>
    </action>
</register>
```

Source Code 1. Registration XML example

Source Code 2 shows an example XML which has been sent from an Android smartphone to the server. In this message, we can see a datum about the accelerometer, and another one to activate for 30 seconds duration time of the vibration, and the proprietary in the service tag.

```
<send>
    <data>
        <datum>8.365426</datum>
        <service>e6644a22aae2cec6-0</service>
    </data>
    <data>
        <datum>30</datum>
        <service>e6644a22aae2cec6-1</service>
    </data>
</send>
```

Source Code 2. Data XML example

Source Code 3 shows an example XML which has been sent from the server to the Smartphone as a response. This contains a petition to make the actions 0 of that Smart Object and 1000 as a parameter. In this case, this means a vibration for 1 second. If the server would have more actions to this object, then, this XML will have more "answer" tags, one per answer.

```
<answers>
    <answer>
        <action>0</action>
        <result>1000</result>
    <answer>
</answers>
```

Source Code 3. Response XML example

Besides, this REST web server offers CRUD operations: create, remove, update, and delete information about the Smart Objects. In addition, using the REST web server it is possible to see the objects, their ID, and the services that are registered in the platform.

After that, Midgar decides, according to the interconnection which can be defined using MOISL, if it has to send a message to another Smart Object or only keep that information [22]. For this reason, if users want to interconnect objects, they need to register the Smart Objects in the IoT platform and create the interconnection using MOISL as we have described in [22] because MOISL is the DSL which allows creating the interconnection among objects. While in this paper, we introduce MOCSL, which is the DSL that creates all the necessary software, and that the objects need to use and interact with an IoT platform.

B. MIDGAR ARCHITECTURE FOR MOCSL

We have included a new part in Midgar platform to implement the new graphic DSL for the creation of software for objects, called MOCSL. Then, we have extended Midgar with a new DSL because the platform has been developing the principles of Model-Driven Engineering (MDE). This is why the platform was developed to be easy to reuse the existing layers and it is easy to modify to change different details. In this paper, we have added a new end-point which the final user has to use, using now MOCSL instead MOISL to define the Smart Objects. However, MOISL is still in use when final users want to define the interconnection among those objects. The other parts that have to be added were the processor and the generation of programs.

The system architecture has three layers, as can be seen in Fig. 1: *Process Definition, Object Generation*, and *Objects*. Each one is a process within the global set of the infrastructure. First of all, the *Process Definition* includes the user's process. In this layer, users (Fig. 1(1)) must define their application using MOCSL, which we have developed as an HTML5 application. When the user finishes the definition of their application (Fig. 1(2)), the model that the user-created with MOCSL is serialized in an XML file.

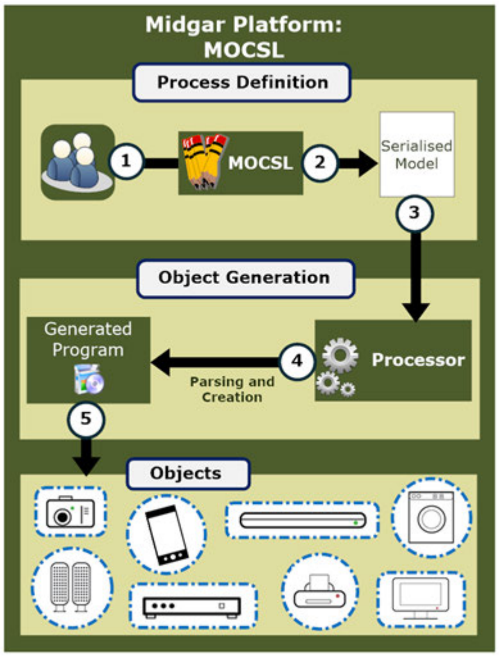


FIGURE 1. Architecture of the Midgar Platform using MOCSL.

The *serialized model* contains all the information about the user's application: the IoT network that he chose, the application name, the selected device, and the exact device type. Afterwards, the second layer, *Object Generation*, receives the *serialized model* and processes it in the *Processor* (Fig. 1(3)). Then, the *Processor* processes the information and creates and compiles the *Generated Program* (Fig. 1(4)). The *Generated Program* has all the functionality that the user had defined using MOCSL and the necessary logic to interconnect the objects with Midgar. When the user has the *Generated Program*, he only needs to upload the program to his device because the *Generated Program* has all the software and logic that that device needs to be a Smart Object and interconnect with an IoT platform (Fig. 1(5)).

C. IMPLEMENTATION

In this subsection, we are going to describe the new components and layers of the Midgar platform and the interaction between each one, the created DSL, MOCSL, how Midgar processed the serialized model which is created by users using MOCSL when they create an application, and how the processor parses it. Afterwards, we are going to talk about Android and Arduino applications.

1) MIDGAR OBJECT CREATION SPECIFIC LANGUAGE (MOCSL)

We have developed MOCSL using the HTML5 canvas element, showing all the configurable options to users. MOCSL can be divided into four different areas as we can see in Fig.2. The first area (Fig. 2(A)) contains the application data about the IoT network to be used, the application name, the device type to choose between different devices like a smartphone or an Arduino, and the exact device type such as a Nexus 4 or a Motorola MB525 in the case that you have chosen a smartphone or different Arduino types in the other case. Depending on the ''device type'', the next combo box can contain only smartphones or only microcontrollers. Then, when a user selects the ''exactly type'' in this combo box, the canvas shows the exact selected element. In our example, the Arduino Uno. Fig. 2(B) the user can select the analogue pins, while Fig. 2(C) contains the digital pins. On the right, the user has the serialized model which represents the object that the user-defined (Fig. 2(D)).

When a user selects a pin, he receives a popup with different sensors and actuators as we can see in Fig. 3. In this popup, the user can select a sensor or an actuator for the pin that he selected previously. If the user selects ''Cancel'' then, the pin will return to its default state. To include more sensors and actuators is necessary to develop or include the code of that exact object. However, it is only necessary to add the code one time because the next time this code will be reused.

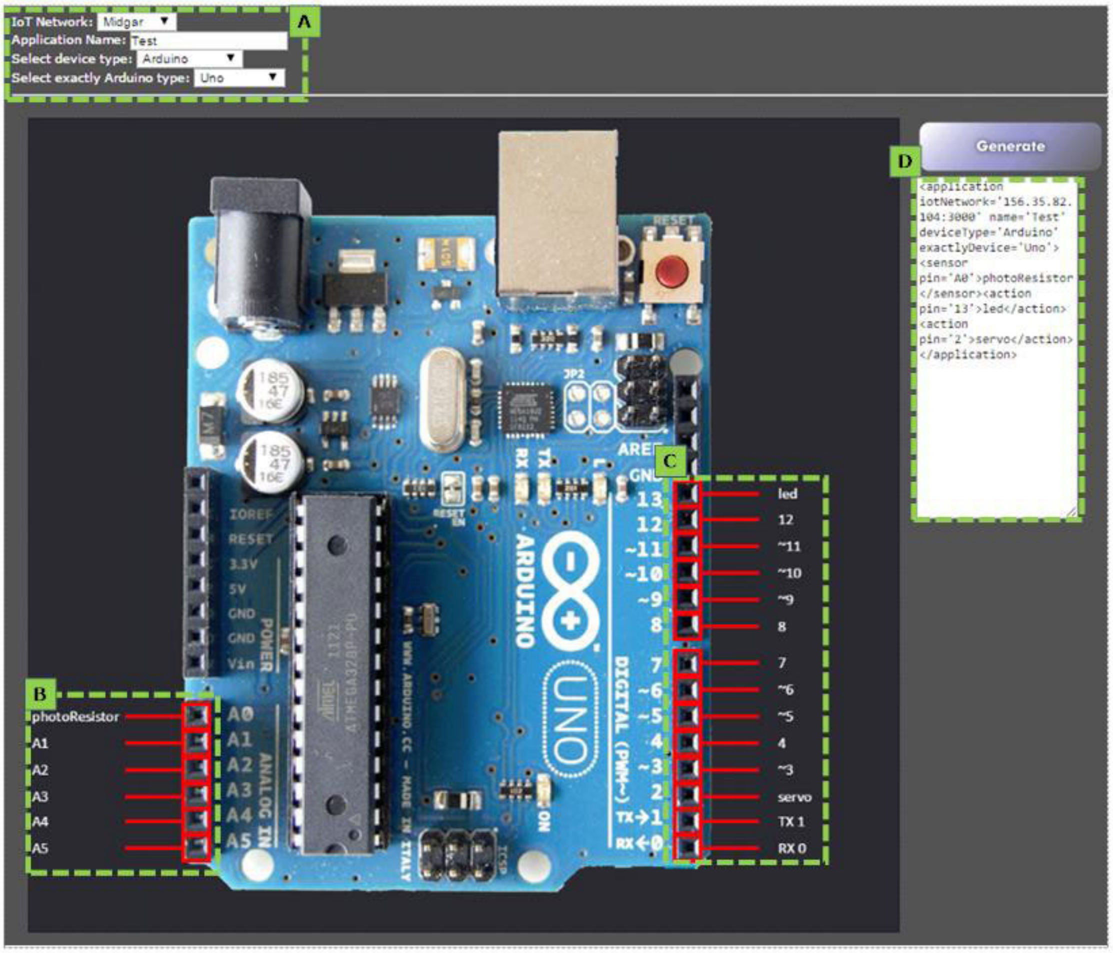
In Fig. 4, we show an example of a smartphone, exactly, a Nexus 4. The user sees the sensors of the Nexus 4 (a green tick or a red cross). and the sensors with no support by their version or sensors which do not have in their smartphone (a red dash). The user only needs to click on the sensor that he wants to change the state between a green tick, if he wants it, or to a red cross, which is the default state, if he does not want it.

Fig. 5 shows the actions in the example of the Android smartphone. This part functions exactly as the sensors part.

2) SERIALIZED MODEL

MOCSL creates a serialized model using XML syntax. We have respected in the serialization model the same nodes that we have in the graphic concrete syntax and the abstract syntax (Fig. 6), which we made with Ecore. Fig. 6 shows the metamodel and the abstract syntax which is the base of the concrete syntax and the semantic one, which are represented in the DSL (MOCSL). Besides, this avoids ambiguities because they are based on Ecore that is an implementation of MOF, which is the standard of used for this type of applications.

We have five main nodes: ''Application'', ''Network'', ''Smart Object'', ''Sensor'', and ''Actuator''. ''Application''



IoT Network: Midgar
Application Name: Test
Select device type: Arduino
Select exactly Arduino type: Uno

```
<application
iotNetwork='156.35.82.
104:3000' name='Test'
deviceType='Arduino'
exactlyDevice='Uno'>
<sensor
pin='A0'>photoResistor
</sensor><action
pin='13'>led</action>
<action
pin='2'>servo</action>
</application>
```

FIGURE 2. **MOCSL for the object creation of the Midgar platform for Arduino.**

is the parent node and it contains the name of the applications which will be deployed in the different devices: "Smartphone", "Arduino", or "Raspberry Pi". The "application" has a "network" which has its IP and its name because this is where the network in which the application will be working. The "network" can have different objects (Smart or not): a "Smart Object" which is composed of other Smart Object or by Sensors and/or Actuators.

In addition, the Smart Object abstract class has two parameters which contain the type of the device (Smartphone, Arduino, or Raspberry Pi) and the exact type of that device (Smartphone: Nexus 4, Moto G5S Plus,...; Arduino: Uno, Mega,...; Raspberry Pi: 1A, 1B, 1B+, 2B,...). Besides, the Smart Object can be a smartphone, an Arduino, or a Raspberry Pi. On the one hand, we have the first not-Smart Object, the "Sensor". We have created three abstract classes which can include the most used sensor in one of these ones: "SensorNeedAction", "SensorOneValue", and "SensorMoreThanOneValue". The first one includes sensors which need an action to work, like ultrasonic sensors which need the time in which you want to throw the ultrasonic sound. The second one is the abstract class for sensors which return only one value like temperature or humidity sensor. The last one is for sensors which need to return more than one value like Accelerometer (three axes), and DHT11 and DHT 22 (both have temperature and humidity together). On the other hand, we have the "Actuator". This abstract class has three abstract classes that could include almost any type of actuator: "ActuatorNoParameters", "ActuatorNeedOneValue", and "ActuatorSpecialised". The first one is for actuators which do not need anything, and only can turn on or turn off like a LED. The second one is for actuators that need a value, this is why we have the float parameter value. For instance, motors need the speed or the direction or the value to open or close a relay.

The last one is for the actuators that are specialized in one thing like the speaker, which needs two lists for the melody and the duration of each note. In both cases, "Sensor" and "Actuator", have four parameters to keep the data or the sensor of actuator that they have, for instance: the name, the pin, the type, and the source code if people want to introduce their own source code.

We show an example in Source Code 4 where we show the serialized model of an Android smartphone, exactly the

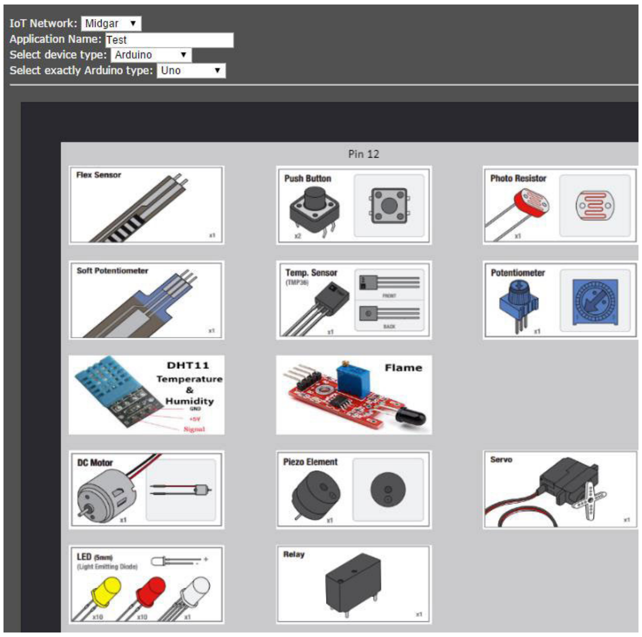

IoT Network: Midgar
Application Name: Test
Select device type: Arduino
Select exactly Arduino type: Uno

FIGURE 3. Arduino sensors and actions in the MOCSL selector.

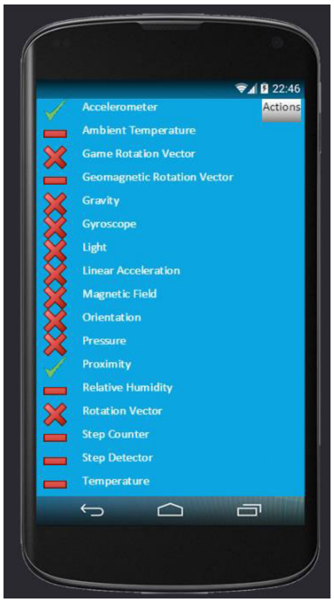


FIGURE 4. MOCSL for the object creation of the Midgar platform for Android smartphones: selecting sensors.

Nexus 4, which has an accelerometer and proximity sensors and the action "Toast".

The nodes "*sensor*" and "*action*" are similar but with a different name because they have semantic meaning to the processor. They can have the attribute "*pin*" when the device type is a microcontroller. In this attribute, we store the pin of that sensor or action. In Source Code 5, we show an Arduino Uno with a photoresistor sensor in the pin "A0" and three actions: a led in the pin 13, a DC motor in the pin 7, and a servo in the pin 2.

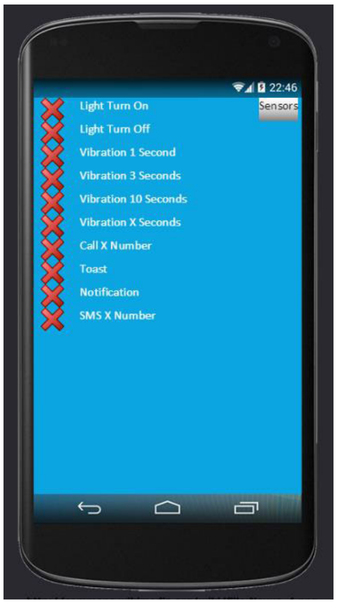


FIGURE 5. MOCSL for the object creation of the Midgar platform for Android smartphones: selecting actions.

```
<application iotNetwork='156.35.82.104:3000' name='Test'
deviceType='Smartphone' exactlyDevice='Nexus 4'>
  <sensor>Accelerometer</sensor>
  <sensor>Proximity</sensor>
  <action>Toast</action>
</application>
```

Source Code 4. Serialized model of an Android smartphone example

```
<application iotNetwork='156.35.82.104:3000' name='Test'
deviceType='Arduino' exactlyDevice='Uno'>
  <sensor pin='A0'>photoResistor</sensor>
  <action pin='13'>led</action>
  <action pin='7'>dcMotor</action>
  <action pin='2'>servo</action>
</application>
```

Source Code 5. Serialised model of an Arduino example

3) OBJECT GENERATION

As Fig. 7 shows, the second layer is composed of one process. The second layer receives the *Serialised Model* in XML format. Then, the processor parses the nodes and node attributes in order to obtain the necessary information about the application, which was defined by the user. With this information, the processor replaces in the templates the string tokens with the necessary information about Midgar, and the sensors and actions that were defined by the user. Clearly, the processor takes the corresponding template depending on the attribute "*deviceType*" and "*exactlyDevice*". When the processor finishes, the parsing creates the files and obtains the *Generated Application*. The *Generated Application* is the application that the user needs to deploy in their smartphone or microcontroller. This application has all the logic that that device needs to work properly, to interconnect with an IoT

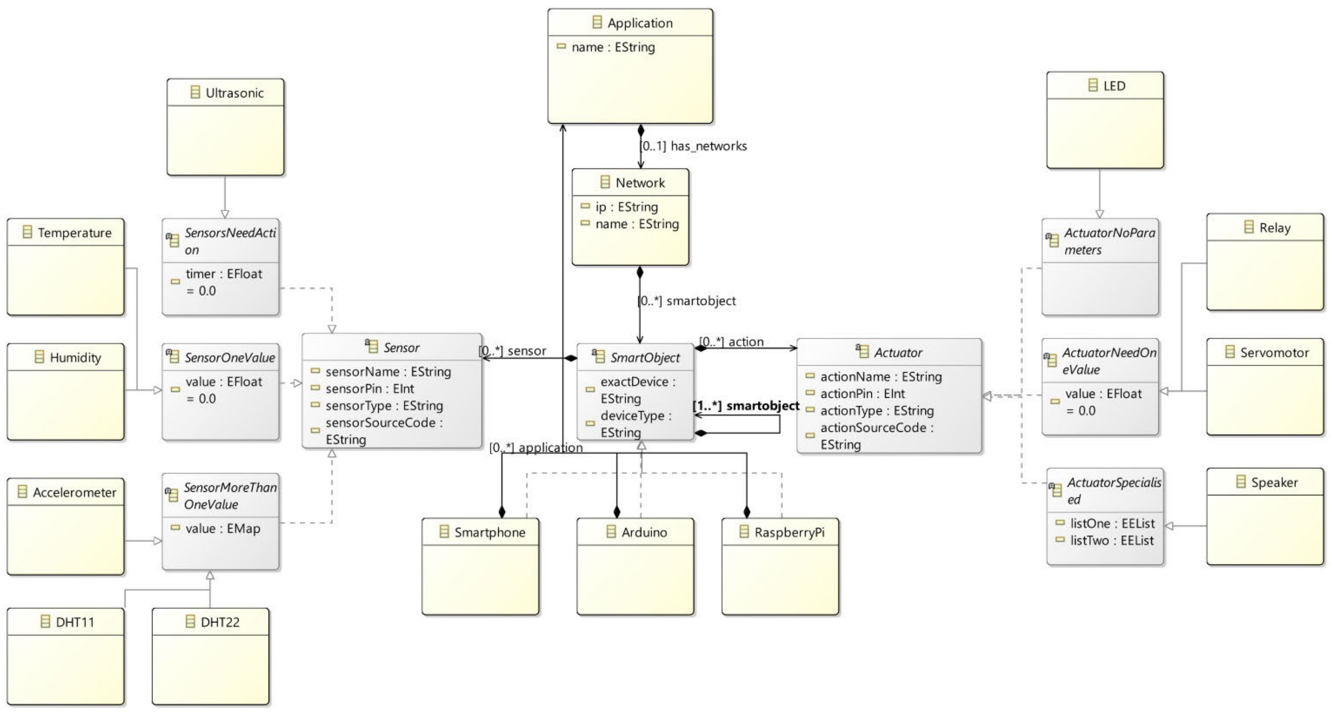


FIGURE 6. Metamodel and abstract syntax.

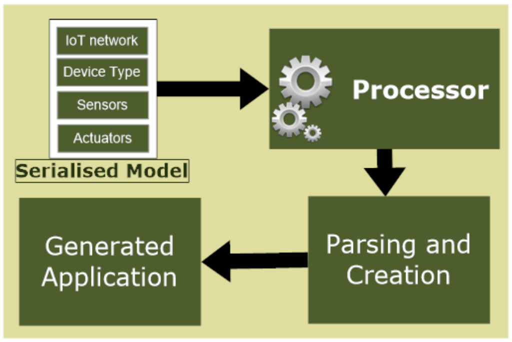


FIGURE 7. Internal working of the object generation.

FIGURE 8. Some of the used Android smartphones.

platform, all the logic to manage the sensors and actuators of that device. Then, in an automated way, this application has all the requirements to communicate with the Midgar IoT server without programming this part.

For instance, in the case of an Arduino, the processor creates the Arduino application and imports the necessary libraries for the corresponding sensor and actuators that were created before, the native libraries, or different libraries in C++, if the defined application needs someone to manage the sensors and actuators. On the other hand, the *Generated Application* for Android devices has all the logic in Java for Android, all the necessary import of different libraries, the logic to access to the sensors of that smartphone, and the logic to manage the different actions and actuators like the vibration, the flash, and so on.

4) ANDROID

We have developed an Android application that supports from the older Android version that is used, which is Froyo 2.2, to the newest Android version, which is Oreo 10. Thus, we have sought to ensure compatibility with all the supported and used versions in our current daily life. In Fig. 8, we can see three of the used Android smartphones. The application can work with each sensor between both versions. This Android application shows a list of the device sensors, the sensor values, and it supports the communication with Midgar: registering the application in Midgar and starting/stopping the data sending to Midgar. Based on this application, we created our template. This is why we change every part which MOCSL could modify with string tokens.

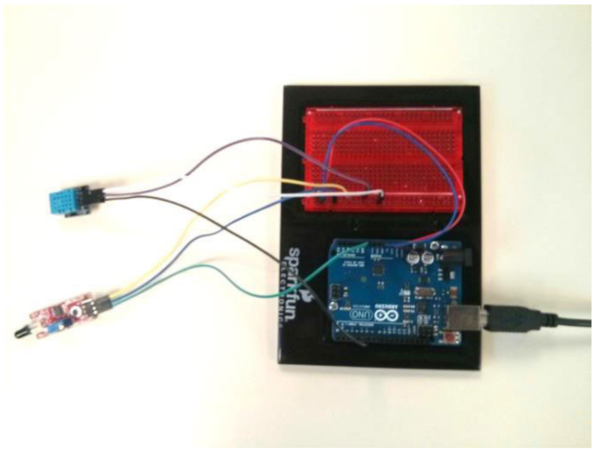


FIGURE 9. **Arduino picture during a testing with the DHT11 and flame sensors.**

5) ARDUINO

To solve the Arduino microcontroller part, we use an Arduino that uses the USB connection with our Personal Computer (PC) (Fig. 9). Then, we have used a Java application as an intermediary that reads and sends data through the USB to the Arduino and interconnects the Arduino and Midgar through the Internet. Using this intermediary, we could implement more tasks in it like saving all data in our PC for other intentions, for instance, to apply Big Data to improve our industrial processes. Thus, we have two applications: The Java intermediary and the Arduino source code in C or C++, it depends on the sensor that we use. Afterwards, to check some sensor and their possibilities with the Arduino, we created the template. We changed all the parts that MOCSL could modify with string tokens. Besides, we have in different files the source code of each sensor. Then, when we work with the Arduino template, we need to include code in the two parts: The Java application and the Arduino application but we process the template as only one application.

D. USED SOFTWARE AND HARDWARE

To develop this research work, different software types were required:

1) The Midgar server is based on Ruby 2.5.1p57 and it uses the Rails framework 5.2.1.
2) Thin web server 1.7.2
3) MariaDB 10.1.29-MariaDB-6
4) The graphic DSL, MOCSL, was developed by using the element HTML5 canvas and JavaScript without the use of any external library and using the current standards.
5) The application generator module was developed using Java 10 and work from Java 6.
6) Minibloq 0.83
7) Libraries used in the generated applications:
8) Arduino: RXTXcomm.jar for Arduino and Java, and HTTPComponents of Apache Software Foundation.
9) Android: HTTPComponents of Apache Software Foundation.

For the evaluation of the proposal, we used the next hardware components:

1) One Raspberry Pi 2 Model B as a dedicated server with Raspbian 8 (Jessie) and with the Kernel Version 4.9.75-v7+.
2) Five Android smartphones: A Moto G5S Plus running version 8.1, a Nexus 5GS Plus running version 7.1, a Nexus 4 running version 5.1.1, a Motorola with version 2.2.2, and a Samsung Galaxy Mini S5570 with version 2.3.6.
3) One Android emulator with Android 8.1
4) One Arduino Uno SMD microcontroller board based on the ATmega328.
5) During the various tests we used: the temperature sensor TMP36, a speaker, a servomotor, a DC motor, several LEDs, two buttons, a photoresistor, a temperature, and humidity sensor DHT11, and a flame sensor.

IV. EVALUATION AND DISCUSSION

In this section, we are going to explain in detail a related work comparison and the process of the evaluation that we used. Afterwards, we are going to show the results that we obtained during the evaluation. Firstly, we are going to talk about the methodology that we used to perform the evaluation. After that, we are going to show the obtained results and discuss them.

A. RELATED WORK COMPARISON

Compared with our proposal, current alternatives for smartphones and microcontrollers present three main drawbacks; 1) They do not allow incorporating the use of the complete sensor set like accelerometer, pressure, proximity, ambient light, etc. in their application. The only sensor that they usually include is GPS. This is a problem when users want to create an application for the IoT to interconnect their smartphones; 2) Editors are usually very generic with many functionalities that do not apply to the context of IoT applications and that make the creation of applications more complex and tedious for non-experts; and 3) Current solutions are not based on the facto standard of the industry for DSL development, in which a metamodel based on MOF/Ecore is created allowing portability, reusability, and interoperability with other applications and tools [43]. That makes current solutions very specific and not open to be adapted to other alternative scenarios.

We have designed a graphic DSL which supports the application creation for the IoT as a possible solution to the main drawbacks identified above. Thus, our proposal is a more abstract graphic DSL, based on standards, to simplify the application creation, and generating only the necessary source code for the application that is being designed.

Table 1 shows a comparison between the different existing approaches and this paper (MOCSL). We have evaluated the different characteristics that we propose in this paper to improve the IoT. From left to right:

TABLE 1. Comparison of the different proposals.

	Development Skill?	*Arduino?*	*Smartphones?*	*Sensors?*	*Web Services?*	*Personalization Level*	*Modify Code?*
AppsGeyser	No	No	Yes	No	No	Medium	No
iBuildApp [72]	No	No	Yes	No	No	Medium	No
Andromo [73]	No	No	Yes	No	No	Medium	No
AppsBuilder [74]	No	No	Yes	No	No	Medium	No
InfiniteMonkeys [75] / Appypie [76]	No	No	Yes	No	No	Medium	No
Como (Swiftic) [77]	No	No	Yes	No	No	Medium	No
AppMachine [78]	No	No	Yes	No	Partial	High	No
MiniBloq [79]	Yes	Yes	No	Yes	No	Medium	No
ArduBlock [81]	Partial	Yes	No	Yes	No	High	Yes
Scratch for Arduino [82]	Partial	Yes	No	Yes	No	Medium	No
Modkit [84]	No	Yes	No	N/D	No	N/D	N/D
Atmel Studio [85]	Yes	Yes	Partial	Partial	Partial	High	Yes
XOD [86]	Partial	Yes	No	Yes	No	Medium	No
MOCSL	No	Yes	Yes	Yes	Yes	High	Yes

1) Development skill?: Do People need to develop or have knowledge about development to create the application?
   a. Partial: Ardublock, Scratch, and XOD need that people know what a variable, a loop, the setup method, and different basic knowledge are.
2) Arduino? Does this application create software to Arduino?
3) Smartphones? Does this application create software for smartphones?
4) Sensors?: Does this application work with the sensor of the device (Arduino or Smartphone)?
5) Web Services?: Does this application sends to and receives from external web services (information, data, messages,...)?
   a. Partial: AppMachine allows downloading XML and JSON data to process them in the Smartphone, but not send data to a Web Service. Besides, you have to create the parser for that data.
6) Personalization level?: restrictions according to its domain (Arduino/Smartphone) and without counting the characteristic of the other columns of this table
   a. Medium: they have different templates with default images and sizes, and the users can change some parameters like the background, hyperlink, titles, index, lists, add more item in a list, etc.
   b. High: it allows creating an application without too many restrictions according to its domain and allows including more things than others.
      i. AppMachine: databases from excel and developer access.
      ii. Ardublock: operations (copy, glue) on the serial, write and read from the I2C, and storage.
      iii. MOCSL: work with IoT servers. Besides, you do not need to know how the sensor works, you only have to select the pin and the sensor.
7) Modify code?: Do the users modify the source code generated by this application?

8) Other important notes:
    a. N/D: now, the webpage of Modkit is not working correctly and the project seems to be abandoned. Then, it is impossible to verify this exact characteristic.
    b. Atmel Studio: This is an IDE like Visual Studio. Then, you can create anything, but you need high program knowledge. This is why has four columns with the partial value, you can create more or less "anything" but you need a lot of knowledge to do it.

B. METHODOLOGY

The main aim of the evaluation process is to validate if our solution is useful to enable users without programming skills to create Smart Objects, and moreover, if our solution is more efficient than other alternatives according to the time, the clicks, and the movement of the mouse required to perform a task. For this purpose, we split the evaluation process into two different phases.

1) Phase 1: In this first phase, users with two different profiles had to create a basic Arduino application using MOCSL and another graphic editor, whereas we were measuring the time, the clicks, and the movement of the mouse that they needed to complete the task. The two profiles are formed by people who are aware of Smart Objects and people who are not aware of Smart Objects. In this phase, we have chosen MiniBloq because this is the one which represents better the different alternatives for Arduino. We have not chosen any other from the smartphone alternatives because they only allow using templates and they do not have support for sensors.
2) Phase 2: In this phase, the participants who had done the first phase made a survey based on the 5-points Likert scale [88] where they gave their opinion about some declarations. These declarations are based on phase 1 and they are about how our research could improve some aspects of the IoT.

We tried to obtain a complete evaluation to verify in the best way our hypotheses using two different evaluations: a quantitative evaluation for phase 1 and another qualitative evaluation for phase 2.

1) PHASE 1

As we have already said, in this phase, participants from both profiles had to create a basic Arduino application using MOCSL and another graphic editor. We chose MiniBloq as the other graphic editor because it is a graphic editor for Arduino programming. Whereas participants were performing an assigned task on each platform, we measured the next parameters: time in seconds to complete the task, the centimeters that the participants moved the mouse, the clicks in the right mouse button, and the clicks in the left mouse button. To obtain these data we used the Mousotron tool [89]. Later, in the results subsection, we will handle all clicks together by summing them.

Firstly, we defined a basic task for participants to emulate a real scenario, but we had to be cautious with the sensors and actuators of each platform because some types could not be supported.

The assigned task was to create an object with one sensor and two actuators. We chose a photoresistor sensor, and a led and a servomotor actuator. The task was to create an object that simulates a motor capable of renovating the air when a photoresistor receives light. Moreover, when the motor is running, a status LED must be lighting.

Here, we chose an easy and trivial task because the majority of the IoT applications are to manage and watch basic parameters like the temperature, humidity, open/close a window, and so on. However, to develop this simple task, people need to spend hours developing an application, in the case that they have the necessary knowledge to program. Besides, if people would want to add their own source code, they could do it using the Java Code tag.

2) PHASE 2

After finishing the first phase, we wanted to obtain the participants' opinion about our proposal. Therefore, the participants had to complete an anonymous survey based on the 5-points Likert scale because it is the most used in the design of scales. The given options were the following: 1 as strongly disagree, 2 as disagree, 3 as neutral, 4 as agree, and 5 as strongly agree.

Furthermore, we decided to use this method because it is a very used one in the software engineering field to obtain information that effectively supports decision-making [90]. This is exactly our case because we can measure neither the efficiency of object creation through a DSL nor its potential. Hence, we have used the survey to obtain more information that could help us to check our hypotheses.

The survey is composed of a set of twelve declarations that are shown in Table 1.

C. RESULTS

In this section, we are going to show and discuss the obtained results in each phase. In the first subsection, we are going to show the results of the first phase. After that, we are going to show the results of the second phase. These results are going to be analyzed by an inter-subject study because of the existence of two different groups, people who have knowledge about Objects (10 users) and people who are not aware of Smart Objects (9 users).

1) PHASE 1

In this first phase, we measured for each participant the time in seconds to do the task, the clicks in the primary mouse button, the clicks in the secondary mouse button, and the centimeters that each participant moved the mouse to finish the task.

We divided the analysis of the results in three parts. Firstly, we are going to compare the results collected from

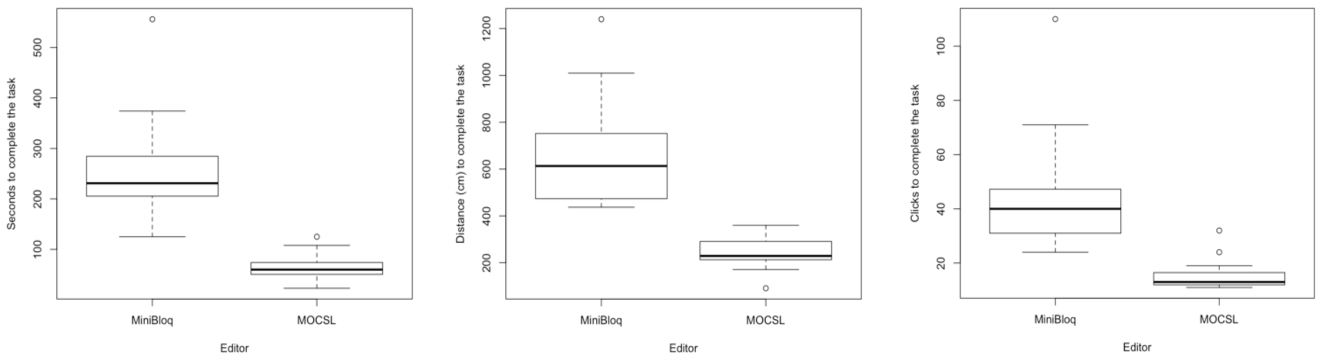


FIGURE 10. Differences between time, distance, and clicks needed to complete the task with both editors.

both editors, MOCSL and MiniBloq, to look for significant differences in order to check if our proposal produces an improvement of the measures. Secondly, we are going to compare the results from both profiles using MOCSL to look for significant differences in order to check if our proposal is useful for everybody to the same extent. Finally, we are going to analyze the measured data relative to the time, the clicks, and the movement of the mouse that each participant required to complete the assigned task.

a: COMPARISON BETWEEN EDITORS

In the first part of the analysis of the results, we are going to perform statistical tests to compare MOCSL and MiniBloq. Before performing a statistical test that compares the editors, we must determine if the sample data follow a normal distribution and perform a homoscedasticity test. In the following enumeration, we are going to check the normality and the homoscedasticity, and we are going to check if there are significant differences between each variable from both editors.

- *Time*: The data from MiniBloq (Shapiro-Wilk test $\rightarrow$p=0.0179) do not follow a normal distribution whereas the data from MOCSL (Shapiro-Wilk test$\rightarrow$p=0.0944) follow a normal distribution. Moreover, the variances are not homogeneous (Levene's test$\rightarrow$p=0.0085). After applying a Wilcoxon test for a paired sample, we obtained that there ARE significant differences (p=3.815×10^{-6}) between the time spent to complete the task with both editors.
- *Distance*: The data from MiniBloq (Shapiro-Wilk test$\rightarrow$p=0.014) do not follow a normal distribution whereas the data from MOCSL (Shapiro-Wilk test$\rightarrow$p=0.6878) follow a normal distribution. Moreover, the variances are not homogeneous (Levene's test$\rightarrow$p=0.0025). After applying a Wilcoxon test for a paired sample, we obtained that there are significant differences (p=3.815×10^{-6}) between the distances of the mouse needed to complete the task with both editors.
- *Clicks*: The data from MiniBloq (Shapiro-Wilk test$\rightarrow$p=0.001) and MOCSL (Shapiro-Wilk test $\rightarrow$p=0.0003) do not follow a normal distribution. Moreover, the variances are not homogeneous (Levene's test $\rightarrow$p=0.0136). After applying a Wilcoxon test for a paired sample, we obtained that there ARE significant differences (p=0.0002) between the clicks needed to complete the task with both editors.

Finally, we obtained that there are significant differences between the data from MiniBloq and the data from MOCSL as Fig.10 Figure 10 shows. Moreover, Fig.10 also shows that the values obtained from MOCSL are lower than the data from MiniBloq. We can interpret these results as our proposal provides different and more efficient results than MiniBloq. Thus, we have validated a part of our hypotheses, our proposal is more efficient than the other alternatives according to the time, the clicks, and the movement of the mouse required to create the Smart Objects software.

b: COMPARISON BETWEEN PROFILES

In the second part of the analysis of the results, we are going to perform statistical tests to compare the results from the two different profiles, people who are aware of Smart Objects (Smart Objects Aware), and people who are not aware of Smart Objects (Not Smart Objects Aware), using MOCSL. Before performing a statistical test that compares the results from both profiles, we must determine if the sample data follow a normal distribution and perform a homoscedasticity test. In the following enumeration, we are going to check the normality and the homoscedasticity, and we are going to check if there are significant differences between each variable from both profiles.

- *Time:* The data from Smart Objects Aware (Shapiro-Wilk test$\rightarrow$p=0.3145) and Not Smart Objects Aware (Shapiro-Wilk test$\rightarrow$p=0.3631) follow a normal distribution. Moreover, the variances are homogeneous (F test$\rightarrow$p=0.0751). After applying a T-test, we obtained that there are NOT significant differences (p=0.5974) between the time spent to complete the task by both profiles.
- *Distance:* The data from Smart Objects Aware (Shapiro-Wilk test$\rightarrow$p=0.8759) and Not Smart Objects Aware (Shapiro-Wilk test$\rightarrow$p=0.173) follow a normal distribution. However, the variances are not homogeneous (F test$\rightarrow$p=0.0408). After applying a

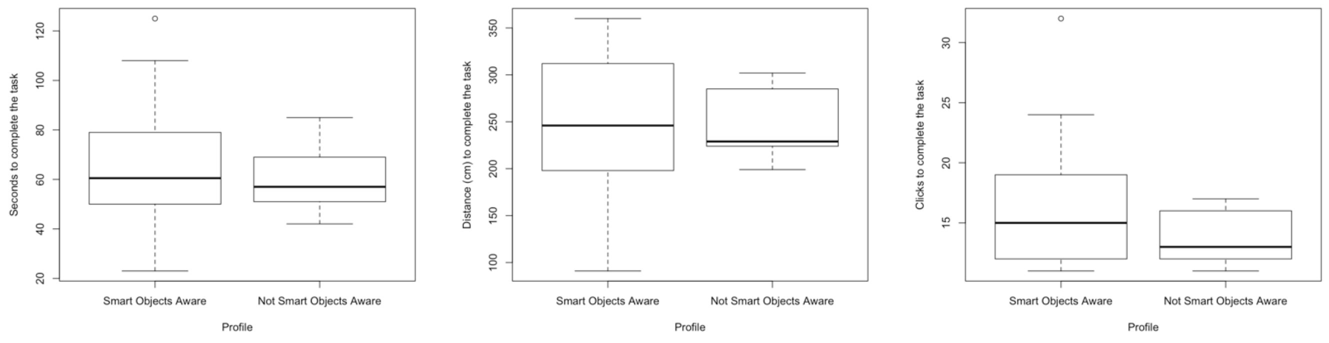


FIGURE 11. Differences between time, distance, and clicks needed to complete the task by both profiles.

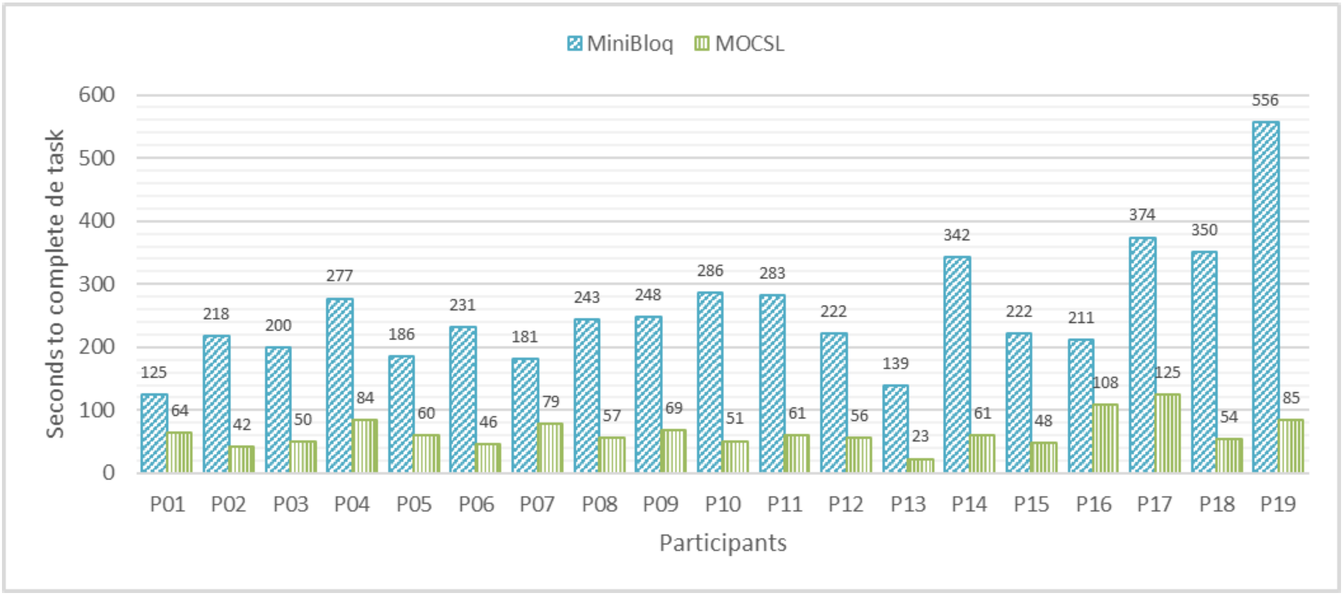


FIGURE 12. Seconds needed by participants to complete the task with each editor.

Mann-Whitney U, we obtained that there are NOT significant differences (p=0.9025) between the distances of the mouse needed to complete the task by both profiles.

- *Clicks:* The data from Smart Objects Aware (Shapiro-Wilk test→p=0.0232) do not follow a normal distribution whereas the data from Not Smart Objects Aware (Shapiro-Wilk test→p=0.0581) follow a normal distribution. Moreover, the variances are homogeneous (Levene's test→p=0.1342). After applying a Mann-Whitney U test, we obtained that there are NOT significant differences (p=0.3849) between the clicks needed to complete the task by both profiles.

Finally, we obtained that there are NOT significant differences between the data from users who are aware of Smart Objects and the data from users who are not aware of Smart Objects as Fig. 11 shows. Differences between time, distance, and clicks needed to complete the task by both profiles. We can interpret these results as our proposal (MOCSL) is valid for both profiles, so users do not need experience with Smart Objects to benefit from MOCSL. Thus, we have validated a part of our hypotheses: it is possible to create a DSL that enables people without experience in the development of applications for Smart Objects to create Smart Objects.

c: ANALYSIS OF THE MEASURED DATA

As we have already said, the third part of the analysis of results is the analysis of the measured data. In the following lines, we are going to present the data about the time, the clicks, and the movement of the mouse that participants needed to complete the assigned task. Fig. 12 shows the time that each participant needed to make the same task on each platform.

Analysing this chart, we can suggest the following interpretations:

- The faster participant in each platform needed 125 seconds in MiniBloq and 23 seconds in MOCSL. Then MOCSL is the quickest platform based on the best results.

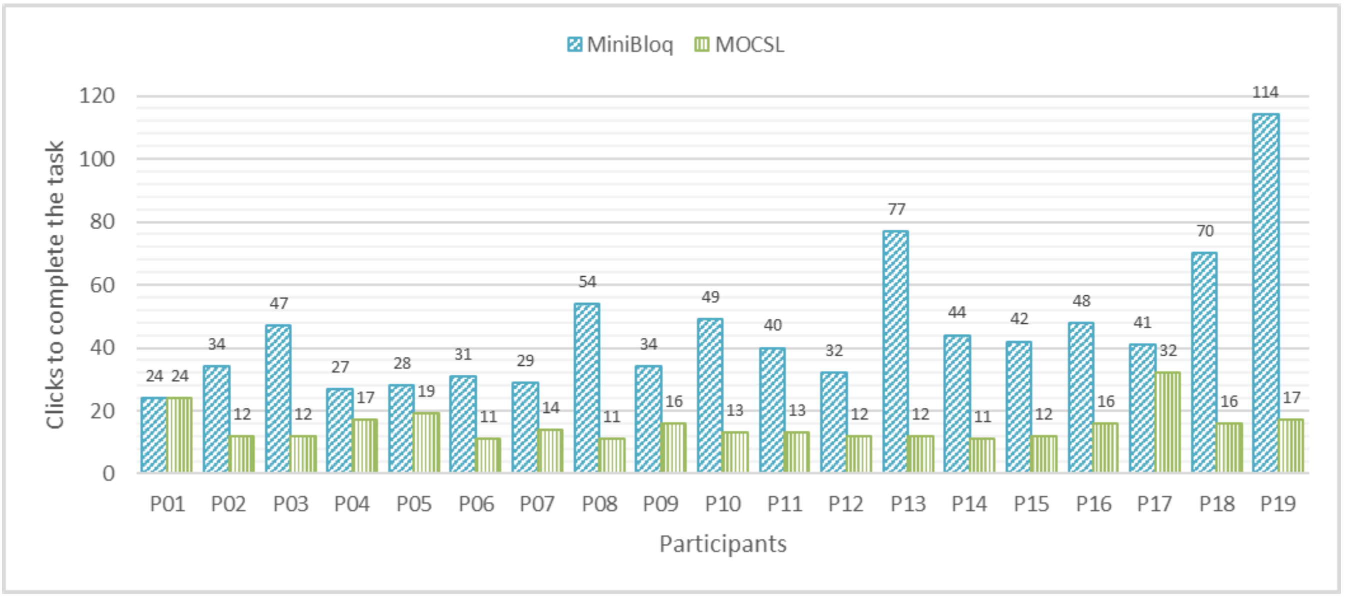


FIGURE 13. Clicks needed by participants to complete the task with each editor.

- The slower participant in each platform needed 556 seconds in MiniBloq and 85 seconds in MOCSL. Then MOCSL is the quickest platform based on the worst results.
- The best result using MiniBloq was of 125 seconds, meanwhile, the worst result using MOCSL was 125. Only if we compare the best time in MiniBloq and the worst time in MOCSL, we obtain the same time.
- The average time for the nine participants were 258 seconds for MiniBloq and 64 for MOCSL. The participants in MOCSL only need 24.8% of the time than MiniBloq to make the same application.
- MiniBloq is the slowest in every case and MOCSL is faster than the MiniBloq to create objects in all cases. This indicates that MOCSL offers more efficiency to solve the object creation.

Fig. 13 shows the total clicks (the addition of primary clicks and secondary clicks) needed by each participant to complete the given task with each editor. However, in the case of MOCSL, it avoids the use of the secondary button to reduce the complexity and it means that MOCSL only needs the primary button. In the case of MiniBloq, the secondary button has shortcuts and the delete option.

If we analyse this chart we can interpret:

- The biggest number of mouse clicks in each platform was 114 clicks in MiniBloq and 32 in MOCSL. Basing on this, MOCSL needs fewer clicks than the other platform in the worst case.
- The lowest number of mouse clicks in each platform was 24 in MiniBloq and 11 in MOCSL. According to the best participants, MOCSL needs fewer clicks.
- The average of clicks in each platform was 43.36 in MiniBloq and 15.26 in MOCSL. Again, MOCSL needs almost a third part of clicks than MiniBloq to create an object.
- MOCSL is always the platform with fewer clicks. This indicates that MOCSL is the most useful platform.

We show in Fig. 14 the distance in centimetres that each participant needed in each platform.

If we analyze this last chart, we can suggest the next interpretations:

- The biggest distance in each platform was 1,240 centimeters in MiniBloq and 360 in MOCSL. This indicates that in the worst-case MOCSL only needs almost a fourth part of clicks than MiniBloq.
- The lowest distance in each platform was 437 centimeters in MiniBloq and 91 in MOCSL, which indicates that in the best case, MOCSL needs four times less than MiniBloq.
- The average distance in each platform was 651.15 centimeters in MiniBloq and 242.47 in MOCSL. Then, BitBloq is the platform that requires more mouse movement to make the task and MOCSL is the platform which needs less distance to solve the object creation problem, exactly, almost three times less.
- The worst distance in MOCSL, 360 centimeters, is less than the best distance in MiniBloq, 437 centimeters. Then, MOCSL is always the platform which needs less distance to solve the task.

2) PHASE 2

In the second phase, participants had to fill a 5-point Likert Scale [88] survey. As we explained before, the options of the survey were: Strongly Disagree, Disagree, Neutral, Agree, and Strongly Agree. To make it easier to analyze, we associate

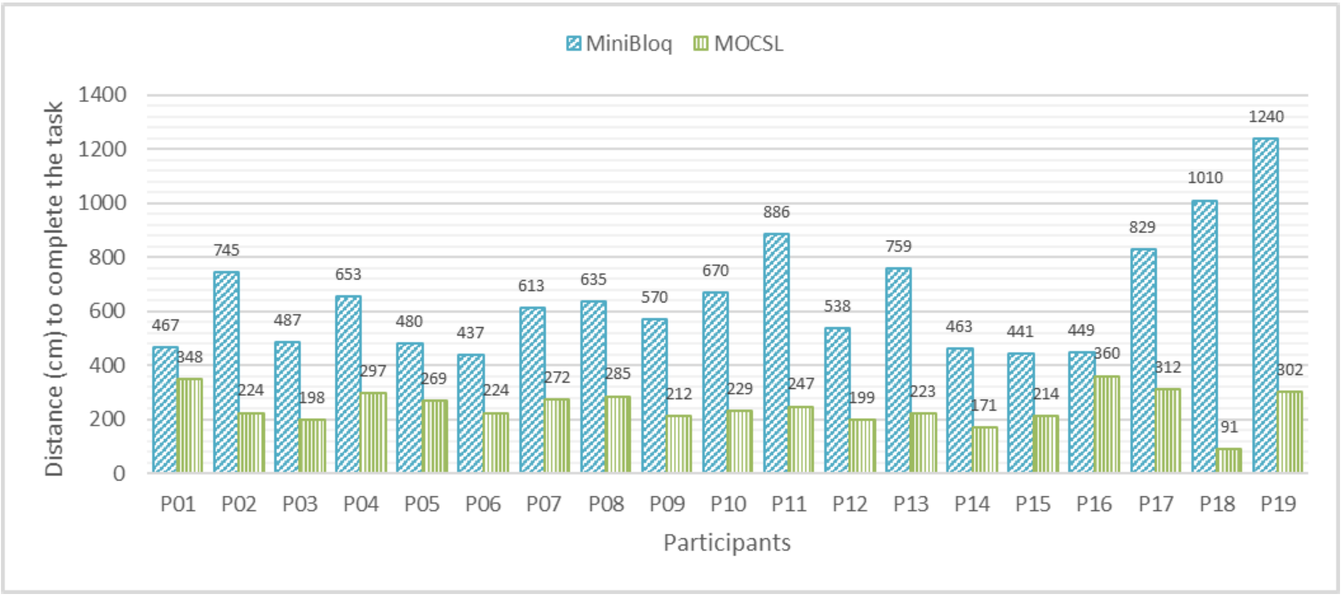


FIGURE 14. Mouse movement in centimeters needed by participants to complete the task with each editor.

TABLE 2. Survey given to participants.

#	Declaration
D01	The user understands the functionality of the DSL elements and their role in the application creation process.
D02	This DSL allows creating Smart Objects in an easy way, using a few clicks and without having to program the source code.
D03	The DSL approach makes it difficult to make mistakes while the user is modelling the application.
D04	This solution offers a fast way of developing the indicated task.
D05	This solution provides assistance to create applications for Smart Objects.
D06	Based on previous experience with the use of other tools, this DSL provides a greater ability to generate such applications.
D07	The DSL does not require that the user has to use complex programming skills, as in traditional application developments.
D08	The DSL includes enough elements and functionality for the user to create a wide range of applications to Smart Objects.
D09	This proposal is a positive contribution to encourage the development of services and applications for Smart Objects.
D10	The Internet of Things and Smart Objects will benefit from this solution.
D11	This DSL could be used to simplify the classic development process of software applications in other areas.
D12	This DSL is easier to use than MiniBloq.

numeric values to each option from 1 to 5 according to the worst and the best opinion

Table 2 shows the responses of each participant in an anonymous way by indicating the profile of each participant, the numeric value of each answer, and the total score of each participant.

From the data collected, we want to conclude if the relationship between having experience with Smart Objects and the opinions, expressed via the Likert survey, is significant. In this way, we calculated a total score per participant, although it is important to mention that the total score is an ordinal value because different scores in an answer in Likert Survey represent different grades of opinion but not in an equidistant way. Thus, a greater score represents a better opinion, but it cannot represent how much better that opinion is. Therefore, we cannot perform statistical analyses that compare the averages to validate our hypotheses, consequently, we will use a non-parametric test even though the data would fit a normal distribution.

We checked the homogeneity of variances with the F test because the data from both profiles follow a normal distribution (Shapiro-Wilk test$\rightarrow$p=0.3687, p=0.4483), and we applied a Mann-Whitney U test because of the homogeneity of variances (p=0.9311) and the ordinal nature of the values. Finally, we obtained that there are NOT significant differences (p=0.7119) between the opinions of users from both profiles as Fig. 15 also shows.

At this point, we can assume that being aware of Smart Objects, does not influence the participant's opinions given via the Likert survey. Therefore, we can analyse the results of the Likert survey globally, without discriminating both profiles.

Table 3 shows the descriptive statistics of the survey. This table is composed of the itemisation of each declaration: the minimum, the first quartile, the median or second quartile, the third quartile, the maximum, the range between quartiles, and the mode. We can see in Fig. 16 the same data but in a Box and Whiskers plot diagram.

We can suggest the next interpretations by analysing Table 3 and Fig. 16:

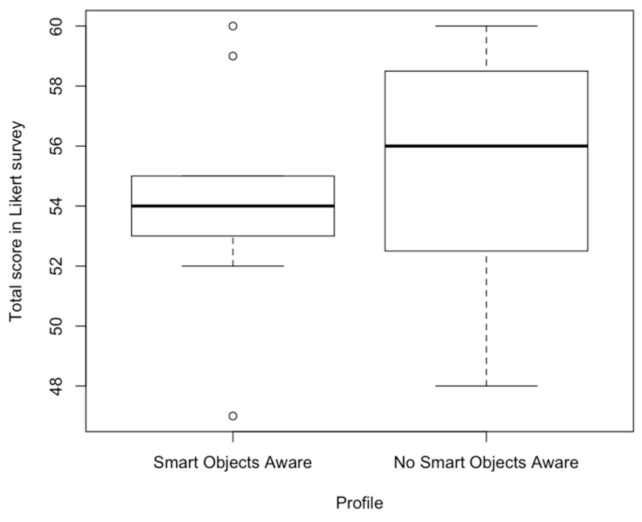


FIGURE 15. **Box and whiskers plot for the total score in Likert survey per profile.**

- D2, D3, D4, and D10 have the highest minimum, 4 out of 5. This means that all participants agree with these declarations, so in this case, they at least agree. On the other hand, with the lowest minimum is the D11. This indicates that D11 is the question with the highest difference of opinions between the participants.
- D1, D2, D3, D4, D5, D6, D7, D9, D11, and D12 have the highest median, exactly, 5 out of 5. From this data, we can interpret that most of the participants strongly agree with these declarations. Another question is Q10 with a median of 4, then, all participants are minimum agree.
- D2 and D4 are questions with a range of 1. This demonstrates that almost all the participants have a very close opinion on these declarations. In our case, the participants' opinion in these questions is "Agree" and

TABLE 3. **Participants' responses to each question.**

	Smart Objects Aware	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10	D11	D12	Total
P01	Yes	5	5	4	5	5	4	4	5	5	5	1	5	53
P02	No	5	5	5	5	3	5	5	4	4	4	5	5	55
P03	Yes	3	5	5	5	5	5	3	3	5	4	4	5	52
P04	No	3	5	4	5	4	5	5	3	4	4	1	5	48
P05	Yes	5	5	5	4	5	4	5	5	3	4	4	5	54
P06	No	5	5	5	5	5	5	5	5	5	5	5	5	60
P07	Yes	5	5	4	5	5	5	5	5	5	5	5	5	59
P08	No	5	5	4	5	5	5	5	5	5	4	5	5	58
P09	No	5	5	5	5	4	4	5	4	5	5	5	5	57
P10	No	5	5	4	5	5	3	5	3	4	5	2	5	51
P11	No	4	5	4	4	4	5	5	4	5	5	5	4	54
P12	No	5	5	5	5	5	5	5	5	5	5	4	5	59
P13	Yes	5	5	4	5	5	5	4	3	5	5	4	5	55
P14	Yes	5	5	5	5	5	4	5	5	4	4	4	4	55
P15	Yes	5	5	5	5	5	5	5	5	5	5	5	5	60
P16	Yes	3	5	5	5	5	4	4	4	5	5	4	4	53
P17	Yes	4	4	5	5	4	3	5	3	4	4	3	3	47
P18	Yes	5	5	5	5	5	5	5	5	5	5	5	5	60
P19	No	5	5	5	5	5	4	4	4	5	5	5	5	57

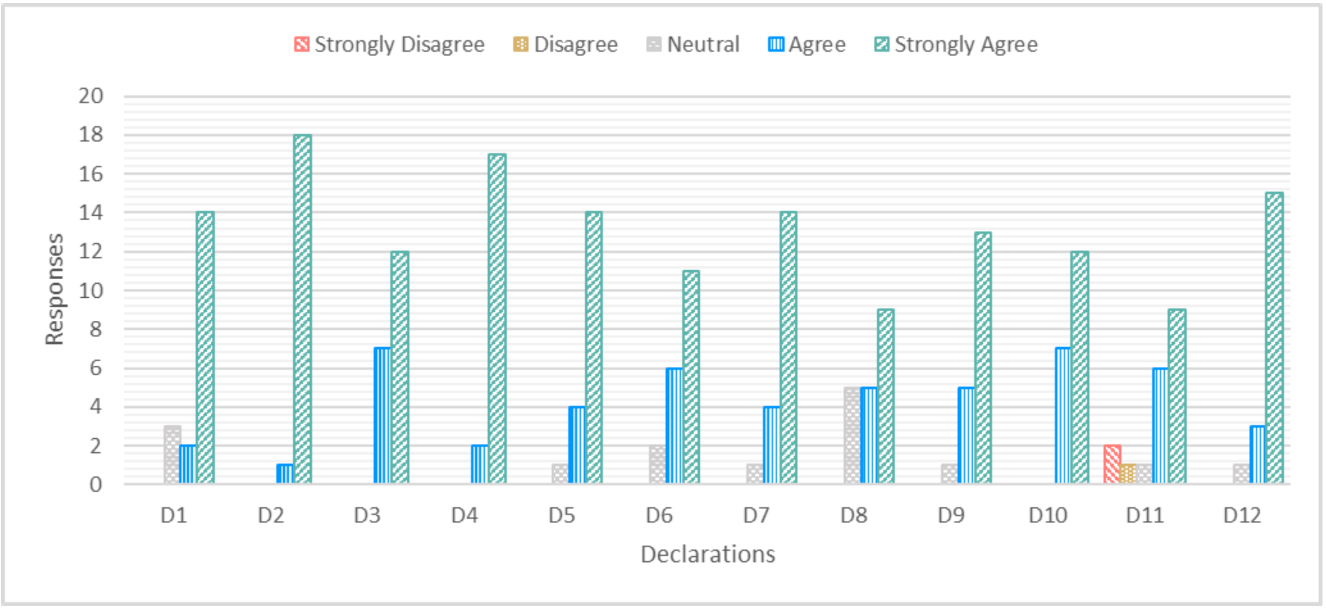


FIGURE 16. **Overall response distribution.**

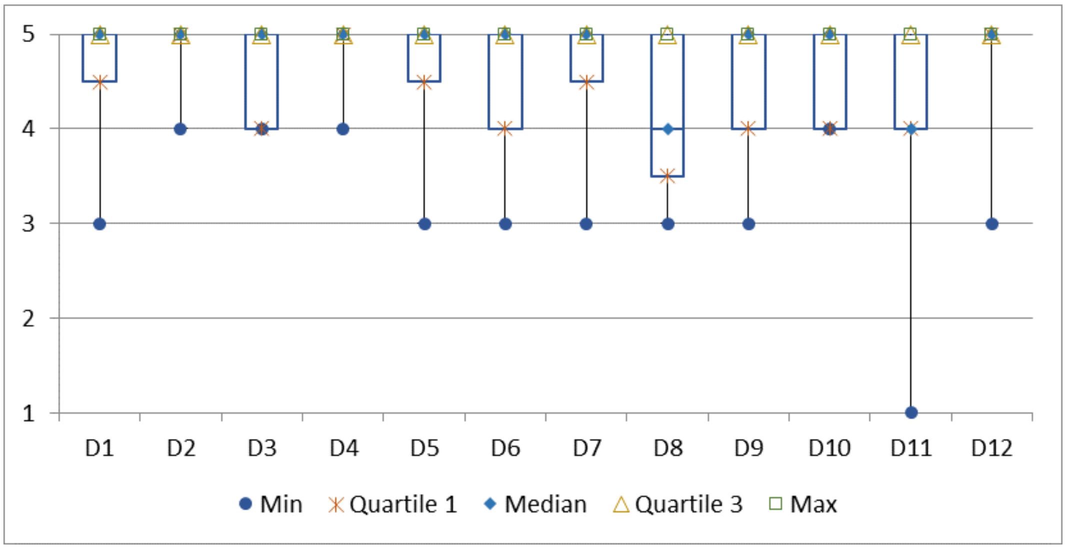


FIGURE 17. Box and whiskers plot for each question.

TABLE 4. Table with the general descriptive statistics.

	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10	D11	D12
Min	3	4	4	4	3	3	3	3	3	4	1	3
Quartile 1	4.5	5	4	5	4.5	4	4.5	3.5	4	4	4	5
Median	5	5	5	5	5	5	5	4	5	5	4	5
Quartile 3	5	5	5	5	5	5	5	5	5	5	5	5
Max	5	5	5	5	5	5	5	5	5	5	5	5
Range	2	1	1	1	2	2	2	2	2	1	4	2
Inter Qrt.-Range	0.5	0	1	0	0.5	1	0.5	1.5	1	1	1	0
Mode	5	5	5	5	5	5	5	5	5	5	5	5

"Strongly Agree". On the contrary, D11 has a range of 4. This expresses a high difference between participants' opinion about this question.

- Regarding the mode, all questions have a mode of 5 which indicates that the most chosen answer was "Strongly agree".
- If we study the maximum, we can see that all questions have a 5 out of 5. This indicates that in all questions, somebody chose "Strongly agree".
- About the Interquartile range, D2, D4, and D12 have a 0. This indicates that the participants had a very close decision between themselves.

Table 4 shows the different frequencies for each question based on the participants' answers. In this table, we have a breakdown of each question to show the number of votes for each decision and the corresponding percentage for them. In Fig. 17, we show the same data but using a graph bar with the frequency of the answers.

TABLE 5. Frequency table for the general responses.

		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
D1	#	0	0	3	2	14
	%	0%	0%	16%	11%	74%
D2	#	0	0	0	1	18
	%	0%	0%	0%	5%	95%
D3	#	0	0	0	7	12
	%	0%	0%	0%	37%	63%
D4	#	0	0	0	2	17
	%	0%	0%	0%	11%	89%
D5	#	0	0	1	4	14
	%	0%	0%	5%	21%	74%
D6	#	0	0	2	6	11
	%	0%	0%	11%	32%	58%
D7	#	0	0	1	4	14
	%	0%	0%	5%	21%	74%
D8	#	0	0	5	5	9
	%	0%	0%	26%	26%	47%
D9	#	0	0	1	5	13
	%	0%	0%	5%	26%	68%
D10	#	0	0	0	7	12
	%	0%	0%	0%	37%	63%
D11	#	2	1	1	6	9
	%	11%	5%	5%	32%	47%
D12	#	0	0	1	3	15
	%	0%	0%	5%	16%	79%

Based on these, we can suggest the following interpretations:

- D2 has 89% of the votes in "Strongly Agree". This indicates that almost all participants strongly agree with this sentence, except 1 person.
- With 89% of votes, we have the D4. This is an amount of 17 votes for "Strongly Agree" and two votes for "Agree".
- D3 and D10 have a minimum of 37% of the votes in "Agree". This indicates that all participants agree with these sentences.

- D1, D5, D6, D7, D8, D9, and D12 have a minimum of 58% of votes in "Strongly Agree". These are 11 votes. Meanwhile, D8 is the declaration with more votes in "Neutral" with 5. This indicates that the majority of people agree with these sentences but there is a minority with some doubt.
- D8 has a 47% of votes for "Strongly Agree" and 26% of votes for "Agree" and "Neutral". These numbers correspond with 9, 5, and 5, respectively. This indicates that the majority of participants agree but there is an important percentage indecisive or who do not believe in this declaration.
- D11 is the only declaration with the five options chosen: 47% of the votes in "Strongly Agree", 32% in "Agree", 5% in "Neutral", 5% in "Disagree", and 11% in "Strongly Disagree". In numbers, this means 9, 6, 1, 1, and 2, respectively. This indicates that the participants were indecisive about this question or maybe we had formulated it in a wrong way.

V. CONCLUSION

In this paper, we improve the Midgar platform with a novel proposal about a new graphic DSL called MOCSL, which provides a solution to create Smart Objects and all the logic that they need to interconnect them through Midgar using MDE. This verifies our first hypothesis. Normally, the creation of Smart Objects is very complex because this process needs that people create an application to recollect the object data, process it, and send it to another object or an IoT platform. Therefore, this process requires complex programming skills because of the different available technologies, APIs, and types of sensors and actuators.

Until now, Midgar had a partial solution: It had enabled interconnecting different registered objects using MOISL. But now, Midgar also enables the creation of the software that these objects need to obtain the data, work with their actuators, and the ability to interconnect with the platform. We have created a solution using a graphic DSL to obtain an abstraction to facilitate this task for any people without development knowledge, called MOCSL. In our case, people only need knowledge about the domain, which is the IoT and their smartphone or Arduino microcontroller, what they want to do, and how their objects are. These people can be workers of a company specialized in the IoT or a similar system, a government team to improve the city or the buildings, or maybe people who like the DIY.

In order to validate the effectiveness of our proposal, we compared MOCSL with other graphic editors that offer a similar solution. First of all, in the state of the art, we can see that Minibloq and other tools have difficulties for the industrial area because they are very generic with functionalities outside the IoT, other ones do not incorporate the use of sensors, which are crucial in IoT scenarios, and these solutions are not based on any standard. After, we made an evaluation to check exhaustively our proposal comparing it with MiniBloq for Arduino microcontrollers. This evaluation was composed of two different phases in which we did a comparison of data that we had measured while participants were performing a specific task with two different editors, MOCSL and MiniBloq. We also did a comparison of data from two different participant profiles according to their past experiences with Smart Objects, and we collected participants' opinion through a Likert survey.

The quantitative evaluation demonstrated that MOCSL is the fastest editor because the participants only needed an average of *64 seconds* to do the task. In comparison with the other platform, this supposed *24.8% of the time than MiniBloq* to create the same application. Furthermore, MOCSL only needed *almost a third part of clicks than MiniBloq*. Although MOCSL avoids the use of the mouse secondary button and MiniBloq uses this button to delete or offer other shortcuts. This demonstrates that MOCSL reduces the complexity in the control without incrementing the number of clicks. Besides, MOCSL is the platform, which needed less distance to create the same task, exactly an average of *242.47* centimeters versus 651.15 centimeters in MiniBloq. These last two data suppose that MOCSL is more usable. This verifies our second hypothesis as true. Furthermore, if people do not use these types of programs, they will need hours and knowledge about software development to create one application.

Respecting the qualitative evaluation, we could interpret that the participants chose "Strongly Agree" in 69% of the declarations and they chose "Agree" on 22% of occasions. Thus, 91% of the declarations have a positive or very positive assessment. In this way, the participants think that MOCSL accomplishes with its functionality to facilitate the object creation in an easy way without the necessity of writing source code. In addition, they believe that MOCSL could offer benefits for the Internet of Things and Smart Objects.

Noteworthy that some participants chose in 8% of responses as "Neutral" and in 1% as "Strongly Disagree". The meaning in the first one is that we can improve some aspects like usability and include new elements to offer more options. In the case of the "Strongly Disagree" answers, we can see that these correspond with D11. The reason is that MOCSL allows creating a great number of applications for different devices, but it is impossible to allow modelling any imaginable application. In future extensions, we will give a special emphasis on new modelling elements to allow new extensions and possibilities for application development like the inclusion of fuzzy logic, artificial intelligence, other protocols, and other complex tasks. Nevertheless, when making these extensions, we must respect the domain of MOCSL to avoid the inclusion of functionalities with undue complexity.

Thus, taking into consideration these results, we can say that MOCSL can be very useful to facilitate object creation and that it reduces the time and the complexity of creating this type of applications. Then, MOCSL is a DSL which complements the Midgar platform and helps people without development knowledge to create objects for the Internet of Things in an easy and fast way. So, we have answer positive the research question and the contribution of this research.

Communication amongst Smart Objects through the Internet is one of the goals of the Internet of Things. Nonetheless, not all people have the necessary knowledge or time to develop the necessary application. This is why we need to bring certain facilities to the industry and daily life. This is the purpose of Midgar, exactly, with this last research, MOCSL.

VI. FUTURE WORK

The Internet of Things has many improvements to do and researches to investigate. As it was mentioned before, there are still many problems to solve. In the next points, we explain some lines, which could be solved by this investigation.

- Merging of MOISL and MOCSL: Merging MOISL and MOCSL for allowing the object creation with some internal restriction or to have only one DSL to create all the process in one-step.
- Incorporate inside the Smart Objects Artificial Intelligence using a DSL: Create a graphic DSL to allow incorporating and defining the required Artificial Intelligence by people in an easier and quicker way.
- Security and privacy in IoT: Investigating secure methods to register objects in the IoT platforms and networks to prevent the steal and modification of transferred data when an object needs to register in the platform and generate secure unique identifiers for objects.
- Scalability of IoT platforms: Studying and testing different implementations in the IoT platforms to obtain a correct way of supporting the maximum number of Smart Objects.
- The performance of Smart Objects: Researching possible improvements in an application like protocols, messages, and others, to improve the object's performance when the object processes data and sends data to save battery and resources.
- Adding a system to upload libraries: This will be very interesting because this system could allow people to create their own libraries to any sensor and actuator and share them in the platform with the rest of the world.
- Include an ontology: Including one allows us to structure the information in a better way and to make possible a new communication way which is smarter between them.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010, doi: 10.1016/j.comnet.2010.05.010.

[2] K. A. Hribernik, Z. Ghrairi, C. Hans, and K. Thoben, "Co-creating the Internet of Things—First experiences in the participatory design of intelligent products with Arduino," in *Proc. 17th Int. Conf. Concurrent Enterprising*, Aachen, Germany, Jun. 2011, pp. 1–9.

[3] L. Tan and N. Wang, "Future Internet: The Internet of Things," in *Proc. 3rd Int. Conf. Adv. Comput. Theory Eng. (ICACTE)*, Chengdu, China, vol. 5, Aug. 2010, pp. V5-376–V5-380.

[4] K. Gama, L. Touseau, and D. Donsez, "Combining heterogeneous service technologies for building an Internet of Things middleware," *Comput. Commun.*, vol. 35, no. 4, pp. 405–417, Feb. 2012, doi: 10.1016/j.comcom.2011.11.003.

[5] *La Sociedad de la Información en España 2016*, Fundación-Telefónica, Barcelona, Spain, 2016.

[6] A. Piras, D. Carboni, and A. Pintus, "A platform to collect, manage and share heterogeneous sensor data," in *Proc. 9th Int. Conf. Netw. Sens. (INSS)*, Antwerp, Belgium, Jun. 2012, pp. 1–2.

[7] T. Yamanoue, K. Oda, and K. Shimozono, "A M2M system using Arduino, Android and Wiki software," in *Proc. IIAI Int. Conf. Adv. Appl. Informat.*, Fukuoka, Japan, Sep. 2012, pp. 123–128.

[8] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–114, Aug. 2002, doi: 10.1109/MCOM.2002.1024422.

[9] G. G. Meyer, K. Främling, and J. Holmström, "Intelligent products: A survey," *Comput. Ind.*, vol. 60, pp. 137–148, Apr. 2009, doi: 10.1016/j.compind.2008.12.005.

[10] C. Perera, C. H. Liu, and S. Jayawardena, "The emerging Internet of Things marketplace from an industrial perspective: A survey," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 4, pp. 585–598, Dec. 2016, doi: 10.1109/TETC.2015.2390034.

[11] C. Perera, C. H. Liu, S. Jayawardena, and M. Chen, "A survey on Internet of Things from industrial market perspective," *IEEE Access*, vol. 2, pp. 1660–1679, 2014, doi: 10.1109/ACCESS.2015.2389854.

[12] K. Aberer, "Smart Earth: From pervasive observation to trusted information," in *Proc. Int. Conf. Mobile Data Manage.*, Mannheim, Germany, May 2007, pp. 3–7.

[13] S. Wang, "Spatial data mining under smart Earth," in *Proc. IEEE Int. Conf. Granular Comput.*, Kaohsiung, Taiwan, Nov. 2011, pp. 717–722.

[14] G. C. González and J. P. Espada, "MUSPEL: Generation of applications to interconnect heterogeneous objects using model-driven engineering," in *Handbook of Research on Innovations in Systems and Software Engineering*, V. G. Díaz, J. M. C. Lovelle, and B. C. P. García-Bustelo, Eds. Hershey, PA, USA: IGI Global, 2015, pp. 365–385.

[15] A. Martinez-Balleste, P. Perez-martinez, and A. Solanas, "The pursuit of citizens' privacy: A privacy-aware smart city is possible," *IEEE Commun. Mag.*, vol. 51, no. 6, pp. 136–141, Jun. 2013, doi: 10.1109/MCOM.2013.6525606.

[16] L. Hao, X. Lei, Z. Yan, and Y. ChunLi, "The application and implementation research of smart city in China," in *Proc. Int. Conf. Syst. Sci. Eng. (ICSSE)*, Dalian, China, Jun./Jul. 2012, pp. 288–292.

[17] M. C. Falvo, R. Lamedica, and A. Ruvio, "An environmental sustainable transport system: A trolley-buses line for Cosenza city," in *Proc. Int. Symp. Power Electron. Power Electron., Electr. Drives, Autom. Motion*, Jun. 2012, pp. 1479–1485.

[18] D. Ding, R. A. Cooper, P. F. Pasquina, and L. Fici-Pasquina, "Sensor technology for smart homes," *Maturitas*, vol. 69, no. 2, pp. 131–136, Jun. 2011, doi: 10.1016/j.maturitas.2011.03.016.

[19] M. Rothensee, "A high-fidelity simulation of the smart fridge enabling product-based services," in *Proc. 3rd IET Int. Conf. Intell. Environ. (IE)*, 2007, pp. 529–532.

[20] G. M. Lee and J. Y. Kim, "Ubiquitous networking application: Energy saving using smart objects in a home," in *Proc. Int. Conf. ICT Converg. (ICTC)*, Jeju Island, South Korea, Oct. 2012, pp. 299–300.

[21] D. Meana-Llorián, C. G. García, B. C. P. G-Bustelo, J. M. C. Lovelle, and N. Garcia-Fernandez, "IoFClime: The fuzzy logic and the Internet of Things to control indoor temperature regarding the outdoor ambient conditions," *Future Gener. Comput. Syst.*, vol. 76, pp. 275–284, Nov. 2017, doi: 10.1016/j.future.2016.11.020.

[22] C. G. García, B. C. P. G-Bustelo, J. P. Espada, and G. Cueva-Fernandez, "Midgar: Generation of heterogeneous objects interconnecting applications. A domain specific language proposal for Internet of Things scenarios," *Comput. Netw.*, vol. 64, pp. 143–158, May 2014, doi: 10.1016/j.comnet.2014.02.010.

[23] G. C. González, D. Meana-Llorián, B. C. P. G-Bustelo, and J. M. C. Lovelle, "A review about smart objects, sensors, and actuators," *Int. J. Interact. Multimedia Artif. Intell.*, vol. 4, no. 3, pp. 7–10, 2017, doi: 10.9781/ijimai.2017.431.

[24] C. G. Garcia, J. P. Espada, E. R. N. Valdez, and V. G. Diaz, "Midgar: Domain-specific language to generate smart objects for an Internet of Things platform," in *Proc. 8th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput.*, Birmingham, U.K., Jul. 2014, pp. 352–357.

[25] S. Luo, H. Xia, Y. Gao, J. S. Jin, and R. Athauda, "Smart fridges with multimedia capability for better nutrition and health," in *Proc. Int. Symp. Ubiquitous Multimedia Comput.*, Hobart, ACT, Australia, Oct. 2008, pp. 39–44.

[26] *Six Technologies With Potential Impacts on US Interests out to 2025*, U.S. Nat. Intell. Council, Washington, DC, USA, 2008.

[27] K. Ashton, "That 'Internet of Things' thing," *RFiD J.*, vol. 22, no. 7, pp. 97–114, 2009.

[28] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, "Smart objects as building blocks for the Internet of Things," *IEEE Internet Comput.*, vol. 14, no. 1, pp. 44–51, Jan. 2010, doi: 10.1109/MIC.2009.143.

[29] H. Gu and D. Wang, "A content-aware fridge based on RFID in smart home for home-healthcare," in *Proc. 11th Int. Conf. Adv. Commun. Technol.*, Phoenix Park, South Korea, Feb. 2009, pp. 987–990.

[30] C. Han, J. M. Jornet, E. Fadel, and I. F. Akyildiz, "A cross-layer communication module for the Internet of Things," *Comput. Netw.*, vol. 57, no. 3, pp. 622–633, Feb. 2013, doi: 10.1016/j.comnet.2012.10.003.

[31] Y. Sun, Y. Xia, H. Song, and R. Bie, "Internet of Things services for small towns," in *Proc. Int. Conf. Identificat., Inf. Knowl. Internet Things*, Beijing, China, Oct. 2014, pp. 92–95.

[32] A. J. Jara, Y. Sun, H. Song, R. Bie, D. Genooud, and Y. Bocchi, "Internet of Things for cultural heritage of smart cities and smart regions," in *Proc. IEEE 29th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, Gwangiu, South Korea, Mar. 2015, pp. 668–675.

[33] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, 2nd ed. London, U.K.: Pearson Higher Education, 2004.

[34] S. Kent, "Model driven engineering," in *Proc. Int. Conf. Integr. Formal Methods*, vol. 2335, 2002, pp. 286–298.

[35] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*. Reading, MA, USA: Addison-Wesley, 2003.

[36] R. C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Reading, MA, USA: Addison-Wesley, 2009.

[37] M. Fowler, *Domain Specific Languages*. Reading, MA, USA: Addison-Wesley, 2010.

[38] M. Eysholdt and H. Behrens, "Xtext: Implement your language faster than the quick and dirty way," in *Proc. ACM Int. Conf. Companion Object Oriented Program. Syst. Lang. Appl. Companion (SPLASH)*, New York, NY, USA, 2010, pp. 307–309.

[39] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, Dec. 2005, doi: 10.1145/1118890.1118892.

[40] G. C. González, J. P. Espada, B. C. P. G-Bustelo, and J. M. C. Lovelle, "Swift vs. Objective-C: A new programming language," *Int. J. Interact. Multimed. Artif. Intell.*, vol. 3, no. 3, pp. 74–81, 2015, doi: 10.9781/ijimai.2015.3310.

[41] T. Stahl, M. Voelter, and K. Czarnecki, *Model-Driven Software Development: Technology, Engineering, Management*. Hoboken, NJ, USA: Wiley, 2006.

[42] *MDA Guide Rev. 2.0.*, Object Management Group Inc., Needham, MA, USA, 2014.

[43] E. Biermann, K. Ehrig, C. Köhler, G. Kuhns, G. Taentzer, and E. Weiss, "Graphical definition of in-place transformations in the Eclipse modeling framework," in *Model Driven Engineering Languages and Systems* (Lecture Notes in Computer Science), vol. 4199, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, Eds. Berlin, Germany: Springer, 2006, pp. 425–439.

[44] Eclipse. (2010). *Ecore*. Accessed: Nov. 1, 2019. [Online]. Available: http://wiki.eclipse.org/Ecore

[45] LogMeIn. (2013). *Xively*. Accessed: Nov. 1, 2019. [Online]. Available: https://xively.com/

[46] Exosite. (2013). *Exchange IoT Marketplace*. Accessed: Nov. 1, 2019. [Online]. Available: http://exosite.com/

[47] Parker Hannifin Corp. (2020). *Sensor Cloud*. Accessed: Nov. 2, 2019. [Online]. Available: http://www.sensorcloud.com/

[48] Digi International Inc. (2008). *Etherios & Google Project, The Data Sensing Lab, Wins Postscapes Internet of Things (IoT) Award*. Accessed: Nov. 2, 2019. [Online]. Available: https://www.digi.com/news/press-releases/905

[49] PTC. (2020). *ThingWorx*. Accessed: Nov. 2, 2019. [Online]. Available: http://www.thingworx.com/

[50] I. Altair Engineering. (2020). *Carriots*. Accessed: Nov. 2, 2019. [Online]. Available: https://www.carriots.com/

[51] Microsoft. (2015). *Azure IoT Suit*. Accessed: Nov. 3, 2019. [Online]. Available: https://azure.microsoft.com/en-us/free/iot/

[52] Amazon Web Services. (2020). *AWS IoT*. Accessed: Nov. 3, 2019. [Online]. Available: https://aws.amazon.com/es/iot-core/features/

[53] IBM. (2015). *IBM Internet of Things*. Accessed: Nov. 3, 2019. [Online]. Available: https://www.ibm.com/es-es/internet-of-things

[54] G. Sánchez-Arias, C. G. García, and B. C. P. G-Bustelo, "Midgar: Study of communications security among smart objects using a platform of heterogeneous devices for the Internet of Things," *Future Gener. Comput. Syst.*, vol. 74, pp. 444–466, Sep. 2017, doi: 10.1016/j.future.2017.01.033.

[55] C. G. García, D. Meana-Llorián, B. C. P. G-Bustelo, J. M. C. Lovelle, and N. Garcia-Fernandez, "Midgar: Detection of people through computer vision in the Internet of Things scenarios to improve the security in smart cities, smart towns, and smart homes," *Future Gener. Comput. Syst.*, vol. 76, pp. 301–313, Nov. 2017, doi: 10.1016/j.future.2016.12.033.

[56] Paraimpu. [Online]. Available: http://www.paraimpu.com/

[57] A. Pintus, D. Carboni, and A. Piras, "Paraimpu: A platform for a social Web of things," in *Proc. 21st Int. Conf. Companion World Wide Web (WWW Companion)*. New York, NY, USA: ACM, 2012, pp. 401–404.

[58] QuadraSpace. (2010). *Open Sensor Network Services*. Accessed: Aug. 7, 2017. [Online]. Available: http://www.quadraspace.org/

[59] Microsoft. (2020). *SenseWeb*. Accessed: Mar. 3, 2020. [Online]. Available: http://research.microsoft.com/en-us/projects/senseweb/

[60] A. Kansal, S. Nath, J. Liu, and F. Zhao, "SenseWeb: An infrastructure for shared sensing," *IEEE Multimedia Mag.*, vol. 14, no. 4, pp. 8–13, Oct. 2007, doi: 10.1109/MMUL.2007.82.

[61] D. Guinard and V. Trifa, "Towards the Web of things: Web mashups for embedded devices," in *Proc. 2nd Workshop Mashups, Enterprise Mashups Lightweight Composition Web*, Madrid, Spain, 2009, p. 8.

[62] Department of Electrical and Electronic Engineering (University of Cagliari). (2012). *SIoT*. Accessed: Feb. 5, 2018. [Online]. Available: http://platform.social-iot.org/

[63] Oak Ridge National Laboratory. (2009). *Sensorpedia*. Accessed: Feb. 5, 2018. [Online]. Available: http://www.sensorpedia.com/

[64] B. L. Gorman, D. R. Resseguie, and C. Tomkins-Tinch, "Sensorpedia: Information sharing across incompatible sensor systems," in *Proc. Int. Symp. Collaborative Technol. Syst.*, May 2009, pp. 448–454, doi: 10.1109/CTS.2009.5067513.

[65] Evrythng. (2012). *The EVRYTHNG Product CloudTM for Your Industry*. Accessed: Mar. 7, 2018. [Online]. Available: https://www.evrythng.com/

[66] OpenSense. (2015). *A Participatory Open Sensor Data Platform*. Accessed: Mar. 7, 2018. [Online]. Available: https://opensense.network/

[67] The MathWorks Inc. (2020). *ThingSpeak for IoT Projects*. Accessed: Mar. 3, 2020. [Online]. Available: https://www.mathworks.com/products/thingspeak.html

[68] Nimbits Inc. (2015). *Process Control and Automation and Evolved Into An IoT API*. Accessed: Jan. 20, 2019. [Online]. Available: https://github.com/bsautner/com.nimbits

[69] KaaIoT Technologies LLC. (2014). *KAA*. Accessed: Feb. 2, 2018. [Online]. Available: http://www.kaaproject.org

[70] B. Kaucic and T. Asic, "Improving introductory programming with Scratch?" in *Proc. 34th Int. Conv. MIPRO*, Opatija, Croatia, May 2011, pp. 1095–1100.

[71] Besttoolbars. (2020). *Free and Simple Tool to Create, Download, Distribute and Monetize Your App*. Accessed: Mar. 3, 2020. [Online]. Available: http://www.appsgeyser.com

[72] iBuildApp Inc. (2020). *Crear Aplicaciones Para Android y iPhone*. Accessed: Mar. 3, 2020. [Online]. Available: http://ibuildapp.com/

[73] Andromo. (2016). *Create the App You Want*. Accessed: Mar. 7, 2020. [Online]. Available: https://www.andromo.com/

[74] Paperlit S.R.L. (2015). *AppsBuilder*. Accessed: Mar. 7, 2020. [Online]. Available: http://www.apps-builder.com/

[75] Infinite Monkeys. (2016). *App Builder to Make Your App Without Coding*. Accessed: Feb. 4, 2018. [Online]. Available: http://www.infinitemonkeys.mobi

[76] Appy Pie. (2013). *No-Code Platform for Building Digital Products*. Accessed: Mar. 7, 2020. [Online]. Available: https://www.appypie.com/

[77] Swiftic. (2020). *Create a Mobile App in 3 Easy Steps*. Accessed: Feb. 1, 2018. [Online]. Available: http://my.como.com

[78] AppMachine. (2020). *Create Your Own App Or Become a Reseller and Build Apps for Others*. Accessed: Feb. 1, 2018. [Online]. Available: https://www.appmachine.com/

[79] MiniBloq. (2016). *miniBloq is an Open Source Graphical Programming Environment for Multiplo, Arduino, Physical Computing Devices and Robots*. Accessed: Oct. 27, 2016. [Online]. Available: http://blog.minibloq.org/

[80] Arduino. (2020). *Arduino/Processing Language Comparison*. Accessed: Feb. 1, 2020. [Online]. Available: https://www.arduino.cc/en/Reference/Comparison

[81] Ardublock. (2020). *A Graphical Programming Language for Arduino*. Accessed: Feb. 1, 2020. [Online]. Available: http://blog.ardublock.com/

[82] Citilab. (2020). *EduLab—Scratch for Arduino*. Accessed: Feb. 1, 2018. [Online]. Available: https://www.citilab.eu/

[83] Citilab. (2020). *S4A*. Accessed: Feb. 1, 2020. [Online]. Available: http://s4a.cat/

[84] Modkit LLC. (2020). *Program Your World: Drag & Drop Programming You Can Touch*. Accessed: Nov. 4, 2019. [Online]. Available: http://modkit.com/

[85] Microchip Technology Inc. (2020). *Atmel Studio 7*. Accessed: Feb. 1, 2020. [Online]. Available: https://www.microchip.com/mplab/avr-support/atmel-studio-7

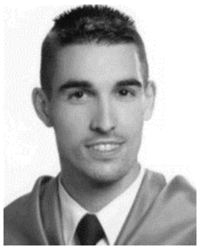
[86] XOD Inc. (2017). *A Visual Programming Language for Microcontrollers*. Accessed: Nov. 4, 2019. [Online]. Available: https://xod.io/

[87] J. S. A. S. M. Bauer, M. Boussard, N. Bui, F. Carrez, C. Jardak, J. Loof, C. Magerkurth, S. Meissner, A. Nettsträter, A. Olivereau, M. Thoma, and J. Walewski, "Deliverable D1.5—Final architectural reference model for the IoT v3.0," Eur. Union, Brussels, Belgium, Tech. Rep., 2013.

[88] R. Likert, "A technique for the measurement of attitudes," *Arch. Psychol.*, vol. 22, no. 40, pp. 5–55, Jun. 1932.

[89] Blacksun Software. (2016). *Mousotron?: Mouse and Keyboard Activity Monitor*. Accessed: Nov. 9, 2019. [Online]. Available: http://www.blacksunsoftware.com/mousotron.html

[90] M. Kasunic, "Designing an effective survey," Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., 2005.

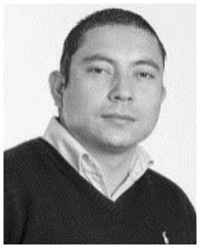


CRISTIAN GONZÁLEZ GARCÍA received the M.Sc. degree in Web engineering and the Ph.D. degree in computers science. He is currently an Assistant Professor with the Department of Computer Science, University of Oviedo, Spain. He is also a Technical Engineer in computer systems. He has been a visiting Ph.D. student with the University of Manchester. Besides, he has been with the University of South Florida, as a Visiting Professor. His research interests include the Internet of Things, Web engineering, mobile devices, artificial intelligence, big data, and modeling software with DSL and MDE.

DANIEL MEANA-LLORIÁN received the Engineering degree in computer systems and the M.S. degree in Web engineering from the School of Computer Engineering, University of Oviedo, Oviedo, Spain, in 2014 and 2016, respectively. He is currently pursuing the Ph.D. degree in computer science.

VICENTE GARCÍA-DÍAZ received the Ph.D. degree in computer engineering from the University of Oviedo. He is currently an Associate Professor with the Computer Science Department, University of Oviedo. His research interests include domain-specific languages, model-driven engineering, business process management, machine learning, the Internet of Things, and eLearning.

ANDRÉS CAMILO JIMÉNEZ received the Ph.D. degree in engineering from the University Francisco Jose de Caldas. He is currently an Associate Professor with the Electronics and Systems Department, University Foundation Los Libertadores. His research areas are related to image processing, digital systems, and cooperative robotic.



JOHN PETEARSON ANZOLA received the M.Sc. degree in information sciences and communications from the University Francisco Jose de Caldas. He is currently an Associate Professor with the Electronics and Systems Department, University Foundation Los Libertadores. His research interests include ad-hoc networks, image processing, machine learning, and the Internet of Things.