ABSTRACT Systems based on the Internet of Things (IoT) are continuously growing in many areas such as smart cities, home environments, buildings, agriculture, industry, etc. This system integrates heterogeneous technologies into a complex architecture of interconnected devices capable of communicating, processing, analysing or storing data. There are several IoT platforms that offer several capabilities for the development of these systems. Some of these platforms are Google Cloud’s IoT Platform, Microsoft Azure IoT suite, ThingSpeak IoT Platform, Thingworx 8 IoT Platform or FIWARE. However, they are complex IoT platforms where each IoT solution has to be developed ad-hoc and implemented by developers by hand. Consequently, developing IoT solutions is a hard, error-prone and tedious task. Thus, increase the abstraction level from which the IoT systems are designed helps to tackle the underlying technology complexity. In this sense, model-driven development approaches can help to both reduce the IoT application time to market and tackle the technological complexity to develop IoT applications. In this paper, we propose a Domain-Specific Language based on SimulateIoT for the design, code generation and simulation of IoT systems which could be deployed on FIWARE infrastructure (an open-source IoT platform). This implies not only designing the IoT system for a high abstraction level and later on code generation, but also designing and deploying an additional simulation layer to simulate the system on the FIWARE infrastructure before final deployment. The FIWARE IoT environment generated includes the sensors, actuators, fog nodes, cloud nodes and analytical characteristics, which are deployed as microservices on Docker containers and composed suitability to obtain a service-oriented architecture. Finally, two case studies focused on a smart building and an agricultural IoT environment are presented to show the IoT solutions deployed using FIWARE.

INDEX TERMS Model-driven development, Internet of Things, IoT simulation, services-oriented, FIWARE.

I. INTRODUCTION

The Internet of Things (IoT) is widely applied in several areas such as smart cities, home environments, agriculture, industry, intelligent buildings, etc. [45]. In order to build IoT applications, multiple technologies are available from configuring a specific sensor to analysing a vast amount of data in real-time. Thus, data should be, among other actions, stored, communicated, analysed, visualised and notified. For this, multiple IoT cloud platforms have emerged for development such as Google Cloud’s IoT Platform [17], Microsoft Azure IoT suite [23], ThingSpeak IoT Platform [47], Thingworx 8 IoT Platform [48] or FIWARE [13].

Each IoT platform has its own characteristics and mechanisms to define devices, connect them, store and analyse data or carry out notifications that provoke the well-known (vendor lock-in problem [36]). Likewise, each IoT
platform offers different services and QoS which should be managed ad-hoc.

However, it is possible to define IoT solutions independent of the IoT platforms on which they will be deployed. For them, it is necessary to focus on the IoT application domain and not on their specific technological issues. Model-Driven Development (MDD) [43, 51] is able to tackle this heterogeneous technology (vendor lock-in problem) increasing the abstraction level where the software is developed, focusing on the domain concepts and their relationships. Thus, the IoT concepts and relationships are defined by a model which can be analysed and validated.

In this sense SimulateIoT [3] is an approach based on Model-Driven Development (MDD) to define IoT environments (Set of components, such as sensors, actuators, Fog or Cloud nodes, etc. that are part of an IoT architecture), generate its code and deploy it. Later on, the IoT environment generated could be simulated. This is because the MDD allows 1) the definition of a Domain Specific Language (DSL), able to model IoT environments, and 2) a model-to-text (M2T) transformations needed to code-generation and deploy the IoT environment.

However, SimulateIoT is limited to code generation towards proprietary infrastructure based on microservices. In order to show that an MDD approach is able to generate code on different technological platforms, it is interesting extending the code generation to other technological approaches such as cloud open-source IoT environments such as FIWARE [13]. It is an open-source project that provides a large catalogue of components for the development of IoT environments, including, among other functions and components to manage, analyse or store the data which are generated and shared in an IoT environment [13]. This paper presents the extension of the SimulateIoT MDD platform to deploy the IoT environments modelled in the FIWARE open-source IoT Platform. Consequently, it shows that is possible to model IoT environments independently of technology and deploy them on concrete IoT cloud platforms such as FIWARE. Thus, the IoT concepts and relationships are defined by a model which can be analysed and validated. Besides, the IoT environment code, including all the artefacts needed, can be generated from a model using model to text transformations, decreasing error-proneness and increasing the user’s productivity.

The main contributions of this paper include:

- A proposal that shows Model-Driven Development is a suitable approach to develop tools and languages to tackle the complexity of heterogeneous technology successfully in the context of IoT environments such as sensors, actuator, databases, complex-event processing engines, communication protocols, etc.

- A Model-Driven Development proposal to generate IoT solutions based on FIWARE infrastructure, hiding the complexity of a cloud IoT framework.

- An extended version of SimulateIoT Domain Specific Language named SimulateIoT-FIWARE that can be used to define IoT environments and generate their implementation based on the components provided by FIWARE. That means reusing both SimulateIoT Abstract Syntax (Metamodel and OCL constraints) and SimulateIoT Concrete Syntax, while the M2T transformations have been improved and adapted to generate, configure and deploy FIWARE artefacts.

- Two case studies have been developed following the methodology and tools presented, focusing on different kinds of IoT systems. Note that, these two use cases are the same as those defined in [3], demonstrating that it is possible to deploy these same environments on the FIWARE platform.

The rest of the paper is structured as follows. Section 2 introduces the FIWARE architecture and how IoT systems should be implemented on it. Section 3 describes shortly SimulateIoT Domain Specific Language. Section 4 presents the integration of SimulateIoT DSL with the FIWARE architecture and artefacts. Section 5 describes the aspects related to code generation towards FIWARE technology from models. Then, Section 6 illustrates the use of the Model-Driven approach presented in two different case studies: Smart Building and Smart Agro. In Section 7 the discussion and limitations of the approach are described, before presenting the related works in Section 8 and the conclusions in Section 9.

II. THE FIWARE ARCHITECTURE

FIWARE is an open-source project that defines and implements a universal set of standards for context data management with the aim of optimising the development of IoT environments in different fields, such as Smart Cities [16], Smart Buildings [15], Smart Agro [25], Smart Energy, Smart Industry [13], etc. FIWARE makes IoT simpler by means of driving key standards for breaking the information silos, transforming Big Data into knowledge, enabling data economy and ensuring sovereignty on your data [13]. Consequently, using FIWARE to design, develop and manage an IoT environment makes it possible reuse the advantages aforementioned and to reuse the knowledge and tools developed as part of FIWARE. In this sense, although FIWARE has several components to support the developing of IoT environments in the scenarios aforementioned, the main and only mandatory component of any FIWARE solution is FIWARE Context Broker. Mention that the concept of context within FIWARE, is the state in which the IoT environment is at a given time. Thus, the context elements or data are those that give context to the environment, i.e., they define a characteristic of the environment, such as climatic data of the environment as temperature or wind speed and also data of the architecture of the environment, such as geoposition of an element or the speed at which it moves.

The FIWARE implementation is based on a set of layers and integrated elements such as i) Interface to IoT, Robotics and third party systems; ii) Core Context Management, iii) Context Processing, Analysis and Visualisation and iv) Data/API management and Publication and Monetisation of Context Information. Each layer is supported by several
tools such as Context Broker, Complex Event Processing, IoT Broker or IoT Backend Device Manager that constitute the architecture that can be seen in Figure 1. These elements communicate among themselves through the NGSI protocol [14], although FIWARE has a middleware that acts as a bridge between NGSI and protocols such as MQTT or HTTP to communicate with external elements. Next, the most important elements of the FIWARE architecture (Figure 1) are defined:

- The Context Broker element named Orion (Figure 1-1) is the core of the FIWARE architecture. Orion allows the management of the complete data lifecycle including updates, queries, registrations and subscriptions. In other words, Orion allows creation and registration of the context elements, such as sensors, actuators, CEP engines, etc. and manages them through updates and queries. In addition, devices can subscribe to context information so when some condition occurs these devices receive a notification [10]. Moreover, it should be mentioned that Orion Context Broker includes MongoDB [30], a NoSql database [22] for the data persistence required to perform the above-mentioned functions as well as those of other FIWARE elements.

- A CEP (Complex Event Processing) (Figure 1-2) [8] element allows more complex analysis techniques than Orion Context Broker subscriptions. Thus, in order to develop CEP applications in FIWARE, a CEP component named CEP Perseo [12] has been included in the FIWARE architecture. CEP Perseo is a CEP software based on the Esper language [9], i.e., software that listens for events that come from context information to identify event patterns described by rules, in order to immediately react to them by triggering actions [12]. CEP Perseo is composed of two basic elements, Perseo front-end and Perseo core. Perseo front-end stores the event rules (written using Event Processing Language (EPL) [37]) on MongoDB and then, processes the incoming events sending them to Perseo Core. Next, Perseo Core checks incoming events against the event rules and notifies Perseo front-end if an action must be executed [11]. Finally, Perseo front-end sends notifications to the appropriate devices.

- The IoT Broker (Figure 1-3) element allows developers to use a message broker such as Mosquitto [31] (based on MQTT protocol) to ensure message exchange among the devices or components defined in an IoT environment. It implements a publish/subscribe communication protocol that makes it possible to interconnect the IoT devices and components such as Sensors, Actuators or other FIWARE components.

- The IoT Backend Device Management (Figure 1-4) consumes data from Sensors (Figure 1-5) and sends it to the Actuators (Figure 1-6). IoT-Agent carries out this task. Thus, IoT-Agent acts as a bridge between the NGSI protocol and other protocols such as MQTT or HTTP. In this way, the IoT-Agent brings a standard interface to all IoT interactions at the context information management level (Orion Context Broker) allowing each IoT device to be able to use its own protocols to communicate with FIWARE.

The elements described above are the main components of the FIWARE platform which are enough to develop an IoT environment on the FIWARE platform. However, as can be seen in Figure 1, FIWARE offers a larger number of components. These components aim to meet specific needs such as service orchestration, Big Data processing, payment management, etc.

For instance, supposing a general Smart Building case study where several sensors are deployed in the building, these sensors send data to a FIWARE instance, which are analysed in real-time by a CEP component in order to notify different event rules detected. Additionally, data processed is stored for later analysis. The system architecture deployed to support this IoT environment can be observed in Figure 2.

Developing this case study includes, among others, the following:

- Define the sensors and actuators into Orion context broker.
- Define and configure the messages that should be interchanged from/to devices to FIWARE architecture.
- Configure and deploy each node for the FIWARE infrastructure, including Orion context broker, CEP perseo, databases, messages brokers, and so on.
- Define the EPL rules and deploy them on the CEP Perseo.
- etc.

Additional issues should be taken into account and they should be resolved by implementing additional ad-hoc modules:

- Components such as Perseo notifies event matched by HTTP protocol. Consequently, in order to notify Actuators who are subscribed to a specific Topic, an HTTP2MQTT converter should be developed. In Figure 2 this module is named NotificationMiddlewareComponent.
- Originally, event patterns analysis can’t be defined on Topic data. So, an additional infrastructure based on Topic data should be registered on Orion. In Figure 2 is named OrionTopicManager.

Developing an IoT environment by hand involves tedious and error-prone tasks. So, the complexity of the whole process defined previously to implement and deploy the IoT environment using heterogeneous technology should be tackled by increasing the abstraction level of the defined IoT environment. Therefore, the SimulateIoT model-driven approach is extended and used to model and generate IoT environments on FIWARE platform.

### III. SimulateIoT: A MODEL-DRIVEN APPROACH TO DEVELOPING IoT SIMULATION ENVIRONMENTS

In a Model-Driven Development approach like SimulateIoT, the software development is guided through Models (M1)
which conform to a MetaModel (M2). Moreover, a Metamodel conforms to a MetaMetaModel (M3) which is reflexive [2]. The MetaMetaModel level is represented by well-known standards and specifications such as Meta-Object Facilities (MOF) [29], ECore in EMF [46] and so on. A MetaModel defines the concepts and relationships in a specific domain in order to model partial reality based (conceptual model). Additionally, OCL is formal language used to describe semantic expressions on Metamodels such as UML. These expressions typically specify invariant conditions that must hold for the system being modeled [35]. So, Model conforms to a MetaModel requires to validate with this semantic extensions (OCL invariants). Later on, the validated models are used to generate totally or partially the application code by model-to-text transformations [44]. Thus, the software code can be generated for a specific
technological platform, improving the technological independence and decreasing error proneness.

SimulateIoT is a tool that uses model-driven development techniques to manage the IoT environments definition using models, so, the models guide the system description and the code generation. Subsequently, the code generated can be deployed through several hosts or be used to deploy a simulation of the IoT environment. There are three main elements of SimulateIoT tools: 1) a Metamodel definition, 2) a Graphical Concrete Syntax definition and 3) the M2T transformations to generate the code artefacts needed to deploy, monitor and measure the IoT environment. In order to be a self-contained paper, next the SimulateIoT proposal is explained.

Figure 3 defines the domain metamodel including concepts related to sensors, actuators, databases, fog and cloud nodes, data generation, communication protocols, stream processing, and deployment strategies, among others. The relevant elements are summarised below:

- The **Environment** element defines the global parameters of the IoT simulation environment, including simulation speed and the number of messages to interchange among the nodes (numberOfMessages).
- **Node** is an abstract concept to represent each node in the IoT simulation environment. It is extended by several concepts such as **EdgeNode** or **ProcessNode** in order to specialise each kind of node. A **Node** can publish and subscribe to a specific **Topic**. It defines publish or subscribe references towards a **Topic** element in which it is interested. Note that, later on, each concrete kind of **Node** could be defined with specific constraints. Furthermore, the device position can be defined using **latitude** and **longitude** attributes.
- The **EdgeNode** element makes it possible to define simple physical devices such as a sensor or an actuator without process capacities. Each **EdgeNode** could be linked with **ProcessNode** elements by **Topic** elements. Moreover, each **EdgeNode** can be mapped with a physical device such as a temperature sensor, a humidity sensor, a turn on/off light device or an irrigation water flow device in the IoT environment. Additionally, the **CoverageSignalGain** attribute allows users to define the coverage reception capacity (offered by the different **ProcessNodes**) of the device.
- A **Sensor** element extends the **EdgeNode** element and defines a set of characteristics such as id or generation_speed. A **Sensor** element analyses a specific environment issue (temperature, humidity, people presence, people counter, etc.) and sends these data to be analysed later. A **Sensor** element is able to publish on **Topic** elements which propagate data throughout the simulation nodes.
- An **Actuator** element is a device in the IoT environment that can execute an action from a set of inputs. For instance, the inputs could determine that an actuator turns a light on or off; other Actuators could require data input to define the light’s luminosity. In order to receive data, an **Actuator** element should be subscribed to **topics**.
- **Topic** is a central element in this metamodel because it defines the information transmitted among any kind of **Node** elements. Thus, **Topic** elements are defined from **CloudNode** and **FogNode** elements and help users to model a publish-subscribe communication model. Obviously, the **Topic** element is a flexible concept to manage the data interchange.
- **Data** element defines the simple data type to be generated (Boolean, short, integer, real, string). It has a **DataSource** element to model either the **DataGeneration** element or **LoadFromFile** element. The former (Data-Generation element) models how synthetic data are generated, for instance, using an aleatory strategy among two values defined in a **GenerationRange** element. The latter (LoadFromFile element) models the path-file that contains the historic data, for instance, it could be defined by a CSVLoad element. In addition, external tools such as [1], [19] can be linked to increase the capabilities to offer additional data generation patterns.
- The **ProcessNode** element defines an IoT node with process capability. For this, two subtype nodes could be defined: **CloudNode** and **FogNode**. Essentially, both have the same properties and only differ in their process capability. Thus, in order to classify the **ProcessNode** capacity (the **size** attribute) related to batteries, CPU, memories, etc. a set of granularity values have been defined (XS, S, L, XL and XXL) They make it possible to define different kinds of nodes. This strategy allows specifying the **ProcessNode** element capacity and associating specific constraints, for example, in an XS **ProcessNode** a ProcessesEngine such as a Complex Event Processing (CEP) engine cannot be deployed. Hence, granularity labels are used as in a Scrum project development [42] to define task complexity. As mentioned, **ProcessNode** can define **Topic** elements, with which can be referenced by any kind of **Node** elements. Besides, the redirectionTime attribute defines the frequency that stored data are flushed towards the next **ProcessNode** element defined by redirect references. The attribute **BrokerType** defines the message-oriented broker that currently is established by **Mosquitto**. In addition, the **ProcessNode** element hides the complexity of how data should be gathered and processed. For instance, it defines how data will be stored, published or offered to be analysed by stream processing engines (SP) or complex event processing engines (CEP) by defining **Component** elements. Note that either the stream processing or the complex event processing capabilities help to define when an **Actuator** element should carry out an action. Finally, the **CoverageSignalPower** attribute is used to establish the range of coverage offered by a **ProcessNode** for those mobile devices that want to connect to it.
FIGURE 3. SimulateIoT metamodel.
• FogNode allows users to describe fog computing instances [5] which could manage and coordinate several devices or Actuators. Thus, this concept focuses on aggregating data for a limited time or connection conditions, that are released later on. Furthermore, a FogNode element can include persistent data storage and data processing.

• CloudNode extends ProcessNode and allows describing a special node deployed in a public or private cloud computing environment.

• The ProcessEngine element should be linked to a ProcessNode, to allow real-time data analysis defining coming from ProcessNode elements or EdgeNode elements. To do this, defining complex event patterns can be carried out by Rule elements. These patterns analyse Topic data in real-time. Usually, a CEP (Complex Event Processing) engine has a higher process capacity and lower latency than an ESP (Event Stream Processing) engine [4], [26].

• Rule elements are linked with the ProcessEngine elements defined at the ProcessNode element. Rule elements can be defined using the EPL language defined for a concrete ProcessEngine kind.

• Notification elements make it possible to throw alerts by using several notification kinds: TopicNotification or eMailNotification. Obviously, the notification hierarchy could be extended in further metamodel versions.

• Route element allows to define the route throughout the coordinates by which the mobile device must move. For this purpose, 3 different methods have been included for their generation 1) Fog/Cloud Route, 2) Linear Route, 3) Random Route. Fog/Cloud Route allows the user to establish a route from the selection of several Fog/Cloud nodes so that the mobile device will move sequentially among the selected nodes. Linear Route allows the user to define 2 coordinates, in this way the mobile device will move in a linear way between these coordinates. Finally, Random Route generates a random route at run time.

Later on, the SimulateIoT models can be created by using a Graphical Concrete Syntax (Graphical editor) defined by using Eugenia [24] from the SimulateIoT metamodel. Figure 4 shows an excerpt from this graphical editor. It helps users to improve their productivity allowing not only defining models conforming to SimulateIoT metamodel but also their validation using OCL constraints [35].

• Sensors. Sensors have the same components in SimulateIoT and in the SimulateIoT-FIWARE, however, in SimulateIoT-FIWARE Sensors publish their data twice. On the one hand, one of the publications is directly addressed to Orion Context Broker to enable it to record the information published by each Sensor separately, so that the data published by each Sensor can be consulted independently. On the other hand, the other publication is addressed to the OrionTopicManager component which allows Orion Context Broker to record the data published in a certain Topic. In this way,
all the data published in a specified Topic can be consulted independently from the Sensor that published it. In this way, for instance, the CEP component can apply rules by Topic and it does not need to collect and consult the data that each sensor has published in the Topic where it is going to apply the rules. As for the topology of Topics concerning the receipt of publications by each sensor independently, it is as follows: “/token/SensorName+SensorId/attrs” for instance, when token is 1234 and SensorName and SensorId are parameters predefined in a Sensor element within the IoT Environment model, we get “1234/temperaturemeter5/attrs.”

- **NotificationMiddleware.** Since CEP Perseo only sends its notifications by the HTTP protocol, a middleware is necessary to act as a bridge between the HTTP protocol and the MQTT protocol as this is used by the Actuator elements to receive data or to receive notifications in this instance. NotificationMiddleware performs the above actions.

- **OrionTopicManager.** In SimulateIoT, the rules can be applied by Topic. However, due to the internal operation of Orion Context Broker and CEP Perseo, applying rules by Topic becomes more complex. To cope with this complexity, an additional module named OrionTopicManager has been developed, enabling CEP Perseo to apply its rules by Topic as in SimulateIoT. It allows reusing the SimulateIoT metamodel and related tools.

The Sensors logic, NotificationMiddleware component and OrionTopicManager component will be generated from the M2T transformation.

B. KNOWING THE INTERACTIONS OF THE INTERNAL COMPONENTS IN ORDER TO INTEGRATE FIWARE ARTEFACTS

In order to deploy the IoT environments of SimulateIoT on FIWARE, it is necessary to define the relationship and interaction between components. Thus, this section describes these relationships or interactions that allow the deployment of IoT environments on FIWARE.

- **IoTAgent-Json and Mosquitto Broker.** IoTAgent-Json needs to receive the messages published in the Mosquito’s Topics from sensors to send them to Orion Context Broker.

- **Orion Context Broker and IoTAgent-Json.** Orion Context Broker can receive in NGSI protocol the messages published by the Sensors because IoTAgent-Json is able to act as a bridge between Sensors and the Orion Context Broker.

- **Orion Context Broker and MongoDB.** Firstly, MongoDB is a functional dependency of Orion Context Broker. Thus, Orion Context Broker needs to interact with MongoDB to manage all the context data and the devices. Figure 5 shows a sequence diagram illustrating the interactions described up to this point which includes IoT-Agent, Sensors, Mosquito, Orion Context Broker and MongoDB.

- **Orion Context Broker and CEP Perseo.** CEP Perseo needs to interact with Orion Context Broker. Specifically, CEP Perseo subscribes to Orion Context Broker data and it is able to apply event pattern analysis to them.

- **Perseo Core and Perseo-FrontEnd.** CEP Perseo is composed of two components which need to interact...
with each other. Basically, Perseo Core is the CEP engine which applies rules to the data and notifies Perseo-FrontEnd. Perseo-FrontEnd is the component that gets the data from Orion Context Broker and sends them to Perseo Core.

**CEP Perseo and MiddlewareNotificationComponent.** CEP Perseo only sends its notifications through the HTTP protocol, however, in the SimulateIoT code generation, the Actuators can only receive it through the MQTT protocol. MiddlewareNotificationComponent is an additional component developed that listens to CEP Perseo notifications and redirects them to Actuators through MQTT. Figure 6 shows a sequence diagram that illustrates the interactions described up to this point. Note that in Figure 6 data start from Orion to Perseo Front-end. The elements involved in this sequence messages includes PerseoFrontEnd, Orion, PerseoCore, MiddlewareNotificationComponent, Mosquito and Actuators elements.

**OrionTopicManager and Orion Context Broker.** In order to ensure the application of rules based on Topics carried out by CEP Perseo, OrionTopicManager resends all the messages of a concrete Topic to Orion Context Broker. Figure 7 shows a sequence diagram illustrating the interactions described up to this point. The elements involved in this sequence messages includes Orion Topic Manager, Sensors, Mosquito and Orion elements.

At this point, the steps needed to integrate FIWARE components and SimulateIoT artefacts in order to be successfully deployed has been explained. Next, the M2T transformation and the specific IoT environment deployment characteristics are described.

### V. IoT ENVIRONMENT CODE GENERATION AND DEPLOYMENT

This section describes the main characteristics of the $M2T$ transformation and deployment phase based on the analyses about how to integrate FIWARE components and artefacts defined on a SimulateIoT model (Section IV-B).

#### A. MODEL-TO-TEXT TRANSFORMATION

Once the models have been defined and validated, an $M2T$ transformation is able to generate the IoT environments that have been modelled for a specific technology. Thus, the generated software includes FIWARE components such as

| Metamodel Element | SimulateIoT Components | Description | FIWARE Components | Description |
|-------------------|-------------------------|-------------|-------------------|-------------|
| ProcessNode       | Mosquito                | MQTT Broker | Mosquito          | MQTT Broker |
|                   | MQTT Client             | Publish/Subscribe on topics | MQTT client | Publish/Subscribe on topics |
|                   | (internal component)    |             | Orion Context Broker | Device and Context data management |
|                   |                         |             | IoTAgent-Json     | Bridge between MQTT-Json and NGSI |

| Component Database | MongoDB client          | MongoDB Manager | Orion Context Broker | Orion has a client to interact with MongoDB |
|                   |                         |                | NoSql Database      | A dependency of Orion Context Broker. Due to the above fact, the ComponentDatabase is no longer an optional component |

| Component Process Engine | CEP Engine | Apply rules to topics | Perseo | Perseo-FrontEnd: Subscribe to Orion context data and Publish notifications (HTTP), Perseo-Core: Apply rules to Orion context data |
|--------------------------|------------|-----------------------|--------|--------------------------|
|                          | MQTT client | Subscribe on topics, Publish notifications on topics |        |                          |

#### TABLE 1. Relationships among the main metamodel elements with the main target components.
Orion Context Broker, CEP Perseo or IoTAgent, and others components can also be generated like an MQTT messaging broker, device infrastructure, databases, a graphical analysis platform, docker container, a REST API etc. These FIWARE components can be deployed as a part of Node elements defined on a SimulateIoT model. In this regard, Table 2 summarises for each Node type the components that can be generated and deployed including NoSQL database, Database Client, REST API, MQTT Broker, MQTT Client, Orion Context Broker, CEP Perseo, IoTAgent, NotificationMiddleware or OrionTopicManager. Besides, hardware requirements are included with each component. These hardware requirements indicate the minimum hardware power needed to deploy each component of Cloud, Fog or Edge node. The hardware power is represented with the following labels: XS, S, M, L, XL, where XS represents the lowest hardware requirements, for instance, a Raspberry Pi, and XL represents the highest hardware requirements, for instance, a cloud infrastructure.

In addition to these components, the M2T transformation also generates all the configuration files required to deploy all the artefacts successfully. These configuration files include:

- The registration in Orion for each device. An excerpt of the device registry file configuration in Orion Context Broker can be seen in Appendix B.
- The specification of CEP Perseo’s rules. A fragment of the file to configure CEP Perseo can be seen in Appendix C.
- The connection of each component with the others, which is managed with a docker-compose file. An example of the docker-compose file is illustrated in Appendix A.
- The deployment scripts needed to deploy the artefact generated.

### B. IoT ENVIRONMENT DEPLOYMENT ON FIWARE INFRASTRUCTURE

The Execution phase involves deploying all the artefacts generated from the models. So, several software artefacts such as the MQTT messaging broker, device infrastructure, databases, graphical analysis platform, Orion, Perseo, IoTAgent, etc. are configured and deployed.

![Diagram of Orion Context Broker updated from Sensors by means of the IoT-Agent.](image_url)
FIGURE 6. Managing CEP Perseo notifications to notify the event patterns detected to the Actuators.

FIGURE 7. OrionTopicManager operation and interactions.

It should be noted that, although it is a simulation, the deployment of the Fog and Cloud layers of the environment is a real deployment, in other words, the architecture generated could be implemented in a real IoT environment. However, the devices of the Edge layer (sensors and actuators) are fully simulated, interacting with the rest of the layers publishing data, imposing the pace of the simulation (speed of data generation), connecting to and disconnecting from different nodes in the Fog layer (displacement), receiving notifications processed in the Fog or Cloud layer (actuators), etc.

Figure 2 shows the architecture of a Smart Building environment where it is possible to observe the different elements that can be deployed including a CloudNode or FogNode, Sensors and Actuators. Note that CloudNode and FogNode are composed of several elements, including FIWARE elements such as Orion Context Broker, CEP Perseo or IoTAgent.

Furthermore, each CloudNode/FogNode can define a Complex Event Processing Engine or, in other words, the inclusion of CEP Perseo. Besides, it includes Orion Context Broker, IoTAgent-Json, a Non-SQL database, as MongoDB is essential due to it being a dependency of Orion Context Broker, a DataBase Client, a REST API, an MQTT broker (e.g. Mosquitto) and an MQTT Client. Likewise, as can be
observed in Figure 2, all of these elements are interconnected and are deployed on Docker containers. Specifically, all Docker containers are orchestrated using Docker Swarm. Finally, along with the device code generated, a deployment script is included which contains the necessary instructions for deploying the IoT environments. Algorithm 1
Algorithm 1 Deploying an IoT Environment Simulation on FIWARE Architecture and Microservices

begin
  //Step 1) Compilation and docker wrapping of the artefacts.
  while there are components which will have to be compiled do
    Compile component
    if component should be in a docker then
      Wrap component in a docker
    endif
  endwhile
  //Step 2) Pushing of docker images to the local registry.
  while there are docker images to push to local registry do
    Push image
  endwhile
  //Step 3) Creating the Swarm Cluster.
  create manager node 'Environment'
  foreach fogNode do
    Create worker node 'fogNode.name'
    Connect worker node to the manager node
  endforeach
  //Step 4) Configuration of each Swarm node and deploy of the FIWARE components on them.
  foreach worker node do
    Deploy FIWARE components from docker-compose-file
    configure FIWARE components from configuration files
  endforeach
  //Step 5) Pulling of docker images from the local registry in each Swarm node.
  foreach worker node do
    foreach docker image wrapped for this node in the local registry do
      Pull image
    endforeach
  endforeach
  //Step 6) Deployment of the docker components as services from the manager node of the Swarm cluster.
  Connect to the swarm manager node
  foreach pulled docker image in the worker node do
    Deploy docker as service
  endforeach
end

shows the deployment phase of an IoT environment on FIWARE.

VI. CASE STUDIES

Next, two case studies have been defined using SimulateIoT. The first one defines an IoT simulation for a smart building while the second one defines an IoT simulation for an agricultural environment. Note that these two cases are the same as the ones modelled in [3]. Thus demonstrating that the proposal presented in this paper can deploy these same environments with SimulateIoT-FIWARE on the FIWARE platform, without the need to know any technical aspects about it.

A. CASE 01. SCHOOL OF TECHNOLOGY

The first case study presents the simulation of a smart building, more specifically, we have modelled our School of Technologies. It has six buildings (Computer Science, Civil Works, Architecture, Communications, Research and a Common Building). So, each building has its own environment with a set of Sensors, Actuators and analysis information processes.

1) CASE 01. MODEL DEFINITION

Figure 8 shows an excerpt from the School of Technology model. The IoT system modelled includes several Node elements shared throughout the different buildings. Each building takes over its own ProcessNode (Figure 8 references 1.1, 1.2 and 2) which gathers all the information produced by the Sensors (Figure 8 references 3.1 and 3.2). Thus, these data are suitably stored on specific databases (Figure 8 references 6.1 and 6.2), analysed and monitored by the ProcessNode element. In this case study, a FogNode element has been defined for each building. For instance, Common_Building or Computer_Science have defined FogNode elements (Figure 8 references 1.1 and 1.2).

Furthermore, a CloudNode named SchoolTechnology-CloudNode (Figure 8 reference 2) is defined to store information gathered from the FogNode elements. Both FogNode and CloudNode elements define several Topic elements (Figure 8 references 5.1, 5.2 and 5.3) such as heating_temperature, presence and smoke_detection. These Topic elements communicate data among the Node elements defined in the IoT system.

In order to model the School of Technology case study, several Sensors such as heating_temperature_meter, presence_detector, smoke_detector and so on have been defined in Figure 8. Each of them publishes its own data on a specific Topic element. As can be observed in Figure 8, the Sensor elements publish data to several FogNode through Topic elements.

Note that Sensor elements are EdgeNode elements that generate data, so the data pattern generators should be defined (Figure 8 references 4.1 and 4.2). For instance, in order to describe the synthetic data generated by a temperature sensor a .csv input file has been defined. It makes it possible to reuse historical data. Other Sensors can define their synthetic data generators using a random pattern, incremental pattern, etc. So, the approach can consume synthetic data based on simple data, range data, a specific set of values, the values obtained from a .csv file, data obtained from a URL source or data generated from the external tools such as [1], [19].
the ComputerScience FogNode element addresses the information every thirty seconds, storing the data obtained in a specific NoSQL database. Then all data are flushed to the next node FogNode or CloudNode defined in the architecture and named in the example SchoolTechnologyCloudNode.

On the other hand, the Common_Building FogNode element defines a different behaviour in order to analyse the data and take advantage of being close to the devices that should carry out some action. For instance, the Common_Building FogNode defines a CEP engine component and several Rule elements (Figure 8 reference 7), for example, the rule_heating analyses the data obtained from a specific Topic named heating_temperature to notify a specific action to another Topic named turn_on_heating which is subscribed by a specific Actuator named heating (Figure 8 reference 3.3. Thus, the rule_heating rule analyses the temperature sent to the heating_temperature Topic element from the heating_temperature_meter Sensor. Consequently, it is gathered and analysed by the CEP Engine by means of the rule_heating Rule. As a consequence, when the defined pattern is matched (for instance, if (temperature < 20) then switch on heating), the CEP engine generates an event to turn_on_off_heating Topic.

2) CASE 01. CODE GENERATION AND DEPLOYMENT

Once the model has been defined, the M2T transformation is applied with the following goals: i) to generate Java code that wraps each device behaviour; ii) to generate configuration code to deploy based on FIWARE components. These files include the code necessary to register all devices in Orion Context Broker and the code required to define all rules in CEP Perseo. iii) to generate configuration code to deploy the message brokers necessary (and connect them with FIWARE), including the topic configurations defined; iv) to generate for each ProcessNode and EdgeNode a Docker container which can be deployed throughout a network of nodes using Docker Swarm v) to generate the Swarm cluster to deploy the simulation in orchestrated mode.

Figure 9 shows a simplified excerpt from the School of Technology IoT model deployed (full version available in Figure 12) and it includes the following: Each Node has been deployed on a Docker container using Docker Swarm technology. Each Docker container instance deploys the characteristics defined on the IoT model, including: where the nodes are deployed, and what the components included in each ProcessNode are. Thus, each EdgeNode and ProcessNode element carries out its own functions such as sending messages, processing and storing messages, acting from messages, etc.

Additionally, the code generated can be reused on the final system deployed. For instance, the EdgeNode elements can be replaced by physical devices (both Sensors and Actuators), and the Process Node can be deployed as Docker containers either on-premise or on the cloud. Not only is the simulation code generated, but also the final IoT system code is partially generated.

Finally, executing the simulation modelled and later on deploying it, makes it possible to analyse the final IoT environment before it is implemented and deployed. The analysis that can be carried out is fundamentally based on the log behaviour of each node within the simulation. This log behaviour includes parameters such as: i) Each component performs its functions successfully, such as publishing, receiving, analysing, redirecting data, etc. ii) The resources used by each component, such as CPU or Memory usage iii) The general function of the IoT architecture modelled, in other words, if the IoT environment is satisfying the user needs or requirements iv) The evolution of the above-mentioned parameters over time.

In this sense, users using the simulation logs, could evaluate the behaviour of the environment by exposing it to different levels of stress by experimenting with different number of devices, size of published messages, publication periods, etc. and study parameters such as a) jitter between messages, checking in mongodb the timestamps of the messages of a sensor, b) response delay of a particular component, for instance, checking the CEP engine logs it can be seen when a rule is met and when the notification is sent to the actuator, c) packet loss rate, checking the difference of number messages between the messages published by a sensor (sensor logs) and the messages stored in MongoDB from that sensor, etc.

In short, users can carry out different experiments by creating different models and simulating them, thus determining which aspects can be improved until the version that meets his requirements is achieved.

B. CASE 02. AGRICULTURAL ENVIRONMENT

This case study focuses on designing an IoT system for managing irrigation and weather data to improve crop production. So, the case study has been designed to simulate the Sensors and Actuators distributed over the countryside which can be monitored in real-time. Nowadays, the agricultural domain has several requirements [49], [50]: i) Collection of weather, crop and soil information; ii) Monitoring of distributed land; iii) Multiple crops on a single piece of land; iv) Different fertiliser and water requirements for different pieces of uneven land; v) Diverse requirements of crops for different weather and soil conditions; vi) Proactive solutions rather than reactive solutions.

For instance, Sensors such as temperature Sensors, humidity Sensors, irrigation Sensors, PH Sensors and Actuators such as irrigation artefacts help to monitor and save water, optimising crop production.

This agricultural IoT environment has been designed over ten hectares of soil where tomatoes are being cultivated. So, for each hectare, a set of Sensors and fog nodes has been shared. So, using fog nodes decreases the communication requirements among them.

The sensor network is built by temperature, humidity, irrigation and water pressure Sensors. These Sensors send data to a specific Topic element linked to a FogNode.
In addition, the irrigation Actuators have been defined for controlling irrigation water. The notification events from the FogNode elements are sent to Actuator elements using messages by Topic elements.

1) CASE 02. MODEL DEFINITION
In Figure 10 an excerpt from an IoT model conforming to the SimulateIoT-FIWARE metamodel is defined. It shows different Sensor elements such as (ph_H1, temperature_H1, Humidity_H1, etc.) which generate data for simulation (Figure 10 references 3.1 and 3.2). Moreover, several Fog computing nodes have been defined, although in Figure 10 (for the sake of simplicity) only two FogNode elements are shown (Figure 10 references 1.1 and 1.2). They define several Topics such as Humidity, Temperature, pH, Water_pressure, etc (Figure 10 references 5.1 and 5.2). In addition, each FogNode element defines a CEP engine by means of Perseo elements (Figure 10 references 7.1 and 7.2). Besides, several Rule elements (event pattern definitions) such as rule_Humidity or rule_pH have been defined to analyse the data gathered from Topic elements in real-time. Likewise, when an event pattern is matched, a Notification element such as Low_pH, High_pH, Low_Humidity, High_Humidity and so on is thrown. For instance, the Actuator element named Irrigator (Figure 10 references 3.1) is activated when the Notification element named Low_Humidity is thrown.

2) CASE 02. CODE GENERATION AND DEPLOYMENT
Once the model has been completed and validated, an M2T transformation is carried out obtaining the simulation code, which can be deployed on a specific platform, specifically using FIWARE components. Thus, in order to define a scalable IoT environment, each deployable element (EdgeNode, CloudNode, FogNode, Actuators and ProcessEngine) is defined as a microservice, wrapping each Node element in a Docker container. It is worthy of mention that one component could have a complex architecture and be defined in several microservices. In consequence, these kinds of components will be wrapped in several Docker containers (each defined microservice in a container, as for example in the case of FogNode and CloudNode components). Figure 11 shows a simplified excerpt from the case study deployment architecture (full version available in Figure 13). In Figure 11 the main characteristics of each node can be observed. For instance, each ProcessNode defines an Orion Context Broker with its MongoDB database, an IoTAgent, a Mosquitto MQTT message broker and a CEP
Perseo engine. In addition, the Rule elements defined are processed through the CEP Perseo engine defined.

Each ProcessNode element deployed on a Docker container has its own characteristics:

- **CloudNode**, named AgroTe_CloudNode, is composed of an Orion Context Broker with its MongoDB [30], an IoT Agent and a message-driven broker like Mosquitto (that implements an MQTT communication protocol). Moreover, the CloudNode deploys a Compass instance [7] to monitor the data gathered.

- Each FogNode named Hectare_1 and Hectare_2 respectively, is composed of an Orion Context Broker with its MongoDB [30], an IoT Agent, a message-driven broker like Mosquitto (that implements an MQTT communication protocol) and a Perseo engine. MongoDB stores the temporal data gathered by the FogNode instance. Currently, the main difference between a CloudNode and FogNode is the processing capability. Using the size attribute at the FogNode element makes it possible to define the process capabilities. Consequently, both CloudNode elements and FogNode elements are deployed as Docker containers on hardware nodes such as PC, VM or Raspberry Pi.

- The CEP characteristic defined at ProcessNode deploys a complex event processor to process high amounts of messages in real-time. As can be observed in Figure 11 a CEP Perseo engine is deployed on each FogNode. Later on, each CEP Perseo engine analyses

*FIGURE 10. Case 02. AgroTech model conforming to the SimulateIoT metamodel.*
the incoming events by the Rule elements associated with it.

- The EdgeNode elements including Sensors and Actuators defined in the model are suitably deployed in Docker containers.

Later on, the execution information can be audited querying the MongoDB database or using the monitoring tool available on each ProcessNode. Moreover, each Docker is generating log information during the IoT execution. Finally, the nodes deployed are accessible from a dashboard tool that gathers the available endpoints of each element, for example, to query a MongoDB database or to show information about a Mosquitto broker. In the above-mentioned ways, it is possible to perform the analysis of the environment mentioned at the end of Section VI-A.

VII. DISCUSSION

Model-driven development can be used to model complex IoT environments using domain concepts. They need not be tied to a specific technology, but rather an M2T transformation makes it possible to generate the code needed to deploy and simulate the systems.

The technology used as a target, such as FIWARE, microservices (Thorntail), containers (Dockers), message-oriented middleware, MQTT (Mosquitto) or a container orchestrator (Docker Swarm) can be quickly replaced by other suitable technology if needed. Of course, to change the target technology, an M2T transformation should be implemented.

For the reasons mentioned above, it has been considered to give FIWARE an added value through integration with SimulateIoT. The result of this integration is SimulateIoT-FIWARE, which is able to model and generate an IoT environment with FIWARE artefacts. In short, SimulateIoT-FIWARE, the resulting tool from the integration of FIWARE and SimulateIoT based on Model-Driven development, define an abstraction layer that allows the use of FIWARE artefacts without the need to know how these components work internally, that is, how they interact with each other and with the other components of an environment, how their deployment is configured, how they are configured to work in the environment, etc. with the final purpose of generating and deploying an IoT environment powered by FIWARE.

Finally, the target users could be both: a) professional users and b) students. Professional users could use the methodology and tools presented in this work to define and analyse complex IoT environments where finally heterogeneous technology is used, even though the core comprises components provided by FIWARE. Besides, our approach can be used for teaching purposes because it makes it possible for students to learn about IoT concepts and relationships. In addition, they can deploy the IoT simulation, and study the code generated to learn the technology used to deploy the IoT system. Thus, they can understand IoT cutting-edge technology such as FIWARE, edge technology and integration patterns such as data patterns, IoT characteristics, publish-subscribe communication protocols, MQTT, containers, NoSQL databases, distributed systems and so on.
A. LIMITATIONS

Although the domain-specific language and tools presented offer a wide expressiveness, they have several limitations to take into account:

- The Edge Nodes can be defined as mobile nodes by using several approaches (FogCloudRoute, LinearRoute and RandomRoute). However, IoT mobility is a wide and interesting research area where multiple protocols and mobility mechanisms could be additionally defined.
- For the sake of simplicity, the current version of our simulator IoT environment for FIWARE allows defining connected nodes by TCP/IP, and it is assumed that connectivity is guaranteed.
- It is possible to simulate IoT environments defined using a high-level domain-specific language. However, the hardware simulation is only managed by the size attribute at ProcessNode which implies several constraints to avoid creating specific software elements (see Table 2). Obviously, it could be considered a simplistic approach to tackle this complex problem, but in the end, it helps users to model IoT environments taking hardware restrictions into account.

VIII. RELATED WORK

At this point, several Model-Driven Development approaches have been defined to manage IoT complexity, however, there are no MDD approaches focused on generating code for well-known or global IoT platforms such as FIWARE. Next, additional MDD approaches related to IoT environment definition are analysed. Next, the main Model-Driven approaches to generate IoT systems are reviewed.

FRASAD [33] is a model-driven software development framework to manage the complexity of the Internet of Things (IoT) applications. FRASAD is based on node-centric software architecture and a rule-based programming model that allows designers to describe their applications (IoT environments).

An application within FRASAD could include several sensors with multiple characteristics. For instance, some sensors could publish temperature, others could receive it and, if rules are specified, the centric-node will apply the rules to this temperature data and from the result of the analysis, modify the behaviour of the sensors. It is worthy of mention that, regardless of the characteristics chosen, within FRASAD all devices are sensors, there is not a hierarchy of devices. In addition, each sensor could have multiple inputs and outputs. Although FRASAD provides multiple options to model sensors, users cannot choose the target technology, for example, to apply rules, to communicate each sensor (communication protocol), to store data, etc.

MDE4IoT [6] is a Model Driven Engineering [41] approach that allows the modelling of IoT components and supports intelligence as self-adaptation of Emerging Configurations in the IoT. Within MDE4IoT they call Emergent Configuration (EC) of connected systems a set of things/devices with their functionalities and services that connect and cooperate temporarily to achieve a goal. In short, MDE4IoT allows users to define an IoT environment that is able to adapt the behaviour of its devices at run-time. For instance, MDE4IoT could define and generate an IoT environment where several inter-connected Smart-Lamps adapt their behaviour (light colour, brightness, etc.) depending on the traffic flow or other environmental data such as car speed, the distance between cars, natural light, etc. MDE4IoT allows users to define hardware and software characteristics of a device, being able to define a sensor, an actuator or another kind of device, of course, each with its own characteristics. However, MDE4IoT does not allow users to choose the technology they want to use, for instance, the users cannot choose the database, the rule engine (to manage the EC), the communication protocol, etc. On the other hand, MDE4IoT generates the code to be implemented in the physical devices of the environment, not allowing a simulation of it. Additionally, a global target such as FIWARE is not available.

Another approach such as [38] proposes a model-driven software development framework that allows users to model IoT environments with several types of devices with many modelling features. It proposes that the stakeholders could add features to the framework. These stakeholders are: 1) The sensor Manufacturer/sensor Provider, who could add device features such as device drivers, data models or device interfaces, 2) The Algorithm expert/Algorithm developer who defines algorithm features as CPU/Memory requirements, performance or accuracy, 3) The Domain Expert who manages the model requirements or the mapping of the algorithms to the sensors and 4) The System Administrator who could add features such as CPU/Memory availability or the calculations of the network characteristics through devices and the cloud. In this way, these four stakeholders could develop a powerful framework to generate IoT environments, however, although the abstraction layer to develop IoT environments has been incremented with this framework, the user needs to know several concepts about the domain of these four stakeholders. For instance, this framework incorporates many algorithms that can be added to devices, in this way, the user does not need to know how to implement the algorithms, but they need to know how they work because several algorithms could do the same thing in different ways, and the user needs to know which one best fits their needs and requirements. The above-mentioned example can be extrapolated to the other features which could be modelled with this framework. In short, due to the low abstraction layer that this framework provides, the users need to be experts in the IoT and all the concepts around it, such as the hardware used, algorithms or the IoT domain. In [38] an initial prototype had been developed to cover some of the aforementioned aspects. In addition, the IoT applications defined using this framework are deployed using their own implementation. Consequently, they do not use a global IoT architecture such as FIWARE as a target.
Fog computing is proposed to solve the latency problems of the services offers by the Cloud. Nevertheless, to realise the full potential of Fog and IoT paradigms, it is necessary to design resource management techniques that determine which modules of analytics applications are pushed to each edge device to minimize the latency and maximise the throughput [18].

In [18] iFogSim an IoT simulator is proposed that enables the quantification of the performance of resource management policies on an IoT or Fog computing infrastructure in a repeatable manner. This simulator can measure performance in four different areas: latency, network congestion, energy consumption, and cost.

iFogSim allows users to model an IoT environment with several nodes such as Sensors, Actuators, Fog devices, etc. with different nodes or environmental properties such as 1) Hardware characteristics: accessible memory, processor, storage size, uplink, and downlink bandwidths, 2) Network characteristics: connectivity among devices, latency, network congestion, etc. 3) Data characteristics: Data flow, type of data, etc. among other devices or environment properties.

iFogSim is a simulator that, because of the great capacity of expression that it possesses, can simulate very similar environments to a real one. In consequence, the users that employ this tool need to skillfully manage a lot of concepts about IoT, Networking, Fog and Cloud paradigms, etc. Due to the above-mentioned aspect, this tool is recommended for expert users, and may not be the best option to some purposes or targets such as education, novel users in IoT or engineering, small IoT environments such as a domotic house, or IoT environments that do not need to use Fog computing.

MobIoTSim [39] is an Android IoT simulator which aims to help Cloud application developers to create IoT environments with several devices without buying real sensors. In this way, MobIoTSim allows the simulation of IoT environments where developers can learn, test and demonstrate IoT applications, which works with IoT Cloud providers such as Bluemix [21] or Google IoT Platform, in a fast and efficient way.
MobIoTSim allows, from an Android device, the configuration of several IoT Cloud gateways to allow connection with the devices. Add, edit or delete devices that sends (MQTT and JSON) random data in a range defined by the user with a frequency. In addition, devices can receive data, for instance, in [39] Bluemix is configured to send warning notifications to devices if it detects critical values. Besides, MobIoTSim provides the display of the data published by the simulated devices.

In [32], the authors make an in-depth analysis of the state of the art of deployment and orchestration in IoT environments. Additionally, the authors have developed a taxonomy of DEPO4IOT to classify, analyse, and compare the studies. This taxonomy takes into account factors such as the deployment and orchestration support, design support and other advanced supports. Our proposal takes into account the importance of deployment and orchestration of the IoT environment, including the possibility of deploying them in Docker containers and orchestrating them using Docker Swarm, also generating a deployment script from the models defined by the users, where all the necessary parameters are automatically configured to carry out a reliable deployment.

The expressiveness of SimulateIoT Domain Specific Language determines the expressive capacity of SimulateIoT-Fiware. Therefore, simulation of specific aspects of particular IoT environments require developing an ad-hoc extension.
So, SimulateIoTFiware requires additional changes to model IoT environments focus on particular aspects such as: a) ThinORAM [20], a lightweight client-side ORAM system that substantially improves response time and network usage respect the existing ORAM systems in literature; b) The approach conducted in [34], a decentralised Blockchain-based architecture to manage the roles and permissions of the IoT devices of an IoT environment; c) ProfiloIoT [28], a machine learning approach that, from the traffic generated by a device on the network, is able to determine whether it is an IoT device (and what kind of IoT device it is) or not.

Although SimulateIoTFiware requires additional changes to model an IoT environment with ThinORAM [34] or ProfiloIoT [28], it has the expressive capability to model the IoT architecture of FIWARE-based Edge, Fog and Cloud nodes.

To sum up, although there are several approaches focused on raising the abstraction level from the IoT applications that can be developed, there is a lack of approaches to carry out this process using a global IoT infrastructure as the target. The present proposal is tailored to FIWARE technology, allowing modelling large IoT projects which can be deployed later on using this IoT platform.

**IX. CONCLUSION**

Model-driven development techniques are a suitable way to tackle the complexity of domains integrating heterogeneous technologies. Initially, they focus on modelling the domain, then, by using M2T transformations, the code for specific technology could be generated. SimulateIoT takes advantage of this technology to allow the modelling and the generation of IoT environments.

Moreover, FIWARE has a large catalogue with several components oriented to the IoT which allow the development of complex IoT architectures. Besides, FIWARE is a popular open-source project, and as a consequence, FIWARE has great support and its components are highly tested.

Both technologies (SimulateIoT and FIWARE) have been integrated. In this way, the resulting tool allows users to define and validate models conforming to the SimulateIoT metamodel. Then, an M2T transformation makes it possible to generate the FIWARE components needed to deploy the IoT Simulation defined.

Future projects include new concepts taking into account the FIWARE catalogue. For instance, components such as Cosmos, which enables an easier BigData analysis, FogFlow, to support dynamic processing flows over cloud and edges, or Knowage, which brings a powerful Business Intelligence platform enabling users to perform business analytics over traditional sources and big data systems. Other interesting further work includes the improvement of the SimulateIoT components which cannot be replaced by FIWARE, for instance, the Sensors, which could be improved by defining and generating new kinds of data generation patterns. Finally, it is expected to explore the code generation to other IoT platforms such as Google Cloud’s IoT Platform [17], Microsoft Azure IoT suite [23], ThingSpeak IoT Platform [47] or Thingworx 8 IoT Platform [48].

**APPENDIX A**

```yaml
orion:
  image: FIWARE/orion:2.0.0
  hostname: orion
  container_name: FIWARE-orion
  depends_on:
    - mongo-db
  expose:
    - "1026"*1
  ports:
    - "8082:1026"
  ....

iot-agent:
  image: FIWARE/iotagent-json
  hostname: iot-agent
  container_name: FIWARE-iot-agent
  depends_on:
    - mongo-db
    - Mosquitto
  expose:
    - "4041"*2
  ports:
    - "4041:4041"

environment:
  - IOTA_CB_HOST=orion
  - IOTA_CB_PORT=1026
  - IOTA_NORTH_PORT=4041
  - IOTA_REGISTRY_TYPE=mongodb
  - IOTA_MONGO_HOST=mongo-db
  - IOTA_MONGO_PORT=27017
  - IOTA_MONGO_DB=iotagent-json
  - IOTA_MQTT_HOST=mosquitto
  - IOTA_MQTT_PORT=1883
  - IOTA_REGISTRY_TYPE=mongodb
  - IOTA_MQTT_HOST=mosquitto
  - IOTA_MQTT_PORT=1883
  - IOTA_REGISTRY_TYPE=mongodb
  - IOTA_MQTT_HOST=mosquitto
  - IOTA_MQTT_PORT=1883
  - IOTA_PROVIDER_URL=http://iot-agent:4041

mongo-db:
  image: mongo:3.6
  hostname: mongo-db
  container_name: db-mongo
  expose:
    - "27017"
  ports:
    - "27017:27017"

perseo-core:
  image: FIWARE/perseo-core
  environment:
    - PERSEO_FE_URL=http://perseo-fe:9090
    - MAX_AGE=6000
  depends_on:
    - mongo-db
  environment:
    - PERSEO_FE_URL=http://perseo-fe:9090
    - MAX_AGE=6000

perseo-fe:
  image: FIWARE/perseo
  ports:
    - 9090:9090
  depends_on:
    - perseo-core
  environment:
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPOINT=mongo-db
    - PERSEO_MONGO_ENDPO...
APPENDIX B
CODE FRAGMENT TO CONFIGURE ORION

```bash
curl -iX POST "http://localhost:4041/iot/devices" -H "Content-Type: application/json" -H "FIWARE-service: openiot" -H "FIWARE-servicepath:" -d '{
  "devices": {
    "device_id": "",
    "Sensor_Heating_Temperature_meter_5",
    "entity_name": "urn:ngsi-ld:Sensor_Heating_Temperature_meter_5",
    "type": "Sensor_Heating_Temperature_meter",
    "protocol": "JSON",
    "timezone": "Europe/Berlin",
    "transport": "MQTT",
    "attributes": {
      "object_id": "",
      "name": "value",
      "type": "Integer"
    },
    "static_attributes": {
      "name": "value",
      "type": "Sensor_Heating_Temperature_meter"
    }
  }
}
```

APPENDIX C
CODE FRAGMENT TO CONFIGURE PERSSEO

```bash
curl -iX POST "http://localhost:9090/rules" -H "FIWARE-service: openiot" -H "FIWARE-servicepath:" -H "Content-Type: application/json" -d '{
  "name": "rule0_sensor_heating_temperature_meter",
  "text": "select * from "rule0_sensor_heating_temperature_meter" as ruleName from pattern [everyev IoTEvent (cast(cast(value?, String),float)>25 and id="urn:ngsi-ld:Topic_heatingtemperature:0")],",
  "action": {
    "type": "post",
    "template": "$\{value\}$",
    "parameters": {
      "url": "http://mncsecciontecnologia2:1510/heatings_0",
      "headers": {
        "Content-type": "application/json"
      }
    }
  }
}
```

APPENDIX D
COMPLETE USE CASE DEPLOYMENT ARCHITECTURE

See Figures 11 and 12.

REFERENCES

[1] J. W. Anderson, K. E. Kennedy, L. B. Ngo, A. Luckow, and A. W. Apon, “Synthetic data generation for the Internet of Things,” in Proc. IEEE Int. Conf. Big Data (Big Data), Oct. 2014, pp. 171–176.

[2] C. Atkinson and T. Kühne, “Model-driven development: A metamodelling foundation,” IEEE Softw., vol. 20, no. 5, pp. 36–41, Sep. 2003.

[3] J. A. Barriga, P. J. Clemente, E. Sosa-Sanchez, and A. E. Prieto, “SimulateIoT: Domain specific language to design, code generation and execute IoT simulation environments,” IEEE Access, vol. 9, pp. 92531–92552, 2021.

[4] T. Bass, “Mythbusters: Event stream processing versus complex event processing,” in Proc. Inaugural Int. Conf. Distrib. Event-Based Syst. (DEBS), 2007, p. 1.

[5] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” Softw., Pract. Exp., vol. 41, no. 1, pp. 23–50, 2011.

[6] F. Coccozzi and R. Spalazzese, “Midiowr: Supporting the Internet of Things with model-driven engineering,” in Proc. Int. Symp. Intell. Distrib. Comput. Cham, Switzerland: Springer, 2016, pp. 67–76.

[7] (2018). MongoDB Compass. [Online]. Available: https://www.mongodb.com/products/compass

[8] G. Cugola and A. Margara, “Processing flows of information: From data stream to complex event processing,” ACM Comput. Surv., vol. 44, no. 3, pp. 1–62, 2012.

[9] EsperTech. (Nov. 2016). Esper Cep. [Online]. Available: https://www.espertech.com/esper/

[10] Fiware. (2019). Orion Context Broker. [Online]. Available: https://fiware.readthedocs.io/en/latest/architecture/architecture/

[11] Fiware. (2019). Perseo Architecture. [Online]. Available: https://perseo.readthedocs.io/en/latest/about-us/

[12] Fiware. (2021). Fiware. [Online]. Available: https://www.fiware.org/about-us/

[13] Fiware. (2021). Ngsi Protocol. [Online]. Available: https://knowage.readthedocs.io/en/6.1.1/user/NGSI/README/index.html

[14] E. Fotopoulou, A. Zafeiropoulos, F. Terroso-Sáenz, U. Şimşek, A. González-Vidal, G. Tsolias, P. Gouvas, P. Liapis, A. Fensel, and A. Skarmeta, “Providing personalized energy management and awareness services for energy efficiency in smart buildings,” Sensors, vol. 17, no. 9, p. 2054, Sep. 2017.

[15] M. García, “New businesses around open data, smart cities and fiware,” Eur. Public Sector Inf. Platform, Tech. Rep. 4, 2015. [Online]. Available: https://data.europa.eu/sites/default/files/report/2015_new_businesses_around_open_data_smart_cities_and_fiware.pdf

[16] L. Gutiérrez-Madroñal, I. Medina-Bulo, and J. J. Domínguez-Jiménez, “IoT simulation environments,” IEEE Access, vol. 5, pp. 784–803, Mar. 2017.

[17] Google. (2017). Google Cloud IoT. [Online]. Available: https://cloud.google.com/solutions/iot/

[18] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, “IFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments,” Softw., Pract. Exp., vol. 47, no. 9, pp. 1275–1296, 2017.

[19] L. Guadet and M. Madronal, “Perseo: An architecture for sensor data processing and mobility,” Sensors, vol. 17, no. 9, pp. 1–62, 2017.

[20] Y. Huang, B. Li, Z. Liu, J. Li, S.-M. Yiu, T. Baker, and B. B. Gupta, “ThinORAM: Towards practical oblivious data access in fog computing environment,” IEEE Trans. Services Comput., vol. 13, no. 4, pp. 602–612, Jul. 2020.

[21] IBM. (2014). IBM Cloud. [Online]. Available: https://www.ibm.com/cloud/bluemix/

[22] J. Han, E. Hailong, G. Le, and J. Du, “Survey on NoSQL database,” in Proc. 6th Int. Conf. Pervasive Comput. Appl. (ICPCA), Oct. 2011, pp. 363–366.
[23] S. Klein, *IoT Solutions* Microsoft’s Azure IoT Suite. New York, NY, USA: Springer, 2017.

[24] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, “Eugenia: Towards disciplined and automated development of OMG-based graphical model editors,” *Softw. Syst. Model.*, vol. 16, pp. 1–27, Feb. 2015.

[25] J. A. López-Riquelme, N. Pavón-Paide, H. Navarro-Hellín, F. Soto-Valles, and R. Torres-Sánchez, “A software architecture based on FIWARE cloud for precision agriculture,” *Agricul. Water Manage.*, vol. 183, pp. 123–135, Mar. 2017.

[26] Arun Mathew, “Benchmarking of complex event processing engines for IoT applications,” *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 506–509.

[27] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, “Eugenia: Towards disciplined and automated development of OMG-based graphical model editors,” *Softw. Syst. Model.*, vol. 16, pp. 1–27, Feb. 2015.

[28] J. A. López-Riquelme, N. Pavón-Paide, H. Navarro-Hellín, F. Soto-Valles, and R. Torres-Sánchez, “A software architecture based on FIWARE cloud for precision agriculture,” *Agricul. Water Manage.*, vol. 183, pp. 123–135, Mar. 2017.

[29] Arun Mathew, “Benchmarking of complex event processing engines for IoT applications,” *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 506–509.

[30] O. Novo, “Blockchain meets IoT: An architecture for scalable access management in IoT,” *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1184–1195, Apr. 2018.

[31] OMG Object Constraint Language (OCL), Version 2.3.1, OMG, Milford, MA, USA, Jan. 2012.

[32] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, “Eugenia: Towards disciplined and automated development of OMG-based graphical model editors,” *Softw. Syst. Model.*, vol. 16, pp. 1–27, Feb. 2015.

[33] J. A. López-Riquelme, N. Pavón-Paide, H. Navarro-Hellín, F. Soto-Valles, and R. Torres-Sánchez, “A software architecture based on FIWARE cloud for precision agriculture,” *Agricul. Water Manage.*, vol. 183, pp. 123–135, Mar. 2017.

[34] Arun Mathew, “Benchmarking of complex event processing engines for IoT applications,” *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 506–509.

[35] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, “Eugenia: Towards disciplined and automated development of OMG-based graphical model editors,” *Softw. Syst. Model.*, vol. 16, pp. 1–27, Feb. 2015.

[36] J. A. López-Riquelme, N. Pavón-Paide, H. Navarro-Hellín, F. Soto-Valles, and R. Torres-Sánchez, “A software architecture based on FIWARE cloud for precision agriculture,” *Agricul. Water Manage.*, vol. 183, pp. 123–135, Mar. 2017.

[37] Arun Mathew, “Benchmarking of complex event processing engines for IoT applications,” *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 506–509.

[38] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, “Eugenia: Towards disciplined and automated development of OMG-based graphical model editors,” *Softw. Syst. Model.*, vol. 16, pp. 1–27, Feb. 2015.

[39] J. A. López-Riquelme, N. Pavón-Paide, H. Navarro-Hellín, F. Soto-Valles, and R. Torres-Sánchez, “A software architecture based on FIWARE cloud for precision agriculture,” *Agricul. Water Manage.*, vol. 183, pp. 123–135, Mar. 2017.

[40] Arun Mathew, “Benchmarking of complex event processing engines for IoT applications,” *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 506–509.

[41] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, “Eugenia: Towards disciplined and automated development of OMG-based graphical model editors,” *Softw. Syst. Model.*, vol. 16, pp. 1–27, Feb. 2015.

[42] J. A. López-Riquelme, N. Pavón-Paide, H. Navarro-Hellín, F. Soto-Valles, and R. Torres-Sánchez, “A software architecture based on FIWARE cloud for precision agriculture,” *Agricul. Water Manage.*, vol. 183, pp. 123–135, Mar. 2017.

[43] Arun Mathew, “Benchmarking of complex event processing engines for IoT applications,” *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 506–509.

[44] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, “Eugenia: Towards disciplined and automated development of OMG-based graphical model editors,” *Softw. Syst. Model.*, vol. 16, pp. 1–27, Feb. 2015.

[45] J. A. López-Riquelme, N. Pavón-Paide, H. Navarro-Hellín, F. Soto-Valles, and R. Torres-Sánchez, “A software architecture based on FIWARE cloud for precision agriculture,” *Agricul. Water Manage.*, vol. 183, pp. 123–135, Mar. 2017.

[46] Arun Mathew, “Benchmarking of complex event processing engines for IoT applications,” *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 506–509.

[47] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, “Eugenia: Towards disciplined and automated development of OMG-based graphical model editors,” *Softw. Syst. Model.*, vol. 16, pp. 1–27, Feb. 2015.

[48] J. A. López-Riquelme, N. Pavón-Paide, H. Navarro-Hellín, F. Soto-Valles, and R. Torres-Sánchez, “A software architecture based on FIWARE cloud for precision agriculture,” *Agricul. Water Manage.*, vol. 183, pp. 123–135, Mar. 2017.

[49] Arun Mathew, “Benchmarking of complex event processing engines for IoT applications,” *Proc. Symp. Appl. Comput.*, Apr. 2017, pp. 506–509.

[50] N. Wang, N. Zhang, and M. Wang, “Wireless sensors in agriculture and food industry—recent development and future perspective,” *Comput. Electron. Agricult.*, vol. 50, no. 1, pp. 1–14, 2006.

[51] A. Z. Abbasi, N. Islam, and Z. A. Shaikh, “A review of wireless sensors and networks’ applications in agriculture,” *Comput. Standards Interfaces*, vol. 36, no. 2, pp. 263–270, Feb. 2014.

*JOSÉ A. BARRIGA* received the degree in computer science from the University of Extremadura, in 1998, and the Ph.D. degree in computer science, in 2007. He is currently an Associate Professor with the Engineering of Computer and Telematics Systems Department, University of Extremadura. He has published numerous peer-reviewed papers in international journals, workshops, and conferences. His research interests include component-based software development, aspect orientation, service-oriented architectures, business process modeling, and model-driven development. He is involved in several research projects. He has participated in many workshops and conferences as speaker and member of the program committee. He has been the Head of the Engineering of Computer and Telematics Systems Department, University of Extremadura, since February 2018.

*PEDRO J. CLEMENTE* received the B.Sc. degree in computer science from the University of Extremadura, Spain, in 1998, and the Ph.D. degree in computer science, in 2007. He is currently a Full Professor in languages and the Head of the Quercus Software Engineering Group, University of Extremadura. His research interests include service-oriented computing, cloud computing, and model driven development. He is involved in several research projects as responsible and senior researcher related to these subjects. He has published the results of his research in more than 150 papers in international journals, conference proceedings and book chapters. He has participated in many workshops and conferences as a speaker and a member of the program committee. He is currently the Vice President of SISTEDES, the Spanish Society of Software Engineering and Software Development Technology, and the Vice-Chancellor for Digital Transformation with the University of Extremadura.

*MIGUEL A. PÉREZ-TOLEDANO* received the M.Sc. degree in computer science from the Polytechnic University of Catalonia, in 1993, and the Ph.D. degree in computer science from the University of Extremadura, in 2008. He is currently an Associate Professor with the Engineering of Computer and Telematics Systems Department, University of Extremadura. He belongs to the Quercus Software Engineering Group. His research interests include software architecture, component-based software development, software coordination and adaptation, and aspect oriented software development. He has participated as an organizer of different editions of the workshop and conferences. He was the Head of the Engineering of Computer and Telematics Systems Department, University of Extremadura, from September 2009 to February 2018.

*JUAN HERNÁNDEZ* received the B.Sc. degree in mathematics from the University of Extremadura, Spain, and the Ph.D. degree in computer science from the Technical University of Madrid. He is currently a Full Professor in languages and the Head of the Quercus Software Engineering Group, University of Extremadura. His research interests include service-oriented computing, cloud computing, and model driven development. He is involved in several research projects as responsible and senior researcher related to these subjects. He has published the results of his research in more than 150 papers in international journals, conference proceedings and book chapters. He has participated in many workshops and conferences as a speaker and a member of the program committee. He is currently the Vice President of SISTEDES, the Spanish Society of Software Engineering and Software Development Technology, and the Vice-Chancellor for Digital Transformation with the University of Extremadura.