Pervasive and Mobile Computing 89 (2023) 101751

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Design, code generation and simulation of IoT environments with mobility devices by using model-driven development: SimulateIoT-Mobile[☆]

José A. Barriga [*], Pedro J. Clemente, Miguel A. Pérez-Toledano, Elena Jurado-Málaga, Juan Hernández

INTIA Research Institute, Quercus Software Engineering Group, Spain [1]
Department of Computer and Telematic Systems Engineering, Universidad de Extremadura, Av. Universidad s/n, 10003, Cáceres, Spain

A R T I C L E   I N F O

Article history:
Received 15 January 2022
Received in revised form 30 December 2022
Accepted 11 January 2023
Available online 20 January 2023

Keywords:
IIoT systems
IoT simulation
IoT mobility
Model-driven development
Model to text transformation

A B S T R A C T

Systems based on the Internet of Things (IoT) are continuously growing in many areas such as smart cities, home environments, buildings, agriculture, industry, etc. Device mobility is one of the key aspects of these IoT systems, but managing it could be a challenge. Mobility exposes the IoT environment or Industrial IoT (IIoT) to situations such as packet loss, increased delay or jitter, dynamism in the network topology, new security threats, etc. In addition, there is no standard for mobility management for the most commonly used IoT protocols, such as MQTT or CoAP. Consequently, managing IoT mobility is a hard, error-prone and tedious task. However, increasing the abstraction level from which the IoT systems are designed helps to tackle the underlying technology complexity. In this regard, 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, a Domain-Specific Language based on SimulateIoT is proposed for the design, code generation and simulation of IoT systems with mobility management for the MQTT protocol. The IoT systems generated integrate the sensors, actuators, fog nodes, cloud nodes and the architecture that supports mobility, which are deployed as microservices on Docker containers and composed suitability. Finally, two case studies focused on animal tracking and a Personal mobility device (PMD) based on bicycles IoT systems are presented to show the IoT solutions deployed.

© 2023 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

The Internet of Things (IoT) and Industrial Internet of Things (IIoT) are being exploited in several areas such as smart-cities, home environments, agriculture, industry, intelligent buildings, etc. [1]. As can be seen, IoT applications can be

☆ This work was funded by the Government of Extremadura, Council for Economy, Science and Digital Agenda under the grant GR21133 and the project IB20058 and by the European Regional Development Fund (ERDF); and Cátedra Telefónica de la Universidad de Extremadura (Red de Cátedras Telefónica).

* Corresponding author at: Department of Computer and Telematic Systems Engineering, Universidad de Extremadura, Av. Universidad s/n, 10003, Cáceres, Spain.

E-mail addresses: jose@unex.es (J.A. Barriga), pjclemente@unex.es (P.J. Clemente), toledano@unex.es (M.A. Pérez-Toledano), elenajur@unex.es (E. Jurado-Málaga), juanher@unex.es (J. Hernández).

1 http://quercusseg.unex.es.

https://doi.org/10.1016/j.pmcj.2023.101751
1574-1192/© 2023 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

very different from each other and therefore have different requirements and needs. Thus, one of the more interesting requirements of an IoT environment is the mobility of its devices [2,3]. This is because in certain environments some devices need to be mobile to perform their tasks, e.g. personal mobility devices such as bicycles, scooters, etc. that can be rented in any city, or GPS sensors that may be placed on animals in extensive farms [4,5]. In the same way, manufacture and industrial processes could demand the deployment IoT mobile devices throughout on industrial factory [6].

However, managing the mobility of a device through an IoT environment can be challenging. In this sense, according to the design of the environment, mobility can lead to periods of disconnection, resulting in packet loss [7]. Besides, mobility can also lead to increased delay or, in distributed systems, increased jitter [8], e.g. when a device is far away from its gateway or migrates to another gateway. Mobility also means, at the network level, dealing with network dynamism [9]. As for the most commonly used communication protocols in IoT, such as MQTT [10] or CoaP [11], they do not offer mechanisms for mobility management. In addition to these issues, there are also some concerns such as security [12], efficient battery management of the devices [13], etc.

Approaches to managing all these problems need to be measured and tested in order to handle mobility in an efficient way. For example, in order to avoid packet loss during a disconnection period, the Intermediate Buffering technique [14] can be applied, however several tests are necessary to choose the optimal buffer size for each device. Another example is the need to measure jitter [8], as for some critical devices this parameter should not exceed certain limits. On the other hand, it is necessary to measure and efficiently use the energy of each device to guarantee the correct functioning of the device until the next load. It may also be interesting to test the behaviour of the environment if one of the mechanisms supporting mobility goes down (e.g. the neighbour discovery service to deal with the dynamism of the network topology).

Taken into account the aforementioned problems, several research questions could be defined:

RQ1. How could mobility be managed in IoT systems where the MQTT protocol is used?
RQ2. How might model-driven techniques be applied to model IoT systems with mobile nodes?
RQ3. To what extent is it possible to generate the code needed to simulate an IoT system with mobile nodes from a model of the system?
RQ4. To what extent could simulations of mobile IoT systems be useful for optimising the real system?

Taken into account the aforementioned problems and limitations, in this paper, we propose the use of a Model-Driven Development (MDD) [15,16] approach to design, simulate and generate the IoT mobility systems. In this context, MDD helps domain experts to model the system using high level tools based on models which can be modelled, validated and used to generate the IoT code. Using MDD for developing IoT environment with mobility support helps users reason about the IoT system focusing on the specific domain more than the specific code or framework to use.

SimulateIoT [17] is an MDD approach that makes it possible to design and simulate IoT systems. The IoT systems designed with SimulateIoT can include different IoT nodes such as Cloud, Fog, or Edge nodes and multiple computing services such as Complex Event Processing service, Publish/Subscribe service or Storage service.

However, it cannot model mobile devices or nodes. Mobility could be an interesting extension in order to facilitate the description and simulation of complex IoT environments where IoT mobility represents a key factor. In this way, solutions to device mobility can be measured and tested by means of simulations, thus helping IoT developers to handle mobility efficiently within an IoT system.

In this work, SimulateIoT-Mobile, an extension of SimulateIoT that includes support for simulating IoT systems with mobile nodes, is presented. In this regard, note that the content described in this communication only focuses on describing new contributions or features added as part of the SimulateIoT-Mobile extension.

Thus, the main work contributions are the following:

- This work shows that the use of Model-Driven Development techniques is suitable for developing tools and languages to tackle successfully the complexity of IoT environments where devices mobility is a key factor.
- This work includes a metamodel to model IoT environments with mobile nodes. It includes model restrictions and a Graphical Concrete Syntax.
- A Model-to-text transformation to code generate for specific IoT platform.
- Two case studies have been designed and evaluated in order to validate the proposal.

The rest of the paper is structured as follows. In Section 2, we give an overview of existing IoT simulation approaches centred on both low-level and high-level IoT simulation environments. Next, Section 3 introduces SimulateIoT-Mobile. In Section 4 the MQTT Mobility Management Model is defined. Next, Section 5 presents the *SimulateIoT-Mobile* taken into account design and implementation phases including the *SimulateIoT-Mobile* metamodel and the graphical editor. In Section 6 the model-to-text transformation from SimulateIoT-Mobile models to code is explained. Section 7 shows the IoT environment simulation outputs and how they could be analysed. In Section 8 two case studies are presented. Finally, Section 9 elaborates on the discussion of the presented approach before Section 10 concludes the paper.

2. Related works

Mobility devices in IoT environment has been addressed for multiple points of view: (1) including extending communication protocols, (2) including communication protocols with additional devices characteristics such as the battery, QoS, latency, etc. or (3) using high-level proposals for managing IoT mobility devices.

On the one hand, several protocols allow mobility in IoT environments as reflected in [18]. This article discusses and compares various communication protocols for Wireless Sensor Networks based on 6LoWPAN technology. Some of these protocols are MIPv6, HMIPv6, ZoroMSN or LoWMob among others. All of them try to achieve optimal performance in terms of QoS, Resource Management, Security, Topology control and Routing protocol. However, the work [18] concludes that there is no efficient solution to meet all requirements and constraints of WSN with 6LoWPAN technology.

Similar studies such as [19] communicates that due to resource constraints in IoT or WSNs, the design of new communication protocols is required. Furthermore, studies such as [20,21] come to the same conclusions and present their own proposals to solve this issue. However, IoT is a heterogeneous area, where systems have different requirements and where technologies such as 6LoWPAN are still being studied and applied in IoT systems nowadays [22–24].

An example of such studies is [25], that conducts a comparative study between classical and bio-inspired mobility. This study addresses different schemes of mobility within a WSN, however, a greater effort is made to optimise the so-called sink node. The sink nodes are nodes that move through the WSN collecting data sensed by different devices on the network. In this way, the use of energy of the other devices is harvested by avoiding the use of multi-hop communication. Besides, mobile sink nodes offer other features such as load balancing of the network, in the sense that they can transfer the data collected anywhere in the WSN. Therefore, by optimising the sink nodes, the entire WSN is optimised. To this end, the authors compare the different classical mobility protocols with the bio-inspired ones, concluding that the latter surpasses the classics in several aspects such as network congestion, computational complexity or latency among others. However, although these types of protocols are promising, major research efforts are still needed to effectively implement them in a real WSN.

With the aim of addressing the resource constraints of some IoT systems or WSNs, protocols such as MQTT, Coap or DDS have been developed, becoming in the most widely used in the IoT due to their high performance [26]. However, this protocols lack mechanisms for the use of mobile nodes. In this regard, several works [8,27,28] focus on addressing the challenges of mobility management of these protocols.

In [27] the authors propose a solution to avoid the loss of information when mobile devices are not connected to any MQTT Broker, such as when devices are migrating from one Broker to another. This proposal is based on a technique called intermediate buffering. This technique decouples the production of messages from their publication, establishing between these two phases an intermediate buffer where the messages are stored in an ordered manner with the aim of publishing them in the first instance when the mobile device in question recovers the connection. This technique avoids the loss of information, however, it has not been tested in large scale IoT environments, and it only partially solves one of the problems associated with mobility in IoT when MQTT protocol is being used.

In [28] a protocol is proposed to allow mobility nodes in CoAP IoT environments. For this, an architecture based on three elements is used, these elements are: (1) CoAP Node (Server), (2) CoAP Node (Client), (3) Mobility Management Table (MMT). Thus, the CoAP Node Client can request data from some CoAP Node Server through the Mobility Management Table. The Mobility Management Table stores relevant information about CoAP nodes such as their IP address, temporal IP address, state of the nodes (handover data or not), etc. making possible the communication between nodes whether they are moving or not. Besides, mechanisms to avoid packet loss have been included in this protocol, this mechanism put in "hold" mode the CoAP nodes to avoid wrong publications, for instance, when a node is changing its IP. However, is not taken into consideration the loss of connection with the Mobility Management Table, the connection to another Mobility Management Table, the new data that a node could retrieve during the "hold" mode and its storage (intermediate buffer), etc. which can result in packet loss and also reduced scalability in the sense that devices can only use one Mobility Management Table.

The authors of [8] define a proposal for mobility handling in IoT applications using the MQTT protocol. Since MQTT is not a protocol adapted to handle mobility issues, the authors rely on Intermediate Buffering to guarantee that there is no packet loss in hand-off periods due to mobility nodes. Thus, several experiments were carried out to study the behaviour of intermediate buffers. These experiments include parameters such as access point migration, no available access point periods, the size of the messages published, the number of publisher devices, the inter-message delay, etc. The results of these experiments indicate the optimal buffer size to avoid packet loss depending on the situation to which the mobile environment is subjected. However, Although the experiments carried out deal with a wide range of situations, no mechanism is provided for the user to determine the size of the intermediate buffers in their specific situation.

SimulateIoT-Mobile, the proposal that will be described in this communication, has the aim of helping to develop this kind of IoT systems, i.e. the development of IoT systems with mobile nodes that use publish/subscribe protocols. In this sense, a literature review has been carried out to identify key concepts for managing mobility in IoT systems. Specifically, for those that use the MQTT protocol. So, SimulateIoT-Mobile includes in its implementation an MQTT mobility management model (Section 4). Thus, being able of simulate this kind of IoT systems. Developers can therefore use SimulateIoT-Mobile to model, validate, generate and simulate their IoT systems with mobility characteristics, and use the simulation results to optimise them, identifying weaknesses or errors in their designs (Sections 5–7).

3. Introduction to SimulateIoT-Mobile

SimulateIoT-Mobile is an extension of SimulateIoT [17]. For the sake of clarity, the aim of this Section is to outline the new features added as part of this extension. Thus, differentiating between what was the previous work (SimulateIoT) and what is new (SimulateIoT-Mobile).

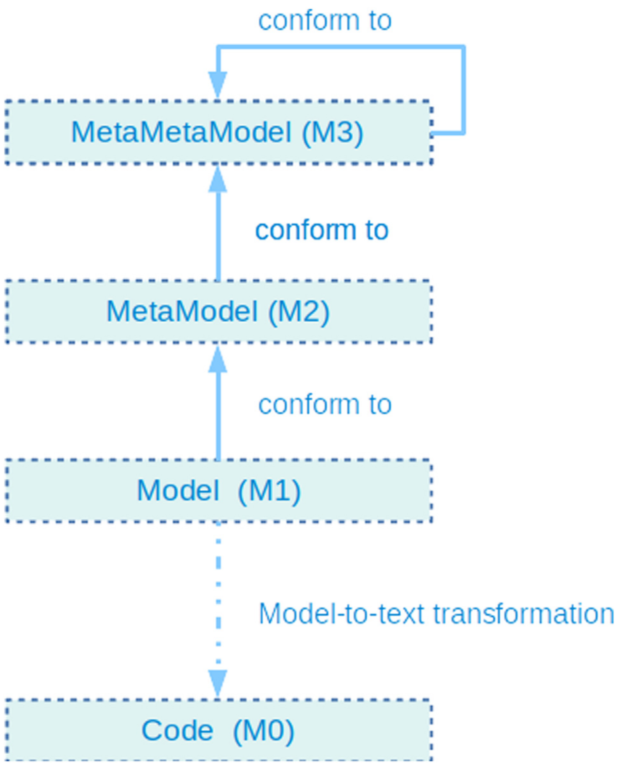


Fig. 1. The four layers of metamodeling. In SimulateIoT [17]: (a) M3 is Ecore, (b) M2 is SimulateIoT Metamodel (c) M1 is a model conform to SimulateIoT Metamodel and (d) Code is generated using the model-to-text transformations defined in SimulateIoT approach.

In this regard, SimulateIoT and therefore SimulateIoT-Mobile are based on the MDD, which is an emerging software engineering research area that aims to develop software guided by models based on Metamodeling technique. Metamodeling is defined by four model layers (see Fig. 1). Thus, a Model (M1) is conform to a MetaModel (M2). Moreover, a Metamodel conforms to a MetaMetaModel (M3) which is reflexive [29]. So, a MetaModel defines the domain concepts and relationships in a specific domain in order to model partial reality. A Model (M1) defines a concrete system conform to a Metamodel. Then, from these models it is possible to generate totally or partially the application code (M0 - code) by model-to-text transformations [30]. Thus, high level definition (models) can be mapped by model-to-text transformations to specific technologies (target technology). Consequently, the software code can be generated for a specific technological platform, improving the technological independence and decreasing error proneness.

Therefore, in order to extend SimulateIoT towards SimulateIoT-Mobile, it is required to work in these metamodelling layers. Specifically, it is required to extend: (1) The Metamodel or Abstract Syntax (M2), (2) The Graphical Concrete Syntax or the element that allows to graphically design models (M1) from the Metamodel (M2) and (3) Model-to-Text Transformations (M2T), the element that carry out the code generation (M0) from models (M1).

In this regard, as indicated in Section 1, SimulateIoT does not support mobile devices. Therefore, all the new features added to SimulateIoT (above mentioned metamodelling layers) are focused on allowing it to integrate mobile devices in its simulations (Sections 5 and 6). In addition, SimulateIoT uses the MQTT protocol and this protocol does not natively support device mobility. So, firstly, it is necessary to define, develop and integrate a MQTT mobility model (Section 4) to SimulateIoT to allow it to support mobile devices.

This mobility model aims to manage mobile devices, i.e. this model assumes that mobile devices exist. However, as aforementioned, SimulateIoT, the previous version of this work, is not able to generate or simulate IoT environments with mobile devices. Therefore, additional mobility-related concepts such as the mobile devices itself, their movement logic, the route that each mobile device will follow, or the battery consumption of these devices also have to be part of the SimulateIoT-Mobile metamodel.

Fig. 2 shows, from a high level of abstraction, the deployment of a generic simulation generated by using model-to-text transformation from a SimulateIoT-Mobile model. In Fig. 2 is possible to differentiate the main components included as extensions in this communication of the components belonging to the previous version of the simulator.

In this regard, the Edge/Mist, Fog and Cloud layers have been extended. On the one hand, the Mist/Edge layer has been extended with mobile devices. These devices can follow a (user-defined) route, connect to different fog nodes (to their brokers) during their route, so publishing their data to different brokers, receive coverage signals from different fog

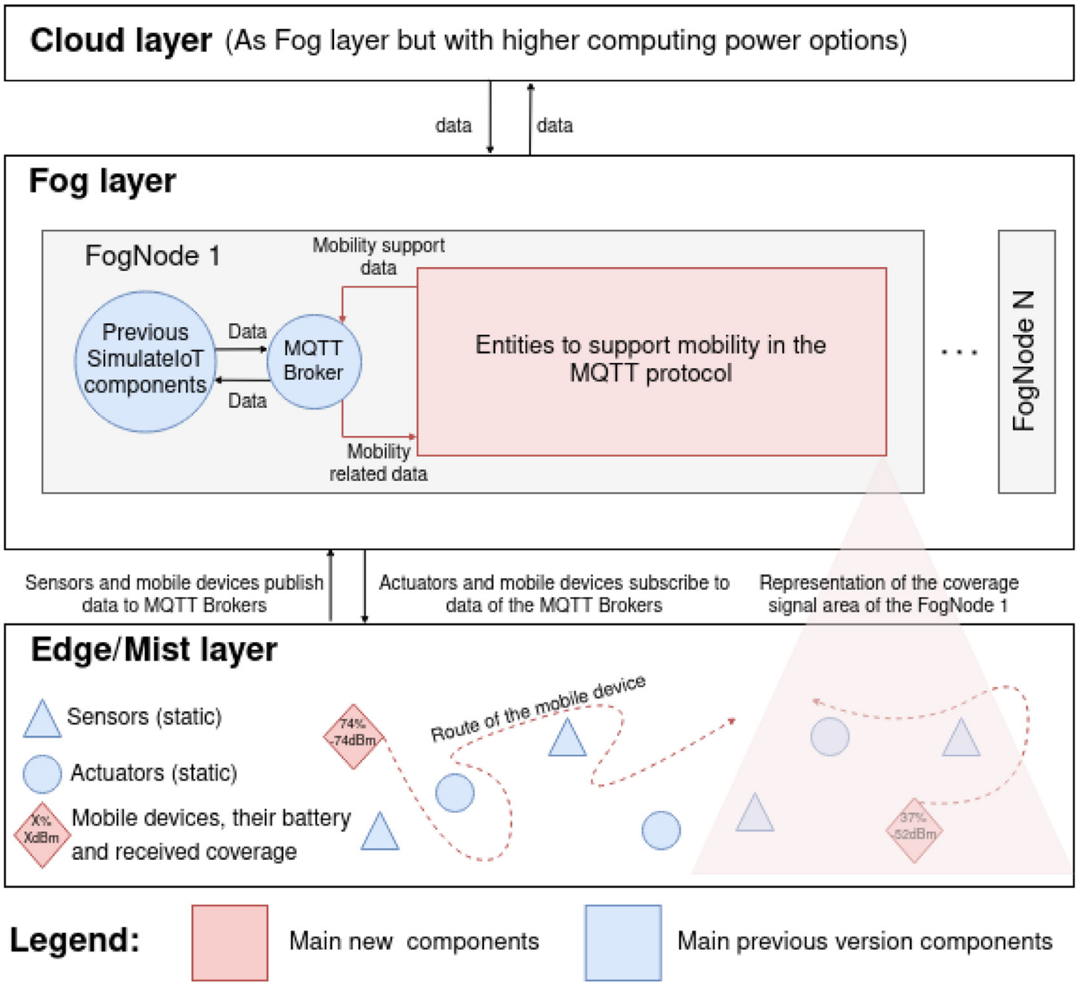

Cloud layer (As Fog layer but with higher computing power options)
data
data
Fog layer
FogNode 1
Mobility support data
Previous SimulateIoT components
Data
Data
MQTT Broker
Entities to support mobility in the MQTT protocol
Mobility related data
FogNode N
Sensors and mobile devices publish data to MQTT Brokers
Actuators and mobile devices subscribe to data of the MQTT Brokers
Representation of the coverage signal area of the FogNode 1
Edge/Mist layer
Sensors (static)
Actuators (static)
Mobile devices, their battery and received coverage
X% XdBm
74% -74dBm
Route of the mobile device
37% -52dBm
Legend:
Main new components
Main previous version components

Fig. 2. A generic model of an IoT system conforms SimulateIoT-Mobile metamodel.

nodes, carry out the simulation of battery consumption due to these new mobility-related concepts, etc. On the other hand, the Fog/Cloud layer has been extended by a set of components that aim to support mobile devices when using the MQTT protocol. These components represent the integration the mobility model (Section 4) in the IoT simulator.

All these extensions are described in detail as follows. First, the MQTT mobility model is presented in Section 4. The extension made to the Metamodel and to the Concrete Syntax is presented in Section 5. The extension made to the M2T is presented in Section 6. The new knowledge that can be obtained to optimise the real system from this extension, is described in Section 7. Besides, Section 8 presents two case of study focused on show the simulations that can be carried out with the extension presented. Finally, note that everything described in these sections is focused on showing the new contributions carried out on SimulateIoT-Mobile and it was not part of SimulateIoT.

4. MQTT mobility management model

In this section, the envisioned model to support mobility in protocols that follow the publish/subscribe paradigm is described. Specifically, the model has been designed for the MQTT protocol, one of the most widely used publish/subscribe protocols in the IoT [26].

First, in Section 4.1 a set of preliminary considerations are made based on the analysis of the protocols that traditionally support mobility and publish/subscribe protocols. Second, in Section 4.2, the necessary entities proposed to provide mobile support for the MQTT protocol are identified and described. In Section 4.3, a solution to mitigate packet loss in the mobility model is proposed. Section 4.4 describes a basic security mechanism to address the vulnerabilities of the model. Next, in Section 4.5 the deployment of the main elements required and their interactions are described. Finally, Section 4.6 presents a review of envisioned scenarios where the MQTT mobility model could be applied.

Note that this model is not intended to be a standard for mobility in publish/subscribe protocols. The MQTT mobility management model proposed is claimed to simulate IoT environments using the MQTT protocol and mobile nodes.

4.1. Preliminary considerations

When traditional protocols address mobility in the literature, they generally address issues such as the entities that carry out the support of mobility, e.g., the Home Agent in the Mobile IP protocol [31], or the Mobility Anchor Point in the HMIP protocol [32]. Also, they address the required data that needs to be shared by the nodes to support mobility, such as the Identifier (Home Address) or the Locator (Care of Address) of mobile nodes in the Mobile IP protocol [31].

On the other hand, interactions between entities are also addressed, such as the interactions needed to start direct communication with a mobile node in the Host Identity Protocol. The Correspondent Node (CN) needs to send the Host Identity Tag to the DNS to get the IP address of the Rendezvous Server. Later, after sending the first packet, the CN and the mobile node can start communication on the direct path [33].

In short, it addresses those issues that allow managing the IPs of the mobile nodes in an efficient and effective way so that changes in the IP by the mobile nodes do not affect the communication between the nodes of the network, and that this management affects as less as possible the parameters involved in the QoS (delay, etc.).

However, the main protocols used in IoT includes publish/subscribe protocols such as MQTT, AMQP or JMS [26]. In the publish/subscribe paradigm, there is no CN to start communication and no mobile node to start communication with. In publish/subscribe protocols there are Brokers that provides Topics, where devices can publish/subscribe to data. For example, in a hypothetical scenario of "intelligent temperature management" of a room, the devices that measure the temperature of the room would publish their measurements in a Topic "Temperature", to which the devices that control the temperature would subscribe.

In our proposal the Brokers are kept static (located on Fog nodes and Cloud nodes), being the devices publishing or subscribing to Topics the mobile nodes. In this regard, the mobile nodes will always be able to communicate with the Broker (static IP) and the Broker will be responsible for redirecting the data. Therefore, mobility management in publish/subscribe protocols (where the Broker remains static) differs from traditional mobility management proposals (no identifiers, locators, mappings, etc.).

4.2. Entities to support mobility in the MQTT protocol: The Broker Discovery Service and the Topic Discovery Service

Thus, focusing on a publish/subscribe protocol such as MQTT for IoT environments there are several key elements: publishers, subscribers, topics and brokers. Data are organised on Topics that are deployed by a Broker and where several elements (IoT nodes) are subscribed and where other elements (IoT nodes) have the role of data publishers. In this context, if a device needs to move through an IoT environment to carry out its own behaviour then it will be needed to connect to different Brokers in order to publish and receive data. Thus, specific entities or services to manage this issue are required. For this regard, a *Broker Discovery Service (BDS)* will be necessary.

On the other hand, publications and subscriptions are addressed to Topics. Since each Topic can be used very differently, a service is needed to analyse the Topics offered by a Broker. Thus, a mobile node can determine whether or not it is feasible to publish/subscribe to a particular Topic. Consequently, an additional service named *Topic Discovery Service (TDS)* is also needed.

Using the Broker and Topic discovery services, a device will be able to: (a) Connect to different Brokers in case it needs it; and (b) Publish/subscribe to compatible Topics that allows it to continue performing their tasks.

In this sense, the BDS allows devices to know the Brokers with which they can establish a connection (Ip). For this, each BDS is deployed together with the rest of the services on a Fog node (one BDS node per Fog/Cloud node). So, the BDS subscribes to the Topic "BDS" where it will receive requests from the mobile nodes. Later, it will publish the responses to each request on a specific Topic for each mobile node ("BDS+DeviceId"). Note that the data shared by the BDS include valuable information such as the distance measure between the device and each Broker.

However, in order to establish a suitable connection with the *Fog* node, not only should be reachable a Fog node (data obtained from BDS), but also the Topics of the Broker's Fog node should be compatible with the Topics required to publish on. To fulfil this requirement, the TDS is used.

Like the BDS, the TDS is a *Fog/Cloud* node service (one TDS node per Fog/Cloud node). This service allows IoT devices to know which Topics of a Broker are compatible with the requesting device. To carry out the above, the TDS connects to the Fog node's Broker, subscribing to a Topic ("TDS") reserved for listening to requests from the devices and publishing the responses in Topics generated dynamically for this purpose ("TDS+DeviceId"). To determine which Topics of a Broker are compatible with a device, the TDS analyses the information provided by the IoT device in its request (the Topics that the device uses and their characteristics) and compares this information with the information it has about each Broker's Topic.

4.3. Disconnection periods and packet loss

Due to the movement of devices and the topology of the designed IoT environment, periods of disconnection may occur, leading to the loss of packets. In order to handle this issue, the Intermediate Buffering technique is applied. The Intermediate Buffering consists of adding a buffer in each mobile device capable of storing the packets that should have been delivered during the disconnection periods. In this way, once the connection is reached, all buffered packets are delivered.

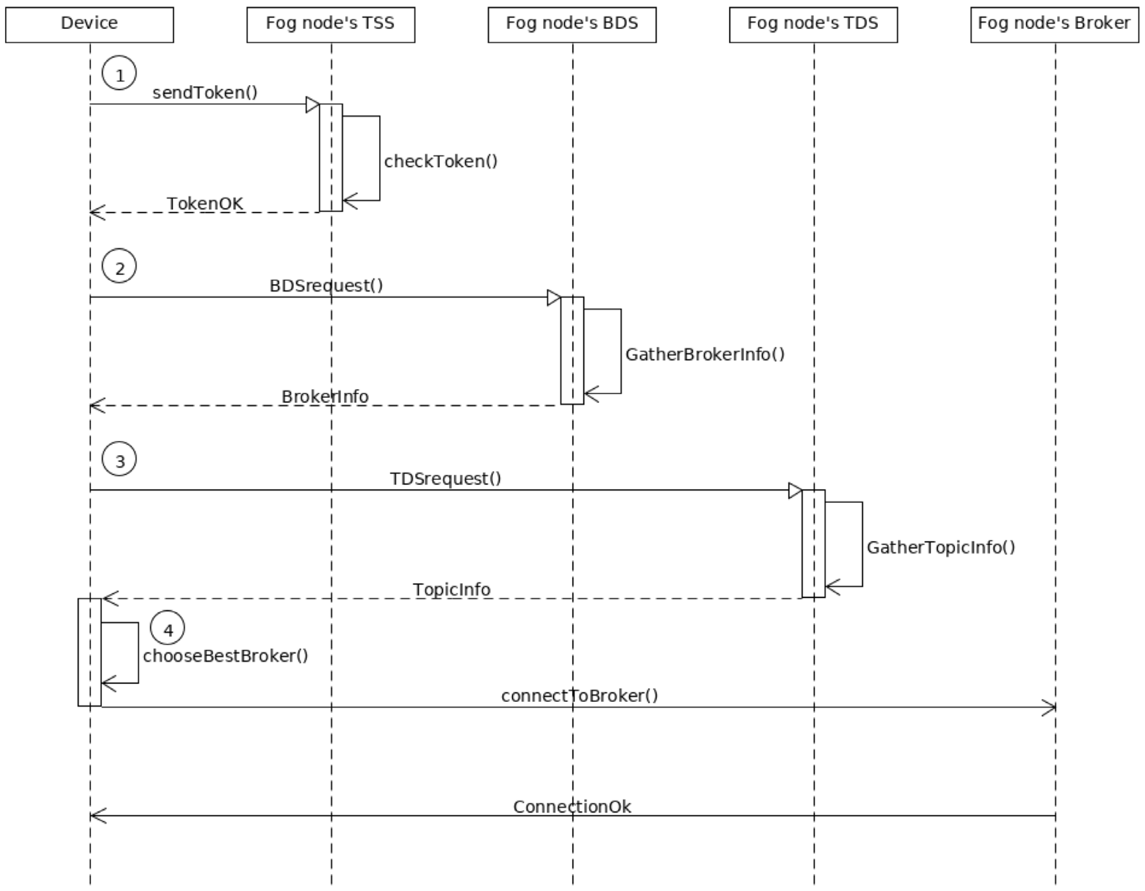


Fig. 3. Topic negotiation protocol to re-connect mobile devices with Fog nodes.

4.4. Security issues

Security is a critical issue in an IoT environment and the mobility of devices leads to the need for additional security services [34]. In this sense, token-based security is usually mandatory, as mobile devices move through the environment making different connections to different nodes and consuming different services in the environment along their way [34]. Thus, a token-based security environment, with the same philosophy as Fiware's token-based security environment [35], has been included in the proposal. Thus, this token-based security environment limits and controls the device connection and access to nodes and services in the IoT environment.

4.5. Model deployment and interactions between entities

A sequence diagram that illustrates the necessary interaction of a device with the mobility architecture (TSS, BDS and TDS) is shown in Fig. 3. It shows four key interactions: Fig. 3-(1) The Device interacts with the Fog node's Token Security System (TSS). In this first interaction, the Device sends its token to the TSS, then the TSS verifies that the token is valid and gives the device the approval to continue with its tasks; Fig. 3-(2). Once the TSS approval is received, the Device requests the BDS to obtain Broker information. The BDS then gathers Broker information and sends it to the Device; Fig. 3-(3). The third interaction is with the TDS, as the device now needs information about the Topics offered in each Broker. Thus, the Device request the TDS, the TDS then gathers information about Topics and sends it to the Device.; Fig. 3-(4). Finally, with all the Brokers and Topics information, the Device can choose the best Broker to connect and establish a connection with it.

4.6. Envisioned scenarios

This section describes several envisioned scenarios or IoT systems for which this MQTT mobility management model has been designed. Thus, being able to be simulated with SimulateIoT-Mobile, taking into account the limitations of the mobility model and the modelling expressiveness of SimulateIoT-Mobile (Section 5). Multiple scenarios could be modelled by using SimulateIoT-Mobile such as Data muling, Animal movement or Smart Cities.

Data muling and ferrying approaches have been vastly investigated. Such proposals focus on managing IoT systems where mobile devices collect data in a specific area, transporting it to a specific nodes of the system [36–39].

Animal movement can be the answer to many biological phenomena, whose understanding could be critical to successfully address challenges such as climate change, species conservation, health and food [40]. Therefore, many IoT-related studies focus their efforts on animal tracking [41–44]. In this regard, Section 8.1 describes a use case where SimulateIoT-Mobile is applied such a system.

Smart cities are IoT systems that can also include mobile nodes. In this sense, mobile nodes can be used for different purposes. The authors of [45] describe an IoT system that tracks vehicles in order to facilitate vehicle parking. The authors of [46] present results from an study where 80 riders of e-bikes discuss their experience with smart mobility. Other studies such as [47] makes a proposal to enable green mobility in cities. This work presents a device that can be integrated into citizens' personal mobility devices, such as segways or electric scooters. This device gathers environmental information to provide personal mobility devices with eco-efficiency services, integrating them in the smart city environment. A use case based on the latter study is carried out in Section 8.2.

In short, SimulateIoT-Mobile is designed to simulate several kinds of IoT systems with mobile nodes, taking into account the limitations outlined in the introduction of this subsection.

To sum up, several of the main characteristics taken into account in IoT environments with mobile devices have been identified. They, together the envisioned scenarios described in Section 4.6, facilitate describing suitably the context where IoT devices should be defined and deployed. In this sense, the next section presents the SimulateIoT-Mobile domain-specific language in order to define IoT environments with mobile devices.

5. Extensions of metamodel and concrete syntax

SimulateIoT-Mobile, as a MDD approach, is composed of three main elements: (1) Metamodel or Abstract Syntax, (2) Graphical Concrete Syntax and (3) Model-to-Text Transformations (M2T). This Section describes the SimulateIoT-Mobile Metamodel and Concrete Syntax.

5.1. *Metamodel extensions*

A Metamodel captures the concepts and relationships in a specific domain in order to model partially reality [15]. Then, it is possible to design models from this Metamodel. These models can be used to generate total or partially the application code. Thus, the software code could be generated for a specific technological platform, improving its technological independence and decreasing the error proneness.

SimulateIoT metamodel [17] defines in deep the core concepts and relationships related to the IoT domain, including elements such as sensors, actuators, edge node, fog node, cloud node, database, complex-event processing services, data definition, topics, message broker, etc. However, it has not enough expressiveness to simulate IoT systems with mobile nodes. Therefore, SimulateIoT-Mobile metamodel, an extension of SimulateIoT metamodel with enough expressiveness to define IoT systems with mobile nodes, has been developed.

For a sake of clarity, Fig. 4 shows an excerpt of the SimulateIoT-Mobile metamodel, concretely the elements required for modelling IoT mobile devices (elements which are numbered and highlighted in Fig. 4 on blue colour). Note that Fig. 14 (Appendix A) shows the complete metamodel, with the elements relating to the extension carried out highlighted in blue.

This extension includes the classes and relationships needed to model the mobility entities and services described in Section 4. Besides, some concepts necessary to capture the specific mobility domain, such as the routes that mobile nodes will follow, are also included. Finally, some concepts useful for the end-user in terms of simulation analysis (Jitter, Battery etc.) are also included.

Thus, in order to describe the SimulateIoT-Mobile metamodel, this section is divided into the domain-specific IoT mobility concepts identified: Device movement, Disconnection periods and packet loss, Jitter, Battery management, the Broker Discovery Service and the Topic Discovery Service and security issues. In this way, each of these subsections include the contributions that make it possible to model the aspects of these mobility concepts (classes and relations shown in Fig. 4).

5.1.1. *Device movement*

In order to model device mobility for simulation purposes, the *Route* concept is introduced. A *Route* is a set of coordinates that specifies the movement of one or more mobile devices. Thus, each mobile device is linked to a *Route* that specifies its movement through the IoT environment.

In this way, SimulateIoT-Mobile metamodel proposes defining several kinds of synthetic routes: *FogCloudRoute*, *LinearRoute*, *RandomRoute*, and *CSV_Route*. These *Routes* have been included as classes in the metamodel and are identified in Fig. 4 with numbers two, five, three and four respectively. Note that the class *Route* identified with the number one is an abstract class and superclass of the Route hierarchy.

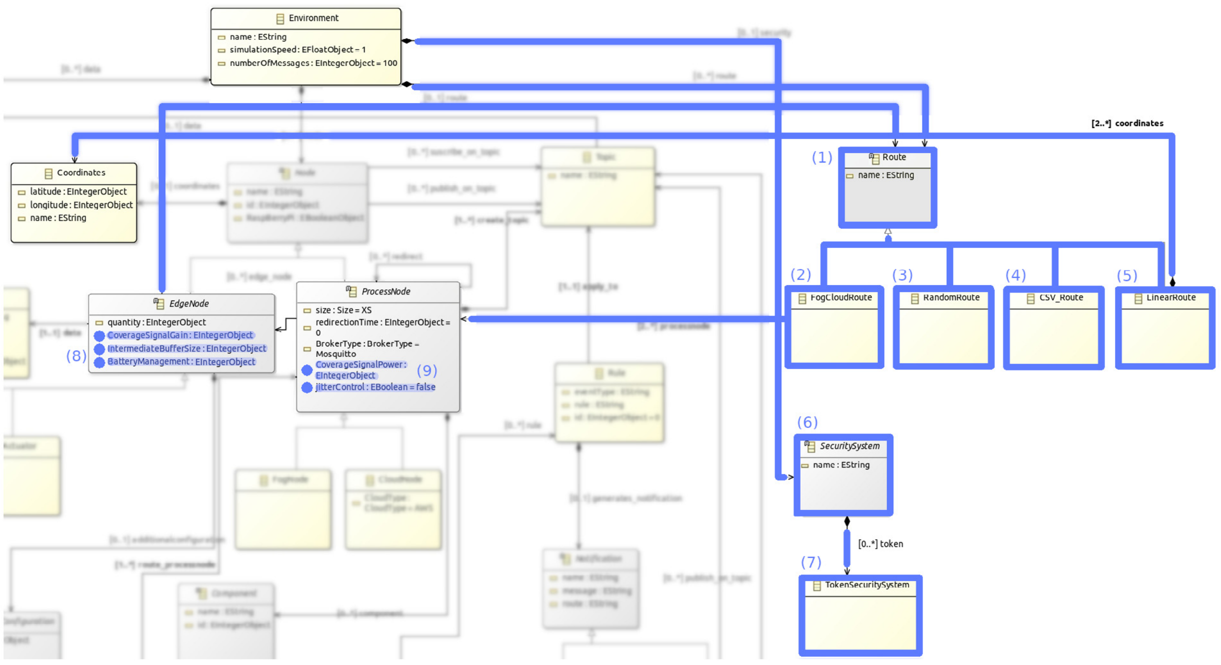

Environment
name : EString
simulationSpeed : EFloatObject = 1
numberOfMessages : EIntegerObject = 100
Coordinates
latitude : EIntegerObject
longitude : EIntegerObject
name : EString
[2..*] coordinates
(1)
Route
name : EString
(2)
FogCloudRoute
(3)
RandomRoute
(4)
CSV_Route
(5)
LinearRoute
EdgeNode
quantity : EIntegerObject
CoverageSignalGain : EIntegerObject
IntermediateBufferSize : EIntegerObject
BatteryManagement : EIntegerObject
(8)
ProcessNode
size : Size = XS
redirectionTime : EIntegerObject = 0
BrokerType : BrokerType = Mosquitto
CoverageSignalPower : EIntegerObject
jitterControl : EBoolean = false
(9)
(6)
SecuritySystem
name : EString
[0..*] token
(7)
TokenSecuritySystem

Fig. 4. Except of SimulateIoT-Mobile metamodel focusing on the mobile concepts. The complete SimulateIoT-Mobile could be found in Appendix A at Fig. 14.

- *FogCloudRoute* class allows users to define a sequence of Cloud and Fog nodes (related to *ProccessNode* class). These nodes are fixed nodes in the IoT environment that has a coordinate (position in the environment). Thus, the sequence of node coordinates defines the *Route* that will be followed by the device linked to it during the whole simulation. This *FogCloudRoute* will be followed by the device linked to it during the whole simulation, from beginning to end and backward.
- *LinearRoute* class allows the user to define routes as a sequence of x/y coordinates (related to *Coordinates* class). So, the mobile device will move throughout this sequence of coordinates indefinitely. Note that once the end of the route is reached, it follows the route in reverse.
- *Random_Route* class makes it possible to generate random routes. These routes start in a specific coordinate and are ad-hoc generated up to the end of the simulation. Note that, from each coordinate is generated the next step direction, avoiding jumps in the route.
- *CSV_Route* class allows the user to load routes defined in a CSV file. The CSV files must include an x/y coordinate in each row. In this way, the device interprets the route and follows it throughout the simulation. As with all other routes, once the end is reached, it retraces the route in reverse.

5.1.2. Disconnection periods and packet loss

As described in Section 4, in order to avoid packet loss, the possibility of using the Intermediate Buffering technique is introduced. Thus, each *EdgeNode* has been extended with the *IntermediateBuffersize* attribute (Fig. 4-(8)). In this way, the end-user is able to specify the amount of memory in terms of Kb that the Intermediate Buffer of a mobile device will have.

5.1.3. Jitter

Currently, a critical aspect in IoT is the delay between the communication of two or more components. In this sense, there are many studies that address this issue which often use simulators to corroborate their hypotheses [48–50]. Therefore, it has been considered appropriate to provide SimulateIoT-Mobile with mechanisms able to model and to measure the delay between components. In particular, when it comes to the delay caused by the mobility of devices (e.g. in a handover period).

Jitter is the variation in the delay of two messages received consecutively by a subscriber from a publisher. This way of measuring delay has been chosen because of the asynchrony of the internal clocks of the devices in a distributed system. When measuring jitter, only the subscriber clock is used. Thus, a possible asynchrony among the internal clocks of the publisher, broker or the subscriber does not affect the measurement [8].

In order to add the concept of jitter in the metamodel, a Boolean attribute called *jitter Controller* (Fig. 4-(9)) has been added to the *ProcessNode* element (nodes where the control services will be deployed). Whether it is specified as *True*, all the services required to measure jitter will be generated in the model-to-text transformations; if it is set to *False*, these services will not be generated.

5.1.4. Battery management

As well as jitter, the battery management is also currently a critical aspect in IoT. An efficient battery consumption in IoT environments is one of the challenges that researchers are currently facing [51,52]. Concerns about energy consumption are even more pronounced in mobile environments, where devices must also expend energy on the movements. Therefore, it has been considered to add the possibility to model the battery of each device so that users can analyse the behaviour of the battery after the simulation.

In this sense, note that there are a large number of IoT devices with different features, so there could be a big difference in consumption from one device to another. Therefore, in order to simulate energy consumption, a count of the tasks that consume energy is carried out. These tasks include: (a) data publishing; (b) data receiving; (c) movement; (d) data processing and storage; and (d) other interactions (e.g. neighbour discovery or security). Thus, the aforementioned parameters are used at simulation run-time to simulate battery consumption.

To model the concept of the battery usage of devices to the metamodel. To do so, an Integer attribute named *batteryManagement* (Fig. 4-(8)) has been added to the *EdgeNode* element (nodes that will be able to simulate their energy consumption). Thus, the user is able to specify the battery milliampere capacity in each device. If it is specified with a value >0, all the services required to simulate the battery consumption will be generated in the model-to-text transformations; if it is set to 0, these services will not be generated.

5.1.5. The Broker Discovery Service and the Topic Discovery Service

The BDS and TDS are two entities introduced in Section 4, designed to manage mobility in the MQTT protocol. These entities are static and their properties are not needed to be modelled by the user. Therefore, the domain-specific features of these entities have not been added to the metamodel. However, there are some concepts related to the execution of these two entities that the user should be able to model, such as the coverage of the access points to these entities and the gain of the devices to sense this coverage.

In order to extend the metamodel in this way, an attribute named *coveragesignalPower* (Fig. 4-(9)) has been added to *FogNode* and *CloudNode* elements. Thus, the end-user is able to define the signal strength of the gateways (included on the *FogNode* and *CloudNode* elements).

During a simulation, this signal strength limits the perimeter within which a mobile node may or may not connect to a gateway. It therefore plays a key role in the design of the architecture of the IoT simulation environment. Thus, users can model the IoT simulation environment and identify areas where there will be no connection, and whether in these areas there are communication problems taken into account properties such as packet loss, size of intermediate buffers, signal strength, etc.

On the other hand, to allow users to model the communications capabilities of a mobile device to detect the gateway coverage signal, an attribute named *coverageSignalGain* has been added to the *EdgeNode* element (Fig. 4-(8)). In this way, the aforementioned coverage perimeters will be variable for each mobile device, thus having different needs (e.g. the size of the intermediate buffer), bringing the simulation closer to reality.

5.1.6. Security issues

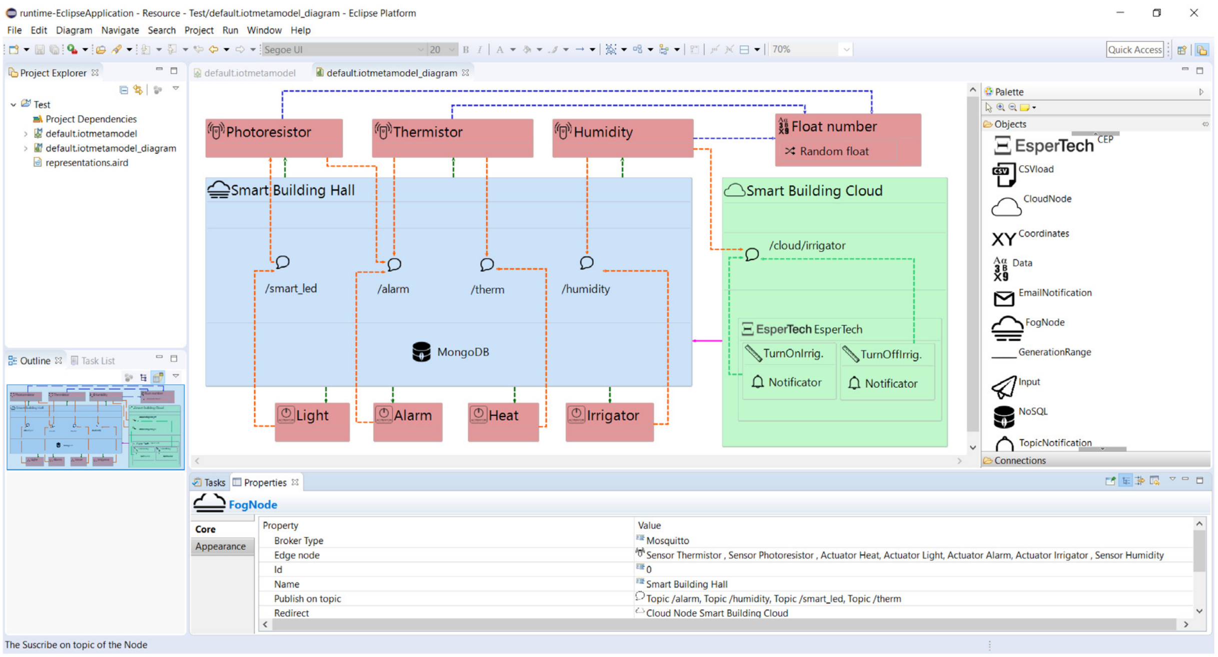
Section 4 describes a token-based security system to address vulnerabilities arising from the proposed mobility management model.

In this regard, the metamodel is extended to provide the user the possibility to choose whether or not to add this security system to the IoT environment. For this purpose, a hierarchy of elements has been added to the metamodel. The superclass of this hierarchy is named *SecuritySystem*(Fig. 4-6 SimulateIoT-Mobile metamodel). This class can contain a security service called *TokenSecuritySystem*(Fig. 4-7 SimulateIoT-Mobile metamodel). If, when modelling an IoT environment, an instance of the *TokenSecuritySystem* class is created, the model-to-text transformations will generate the security architecture necessary to implement the token-based security services discussed in Section 6.6. If it is not instantiated, these services shall not be generated.

5.2. Graphical concrete syntax and validator extensions

*M*odel-Driven Development allows creating models conforming to a metamodel. So, in order to do this, the Eugenia tool [53] makes it possible to generate a Graphical Concrete Syntax (Graphical editor). The Graphical Concrete Syntax generated for SimulateIoT-Mobile metamodel is an extension of the Graphical Concrete Syntax defined in SimulateIoT, which is based on Eclipse GMF (Graphical Modeling Framework) and EMF (Eclipse Modeling Tools). Consequently, models (EMF and OCL (Object Constraint Language) [54] based) can be validated against the defined metamodel (EMF and OCL based). Note that OCL is a standard to define model constraints. Fig. 5 shows an excerpt from this graphical editor. It helps users to improve their productivity allowing not only defining models conforming to the *SimulateIoT-Mobile metamodel* but also their validation using this metamodel and OCL constraints [54].

The graphical concrete syntax (based on an Eclipse plugin) developed offers a suitable way to model the IoT environment by using the high-level concepts defined in the SimulateIoTModel metamodel (Fig. 4). Later on, the graphical concrete syntax will be used to model and validate several case studies.



Property	Value
Broker Type	Mosquitto
Edge node	Sensor Thermistor , Sensor Photoresistor , Actuator Heat, Actuator Light, Actuator Alarm, Actuator Irrigator , Sensor Humidity
Id	0
Name	Smart Building Hall
Publish on topic	Topic /alarm, Topic /humidity, Topic /smart_led, Topic /therm
Redirect	Cloud Node Smart Building Cloud

Fig. 5. Graphical editor based on the Eclipse to model conforming to the SimulateIoT-Mobile metamodel.

6. Extensions of model-to-text transformations

As aforementioned, SimulateIoT-Mobile, as a MDD approach, is composed of three main elements: (1) Metamodel or Abstract Syntax, (2) Graphical Concrete Syntax and (3) Model-to-Text Transformations (M2T). In Section 5, the extensions carried out in (1) Metamodel or Abstract Syntax (Section 5.1) and (2) Graphical Concrete Syntax (Section 5.2) were described. Thus, in this section, the extensions for SimulateIoT-Mobile carried out in (3) Model-to-Text Transformations (M2T) are described.

Once the models have been defined and validated conforming to the SimulateIoT-Mobile metamodel (examples of models in the Figs. 8 and 11), a model-to-text transformation defined using Acceleo [55] can generate the IoT environment modelled.

Thus, this section describes the main features of the Model-To-Text transformation carried out in order to generate the IoT environment, focusing in the transformations which allow mobile support (the target of this work). For the sake of clarity, this section is divided into the domain-specific IoT mobility concepts identified (as in Section 5). In this way, each subsection contains the contributions that make it possible to generate the code of each component (M2T transformations) related to these mobility concepts. Finally, a section describing the overall generation and integration of the artefacts is included.

6.1. Device movement

Section 5.1.1 describes the extensions carried out to make it possible to model the movement of mobile devices. In this sense, *Route* is an abstract class that can be specified by different elements: (a) CSV file (*CSV_Route* class), (b) Fog/Cloud nodes (*FogCloudRoute* class), (c) Predefined Coordinates (*LinearRoute* class), and (d) Random Coordinates (*Random_Route* class). In order to manage the *Route* elements and their specifications, the following services are required:

- Mapping services, to map the routes defined in a CSV file, list of Fog/Cloud nodes or Coordinates to a suitable format for the devices.
- A coordinate generation service, to generate realistic random routes in real-time (this service takes care that the direction of the route is consistent, that there are no incoherent movements from one position to another, etc.).
- A route management service, in order to make the mobile devices capable of interpret the routes and move along them during the simulation.

Therefore, Simulate-IoT model-to-text transformations have been extended to generate and integrate these three services on every mobile device in the environment (i.e. on every mobile device modelled by the user in a model).

6.2. Disconnection periods and packet loss

Section 5.1.2 describes the extensions carried out to make it possible to model the application of the *Intermediate Buffering technique* for mobile devices. Thus, in order to implement and apply the Intermediate Buffering technique, it is necessary to include two new services to the mobile devices, a buffer storage service and a buffer publish service.

- Buffer storage service: This service have the capacity (Kb) modelled through the EdgeNode element *Intermediate-Buffersize* attribute (Fig. 4-8). Thus, this service controls the size of the messages that are stored in the buffer and that the memory does not overflow. Whether the buffer is full and the device is still offline, this service acts as a queue, eliminating the oldest messages (packet loss) so that the new ones can be stored, always taking into account the size of each message.
- Buffer publish service: Once the device decides to connect to a gateway, the buffer publish service (integrated with the device's publishing logic) reads and empties the buffer, subsequently publishing all this data.

Therefore, Simulate-IoT M2T transformations have been extended to generate and integrate these two services on every mobile device in the environment (i.e. on every mobile device modelled by the user in a hypothetical model).

6.3. Jitter

Section 5.1.3 describes the extensions needed to make it possible to model whether to deploy the *Jitter* analysis service or not. Thus, in order to generate and deploy the Jitter analysis service, it is necessary to include this service in the Cloud and Fog nodes.

At simulation start, the jitter analysis service is deployed to monitor jitter next to each Broker (deployed at each Fog/Cloud node). Thus, the jitter analysis service subscribes to all Topics, receiving all the messages published in them and registering the reception timestamp of each message. At the end of the simulation, this service calculates the jitter of the messages received by the devices. It should be noted that the data published by each device is structured in JSON format and contains a field reserved for identifying the publisher and the timestamp of each published message [17].

At simulation ends, the jitter control service generates an output with the jitter experienced during the whole simulation so, the average jitter, the maximum jitter and the minimum jitter.

Note that the following expression is used to determine the jitter:

$$Jitter = m'_n - m'_{n-1} - T$$

This expression considers the reception of two messages. The arrival time for message n is defined as m'_n. Note that T is a fixed parameter representing the publishing period of the publisher.

As an instance of the above, consider a situation where a hypothetical sensor has a period T equal to 500 ms, assuming that a message $m'_n - 1$ from the sensor is received by an actuator at instant 0 and the next message m'_n from this sensor is received 621 ms later, the Jitter between these two messages is: $621 - 0 - 500 = 121$ ms.

Thus, Simulate-IoT M2T transformations have been extended in this sense to generate and integrate this service on every Fog or Cloud node modelled in the environment (i.e. on every Fog or Cloud node modelled by the user in a model, whose attribute *jitter_Controller* is setted as True).

6.4. Battery management

Section 5.1.4 describes the extensions carried out to make it possible to model whether to include the *Battery consumption* simulation or not. Thus, in order to generate and deploy the *Battery consumption* simulation, it is necessary to include this simulation module in the Cloud and Fog nodes.

Therefore, the battery simulation is based on the integration of several counters throughout the devices code generated, thus counting each of the tasks carried out by a device. These tasks include: (a) Data publishing, (b) Data receiving, (c) Movement, (d) Data processing and storage, (d) Other interactions (e.g. neighbour discovery or security interactions).

All these parameters are used at simulation run-time to simulate the battery consumption of each device. In addition, once the simulation is finished, the battery simulation service of each device outputs a log with the results of these counters. Thus, the user can then use these parameters to predict more accurately the battery consumption of a specific real device.

Therefore, Simulate-IoT M2T transformations have been extended in this sense to generate and integrate all the aforementioned counters in the device code, thus simulating the battery consumption of each modelled device (i.e. on every device modelled by the user in a hypothetical model).

6.5. Broker Discovery Service and Topic Discovery Service

The BDS and TDS are nodes deployed on each Fog/Cloud node of the system. These nodes are designed to support mobility in IoT environments where the MQTT protocol is used. The behaviour of the BDS nodes and TDS nodes is also described in Section 4.2. However, this section aims to identify and describe individually each of the services generated from the model-to-text transformation for the BDS nodes and the TDS nodes.

The BDS nodes are entities that communicate to mobile devices useful information about the Brokers deployed in the system. In this way, mobile devices can use this information to make an appropriate selection of which Broker to publish to or subscribe to.

In this sense, the device communicates to the BDS (to those within their reach) information about its geographical location. Using this data, the BDS nodes reply to the device with a list of Brokers and details about each one of them, such as their geographical location, IP address or the distance to them in a straight line. Therefore, three services are identified:

- MQTT client: The first service identified is the MQTT client that uses the BDS and the underlying logic to communicate with the target device.
- DataBase client: Secondly, it is identified the client of the database where the BDS queries all the data related to the Brokers.
- To Measure of distance between device and Brokers: Thirdly, it is identified the component that applies the logic necessary to interpret the coordinates of the mobile devices and calculate the distance between it and the Brokers deployed in the system.

On the other hand, the TDS nodes are entities that communicates to mobile devices useful information about the Topics deployed in the system's Brokers. In this regard, the device requests from the TDS nodes data about the Topics deployed in one or several Brokers. Using this list of Brokers, the TDS nodes reply to the device with a list of Topics for each Broker, including information about each of the Topics such as a set of Tags (describing the Topic), its name, etc. For this, two services are identified:

- MQTT client: The first service identified is the MQTT client that uses the TDS and the underlying logic to communicate with the device in question.
- DataBase client: Secondly, the client of the database where the TDS consults all the data related to the Topics of each Broker is identified.

In addition to BDS, TDS code generated, the compilation of its code, the wrapping of it in a Docker, and its deployment and integration with the rest of the system, must also be generated. In this sense, SimulateIoT-Mobile takes these issues into account in the deployment script of the system.

To summarise, Simulate IoT model-to-text transformations have been extended to generate and integrate the BDS and the TDS and each of their services in each Fog or Cloud node of the environment (i.e. on every Fog or Cloud node modelled by the user in a model).

6.6. Security issues

Section 5.1.6 describes the extensions carried out to make possible the modelling of the *Security* services. Thus, in order to secure mobile IoT environments and simulate the impact on the overall performance of the environment, a token-based security system is included in SimulateIoT-Mobile, the *TokenSecuritySystem* (TSS). When the simulation starts, all devices generated from the metamodel share a security token. This token is used by devices when they publish or subscribe to a Topic, so if the Topic is named *temperature*, the device publishes or subscribes to */{token}/temperature*. In this way, if an external device tries to connect to the IoT environment, as it is not in possession of the security token, it will not be able to obtain the data published in any Topic, and will not be able to publish false information in any Topic.

As for the TSS, it is a Fog node's service and it is responsible for managing the tokens. In this way it gives them a random lifespan, generates new tokens when they expire, communicates the new token to the devices, etc.

Therefore, SimulateIoT model-to-text transformations have been extended in this sense to generate and integrate this TSS in all Fog or Cloud nodes of the environment (i.e. on every Fog or Cloud nodes modelled by the user in a model).

6.7. IoT environment generated from M2T transformations

For a better understanding of the extensions carried out in this work and their relationships or interactions, this section describes the overall architecture generated from the M2T transformations from SimulateIoT-Mobile models. To explain the generated architecture, it is divided into the three layers that can constitute an IoT environment defined with SimulateIoT-Mobile: (A) Edge Layer; (B) Fog Layer; (C) Cloud Layer.

(A) Edge Layer

The Edge layer is composed of the set of sensors and actuators of the IoT environment. The architecture of an *Edge* node is illustrated in Fig. 6. In terms of the main elements of the architecture (numbered with the numbers used in Fig. 6):

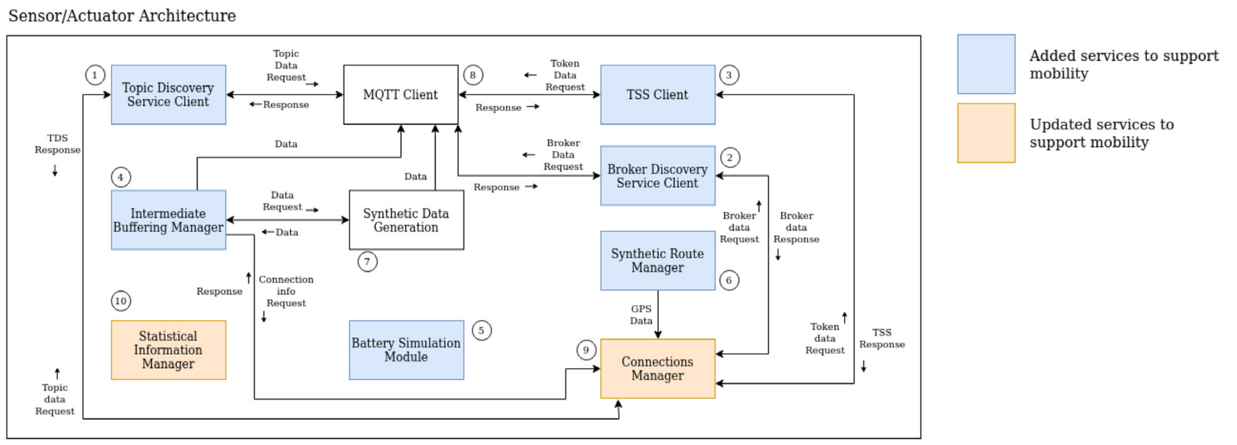


Fig. 6. Software architecture of a Edge node generated.

1. *Topic Discovery Service Client*: Embedded client in the Edge nodes that allows Edge nodes to interact with the Topic Discovery Service offered by the Cloud or Fog nodes. The communication is done through the MQTT protocol (MQTT client relationship) and, once the response is received from the Fog/Cloud node, it is transferred to the Connection Manager component.
2. *Broker Discovery Service Client*: Embedded client in the Edge nodes to interact with the Broker Discovery Service offered by the Cloud or Fog nodes. The communication is done through the MQTT protocol (MQTT client relationship) and, once the response is received from the Fog/Cloud node, it is transferred to the Connection Manager component.
3. *Token Security System Client*: Embedded client in the Edge nodes that allows Edge nodes to interact with the Token Security System offered by the Cloud or Fog nodes. The communication is done through the MQTT protocol (MQTT client relationship) and, once the response is received from the Fog/Cloud node, it is transferred to the Connection Manager component.
4. *Intermediate Buffering Manager*: Intermediate Buffer included in *Edge* nodes to avoid packet loss. This element is related to: (a) The Connections Manager which informs when the connection is *on* or *off*, in order to start or stop storing data. (b) The Synthetic Data Generation element, in order to know which data to store; (c) The MQTT client, to publish the stored data when the connection is *on*.
5. *Battery Simulation Module*: Battery simulation module embedded in the *Edge* nodes to simulate the energy consumption.
6. *Synthetic Route Manager*: It manages, generates or loads routes that Edge devices should follow. This component is linked to the Connections Manager module by sending it the device location. Thus, the Connection Manager module can use the device location to optimise the establishment of new connections.
7. *Synthetic Data Generation*: It manages, generates or uploads the publication of data made by an *Edge* device. It is linked to the MQTT client, thus being able to publish the generated data. It is also related to the Intermediate Buffer so that, in case of disconnection, it stores the generated data.
8. *MQTT Client*: It allows an Edge device to publish or subscribe to Topics on an MQTT. As can be observed in Fig. 6, several components on the Edge node require to publish or subscribe to Topics by using the MQTT Client.
9. *Connections Manager*: It manages the connections among an *Edge* node with the rest of the nodes in the IoT environment. It is related to the Topic Discovery Service, Broker Discovery Service, Token Security System and the Synthetic Route Generation element with the aim of coordinating them when making requests, thus being able to use the responses obtained from each of them to establish optimal connections.
10. *Statistical Information Manager*: It collects data from the device during the simulation in order to integrate them and to produce statistics to be analysed at the end of the simulation for the analysis of the simulation.

(B) Fog Layer

The Fog layer is composed of the set of *Fog* nodes of the IoT environment. The architecture of a *Fog* Node is illustrated in Fig. 7. In terms of the main elements of the architecture (numbered with the numbers used in Fig. 7):

1. *Topic Discovery Service*: Component that implements the Topic Discovery Service explained in Section 6.2. This service is linked to the MQTT client in order to receive requests. In addition, it relates to the MongoDB Client to obtain information about the *Fog* node Topics (response to requests from devices).
2. *Broker Discovery Service*: Component that implements the Broker Discovery Service explained in Section 6.2. This service is linked to the MQTT client in order to receive requests from the Edge nodes.

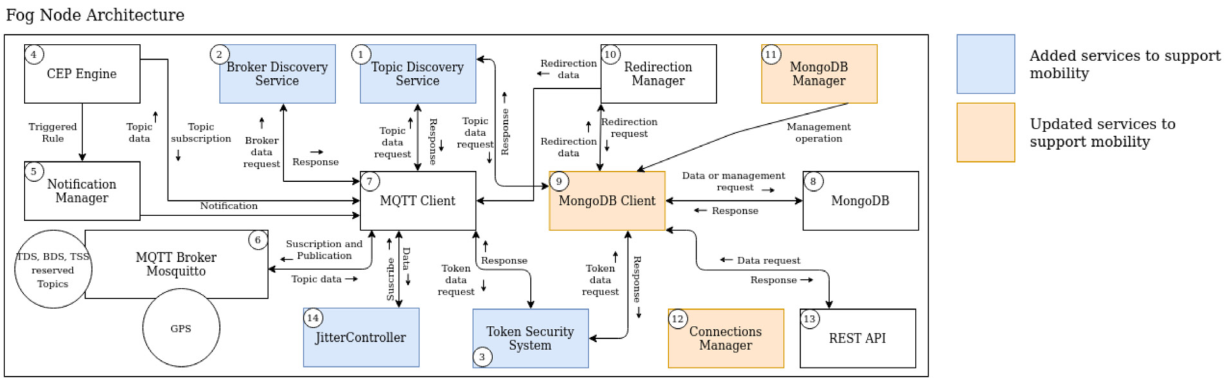


Fig. 7. Software architecture of a Fog node generated.

3. *Token Security System*: Component that implements the Topic Discovery Service explained in Section 6.2. This service is linked to the MQTT client in order to receive requests from the Edge nodes. In addition, it relates to the MongoDB Client to manage Tokens.
4. *Complex Event Processing Engine*: CEP engine that analyses and applies user-defined rules (modelled previously) to data published in the Topics (it is related to MQTT Client). Besides is linked to the Notification Manager element to which it sends its output.
5. *Notification Manager*: Component that collects the analyses carried out by the CEP engine (related to CEP Engine) and publishes them in the Topics that the user has defined for this purpose during modelling phase (relation with MQTT client).
6. *MQTT Broker Mosquitto*: MQTT Broker that supports communication by the MQTT protocol. It is related to the MQTT client of the Fog node to allow it to communicate by using the MQTT protocol.
7. *MQTT Client*: It allows the Fog node to connect to its MQTT Broker and publish or subscribe to its Topics.
8. *MongoDB*: No-Sql database used for data storage on a Fog node. It is related to the MongoDB client as it is the client that performs the queries.
9. *MongoDB Client*: MongoDB client that allows the Fog node to interact with the MongoDB database (related to MongoDB).
10. *Redirection Manager*: Component that allows redirecting data (related to MongoDB Client) among Fog nodes and Cloud nodes.
11. *MongoDB Manager*: Component that includes the necessary interactions with MongoDB (relation with MongoDB Client) in order to ensure the correct performance of the Fog node.
12. *Connections Manager*: Module that manages the connections of a *Fog* node with the rest of the nodes in the IoT environment.
13. *REST API*: REST API that provides information about the *Fog* node to external components. So, internal aspects of the *Fog* node could be requested, for instance, data stored on MongoDB.
14. *JitterController*: Component that measures the jitter produced in the exchange of messages between the different devices in the IoT environment. It has a relationship with the MQTTClient as it needs to subscribe to all Topics in the environment in order to receive the messages published and thus measure the jitter of them.

To sum up, each *Fog* node exposes several interfaces based on different protocols: (a) REST API publish a REST API on port 4000 based on request–response communication schema; and (b) MQTT Broker Mosquito exposes port 18XX that could be used to interchange messages using MQTT protocol based on the well-know publish–subscribe communication schema; (c) The MongoDB database which listens for requests on port 27017 from which queries can be made.

(C) Cloud Layer

The Cloud layer is composed of the set of Cloud nodes of the IoT environment. The architecture of a *Cloud* node is the same as that of a *Fog* node, differing from it only in computational performance, where *Cloud* performance and store capabilities are greater than Fog capabilities. This architecture is illustrated in Fig. 7.

7. Simulation outputs and analysis that can be obtained from the extensions

The main motivation for simulating an IoT system is to gain knowledge to optimise it. Therefore, the benefit that can be derived from an IoT simulator is determined by its outputs. In this regard, SimulateIoT-Mobile provides several outputs that allow to perform several analyses from which to gain knowledge. The main analyses that can be carried out with SimulateIoT-Mobile are the following:

- Whether all messages have been successfully sent from sensors to gateways (*ProcessNode* elements). Data obtained comparing sensors logs and MongoDB storage.
- How many mobile devices have reached the maximum local storage (Intermediate Buffer) due to they do not found a gateway to send data during their routes. Data obtained from each IoT mobile device log.
- Check packet loss rate. Data obtained from each IoT mobile device log.
- Check the jitter produced in the environment during the exchange of messages. Data obtained from the jitter controller component.
- To check the state of the battery of the IoT mobile devices simulated. Data obtained from each IoT mobile device log.
- To check if the gateways deployed (*FogNode* elements) have been enough to attend the IoT mobile devices. Data obtained from each IoT mobile device log and the *FogNode* elements database and logs.
- To check if the complex event processing rules defined have been executed suitable and the *Actuator* elements have executed their actions. Data obtained from each complex event processing event engine log and the message sent to *Actuator* elements.
- Visualise the data interchanged among the IoT mobile devices and the *FogNode* or *CloudNode* elements. Data can be visualised using the view tool *Compass* associated with each *FogNode* or *CloudNode* element.
- To check if there are message bottlenecks on specific *ProcessNode*. It implies that a specific IoT node is a sink of messages which is a potential system risk and a situation that should be avoided. This situation requires to analyse what has been the percentage of messages that cross each *ProcessNode* identifying those which they have a high message rate. Data obtained from different sources, such as the jitter produced at certain times, packet loss rate, node downtime, etc.
- To check if the resources available on the *EdgeNode, FogNode or CloudNode* are enough to deploy suitable the IoT system modelled. Data obtained from different sources, such as the jitter produced at certain times, packet loss rate, node downtime, etc.
- To obtain several statistics related with the number of connections carried out by IoT mobile devices with the gateways deployed (*FogNode* elements). Data obtained from each IoT mobile device log.

8. Case studies

Next, two case studies have been developed using the SimulateIoT-Mobile metamodel and M2T transformations previously presented. The first one defines an IoT simulation of Animals tracking while the second one defines an IoT simulation of Personal mobility devices (PMD) based on public bicycles .

Below is defined a synthesis of the methodology required to use *SimulateIoT-Mobile* and the processes carried out by this tool to simulate these use cases in order to illustrate them more effectively.

1. *Model definition:* This step refers to the modelling of the *IoT Environment* that the user wants to deploy and simulate. This model corresponds to the *DSL* and therefore can contain all the elements defined in it.
2. *M2T transformations and deployment:* Once the model has been defined, the source code of all the elements involved can be generated from it. *Sensors*, *Actuators*, *FogNodes*, *CloudNodes* and all their sub-components and configuration files will be ready for the deployment phase.

8.1. Case 1: Animal tracking

Animal movement can be the answer to many biological phenomena, whose understanding could be critical to successfully address challenges such as climate change, species conservation, health and food [40]. Therefore, many IoT-related studies focus their efforts on optimising the application of these systems in such environments. Moreover, many of these studies corroborate and justify their results through the use of simulations [41–44].

For all these reasons, this first use case is focused on the simulation of an IoT system based on animal movement tracking. So, modelling the behaviour of a system based on GPS devices on animals with MQTT communications *ProcessNode* elements facilitates the animal tracking making it possible to analyse data.

In order to model this IoT system the following aspects are taken into account:

- Each animal has its own GPS devices which communicate with the gateways deployed on the area. So, several *Sensor* elements with mobility capabilities should be included in the model.
- Each *Sensor* element has defined the route that they should follow, this route is a *FogCloudRoute* (Section 5.1.1) that is shared, simulating a flock.
- There are defined several *ProcessNode*, specifically three *FogNode* elements which could be deployed on strategic locations on the area, such as the lagoons where periodically animals should access to drink water.
- Each *Sensor* element (GPS devices) send storage data to the gateway represented by the *FogNode* elements deployed.
- Each *FogNode* element defined, that is the gateways deployed, notifies to a central *CloudNode*.

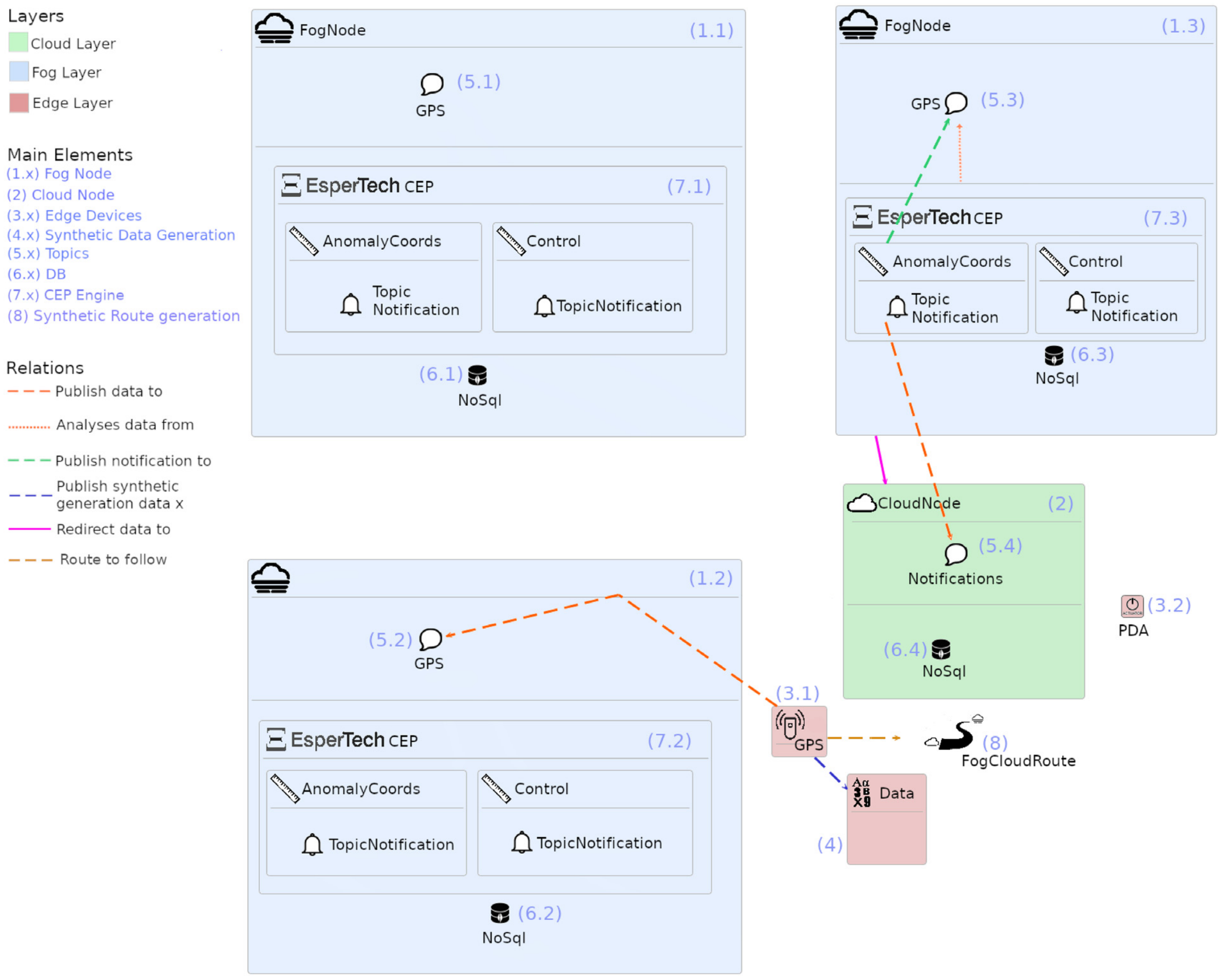


Fig. 8. Case 1. Model simplified conforms to SimulateIoT-Mobile metamodel for animals tracking. Complete version in Fig. 15.

8.1.1. Model definition

Fig. 8 shows an excerpt from the animals tracking model. It also includes numerical references for each node which are then used to describe the use case. Note that, for the sake of clarity this extract is simplified, including only one instance of each possible type of relationship between components. The complete version of this model is shown in Fig. 15.

For the purpose of explaining the model, it is divided into three parts: (1) Edge Layer (Red nodes), (2) Fog Layer (Blue nodes), (3) Cloud Layer (Green nodes).

(1) Edge Layer

The Edge layer contains the definition of the sensors (Fig. 8 label 3.1) and actuators (Fig. 8 label 3.2) of the simulation. This sensor represents the GPS that has been incorporated into each animal. This GPS sensor monitors the different locations of a animal throughout the day. On the other hand, the *PDA* actuator (Personal Digital Assistant) has been modelled bearing in mind that there may be use cases where workers are in charge of keeping the integrity of the animals safe, being the PDA the device where they receive notifications of danger. For instance, receiving notifications when an animal is not in the area where it should be, such as outside of a hypothetical protected area where it might be at risk. In addition to this PDA, notification could be also defined to send a message to user applications such as email.

GPS data is assigned by a synthetic data generation (Fig. 8 label 4) and a *Route* (Fig. 8 label 8). Regarding the publication of the data, GPS could publish data in the Topic called *GPS* (Fig. 8 labels 5.1, 5.2, 5.3) located in the Fog nodes. On the other hand, the PDA actuator (Fig. 8 label 3.2) subscribes to the Topic *Notifications* (Fig. 8 label 5.4) located in the Cloud node (Fig. 8 label 2).

Note that for simulation purposes it is not necessary to re-model all elements of the above for each animal. Since each GPS has a *quantity* attribute to specify how many times it should be generated in the M2T transformation phase. Nevertheless, *Route* and the synthetic generation of data should be defined for each animal, otherwise, it will be the same for each of them.

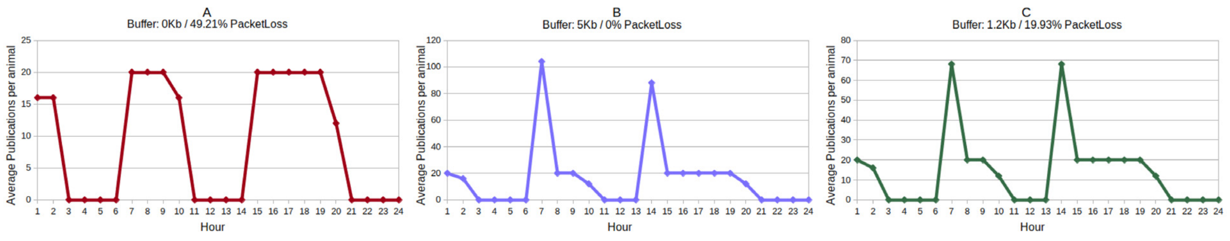


Fig. 9. Case01. Simulation analysis: Intermediate buffer size and Packet loss rate.

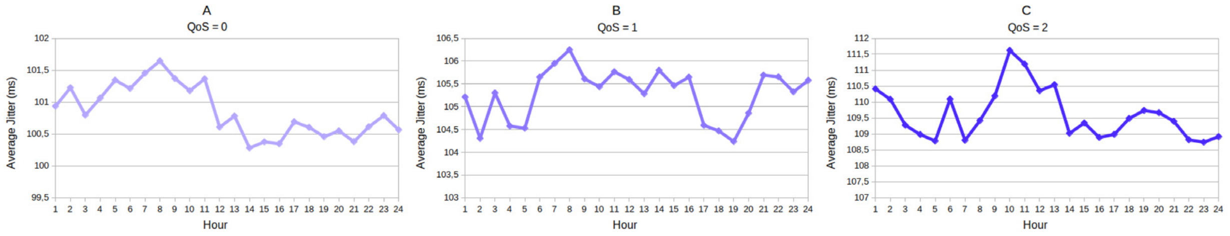


Fig. 10. Case 1. Simulation analysis: Jitter variation during simulation according to the selected QoS (MQTT protocol).

(2) Fog Layer

Fog nodes ((Fig. 8 labels 1.1, 1.2, 1.3) are those that integrate the necessary services for the Edge nodes to carry out their duties. Taking into account the example modelled, each Fog node could be located near watering places where the animals live. For this case study, three Fog nodes have been defined.

The modelling of the Fog nodes is divided into Topics (Fig. 8 labels 5.1, 5.2, 5.3) and the CEP engine (Fig. 8 labels 7.1, 7.2, 7.3). In this use case study, each Fog node offers one Topic, *GPS*, where the GPS incorporated in each animal publishes its location during the day. On the other hand, the CEP engine analyses the data published in these Topics and applies a set of rules to detect anomalies. Specifically, the CEP engine defines two rules: a) *AnomalyCoords* rule, which analyses the data published in the Topic *GPS* and identifies if the location of an animal is inappropriate; (b) Control rule, which analyses the data published in the Topic *GPS* and identifies if an animal does not publish its location for too long a period of time. If one of these rules is met, the CEP engine publishes a notification in the Topic *Notifications* (Fig. 8 label 5.4), located in the Cloud node (Fig. 8 label 2), where the PDA actuator (Fig. 8 label 3.2) is subscribed.

Finally, the Fog nodes are related to the Cloud node. This relationship allows Fog nodes to forward all the data received by their Topics to the Cloud node for storage and future analysis. Note that Fog nodes can also store data if they include a database (Fig. 8 labels 6.1, 6.2, 6.3)

(3) Cloud Layer

As for the Cloud node (Fig. 8 label 2), in this use case it is necessary to model the relationship with the Fog nodes (Fig. 8 labels 1.1, 1.2, 1.3). Thus, it is specified that the Cloud node will receive all the data published in their Topics.

On the other hand, the notifications of the CEP engines incorporated in each Fog node (Fig. 8 labels 7.1, 7.2, 7.3) are sent directly to the Cloud node via MQTT, therefore, it is necessary to define a Topic in the Cloud node. This Topic is *Notifications* (Fig. 8 label 5.4) and is where the PDA actuator is subscribed (Fig. 8 label 3.2), thus receiving any anomaly regarding the animals.

Finally, it is also necessary to model a database to store the received data (Fig. 8 label 6.4).

8.1.2. Model-to-text transformation and deployment

Once the model has been defined, the model-to-text transformation is applied with the following goals: (i) to generate Java, Python, NodeJs, etc. code that wraps each device behaviour; (ii) to generate configuration code to deploy all the generated services, such as the message brokers necessary, including the *topic* configurations defined, the gateway configurations, etc. (iii) to generate the code and deployment configuration files of the architecture that supports mobility (Broker Discovery Service, Topic Discovery Service, Token Security System, etc.). (iv) to generate the configuration files and scripts necessary to deploy the databases and stream processors defined; and finally, to generate the code necessary to query the databases where the data will be stored; (v) to generate for each *ProcessNode* and *EdgeNode* a *Docker* container which can be deployed throughout a network of nodes using *Docker Swarm*.

Consequently, each Edge node, Fog node and Cloud node is generated following the software architecture defined in Section 6 where model-to-text transformation has been defined.

8.1.3. Simulation analysis

SimulateIoT-Mobile allows users to iteratively model, simulate (execute) and analyse their environment as many times as necessary until the final version is achieved. So, having executed a simulation the users can analyse several data (Section 7).

To exemplify the above mentioned, some experiments and analysis have been applied below on Case 01, Animals Tracking. In this sense, the optimal size of the Intermediate Buffer in different situations, the battery behaviour of the devices and the jitter produced in the message exchange are studied.

First, the packet loss rate is analysed. To carry out this analysis, in a first approximation of the model, it is specified that the GPS incorporated in each animal publishes its data every three minutes. In addition, no Intermediate Buffer has been included. The results after one day simulation (2 min real time - simulation accelerated) are 45.82% packet loss on average per animal (Fig. 9-A). In order to reduce this packet loss rate, an Intermediate Buffer of 5 Kb (250 publications) is added to the GPS of each animal. The results of this second approach are 0% packet loss rate (Fig. 9-B). Finally, a series of tests are carried out to optimise the buffer size and keep the packet loss rate below 20% (hypothetical acceptable threshold). The test results indicate that a buffer size of 1.2 Kb would be necessary to keep the packet loss rate below 20% (Fig. 9-C).

As for the battery, different valuable data can be extracted about its consumption. For example, in this use case when the buffer size is set to 1.2 Kb, (around 20% packet loss) during one day each GPS was connected to the internet for an average of 10.88 h, published a total of 340 messages on average, made 3 connections and disconnections of gateways, etc.

On the other hand, it is also possible to analyse the jitter that occurs during the exchange of messages between devices. Jitter can be measured from different perspectives, in this case, the jitter is measured during a normal exchange, ignoring the increase produced by a handover period (gateway switch) or a disconnection period. In this sense, the results obtained are an average jitter of 100.859 ms, a maximum of 102.831 ms and a minimum of 100.116 ms. Fig. 10-A shows the average jitter of each simulated hour when QoS is set to 0 (this case).

One of the factors involved in the Jitter results is the quality of service offered. In this sense, MQTT has three QoS levels. The above tests have been carried out with a QoS of 0 (minimum QoS allowed by MQTT). When using a QoS of 1 (intermediate QoS level in MQTT) the results are an average jitter of 105.280 ms, a maximum of 109.611 ms and a minimum of 104.259 ms. Fig. 10-B shows the average jitter of each simulated hour when QoS is set to 1. Finally, if the QoS is raised to its maximum level (QoS = 2), the results obtained are an average jitter of 109.614 ms, a maximum of 113.459 ms and a minimum of 105.981 ms. Fig. 10-C shows the average jitter of each simulated hour when QoS is set to 2.

In short, with SimulateIoT-Mobile the users can analyse different aspects of the IoT environment in order to optimise or adapt it to their requirements.

8.2. Case 2: Personal mobility device (PMD) based on public bicycles

In recent years, the presence of PMD's such as bicycles or electric scooters has grown significantly in cities. In order to manage these PMD's and ensure the safety of their users, they can be equipped with several sensors that monitor the status of the PMD in real-time [46,47]. Thus, our second case study presents the simulation of a city with a smart PMD system.

In order to model this case study several assumptions should be taken into account:

- Each PMD includes the following sensors: (a) A GPS that publishes data related to its geolocation; (b) A Wheels pressure sensor, that monitors wheels pressure; (c) A Timer, that monitors the time the PMD is used by a user. On the other hand, the PMD incorporates an Actuator that notifies the user of anomalies, e.g. inadequate wheel pressure.
- The PMD route could be defined as *CSV_Route* based on specific routes defined on the map where the PMD and gateways are deployed.
- Each gateway can be defined as a *FogNode* element that is able to manage the data available on each PMD that reaches a gateway. Note that, from our point of view a *FogNode* element can act as gateway gathering data from sensors or sending data to other *FogNode*, *CloudNode* or *Actuator* elements.
- Each *FogNode* element re-send data to a *CloudNode* element which is able to store and analyse all the data available.
- Each *FogNode* element deployed is able to analyse the data send from the PMD in order to automatically notify the device if it has reached the lease term, the battery is low or the pressure of the wheel is not appropriate. Consequently, PMD incorporates an *Actuator* element that is able to notify the user.

8.2.1. Model definition

Fig. 11 shows an excerpt from the PMD based on the public bicycles model. It also includes numerical references for each node which are then used to describe the use case. Note that, for the sake of clarity, this model is simplified, including only one instance of each possible type of relationship between components. The complete version of this model is shown in Fig. 16 (Appendix B).

For the purpose of explaining the model, it is divided into three parts: (1) Edge Layer (Red nodes), (2) Fog Layer (Blue nodes), (3) Cloud Layer (Green nodes).

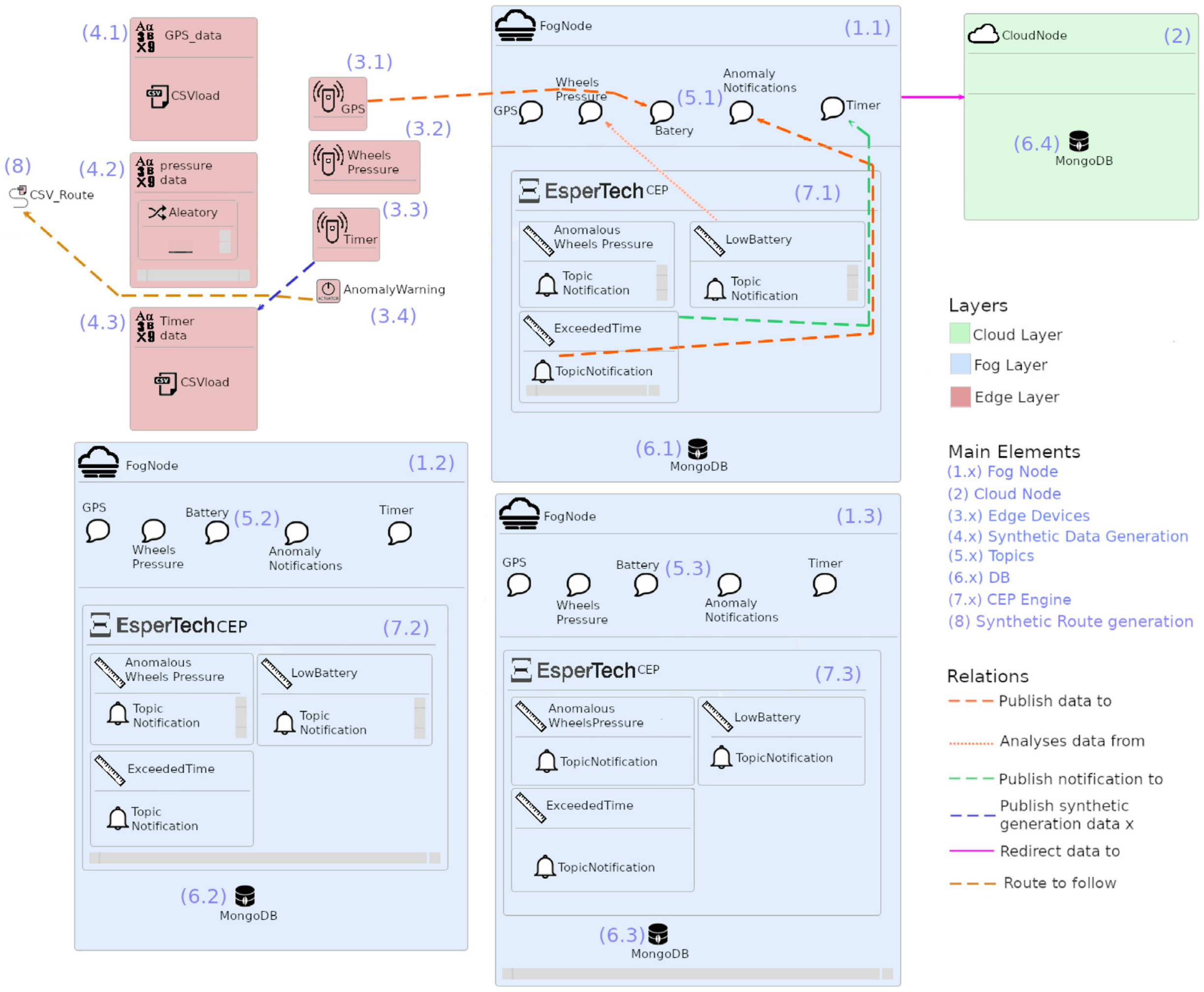


Fig. 11. Case 2. Model conforms to SimulateIoT-Mobile metamodel for Personal mobility device (PMD) based on public bicycles (Simplified version). Complete version in Fig. 16 (Appendix B).

(1) Edge Layer

The Edge layer contains the sensors (Fig. 11 labels 3.1, 3.2, 3.3) and actuators (Fig. 11 label 3.4) of the simulation. This set of devices is the one that has been incorporated into each PMD, thus representing a PMD.

These devices are three sensors and one actuator for each PDM: (a) A GPS (Fig. 11 label 3.1), which monitors the position of the PMD; (b) A pressure sensor (Fig. 11 label 3.2), which monitors the pressure of the wheels; (c) A timer (Fig. 11 label 3.3), which monitors the time the user uses a PMD; (d) An anomaly notifier (Fig. 11 label 3.4), which notifies the user when an anomaly occurs.

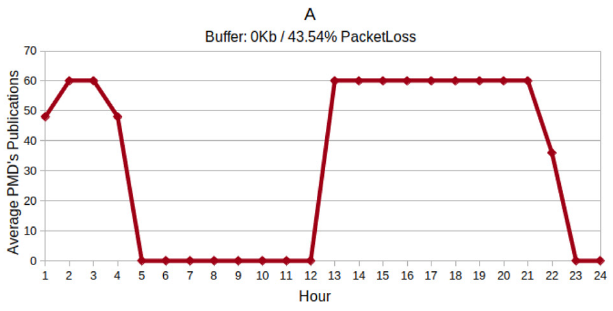
For simulation purposes each sensor has assigned a synthetic data generation (Fig. 11 labels 4.1, 4.2, 4.3) and a *Route* (Fig. 11 label 8). Note that, all devices have assigned the same *Route* (Fig. 11 label 8), consequently, this is the PMD *Route*.

Finally, the sensors and the actuator are linked to several Topics (Fig. 11 label 5.1), where they will publish their data or from where they will receive them respectively.

(2) Fog Layer

Fog nodes (Fig. 11 labels 1.1, 1.2, 1.3) are those that integrate the services necessary for the Edge nodes to carry out their functions. For this case study, three Fog nodes have been defined although other numbers of Fog nodes could be defined if needed.

Modelling of the Fog nodes is divided into Topics (Fig. 11 labels 5.1, 5.2, 5.3) and the CEP engine (Fig. 11 labels 7.1, 7.2, 7.3). In this use case, each Fog node offers five Topics: (a) *GPS*, where the *GPS* publishes the location of the *PMD*; (b) *WheelsPressure*, where the *Pressure* sensor publishes the pressure of the wheels; (c) *Battery*, where the different Edge nodes publish their remaining battery life; (d) *AnomalyNotifications*, where the actuator is subscribed for anomalies and the CEP engine publishes the anomalies identified; (e) *Timer*, where the Timer publishes the remaining leasing time.



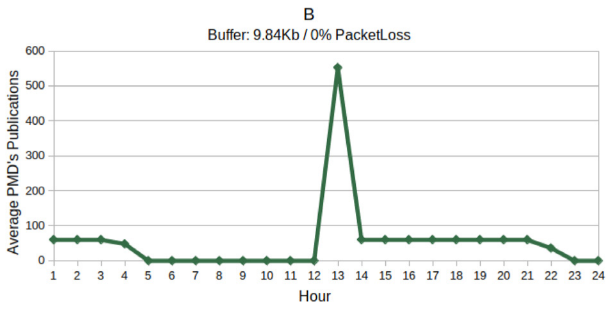


Fig. 12. Case 2. Simulation analysis: Intermediate buffer size required to avoid Packet loss.

On the other hand, the CEP engine analyses the data published in the Topics and applies a set of rules to detect anomalies. Specifically, the CEP engine defines three rules: (a) *AnomalousWheelPressure*, which analyses the data published in the Topic *WheelsPressure* and identifies if the wheel pressure is not adequate; (b) *LowBattery*, which analyses the data published in the Topic Battery and identifies if any device has a low battery; (c) *ExceedTime*, which analyses the data published in the Topic Timer and identifies if the elapsed lease time has expired. If one of the rules is met, the CEP engine publishes a notification in the Topic *AnomalyNotifications*.

Finally, the Fog nodes are related to the Cloud node. This relationship allows Fog nodes to forward all the data received by their Topics to the Cloud node for storage and future analysis. Note that Fog nodes can also store data if they include a database (Fig. 11 labels 6.1, 6.2, 6.3).

(3) Cloud Layer

Cloud node (Fig. 11 label 2) makes it possible to model a node with high capabilities to store and process data. In this case study, the Cloud node (Figure) 11 label 2 stores all data produced in the IoT environment during the simulation process. So, it is needed to model a database to store the received data (Fig. 11 label 6.4). Additionally, it is related to the Fog nodes defined on the model which redirect their data to the cloud node. The Cloud node has defined a Topic named *Notification* which receives all messages thrown several CEP rules defined at the Fog layer.

8.2.2. Model-to-text transformation and deployment

Once the model has been defined, the model-to-text transformation is applied with the same goals as in Case 01 (Section 8.1.2).

Consequently, each Edge node, Fog node and Cloud node is generated following the software architecture defined in Section 6 where model-to-text transformation has been defined.

8.2.3. Simulation analysis

Section 8.1.3 describes and exemplifies some of the experiments and analyses that can be carried out with SimulateIoT-Mobile. This subsection illustrates some additional experiments and analyses that the user could carry out in Case02, a Personal mobility device (PMD). In particular, the impact of a Fog node downtime in terms of packet loss is studied. Besides, the impact of switching brokers on jitter is analysed.

In this use case, the gateways are strategically distributed so that the devices in the environment do not suffer periods of disconnection. Therefore, the use of the Intermediate Buffer is not necessary. However, it is interesting to study the case where one of the Fog nodes goes down (including its gateway) and analyse the number of packets that could be lost in this case.

For this experiment, a device that follows a route that frequents the area with no coverage due to the Fog node downtime has been selected. This device publishes one publication per minute. The output logs of this device show a result of 43.54% of packets lost (Fig. 12-A).

In a hypothetical IoT environment where this Fog node could be down on a regular basis, the user could choose to add an Intermediate Buffer to the devices to avoid packet loss. In this use case, after several tests with SimulteIoTMobile, it is concluded that a 9.84 Kb (492 publications) buffer is needed to avoid packet loss (Fig. 12-B).

On the other hand, this use case studies the impact of switching brokers on jitter. Thus, the jitter of the messages published by a random device has been analysed during simulation execution. This device has switched Brokers approximately 100 times. Each switch involves interacting with the TSS, TDS and BDS, as well as coordinating the requests and responses of these components. The results of this study are an average jitter of 115.668 ms, a maximum of 824.735 ms and a minimum of 100.014 ms. Looking at the maximum jitter it is possible to state that during a Broker switch there is an additional jitter of about 724 ms (worst case). These results may indicate to the user the need to re-model their environment with a view to reducing the impact of jitter in their environment, e.g. critical section that requires a jitter of fewer than 820 ms. Fig. 13 shows an extract of 140 delay measurements where three periods of handover or gateway switching occur.

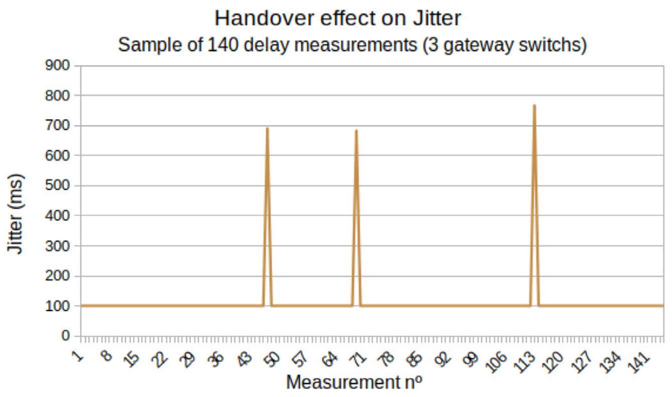


Fig. 13. Case 2. Simulation analysis: Extract of 140 delay measurements where three periods of handover occur.

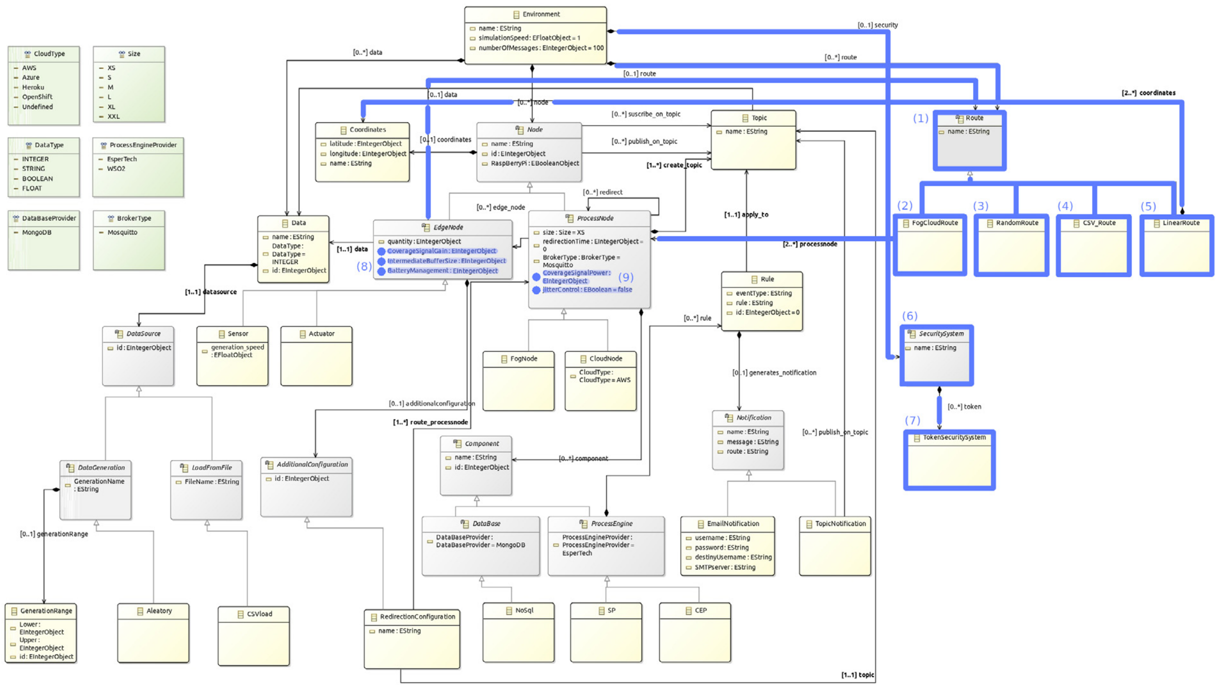


Fig. 14. Complete SimulateIoT-Mobile metamodel.

9. Discussion

In order to discuss the main facts reached by the proposal, the research questions previously defined will be answered.

In relation to RQ1, *"How could mobility be managed in IoT systems where the MQTT protocol is used?"*, in order to manage mobility in IoT systems based on MQTT protocol, several artefacts should be suitably generated (TSS, BDS, TDS) to manage the data among IoT devices and the additional application layer interactions needed to manage IoT mobility should be implemented. Consequently, as has been shown previously it is possible to manage mobility in IoT systems by using MQTT protocol.

Regarding RQ2, *"How might model-driven techniques be applied to model IoT systems with mobile nodes?"*, using model-driven development helps to manage the complexity of heterogeneous technology as a success during an IoT environment development. In this work the IoT systems with mobile nodes are modelled at high abstraction level by using metamodeling techniques. In addition, models obtained could be validated by using OCL (Objects Constrain Language) which guarantees that models are conformed to the metamodel proposed. The metamodel proposed makes it possible to model the target IoT systems using common domain elements.

Concerning RQ3, *"To what extent is it possible to generate the code needed to simulate an IoT system with mobile nodes from a model of the system?"*, modelling IoT environments is a key activity for any IoT project making it possible to focus on the

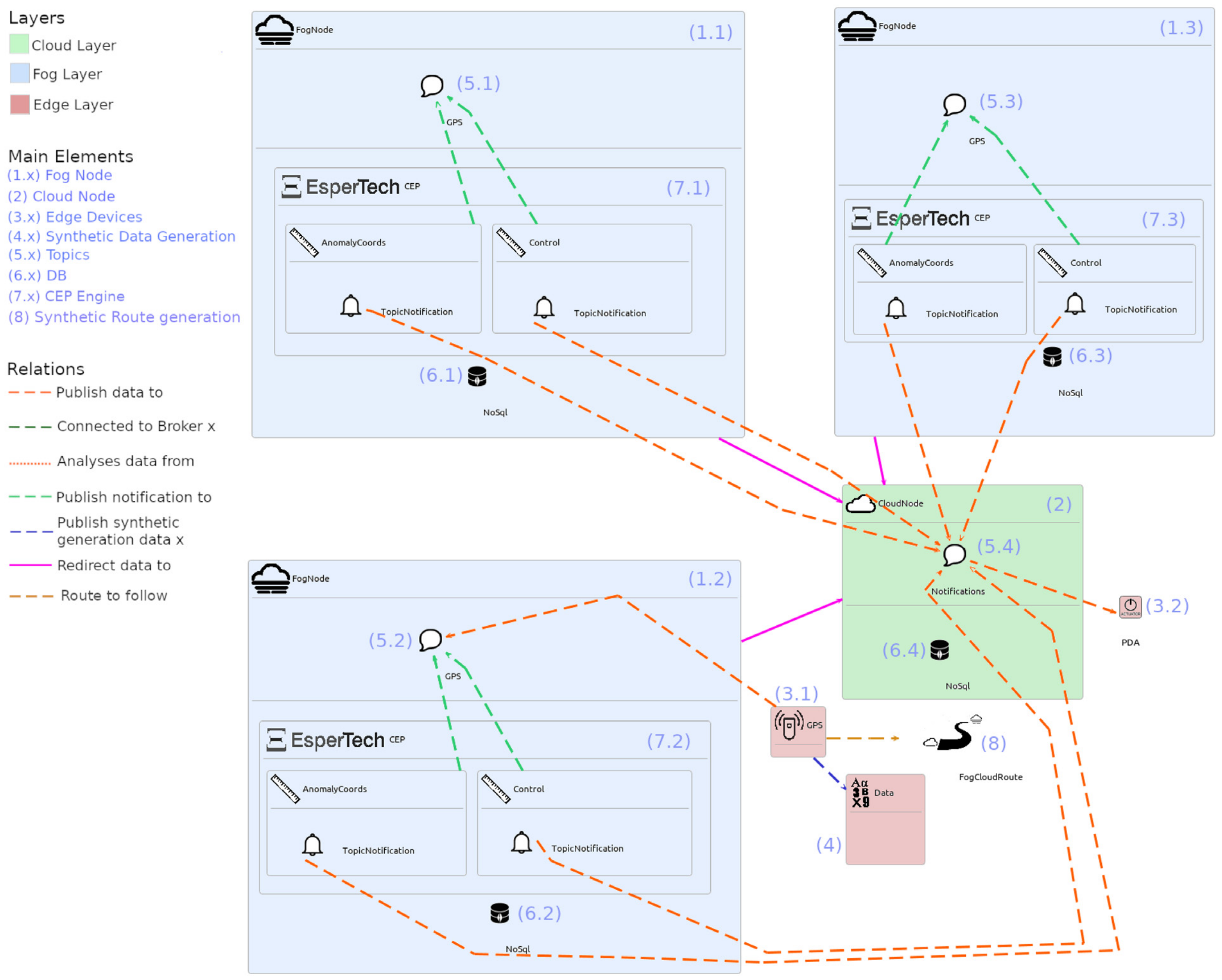


Fig. 15. Case 01. Model conforms to SimulateIoT-Mobile metamodel for animals tracking (complete version).

IoT domain in order to later on generate final code from the models defined. Additionally, modelling and simulating the behaviour of the IoT environments including mobile devices facilitates analysing of several system complex aspects such as battery behaviour, jitter, Intermediate Buffer, storage data, mobile communication protocols and so on. Code generate from the models defined includes multiple artefacts (described in Section 6) which are suitably orchestrated to simulate the IoT environment defined.

Finally, in relation to RQ4, *"To what extent could simulations of mobile IoT systems be useful for optimising the real system?"*, users are able to evaluate the system modifying their characteristics in order to find the better trade-off among the devices and nodes deployed. Specifically, users can use DSL tools such as the Graphical Editor to model the system and the model-to-text transformation to generate the code for deploying the simulation and checking the statistics generated during the simulation.

On the other hand, although there are interesting advantages to using SimulateIoT-Mobile DSL, there are some issues related to the mobility proposal presented.

Firstly, the publish/subscribe communication protocol used is based on MQTT protocol [10], although other publish/subscribe protocols can be used adding it to the model-to-text transformation. Secondly, model-to-text transformation, it has been defined for a concrete target based on microservices deployed on Docker containers which represent the concrete IoT nodes defined on the model. Other technological targets could be defined which implies re-code the model-to-text transformation. Thirdly, the routes of the IoT devices have been defined using common IoT mobility patterns, but additional IoT mobility patterns could be defined. Consequently, it would imply including additional modelling elements and including the new behaviours on the model-to-text transformation. Finally, current version of SimulateIoT-Mobile, for the sake of simplicity, allows defining connected nodes by TCP/IP, and we assume that connectivity is guaranteed.

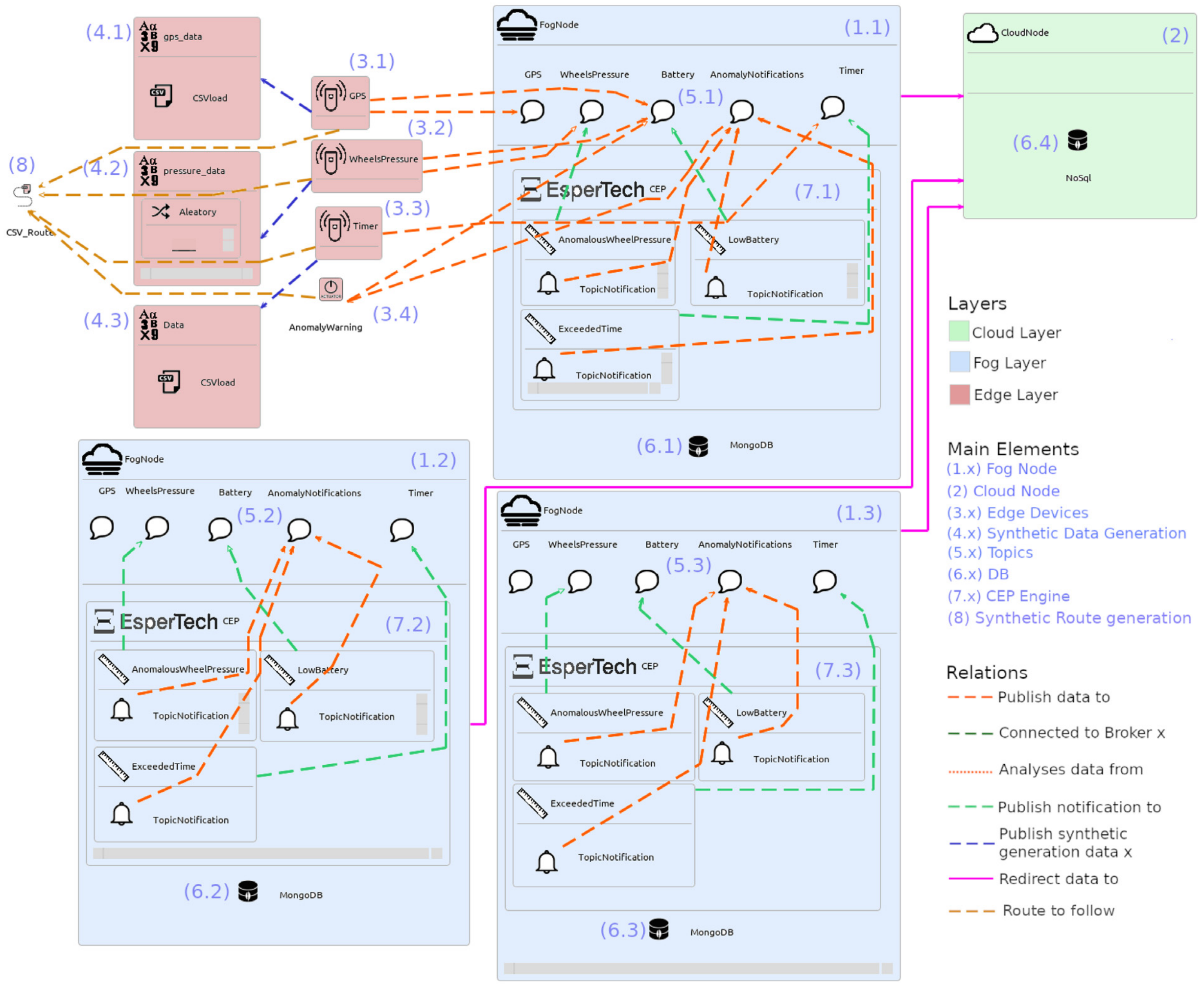


Fig. 16. Case02. Model conforms to SimulateIoT-Mobile metamodel for Personal mobility device (PMD) based on public bicycles (complete version).

10. Conclusions

Model-driven development techniques are a suitable way to tackle the complexity of domains where heterogeneous technologies are integrated. Initially, they focus on modelling the domain by using the well-known four-layer metamodel architecture. Then, by using model-to-text transformations the code for specific technology could be generated.

The IoT simulation methodology and tools proposed in this work help users to think about the IoT system in general and IIoT in particular, to propose several IoT alternatives and policies in order to achieve a suitable IoT architecture, including modelling IoT mobile nodes. In this sense, several kinds of mobile devices and routes can be defined, allowing defining realistic IoT environments. Finally, the IoT environments modelled can be deployed, simulated and analysed.

Future works include extending the metamodel and model-to-text transformation to model additional publish–subscribe communication protocols such as JMS or AMQP; or request–response protocols such as REST. Both extensions facilitate modelling IoT environments taking into account additional heterogeneity technology. Additionally, additional IoT mobile behaviours and routes could be identified and modelled. Finally, the model-to-text transformation could make it possible to generate Cloud support based on well-known Cloud providers such as AWS or Azure. It could open interesting research areas for IoT simulation purposes.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgement

Cofinanciado por la Unión Europea

JUNTA DE EXTREMADURA

Consejería de Economía, Ciencia y Agenda Digital

Appendix A

This Section shows in Fig. 14 the complete metamodel of SimulateIoT-Mobile. This metamodel is composed of the SimulateIoT metamodel and the extension carried out (highlighted in blue). The description of the classes and relationships that are not part of the extension (and that have not been addressed in this article), can be found in the article [17] Section IV, subsection A.

Appendix B

This Section shows the complete version of the models shown in Figs. 8 (Case 01. Animal tracking) and 11 (Case02. Personal mobility device (PMD) based on public bicycles) respectively in Figs. 15 and 16.

References

[1] E. Siow, T. Tiropanis, W. Hall, Analytics for the internet of things: A survey, ACM Comput. Surv. 51 (4) (2018) 74.
[2] S.M. Ghaleb, S. Subramaniam, Z.A. Zukarnain, A. Muhammed, Mobility management for IoT: a survey, EURASIP J. Wireless Commun. Networking 2016 (1) (2016) 1–25.
[3] K. Nahrstedt, H. Li, P. Nguyen, S. Chang, L. Vu, Internet of mobile things: Mobility-driven challenges, designs and implementations, in: 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), IEEE, 2016, pp. 25–36.
[4] H. Teng, Y. Liu, A. Liu, N.N. Xiong, Z. Cai, T. Wang, X. Liu, A novel code data dissemination scheme for internet of things through mobile vehicle of smart cities, Future Gener. Comput. Syst. 94 (2019) 351–367.
[5] L. Nóbrega, A. Tavares, A. Cardoso, P. Gonçalves, Animal monitoring based on IoT technologies, in: 2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany), 2018, pp. 1–5.
[6] F. Almada-Lobo, The industry 4.0 revolution and the future of manufacturing execution systems (MES), J. Prod. Innov. Manage. 3 (4) (2015) 16–21.
[7] M.B. Yassein, S. Aljawarneh, W. Al-Sarayrah, Mobility management of internet of things: Protocols, challenges and open issues, in: 2017 International Conference on Engineering & MIS, ICEMIS, IEEE, 2017, pp. 1–8.
[8] J.E. Luzuriaga, J.C. Cano, C. Calafate, P. Manzoni, M. Perez, P. Boronat, Handling mobility in IoT applications using the MQTT protocol, in: 2015 Internet Technologies and Applications, ITA, IEEE, 2015, pp. 245–250.
[9] L. Farhan, S.T. Shukur, A.E. Alissa, M. Alrweg, U. Raza, R. Kharel, A survey on the challenges and opportunities of the internet of things (IoT), in: 2017 Eleventh International Conference on Sensing Technology, ICST, IEEE, 2017, pp. 1–5.
[10] Oasis, Message queuing telemetry transport (MQTT) v5.0 oasis standard, 2019, URL https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.
[11] CoAP, The constrained application protocol (CoAP) - RFC 7252, 2014, https://datatracker.ietf.org/doc/html/rfc7252.
[12] S.-M. Cheng, P.-Y. Chen, C.-C. Lin, H.-C. Hsiao, Traffic-aware patching for cyber security in mobile IoT, IEEE Commun. Mag. 55 (7) (2017) 29–35.
[13] X. Liu, N. Ansari, Toward green IoT: Energy solutions and key challenges, IEEE Commun. Mag. 57 (3) (2019) 104–110.
[14] J.E. Luzuriaga, M. Perez, P. Boronat, J.C. Cano, C. Calafate, P. Manzoni, Improving mqtt data delivery in mobile scenarios: Results from a realistic testbed, Mob. Inf. Syst. 2016 (2016).
[15] B. Selic, The pragmatics of model-driven development, IEEE Softw. 20 (5) (2003) 19–25.
[16] A. Wortmann, O. Barais, B. Combemale, M. Wimmer, Modeling languages in industry 4.0: An extended systematic mapping study, Softw. Syst. Model. 19 (1) (2020) 67–94.
[17] J.A. Barriga, P.J. Clemente, E. Sosa-Sánchez, A.E. Prieto, SimulateIoT: Domain specific language to design, code generation and execute IoT simulation environments, IEEE Access 9 (2021) 92531–92552.
[18] M. Bouaziz, A. Rachedi, A survey on mobility management protocols in wireless sensor networks based on 6LoWPAN technology, Comput. Commun. 74 (2016) 3–15.
[19] C.C. Sobin, A survey on architecture, protocols and challenges in IoT, Wirel. Pers. Commun. (ISSN: 1572-834X) 112 (3) (2020) 1383–1429, URL https://doi.org/10.1007/s11277-020-07108-5.
[20] R. Silva, J.S. Silva, F. Boavida, A proposal for proxy-based mobility in WSNs, Comput. Commun. 35 (10) (2012) 1200–1216.
[21] R. Silva, J. Sa Silva, F. Boavida, Mobility in wireless sensor networks – Survey and proposal, Comput. Commun. (ISSN: 0140-3664) 52 (2014) 1–20, URL https://www.sciencedirect.com/science/article/pii/S0140366414001911.
[22] B. Bettoumi, R. Bouallegue, LC-DEX: Lightweight and efficient compressed authentication based elliptic curve cryptography in multi-hop 6LoWPAN wireless sensor networks in HIP-based internet of things, Sensors (ISSN: 1424-8220) 21 (21) (2021) URL https://www.mdpi.com/1424-8220/21/21/7348.
[23] H.A. Al-Kashoash, H. Kharrufa, Y. Al-Nidawi, A.H. Kemp, Congestion control in wireless sensor and 6LoWPAN networks: toward the internet of things, Wirel. Netw. 25 (8) (2019) 4493–4522.
[24] M.L. Miguel, E. Jamhour, M.E. Pellenz, M.C. Penna, SDN architecture for 6LoWPAN wireless sensor networks, Sensors 18 (11) (2018) 3738.
[25] R. Hamidouche, Z. Aliouat, A.M. Gueroui, A.A.A. Ari, L. Louail, Classical and bio-inspired mobility in sensor networks for IoT applications, J. Netw. Comput. Appl. 121 (2018) 70–88.

[26] Y. Chen, T. Kunz, Performance evaluation of IoT protocols under a constrained wireless access network, in: 2016 International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT), 2016, pp. 1–7.
[27] J.E. Luzuriaga, J.C. Cano, C. Calafate, P. Manzoni, M. Perez, P. Boronat, Handling mobility in IoT applications using the MQTT protocol, in: 2015 Internet Technologies and Applications, ITA, 2015, pp. 245–250.
[28] S. Chun, J. Park, Mobile CoAP for IoT mobility management, in: 2015 12th Annual IEEE Consumer Communications and Networking Conference, CCNC, 2015, pp. 283–289.
[29] C. Atkinson, T. Kuhne, Model-driven development: a metamodeling foundation, IEEE Softw. 20 (5) (2003) 36–41.
[30] S. Sendall, W. Kozaczynski, Model transformation: The heart and soul of model-driven software development, IEEE Softw. 20 (5) (2003) 42–45.
[31] C. Perkins, Mobile IP, IEEE Commun. Mag. 35 (5) (1997) 84–99.
[32] R. Wakikawa, Z. Zhu, L. Zhang, A survey of mobility support in the internet. RFC 6301, 2011, URL https://www.rfc-editor.org/info/rfc6301.
[33] R. Moskowitz, P. Nikander, P. Jokela, T. Henderson, Host Identity Protocol, Tech. rep., 2008.
[34] A.R. Sfar, E. Natalizio, Y. Challal, Z. Chtourou, A roadmap for security challenges in the internet of things, Digit. Commun. Netw. 4 (2) (2018) 118–137.
[35] C. Thomás Oliveira, R. Moreira, F. de Oliveira Silva, R. Sanches Miani, P. Frosi Rosa, Improving security on IoT applications based on the FIWARE platform, in: 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications, AINA, 2018, pp. 686–693.
[36] E. Tuyishimire, A. Bagula, A. Ismail, Clustered data muling in the internet of things in motion, Sensors (ISSN: 1424-8220) 19 (3) (2019) URL https://www.mdpi.com/1424-8220/19/3/484.
[37] A. Bagula, E. Tuyishimire, J. Wadepoel, N. Boudriga, S. Rekhis, Internet-of-things in motion: A cooperative data muling model for public safety, in: 2016 Intl IEEE Conferences on Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016, pp. 17–24.
[38] O. Tsilomitrou, N. Evangeliou, A. Tzes, Mobile robot tour scheduling acting as data mule in a wireless sensor network, in: 2018 5th International Conference on Control, Decision and Information Technologies (CoDIT), 2018, pp. 327–332.
[39] A. Ismail, E. Tuyishimire, A. Bagula, Generating dubins path for fixed wing uavs in search missions, in: International Symposium on Ubiquitous Networking, Springer, 2018, pp. 347–358.
[40] R. Kays, M.C. Crofoot, W. Jetz, M. Wikelski, Terrestrial animal tracking as an eye on life and planet, Science 348 (6240) (2015) aaa2478, URL https://www.science.org/doi/abs/10.1126/science.aaa2478.
[41] T.M. Behera, S.K. Mohapatra, U.C. Samal, M.S. Khan, Hybrid heterogeneous routing scheme for improved network performance in WSNs for animal tracking, Internet Things (ISSN: 2542-6605) 6 (2019) 100047, URL https://www.sciencedirect.com/science/article/pii/S2542660518301914.
[42] F. Maroto-Molina, J. Navarro-García, K. Prí ncipe Aguirre, I. Gómez-Maqueda, J.E. Guerrero-Ginel, A. Garrido-Varo, D.C. Pérez-Marín, A low-cost IoT-based system to monitor the location of a whole herd, Sensors (ISSN: 1424-8220) 19 (10) (2019) URL https://www.mdpi.com/1424-8220/19/10/2298.
[43] Q.M. Ilyas, M. Ahmad, Smart farming: An enhanced pursuit of sustainable remote livestock tracking and geofencing using IoT and GPRS, Wirel. Commun. Mob. Comput. (ISSN: 1530-8669) 2020 (2020) 6660733, URL https://doi.org/10.1155/2020/6660733.
[44] J.G. Panicker, M. Azman, R. Kashyap, A LoRa wireless mesh network for wide-area animal tracking, in: 2019 IEEE International Conference on Electrical, Computer and Communication Technologies, ICECCT, 2019, pp. 1–5.
[45] P. Sadhukhan, An IoT-based E-parking system for smart cities, in: 2017 International Conference on Advances in Computing, Communications and Informatics, ICACCI, 2017, pp. 1062–1066.
[46] F. Behrendt, Why cycling matters for smart cities. Internet of bicycles for intelligent transport, J. Transp. Geogr. (ISSN: 0966-6923) 56 (2016) 157–164, URL https://www.sciencedirect.com/science/article/pii/S0966692316300746.
[47] R. Sanchez-Iborra, L. Bernal-Escobedo, J. Santa, Eco-efficient mobility in smart city scenarios, Sustainability (ISSN: 2071-1050) 12 (20) (2020) URL https://www.mdpi.com/2071-1050/12/20/8443.
[48] A. Dorri, S.S. Kanhere, R. Jurdak, P. Gauravaram, Blockchain for IoT security and privacy: The case study of a smart home, in: 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2017, pp. 618–623.
[49] Z. Ning, P. Dong, X. Kong, F. Xia, A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things, IEEE Internet Things J. 6 (3) (2019) 4804–4814.
[50] Q. Fan, N. Ansari, Application aware workload allocation for edge computing-based IoT, IEEE Internet Things J. 5 (3) (2018) 2146–2153.
[51] H. Jayakumar, A. Raha, Y. Kim, S. Sutar, W.S. Lee, V. Raghunathan, Energy-efficient system design for IoT devices, in: 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE, 2016, pp. 298–301.
[52] N. Kaur, S.K. Sood, An energy-efficient architecture for the internet of things (IoT), IEEE Syst. J. 11 (2) (2015) 796–805.
[53] D.S. Kolovos, A. García-Domínguez, L.M. Rose, R.F. Paige, Eugenia: towards disciplined and automated development of GMF-based graphical model editors, Softw. Syst. Model. (2015) 1–27.
[54] OMG, OMG Object Constraint Language (OCL), Version 2.3.1, 2012, URL http://www.omg.org/spec/OCL/2.3.1/.
[55] Obeo, Acceleo project , 2012,.