A Formal Modeling and Verification Approach for IoT-Cloud Resource-Oriented Applications

Yasmine Gara Hellal, Lazhar Hamel, Mohamed Graiet, Daniel Balouek

2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 979-8-3503-9566-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/CCGrid59990.2024.00047

A Formal Modeling and Verification Approach for IoT-Cloud Resource-Oriented Applications

1st Yasmine Gara Hellal
Faculty of Sciences of Monastir
University Of Monastir
Monastir, Tunisia
garayesmine@gmail.com

2nd Lazhar Hamel
Computer Science Department, ISIMM
University Of Monastir
Monastir, Tunisia
lazhar.hamel@isimm.rnu.tn

3rd Mohamed Graiet, 4thDaniel Balouek
Inria, LS2N Laboratory
IMT Atlantique
Nantes, France
first.lastname@imt-atlantique.fr

***Abstract*—IoT-Cloud environments are being increasingly adopted for the deployment of applications and particularly resource-oriented ones. However, ensuring correct communications during the execution of IoT applications is not guaranteed. In fact, a substantial class of applications is intended to run on constrained IoT networks. Moreover, IoT devices exchange the data derived from various Cloud providers and in accordance with different protocols. In this paper, we propose a formal approach to model and verify the applications deployed over IoT-Cloud environments. The proposed model encompasses four verification levels: the Structural, Functional, Operational and Behavioral levels. Therefore, we opted for the Event-B formal method that allows gradual problems decomposition by relying on its refinement capabilities. The proposed approach has proven its efficiency for the modeling and the verification of IoT applications. We applied mathematical proof-based method to verify the model since it provides rigorous reasoning. We also employed the ProB animator to proceed in the validation of the model.**

***Index Terms*—IoT applications, Cloud, Behavioral Verification, Operational, Functional, Structural Verification, Event-B.**

I. Introduction

The Internet of Things (IoT) [1] has emerged over the past decade as a dynamically growing data acquisition infrastructure that is being driven by the dynamic rise in a wide range of things including virtual services, physical devices and so on. Currently, it has become more relevant to the world given its provision with advanced technology pillars: Cloud Computing [2], Blockchain [3], Big Data and AI. Besides, IoT things are equiped with Internet connectivity to enable the development of intelligent applications through the collection and exchange of data.

Recently, there has been a growing interest in IoT communications during the execution of applications especially resource-oriented applications that follow the REST [4] architectural style. In fact, a considerable class of such applications is intended to interact and run on constrained IoT networks that may exhibit so sever constraints with unreliable channels, low throughput and insufficient wireless bandwidth. Further, they frequently encounter serious disruptions in end-to-end connectivity, which may in turn lead to deadlock situations while executing IoT applications. Moreover, IoT networks may involve a substantial portion of nodes [5] that are in a tight of processing resources, energy budget and storage, hence, they may face troubles when they attempt to exchange their data through the network. Indeed, many mediators [6] (e.g proxy, gateway) are implicated at various locations to aid the constrained devices to communicate however they may end up in deadlock situations since the number of devices they can assist simultaneously is limited. The constrained devices are also normally-off and reattached to the network only when required to save their available energy, Even so, they may be exhausted and definitely discarded at any point in time which poses serious issues of non-reachability during IoT communications. Besides, IoT devices implement a wide variety of data formats and communication protocols to exchange the data that are in turn derived from various Cloud providers. Consequently, additional incompatibility issues may occur especially with the emergence of Cloud paradigm and the challenges it poses.

As a result of the described accumulate issues, we propose an approach to formally model and verify the resource-oriented applications deployed on IoT-Cloud environments, and also to check the correctness of the communications during their execution. Indeed, this task is not trivial. In fact, several verification axes of IoT applications are covered by our model in order to avoid the inconsistency of their interactions and possible deadlock situation that may occur. The major contribution of this work is a model that comprises four sub-models in respect to four verification axes: the Structural, Functional, Operational and Behavioral models. The suggested approach is new in a sens that it guarantees that the IoT-Cloud environment on which runs a resource-oriented application is properly structured and designed. Also, the required resource is available and the device that hosts it is reachable to avoid deadlock situations while interacting with the endpoints that are exhausted or down.

The remainder of this paper is organized as follows: In section 2 we present our motivations. Section 3 summarizes the related works. In Section 4, we go over the main requirements considered in this article. In Section 5, we expose the Event-B detailed to the extent needed. Section 6 presents our proposed Event-B model. In Section 7, we detail the model verification and validation. Finally, section 8 concludes this paper and presents future works.

979-8-3503-9566-2/24/$31.00 ©2024 IEEE
DOI 10.1109/CCGrid59990.2024.00047

II. Motivations and problem statement

Let us consider the safety-critical Fire Alarm (FA) application that improves the degree of protection in a smart city [7]. As shown in Fig. 1, the FA application is deployed and executed on a constrained IoT system (*system1*) that consists of a constrained-node network (*CNN1*) and a low-power, lossy network (*LLN1*). The characteristics of the *CNN1* are influenced by being composed of a considerable portion of constrained things (e.g *sensor3*, *building1*, *QN*,...), that exhibit severe limits on memory, processing resources, available power and remain unreachable for extended periods. However, the *LLN1* involves a portion of constrained devices (e.g *sensor1*,...) with additional aspects steaming from the network as well namely a high degree of packet loss and interruptions in the connectivity. The IoT devices are responsible for the following resources that encapsulate their functionalities and expose their services:

1) *Gas Sensing Service*: hosted on the *sensor3*.
2) *Window/Door Control Service*: offered by the *sensor3*.
3) *Emergency Notification Service*: hosted on the *sensor3*.
4) *Smoke Sensing Service*: offered by the *sensor2*.
5) *Temperature Record Service*: hosted on the *sensor2*.
6) *Water Sprinkling Service*: hosted on the *sensor2*.

The resource-oriented FA application is executed on the *system1* that emerges Cloud paradigm. Commonly, an IoT device (e.g *sensor3*) within *system1* is expected to communicate in a shared manner with another device (e.g *QN*) or a cloud service and it is intended to manipulate (update, create) its hosted resources (the key abstraction of information in REST [4]) to ascertain whether there is a risk of fire (see Fig. 1). The following FA communications mainly occur:

- ***QN* to *sensor3*:** *QN* queries the *sensor3* about its hosted resource *'Gas Sensing Service'* that is identified by *URI1*. The request is satisfied without the interference of mediators (e.g *proxy1*) since *QN* employs *CoAP* protocol [8] specially designed to transmit and retrieve the data of constrained nodes. Hence, the *sensor3* responds with a payload containing the resource value.
- ***sensor3* to *UnN1*:** *sensor3* exhibits severe limits and cannot talk directly to unconstrained *UnN1* that employs full protocol stack (e.g HTTP [9]). So as the interference of *proxy1* is required to make protocols translations and send *JSON* state of the captured *'Emergency Notification'*.
- ***UnN1* to a global Internet node:** *gateway1* is set up at *system1-system2* boundary to perform viable translations.
- ***sensor1* to *Cloud*:** a Cloud service retrieves the data from the *sensor1* and commands the actuator *UnN2*.

The aforementioned FA application is in charge of analysing the resources' contents retrieved in real-time from the *sensor1* and *sensor3* that detect smoke and gases and so on. These readings are evaluated against a predetermined threshold to determinate whether there is a risk of fire. In the event that a risk is identified, the *Water Sprinkling Service* will be enforced to rapidly extinguish the fire and avoid tragic aftermath.

However, developing the FA resource-oriented application properly and ensuring its correct and consistent behavior during device-to-device, cloud-to-device and/or device-to-cloud communications is an arduous task. In fact, diverse issues may occur when developing and executing the IoT application.

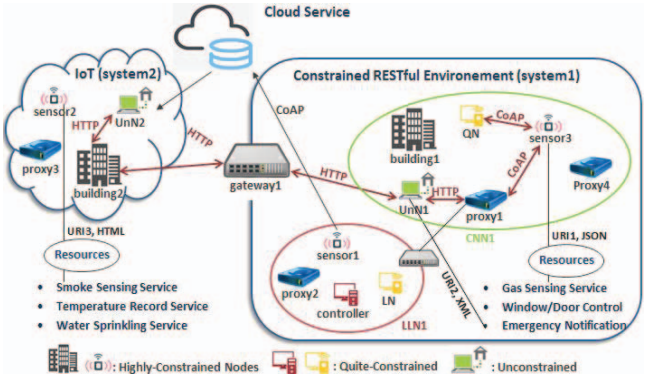


Fig. 1: IoT-Cloud Fire Alarm (FA) Application

- **Non-interoperability:** The format of a resource may be not recognizable by the device that queries about it. The reason for this is due to the mismatch and disparity in data types that each device supports. For example, *QN* attempts to capture the current or intended state of the resource *Gas Sensing Service* in the format JSON that understands it. If it receives the resource in a different format, the resource retrieval will be erroneous.
- **Looping:** If a request is invoked by *sensor1*, it may not reach the endpoint (e.g *QN*). This is due to an end-loop that occurs at a stage during the execution of FA application on *LLN1* such as interruptions in the connectivity that prevent the package from being delivered.
- **Non-reachability:** The constrained device (*sensor3*) has a limited energy budget and it is normally-off and reattached to the network when requisite. However, the *sensor3* may became unreachable when it is exhausted, definitely discarded and down or due to technical failures.
- **Device Capability:** A highly-constrained device (*sensor3*), most likely does not have the resources required to interact with the unconstrained device *UnN1* that supports *HTTP*. For instance, the available capacity of *sensor3* in terms of storage (5 GB) does not fit the needed capacity (12 GB) to communicate with *UnN1* in a reliable manner.
- **Mediator capacity and conflicts:** A non-elastic intermediary, for example, the shareable *proxy1*, cannot be allocated by the *sensor3* while its available capacity in terms of tasks it can handle in parallel (3 tasks) has been exceed. Such that, exceptionable conflicts may occur.

To overcome these problems, we propose an Event-B formal model that covers the requirements organized in four levels : The first, allows the checking of the overall architecture of constrained IoT-Cloud environments.The second checks the proper functioning of resource-based applications deployed on IoT domain. The third checks the correctness of the application's communications while tackling their operational features (Reachability, Availability, Compatibility). And, the last

level verifies the interactions behavioral properties (Deadlock-freeness, brokers interference).

III. Related Work

Research interest for IoT application systems has recently grown, resulting in many studies working on their specification and verification. In this section, we focus on the relevant ones.

In [10], a BiAgents* model based on the judicious combination of bigraphs and formal agents is proposed. In fact, the Bigraphs model the physical part for IoT systems in a mathematically sound way. Subsequently, their behaviors are expressed by means of Reaction Rules. In this work, the key demerit is that the functional and the operational aspects are not considered. Moreover, the approach is unsatisfactory as no mechanism of verification was evoked. Hence, any proofs on the model's reliability are given. In [11], an approach leveraging Thing-In-the-Loop is devised to verify the interactions of IoT components. Thereby, The key limitation is that authors do not model IoT components that appear with peculiar constraints, but instead just focus on their structural requirements. Further, the proposal has no analysis of the operational aspect.

We notice [12] that focuses on the functional and non-functional characteristics of IoT systems by binding them as UML diagrams that enable an intuitive graphical vision of their elements' relationship. Costa et al. [13] introduce a Model-Based methodology furnishing high-level abstractions. In [12], [13] authors pretend abstracting the complexity. However, such systems are sufficiently critical. Thereby, they impose a determinate level of precision. The major drawback of all the stated works that they do not model the systems following REST despite its suitability for IoT newer context.

Meanwhile, we highlight [14] that introduces a novel methodology articulated with event-Calculus, to verify IoT communications during the applications' execution. Similarly, numerous approaches [15], [16] and [17] involve the analysis, of Request/Response events that occur in MQTT and MSC systems. In recent studies by Song, et.al [18], a formal modeling method based on process algebra and lattice structure is introduced to abstract the collective behavior of IoT Systems as a sequence of actions. The major demerit of this study, is that it does not consider the third party infrastructure (e.g mediators interference) while designing the behavioral requirements.

Despite the feasibility indicated through experimental studies, the key shortcomings of all stated works that they treat each side of the verification separately. In this sense, we position our contribution that combines the modeling and verification of four axes of verification: Structural, Functional, Operational and Behavioral. Further, this work (1) addresses some properties (e.g compatibility), (2) deals with the conflicts of interfered mediators, (3) is based on rigor Event-B formal model and (4) does not suffer from the state-space explosion.

IV. Modeling Requirements

In this section, we furnish insight into the major concepts related to the verification of resource-oriented applications executed on IoT-Cloud environments. Modeling requirements are gradually excerpted from the presented concepts.

A. Structural Requirements

The structure of IoT systems is crucial and constitutes the primary artefact of applications life cycle including development and execution. In the following, we verify and address the design concerns of IoT architectural components (systems, networks, nodes) in order to ensure correct execution for IoT applications. Table I lists the IoT structural requirements deduced from [8] that our proposed model would fully satisfy.

TABLE I: Structural Requirements

S-RQ1	An IoT system includes a set of sub-systems *Rsy* where REST guidelines are applied to their design, a set of Non-REST sub-systems *NRsy* and a set of gateways. IoT system involves at least one sub-system $Rsy \neq \varnothing$.
S-RQ2	REST system *rst* is made up of a set of constrained networks *CN* and unconstrained networks *UN*. Each *CN* is classified as either Constrained-Node Network *CNN*, Challenged Network *CHN* (have certain network constraints) or both (Low-Power Lossy Network *LLN*).
S-RQ3	*CNN* contains a set of nodes classified according to their degree of constraints as unconstrained, less/quite-constrained, highly-constrained or proxy.
S-RQ4	A node belongs to at most one *CNN* that involves at least two constrained devices and one proxy.
S-RQ5	The dominant nodes in a *CNN* are the quite and highly-constrained nodes. (*card*(*high*) +*card*(*quite*) > (*card*(*less*) + *card*(*unconstrained*) + *card*(*proxy*).

For instance, in our motivating example (Fig. 1), the FA application is executed on an IoT system that consists of a constrained REST sub-system called *system1*,a general internet sub-system called *system2* and a *gateway1* in respect to the requirement **S-RQ1**. The *system1* includes two networks (*LLN1*, *CNN1*) in regard to **S-RQ2**. The constrained-node network *CNN1* within *system1* involves two highly-constrained nodes (*building1*, *sensor3*), one quite-constrained node (*QN*), one unconstrained node (*UnN1*) and two proxies (*proxy1*, *proxy4*) in respect to **S-RQ3** and **S-RQ4**. The characteristics of *CNN1* are influenced by being composed of a significant portion of constrained nodes (3 devices) which is greater than the remaining unconstrained nodes of the same cluster (**S-RQ5**).

B. Functional Requirements

Substantially, resource-oriented applications are being deployed on IoT-Cloud environments. Table II outlines the functional requirements that will be covered by our suggested model to address the concerns of such application's elements.

Back to our motivating example, the resource *'Gas Sensing'* should be hosted on at least one device (*sensor3*) that must in turn be responsible for at least one resource (**F-RQ1**). Likewise, the **F-RQ2** requires the same *URI1* to only identify the resource *'Gas Sensing'* as it is not able to designate another resource (e.g *'Smoke Sensing'*). Additionally, the **F-RQ3** requires the device *sensor3* to support at least one protocol (*CoAP*) and one data format (*JSON*). A highly-constrained node, for example *building1*, should not employ heavyweight protocols such as *HTTP* which is ensured by

modeling **F-RQ5**. Finally, the **F-RQ6** imposes the *proxy1* to implement *client* and *server* roles at the same time.

TABLE II: Functional Requirements

F-RQ1	A resource runs on at least one node that must host at least a relevant resource.
F-RQ2	A resource is named by at least one *URI* that designates exactly one resource.
F-RQ3	Each node (device) must support at least one protocol and one data format.
F-RQ4	A node that belongs to a Non-REST system is never able to support *CoAP*.
F-RQ5	Highly Constrained nodes cannot employ full protocol stack such as using *HTTP* and *XML*.
F-RQ6	A node excluding brokers must commonly have at least one role: Client, Server. However, the mediators should implement both roles simultaneously.

C. Operational Requirements

In the following, we tackle the fundamental operational requirements that our proposed model would fully satisfy to ensure successful and consistent interactions for IoT resource-oriented applications(see Table III). To do so, we analyse and check the operational properties of connected devices (Reachability, Availability, Compatibility) that are expected to exchange data during the execution of IoT applications. Such that, we find out deadlock situations that mainly occur.

For instance, in our motivating example, we consider the actuator *QN* that attempts to capture the state of the resource *'Gas sensing'* hosted on the constrained *sensor3*. Through that, *QN* should implement client role in respect to **O-RQ1** and the *sensor3* should act as a server. Likewise, the **O-RQ2** requires the *sensor3* to be reachable despite its limited energy budget. Noteworthy, the *sensor3* may be powered-off at any point in time and may became unreachable when it is exhausted or down or due to technical failures. Accordingly, the **O-RQ3** requires the *'Gas sensing'* service to be available, in other words, it needs to be directly offered or dynamically linkable (navigable) by another resource running on the *sensor3*. Indeed, the FA application cannot be executed if one of its mandatory resources is not available. Finally, the compatibility concept is employed to verify the matching of data formats supported by the endpoints (*sensor3*, *QN*) and the captured resource. For example, the *JSON* representation of the resource *'Gas Sensing'* should be understandable by both endpoints, which is ensured by modeling the **O-RQ4**.

TABLE III: Operational Requirements

O-RQ1	Sender node must implement Client role and the receiver implements Server role.
O-RQ2	Each endpoint (e.g *server*) must be reachable against another device (e.g *client*) seeking access to its hosted resources (Reachability).
O-RQ3	The resource queried by a client is hosted on a server (Availability).
O-RQ4	The endpoints support the resource's media type mentioned in their associated URIs (Compatibility).

D. Behavioral Requirements

In order to guarantee proper communications for IoT applications, the connected devices must be behaviorally compatible. In fact, IoT devices appear with a peculiar heterogeneity among other factors as they might own various constraints degrees and interact according to different protocols. In the following, we describe how analyzing the protocols and the resources constraints of a device aid us in identifying any potential interoperability problems. Indeed, we check the devices constraints (e.g processing capability) and the matching of protocols that must be deadlock-free. Two devices have a behavioral mismatch if their protocols get stuck during their interactions. In some cases where behavioral mismatches are detected, it is necessary to interfere brokers in the form of proxies and unconstrained servers to alleviate the devices incompatibility and assist their interactions. Table III sums up the behavioral requirements tackled by our suggested model.

TABLE IV: Behavioral Requirements

B-RQ1	**If the *sender* is highly-constrained & the *sender* and the *receiver* are encompassed into the same *CNN*** $\Rightarrow$ whatever is the *sender*'s protocol (*CoAP*, *HTTP*), proxies of the same cluster must assist the interaction.
B-RQ2	**If the *sender* is quite-constrained & the *receiver* has no restrictions with respect to the constraints degree** $\Rightarrow$ If the *sender* applies *CoAP*, it talks easily to the *receiver* that uses full protocol stack (*HTTP*, *XML*, ...), otherwise, the brokers' support is required.
B-RQ3	**If the *sender* is less-constrained or unconstrained** $\Rightarrow$ The interference of intermediaries is not necessary and the interaction can be directly satisfied.

For instance, in our motivating example, the highly constrained *sensor3* most likely cannot communicate with *QN* in a reliable manner. Therefore, the **B-RQ1** requires this incompatibility task to be supported by neighbour proxies (*proxy1*, *proxy4*) or unconstrained devices acting as servers (*UnN1*). The mediator choice is considered according to its capacity. In fact, it is limited in terms of tasks it can handle simultaneously. Thus, we propose to check all the available brokers' capacities. Each time a proxy (*proxy1*) has exceed its maximum capacity, the system passes the task to a contiguous proxy (*proxy4*). If all proxies have reached their maximum capacity, the task is handled by unconstrained servers (*UnN1*).

V. Overview of the Event-B method

Event-B [19] is a formal technique based on the set theory, that adopts mathematical tools known for their rigour when dealing with safety-critical and complex IoT applications. An Event-B model is organised in terms of two basic constructs representing the context and the machine. A context specifies the static part of a representation; it includes CONSTANTS, carrier SETS along with AXIOMS, and finally THEOREMS which express some properties derivable from the axioms. Regarding the machine, it involves the dynamic portion that describes behavioral properties of a model. Such machine includes VARIABLES describing the state of a machine, INVARIANTS constraining that state by restricting the possible

values to hold by the variables, and EVENTS allowing such state update. An event is modelled through a set of guards and actions following this form: ANY X WHEN Grd THEN Act END. An event is activated when the guard's substitution is evaluated to true. However, the actions maintain the variables evolution at each event call.

VI. The Proposed IoT Applications Event-B Model

In this section, we detail our Event-B approach for the verification of the correctness of resource-oriented applications to avoid their erroneous communication and execution on IoT-Cloud environments. As depicted in Fig. 2, Our model is broken down into its constituent sub-models, while covering four verification axes: Structural, Functional, Operational and Behavioral Models.

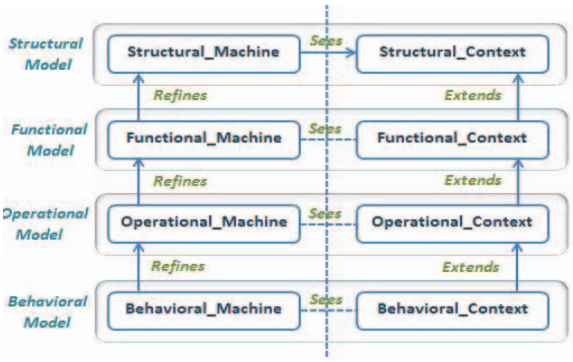


Fig. 2: An Event-B formal approach for IoT applications

A. Modeling the Structural Requirements

Preliminary, we model the elements that constitute the IoT structure to guarantee correct execution for IoT applications. For instance, it is axiomatic to define that an IoT system is made of a non empty set of networks, that a *CNN* network involves a significant portion of constrained nodes. At this level, we construct IoT systems following REST as they are lightweight and enable loose coupling of IoT elements.

To reach such goal, we primarily define in the first abstraction level the *"Structural_Context"* (Fig. 3). IoT elements are introduced thanks to Event-B as SETS named $IoTSystems$, $Networks$, $Gateways$ and $Nodes$, to model the structure of IoT systems. However, the relations between IoT elements are given in the AXIOMS clause where the constants are typed.

S-RQ1 is modeled in the following *axm1*, *axm2* and *axm3*:

- *axm1* specifies that all the defined sets are finite.
- The constant $SysStyles$ denotes the architectural style of each IoT system (*axm2*, *axm3*). It is modeled by a partial function as a network is not required to follow a style.
- IoT includes at least one REST sub-system (*axm3*).

S-RQ2 is modeled in the following *axm4*, *axm5* and *axm6*:

- $NetKinds$ is a constant that maps a given network to its kind (e.g constrained-node network *CNN*) (*axm4*, *axm6*).

CONTEXT **Structural_Context**
SETS
$IoTSystems\ Networks\ Nodes\ Gateways\ \ldots$
CONSTANTS
$SysNetworks\ NetNodes\ NodDeg\ \ldots$
AXIOMS
axm1 : $finite(IoTSystems) \land finite(Networks) \land finite(Nodes) \land finite(Gateways)$
axm2 : $ArchiStyles = \{REST, NonREST\}$
axm3 : $SysStyles \in IoTSystems \nrightarrow ArchiStyles \land SysStyles \sim [\{REST\}] \neq \varnothing$
axm4 : $Kinds = \{CNN, LLN, CHN, UN\}$
axm5 : $SysNetworks \in Networks \twoheadrightarrow IoTSystems$
axm6 : $NetKinds \in Networks \nrightarrow Kinds$
axm7 : $Deg = \{High, Quite, Less, Unconstrain, Proxy\}$
axm8 : $NetNodes \in Nodes \twoheadrightarrow Networks$
axm9 : $NodDeg \in Nodes \nrightarrow Deg$
END

Fig. 3: The Event-B *Structural_Context*

- $SysNetworks$ correlates each network to its potential system (*axm5*). It is modeled by a surjective function since each IoT system must involve at least one network.

S-RQ3 is modeled in the following *axm7*, *axm8* and *axm9*:

- $NodDeg$ denotes the constraints degree of IoT node (e.g quite-constrained, unconstrained) (*axm7*, *axm9*).
- The constant $NetNodes$ associates each node to its corresponding network (*axm8*).

The above context is seen (clause **SEES**) by *"Structural_Machine"* (Fig. 4) that introduces some variables in INVARIANTS to model the dynamic elements of IoT systems designed using REST. For instance, highly-constrained nodes (resp. unconstrained) within *CNN* clusters, are considered to be dynamic as they designate a migratory nature. In fact, IoT nodes can move autonomously without even knowing in advance mobility parameters such as destination locations. Therefore, a *CNN* may turn into an unconstrained network *UN* if the majority of its constrained nodes have left the network.

- We have defined the variable $rstSys$ to model the subset of REST systems (*inv1*).
- The *CNN* networks within REST systems are denoted by the total relation $rstCNN$ (*inv2*).
- To each *CNN* cluster, we have respectively associated its highly and quite-constrained nodes denoted by the relations $cnnHigh$ and $cnnQuit$ (*inv3*, *inv4*).
- Similarly, the less-constrained, unconstrained nodes and proxies are denoted by $cnnLess$, $cnnUnconst$ and $cnnProxy$ (hidden for a better readability).

S-RQ4 and **S-RQ5** are respectively covered in *inv5* and *inv6*. In fact, *inv5* permits to check whether each available *CNN* includes at least two constrained devices and one proxy. Subsequently, *inv6* tackles the density of the fulfilled *CNN* clusters, while monitoring the amount of constrained/unconstrained nodes within them.

MACHINE **Structural_Machine**
SEES **Structural_Context**
VARIABLES
$rstSys\ rstCNN\ cnnHigh\ cnnQuit\ \ldots$
INVARIANTS
inv1 : $rstSys \subseteq SysStyles \sim [\{REST\}]$
inv2 : $rstCNN \in rstSys \nleftrightarrow NetKinds \sim [CNN]$
inv3 : $cnnHigh \in ran(rstCNN) \longleftrightarrow NodDeg \sim [\{High\}]$
inv4 : $cnnQuit \in ran(rstCNN) \longleftrightarrow NodDeg \sim [\{Quite\}]$
$\ldots$
inv5 : $\forall c \cdot c \in ran(rstCNN) \Rightarrow$
$card(cnnHigh[\{c\}]) + card(cnnQuit[\{c\}]) \geq 2$
$\wedge cnnProxy[\{c\}] \neq \varnothing$
inv6 : $\forall c \cdot c \in ran(rstCNN) \Rightarrow$
$card(cnnHigh[\{c\}]) + card(cnnQuit[\{c\}]) \geq$
$card(cnnLess[\{c\}]) + card(cnnUnconst[\{c\}])$
END

Fig. 4: The Event-B *Structural_Machine*

CONTEXT **Functional_Context**
EXTENDS **Structural_Context**
SETS
$Resources\ Protocols\ Roles\ URIs\ MediaTypes$
CONSTANTS
$NodRsc\ EmbeddedRsc\ CoAP\ \ldots$
AXIOMS
axm1 : $NodRsc \in Resources \nrightarrow\!\!\!\!\!\rightarrow NetNodes \sim$
$[SysNetworks \sim [SysStyles \sim [\{REST\}]]]$
axm2 : $EmbeddedRsc \in Resources \leftrightarrow Resources$
axm3 : $ResourcesURIs \in URIs \nleftrightarrow Resources$
axm4 : $Protocols = \{CoAP, HTTP, MQTT, DTN\}$
axm5 : $ProtSupport \in Nodes \leftrightarrow Protocols$
axm6 : $Roles = \{Client, Server\}$
axm7 : $NodesRoles \in Nodes \leftrightarrow Roles$
axm8 : $MediaTypes = \{JSON, XML, HTML\}$
axm9 : $RepSupport \in Nodes \leftrightarrow MediaTypes$
END

Fig. 5: The Event-B *Functional_Context*

The structure of IoT environment over which runs the FA application of our motivating example (Fig. 1) is modeled by:
SETS
$IoTSystems = \{system1, system2, \ldots\}$
$Networks = \{CNN1, LLN1, \ldots\}$
$Nodes = \{sensor3, QN, proxy1, UnN1, LN, UnN2 \ldots\}$
CONSTANTS
$SysStyles = \{system1 \mapsto REST, system2 \mapsto REST, \ldots\}$
$SysNetworks = \{CNN1 \mapsto system1, \ldots\}$
$NetKinds = \{CNN1 \mapsto CNN, LLN1 \mapsto LLN, \ldots\}$
$NetNodes = \{sensor3 \mapsto CNN1, QN \mapsto CNN1, \ldots\}$
$NodDeg = \{sensor3 \mapsto High, QN \mapsto Quite, proxy1 \mapsto Proxy, UnN1 \mapsto Unconstrained, \ldots\}$
VARIABLES
$rstSys = \{system1\}, rstCNN = \{system1 \mapsto CNN1\}$
$cnnHigh = \{CNN1 \mapsto sensor3, CNN1 \mapsto building1\}$
$cnnQuit = \{CNN1 \mapsto QN\}$

B. Modeling the Functional Requirements

So far, we have modeled the basic structure of IoT systems that adhere to REST. In this part, we focus on checking the functional features and requirements of REST elements. Hence, we guarantee a satisfactory development for IoT resource oriented applications as they cannot be executed without their required functions. These features consist mainly on the resources hosted on IoT nodes, their supported protocols and *URIs* designating them. For this aim, we extend *"Structural_Context"* by *"Functional_Context"* (Fig. 5) and refine *"Structural_Machine"* by *"Functional_Machine"* (Fig. 6).

In the second abstraction level, we model **F-RQ1**, **F-RQ2**, **F-RQ3** and **F-RQ6** in the *"Functional_Context"*. To do so, we add some sets and axioms, that are defined as follows:

- *axm1* associates a given resource to its potential REST node, by means of a total surjective relation since a node should host at least one resource (in respect to **F-RQ1**).
- *axm2* allows linking a resource to its embedded resources.
- $ResourcesURIs$ refers to the URIs that enable naming a resource (*axm3*). It is modeled as a total relation since an URI designates exactly one resource (**F-RQ2**).
- To each node, we have assigned at least one protocol (*axm4*, *axm5*) and a data format (*axm8*, *axm9*) (**F-RQ3**).
- Likewise, $NodesRoles$ is a relation that affects at least one role (e.g client) to each node (*axm6*, *axm7*) (**F-RQ6**).

The *"Functional_Machine"* is seen by the *"Functional_Context"* to check whether **F-RQ4**, **F-RQ5** and **F-RQ6** are preserved, respectively through *inv1*, *inv2* and *inv3*. In fact, *inv1* detects the nodes that run on non-REST systems, while checking whether they support *CoAP* (in respect to **F-RQ4**). However, *inv2* monitors the protocols employed by highly-constrained nodes, since heavyweight protocols (*XML*, *HTTP*) are forbidden for such nodes that exhibit severe limits (**F-RQ5**). Regarding *inv3*, it allows checking whether the intermediaries (e.g proxy) implement both client and server roles, as stated in **F-RQ6**.

The functional elements of FA (Fig. 1) are modeled by the following sets and constants:
$Resources = \{SmokeSensingService, Temperature, EmergencyNotification, GasSensingService, \ldots\}$
$NodRsc = \{GasSensingService \mapsto sensor3, EmergencyNotification \mapsto UnN1, \ldots\}$
$ProtSupport = \{sensor3 \mapsto CoAP, QN \mapsto CoAP, proxy1 \mapsto HTTP, proxy1 \mapsto CoAP, UnN1 \mapsto HTTP, \ldots\}$
$RepSupport = \{sensor3 \mapsto JSON, sensor3 \mapsto HTML, QN \mapsto HTML, proxy1 \mapsto JSON, \ldots\}$

C. Modeling the Operational Requirements

Up to this level, we have modeled the architecture of IoT-based systems following REST to check that are properly structured. Subsequently, we have modeled the functional

MACHINE **Functional_Machine**
REFINES **Structural_Machine**
SEES **Functional_Context**
INVARIANTS
inv1 : $\forall n \cdot (n \in dom(ProtSupport) \land n \notin ran(NodRsc))$
$\Rightarrow ProtSupport(n) \neq CoAP$
inv2 : $\forall high \cdot high \in cnnHigh$
$\Rightarrow (ran(\{high\}) \not\subset RepSupport \sim [\{XML\}]) \land$
$ran(\{high\}) \not\subset ProtSupport \sim [\{HTTP\}])$
inv3 : $\forall proxy \cdot proxy \in NodDeg \sim [\{Proxy\}])$
$\Rightarrow (NodesRoles \sim [\{Client\}] \cap \{proxy\} \neq \varnothing) \land$
$(NodesRoles \sim [\{Server\}] \cap \{proxy\} \neq \varnothing)$
END

Fig. 6: The Event-B *Functional_Machine*

properties of REST elements to guarantee a correct execution for IoT resource-oriented applications. Indeed, IoT elements (devices) are expected to exchange and collect the data during the applications' execution. In this section, we model the operational requirements of the connected devices to ensure their consistent communications. Three operational properties are analysed at this level: **Reachability**, **Availability** and **Compatibility**. Wherefore, in this section, we introduce the *'Check-Reachability'* event to check whether the connected endpoints are attainable. For instance, they may remain unreachable when interruptions in end-to-end connectivity are encountered. Such event enables us to avoid deadlock situations where for example a device is exhausted in terms of energy, discarded and not willing to communicate. Subsequently, we perform the *'CheckAvailability'* event to ensure that the mandatory resources to manipulate by a device are available. In fact, they must be directly accessible or dynamically navigable by other resources. We also define the *'CheckCompatibility'* event to verify the matching of data formats supported by the endpoints and the circulating resources. This event enables us to avoid incompatibility issues that may occur when for example the format of a resource is not recognized by the device that queries about it. Therefore, we define the extended *"Operational_Context"* (Fig. 7) and refined *"Operational_Machine"* (Fig. 8), to add operational features to the model.

Initially, we model **O-RQ1** in the *"Operational_Context"* where we define new constants typed in AXIOMS as follows:

- $Exchange$ denotes the pairs of connected devices (sender, receiver), where the sender should implement client role and the receiver acts as a server (**O-RQ1**).
- $NodesModes$ is a partial function that associates to a given node a $Sleep$ or a $Sleepless$ mode (*axm2*, *axm3*).
- To each node, we have assigned a $sleepTiming$ and an estimated maximum sleeping time $EMST$ (*axm4*, *axm5*).
- $QueriedRep$ and $RscLookFor$ correlate respectively a given node to data types and resources that queries about.
- $RscRep$ denotes the resource media type (*axm8*).

The above context is seen by *"Operational_Machine"*. In this third refinement level, we define new variables as follows:

CONTEXT **Operational_Context**
EXTENDS **Functional_Context**
CONSTANTS
$Exchange\ QueriedRep\ EMST\ RscLookFor\ \ldots$
AXIOMS
axm1 : $Exchange \in Nodes \leftrightarrow Nodes \land (\forall r \cdot r \in Exchange$
$\Rightarrow (NodesRoles \sim [\{Client\}] \cap dom(\{r\}) \neq \varnothing)$
$\land (NodesRoles \sim [\{Server\}] \cap ran(\{r\}) \neq \varnothing))$
axm2 : $Modes \in \{Sleep, Sleepless\}$
axm3 : $NodesModes \in Nodes \nrightarrow Modes$
axm4 : $sleepTiming \in Nodes \nrightarrow \mathbb{N}$
axm5 : $EMST \in Nodes \nrightarrow \mathbb{N}$
axm6 : $QueriedRep \in Nodes \nrightarrow MediaTypes$
axm7 : $RscLookFor \in Nodes \leftrightarrow Resources$
axm8 : $RscRep \in Resources \nrightarrow MediaTypes$
END

Fig. 7: The Event-B *Operational_Context*

- $communication$ is a subset of the $Exchange$ set that refers to the current communicating devices (*inv1*).
- $Reachability$, $Compatibility$ are evaluated to TRUE if both sender and receiver are reachable and compatible.
- $Availability$ indicates whether the resource is accessible.

MACHINE **Operational_Machine**
REFINES **Functional_Machine**
SEES **Operational_Context**
VARIABLES
$communication\ Sender\ Reachability\ \ldots$
INVARIANTS
inv1 : $communication \subseteq Exchange$
inv2 : $Sender \subseteq Nodes \land Receiver \subseteq Nodes$
inv3 : $Reachability \subseteq BOOL$
inv4 : $Availability \subseteq BOOL$
inv5 : $Compatibility \subseteq BOOL$
END

Fig. 8: The Event-B *Operational_Machine*

Both connected endpoints must be reachable to avoid possible deadlock situations that occur during IoT communications. Namely, for each device waiting to receive a required message there will be a receiver willing to send it. Such constraint is ensured by modeling **O-RQ2** while performing the *'Check-Reachability'* event (Fig. 9) that we proceed to its description.

During the current communication (*grd1*), the receiver (*grd2*) can move freely without knowing in advance mobility parameters and can even definitely disappear from the IoT system. In the *grd3*, we assume that the receiver physically exists while evaluating the variable $existed$ to TRUE. The receiver is considered to be reachable ($reachable = TRUE$) (*grd5*) when the following requirements are fulfilled : (1) it is awake at a given instant or it is into a $sleep$ mode but its estimated maximum sleeping time has not been yet reached ($exhausted \neq TRUE$) and, (2) ($existed = TRUE$).

CheckReachability $\widehat{=}$
ANY com receiver sender existed exhausted reachable
WHERE
grd1: $com \in Exchange \setminus communication$
grd2: $receiver \in ran(\{com\}) \wedge sender \in dom(\{com\})$
grd3: $NetNodes[\{receiver\}] \neq \varnothing \Rightarrow existed = TRUE$
grd4: $(receiver \in NodesModes \sim [\{Sleep\}]$
$\wedge sleepTiming(receiver) > EMST(receiver))$
$\vee existed \neq TRUE$
$\Rightarrow reachable = FALSE \wedge exhausted = TRUE$
grd5: $(receiver \in NodesModes \sim [\{Sleepless\}]$
$\vee exhausted \neq TRUE) \wedge existed = TRUE$
$\Rightarrow reachable = TRUE$
THEN
act1: $communication := communication \cup \{com\}$
act2: $Receiver := \{receiver\}$ act3: $Sender := \{sender\}$
act4: $Reachability := reachable$
END

Fig. 9: The event *'CheckReachability'*

However, the receiver is unreachable ($reachable = FALSE$) (*grd4*) when (1) it is $asleep$ and its associated $EMST$ has been exceed or, (2) simply when ($existed \neq TRUE$). Once triggered, the variable $Reachability$ is evaluated ($TRUE$ or $FALSE$) (*act4*) and the event verification is launched for next interactions (*act1*). If the variable $Reachability$ of the above event is evaluated to TRUE, it automatically activates the event *'CheckAvailability'* (*grd1*)(Fig.10) that in turn checks the availability of the resource hosted on the $Receiver$ (*act2*) that the $Sender$(*act3*) queries about it, while covering **O-RQ3**.

CheckAvailability $\widehat{=}$
ANY available found
WHERE
grd1: $Reachability = TRUE$
grd2: $RscLookFor[Receiver] \subseteq NodeRsc \sim [Receiver]$
$\vee(\exists r, e.(r \mapsto e \in EmbeddedRsc \Rightarrow$
$RscLookFor[Receiver] = \{e\}$
$\wedge NodeRsc \sim [Receiver] = \{r\}))$
$\Rightarrow available = TRUE \wedge found = TRUE$
grd3: $found \neq TRUE \Rightarrow available = FALSE$
THEN
act1: $Availability := available$
END

Fig. 10: The event *'CheckAvailability'*

To confirm the accessibility of the resource looking for ($available = TRUE$) (*grd2*) (Fig. 10), the following properties should be satisfied: (1) the resource running on the receiver is directly offered or, (2) it is dynamically navigable, it means there exist a resource r hosted on the receiver that is linked to an embedded resource e that in turn conforms to the queried resource. After lookup, if the resource is not found ($found \neq TRUE$), $available$ is set to FALSE (*grd3*). Once launched, this event evaluates the variable $Availability$ (*act1*). If $TRUE$, the event *'CheckCompatibility'* (Fig.11) is activated.

To guarantee correct communications for IoT devices, we formalise the matching process through *'CheckCompatibility'* event in respect to **O-RQ4**. IoT devices are compatible ($compatible = TRUE$) (*grd4*) if the content negotiation is fully meet as follows ($negotiation = TRUE$): (1) the media format of the captured resource $rscLookRep$ (*grd2*) matches with the media type queried by the $Sender$ (client) (*grd3*) and, (2) $rscLookRep$ is supported by the $Receiver$ and, (3) there exist a data format supported by the $Sender$ that matches with $rscLookRep$. If a mismatch is detected, $compatible$ is set to FALSE (*grd5*). Once the negotiation is performed, $act1$ is triggered and the variable $Compatibility$ is evaluated.

CheckCompatibility $\widehat{=}$
ANY rscLookRep compatible negotiation
WHERE
grd1: $Reachability = TRUE \wedge Availability = TRUE$
grd2: $rscLookRep = RscRep[RscLookFor[Receiver]]$
grd3: $QueriedRep[Sender] = rscLookRep$
$\wedge RepSupport[Receiver] \cap rscLookRep \neq \varnothing$
$\wedge(\exists r.r \in \{RepSupport[Sender]\} \Rightarrow r = rscLookRep)$
$\Rightarrow negotiation = TRUE$
grd4: $negotiation = TRUE \Rightarrow compatible = TRUE$
grd5: $negotiation \neq TRUE \Rightarrow compatible = FALSE$
THEN
act1: $Compatibility := compatible$
END

Fig. 11: The event *'CheckCompatibility'*

D. Modeling the Behavioral Requirements

So far, we have modeled the operational properties of the connected IoT devices. In some cases, it is necessary to interfere mediators in the form of proxies to assist IoT devices that do not have the resources required to communicate. At this abstraction level, two behavioral properties are studied: **Proxy Interference** and **Proxy Conflicts**. We preliminary introduce the *'ProxyInterference'* event to analyse the degree of devices' constraints and the matching of their protocols to detect the behavioral mismatches and model the cases where the brokers aid is required. Subsequently, we carry out the *'ProxyConflicts'* event to handle exceptionable conflicts that may occur when a shared proxy assists many devices simultaneously, while its capacity in terms of tasks it can handle in parallel is limited. Therefore, we define the extended *"Behavioral_Context"* to add a constant called $PrxCapacity$ denoting the proxy capacity. The context is seen by the *"Behavioral_Machine"* where we define $PrxTasks$ ($PrxTasks \in Nodes \rightarrow \mathbb{N}$) and $PrxLinks$ ($PrxLinks \in Nodes \leftrightarrow Nodes$) that refer respectively to the number of tasks handled by a given proxy at an instant and the links that join the $Sender$ and $Receiver$ to the proxy.

Fig. 12 illustrates the *'ProxyInterference'* event. To ascertain whether a device does not have the capacity to establish direct communications, we locate the networks over which it runs and we eventually recognize its relative informations (e.g supported protocols, constraints degrees, etc). To sum up, a proxy behaves toward assisting the device ($proxy = TRUE$) if the requirements **B-RQ1** and **B-RQ2** are fully satisfied as follows: (1) the $Sender$ is so severely constrained (*highly-constrained*) and it belongs to the same *CNN* cluster of the $Receiver$ as well (*grd1*) or, (2) the $Sender$ is classified as a *quite-constrained* node and it does not support the protocol $CoAP$ that is especially designed to cope with the devices that have limited resources. A device that is able to interact directly without the mediators' interference ($proxy = FALSE$) should meet the following requirements : (1) the $Sender$ is quite-constrained but employs CoAP protocol to interact or, (2) it is less-constrained (resp. unconstrained) (*grd2*). Once triggered, this event evaluates the variable $Proxying$ (*act1*).

ProxyInterference $\widehat{=}$
ANY proxy
WHERE
grd1: $(cnnHigh \sim [Sender] = NetNodes[Receiver])$
$\vee((cnnQuit \sim [Sender] \neq \varnothing)$
$\wedge(\forall p.p \in \{ProtSupport(Sender)\} \Rightarrow p \neq CoAP))$
$\Rightarrow proxy = TRUE$
grd2: $((cnnQuit \sim [Sender] \neq \varnothing)$
$\wedge(\exists p.p \in \{ProtSupport(Sender)\} \rightarrow p = CoAP))$
$\vee cnnUnconst \sim [Sender] \cup cnnLess \sim [Sender] \neq \varnothing$
$\Rightarrow proxy = FALSE$
THEN
act1: $Proxying := proxy$
END

Fig. 12: The event *'ProxyInterference'*

As stated above, the proxies assist the devices that are not capable to communicate. However, each proxy is limited by a given maximum capacity which may pose serious conflicts when dealing with parallel tasks. In the event *'ProxyConflicts'* (Fig. 13) the selection of the appropriate neighbour proxy (*grd1*) is principally based on the proxies capacity. Each time a proxy has exceed the maximum capacity (*grd2*); the system passes the task to the next neighbour proxy included in the same $Sender$ network. The variable $neighborsProxy$ stores all the candidate proxies. However, the final winner proxy, which is selected for the inter-mediation, is modeled through the variable $proxy$ (*grd4*). Afterwards, the proxies tasks in progress and the internal proxies/devices links are updated for each event trigger (*act1*, *act2*).

VII. Verification and Validation

A. Proof Obligations (POs)

As a result of the mathematical sound way that Event-B model possesses, it is simple to supply experiments that check our proposed model consistency. In fact, The use of formal

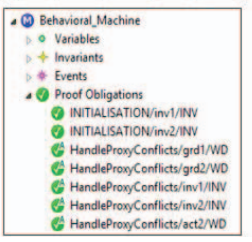
ProxyConflicts $\widehat{=}$
ANY prxNearSender neighbourProxy Links proxy t
WHERE
grd1: $prxNearSender{=}cnnProxy[NetNodes[Sender]]$
grd2: $neighbourProxy \in prxNearSender$
$\wedge PrxTasks(neighbourProxy) <$
$PrxCapacity(neighbourProxy)$
grd3: $Links \in Sender \rightarrow \{neighbourProxy\}$
$\wedge proxying = TRUE \wedge t \in \mathbb{N}$
grd4: $proxy \in ran(Links) \wedge Compatibility = TRUE$
THEN
act1: $PrxTasks(proxy) := PrxTasks(proxy) + t$
act2: $PrxLinks := PrxLinks \cup Links$
END

Fig. 13: The event *'ProxyConflicts'*

methods is nowadays a necessity to provide a rigorous reasoning and mathematical proofs to create reliable applications for complex IoT systems and to avoid their incorrect behavior.

POs are produced by Rodin platform to formally prove the Structural, Functional, Operational and Behavioral properties, in respect to four verification axes. Indeed, vigorous tools assist the proof process established to ensure that the events related to four sub-models preserve their invariants. Our verified model has successfully fulfilled its 57 POs, where 70% of them are automatically discharged by the automatic prover. Fig. 14 shows the proof statistics of the Operational and Behavioral sub-models. As given, the POs denoted by "A" symbol are automatically discharged. However, the remaining POs are interactively discharged.

Element Name	Total	Auto	Man.	Rev.	Und.
TheVerificationMo...	**57**	**40**	**17**	**0**	**0**
Structural_Machine	20	15	5	0	0
Functional_Machine	16	12	4	0	0
Behavioral_Machine	14	9	5	0	0
Operational_Machine	7	4	3	0	0



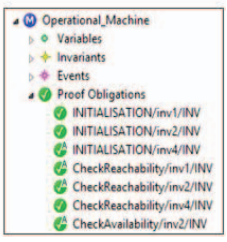


Fig. 14: POs of the *Operational* and the *Behavioral Models*

B. ProB Animation

The second validation of our model is accomplished through the use of ProB-based Rodin that enables us to carry out an automated animation of its specification. It offers the possibility to play different scenarios by shown at each step the variable's values and by distinguishing the specification behavior and distinguishing activated events from deactivated ones. Before proceeding with the animation, we gave values

to carrier sets, constants and variables. The animation consists of the following steps (see Fig. 15):

1) We first trigger the SETUP-CONTEXT event which gives values to the constants and context's carrying sets.
2) Then, we trigger the INITIALIZATION event to place our model in its initial state.
3) We finally proceed to the steps of the scenario to be verified. Considering the communication ($sensor3 \mapsto UnN1$) of our motivating example. Once triggering the event '$CheckReachability$', the variable $Reachability$ is evaluated to True since both the $Sender$ and $Receiver$ are attainable. Hence, the event '$CheckAvailability$' is automatically activated (The green one). Similarly, the variable $Availability$ is set to TRUE since the resource to manipulate by the sender $sensor3$ ('$EmergencyNotification$') is available. And thus, the event '$CheckCompatibility$' is activated. Given that the connected endpoints match in respect to their supported data formats and the captured resource format ($JSON$) and so on, the variable $Compatibility$ is also set to TRUE. Up to this stage, this communication is partially satisfied, as it fully preserves the INVARIANTS and GUARDS related to *"Operational_Machine"*. Subsequently, the event '$ProxyInterference$' of the *"Behavioral_Machine"* is activated seeing that $Compatibility = TRUE$. Once triggered, the variable $Proxying$ is set to TRUE, since the proxies should assist the interaction. Therefore, the message will be routed to $proxy4$ ($PrxTasks(proxy4) = 2$) since $proxy1$ is saturated ($PrxTasks(proxy1) = 1$) after calling the event '$ProxyConflicts$'. The same process is applied to check the consistency of the next interaction ($QN \mapsto sensor3$) that is directly satisfied without the proxies interference.

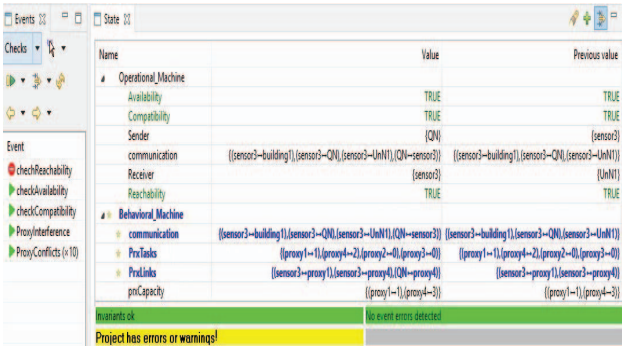


Fig. 15: Validation of the proposed Model

Conclusion

In this paper, we proposed an Event-B based model to verify the architecture of IoT-Cloud environments over which the resources-oriented applications are deployed. Subsequently, we check the correctness of communications that occur during the execution of IoT applications, in order to avoid possible deadlock situation that may occur. Our model comprises four abstract levels with respect to four verification levels: Structural, Functional, Operational and Behavioral. We succeeded to rigorously verify and validate our approach by means of the PROB animator and the POs. We are continuing to work on the remaining issues and will present them in future papers. We plan in the near future, to extend this work by examining the elasticity of IoT communications in order to improve non-functional properties related to QoS and performance.

References

[1] V. Lesch, M. Züfle, A. Bauer, L. Iffländer, C. Krupitzer, and S. Kounev, "A literature review of iot and cps—what they are, and what they are not," *Journal of Systems and Software*, vol. 200, p. 111631, 2023.

[2] M. M. Sadeeq, N. M. Abdulkareem, S. R. Zeebaree, D. M. Ahmed, A. S. Sami, and R. R. Zebari, "Iot and cloud computing issues, challenges and opportunities: A review," *Qubahan Academic Journal*, pp. 1–7, 2021.

[3] A. K. Tyagi, S. Dananjayan, D. Agarwal, and H. F. Thariq Ahmed, "Blockchain—internet of things applications: Opportunities and challenges for industry 4.0 and society 5.0," *Sensors*, vol. 23, p. 947, 2023.

[4] R. T. Fielding, "Architectural styles and the design of network-based software architectures; doctoral dissertation," 2000.

[5] F. Pereira, R. Correia, P. Pinho, S. I. Lopes, and N. B. Carvalho, "Challenges in resource-constrained iot devices: Energy and communication as critical success factors for future iot deployment," *Sensors*, vol. 20, no. 22, p. 6420, 2020.

[6] G. Bouloukakis, N. Georgantas, P. Ntumba, and V. Issarny, "Automated synthesis of mediators for middleware-layer protocol interoperability in the iot," *Future Generation Computer Systems*, vol. 101, 2019.

[7] X. Zhou, Y. Jia, C. Bai, H. Zhu, and S. Chan, "Multi-object tracking based on attention networks for smart city system," vol. 52, p. 102216, 2022.

[8] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7252

[9] D. Gourley and B. Totty, *HTTP: the definitive guide*. " O'Reilly Media, Inc.", 2002.

[10] M. Souad, B. Faiza, and H. Nabil, "Formal modeling iot systems on the basis of biagents* and maude," in *2020 International Conference on Advanced Aspects of Software Engineering (ICAASE)*. IEEE, 2020, pp. 1–7.

[11] V. Riccio, A. Fasolino, N. Amatucci, V. De Simone, and D. Amalfitano, "Towards a thing-in-the-loop approach for the verification and validation of iot systems," 11 2017.

[12] A. Fortas, E. Kerkouche, and A. Chaoui, "Formal verification of iot applications using rewriting logic: An mde-based approach," vol. 222, p. 102859, 2022.

[13] B. Costa, P. F. Pires, and F. C. Delicato, "Modeling iot applications with sysml4iot," in *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2016, pp. 157–164.

[14] K. Inçki, İ. Arı, and H. Sözer, "Runtime verification of iot systems using complex event processing," in *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*. IEEE, 2017, pp. 625–630.

[15] M. Tappler, B. K. Aichernig, and R. Bloem, "Model-based testing iot communication via active automata learning." IEEE, 2017, pp. 276–287.

[16] K. İnçki and I. Ari, "A novel runtime verification solution for iot systems," *IEEE Access*, vol. 6, pp. 13 501–13 512, 2018.

[17] M. Diwan and M. D'Souza, "A framework for modeling and verifying iot communication protocols," 10 2017, pp. 266–280.

[18] J. Song, D. Karagiannis, and M. Lee, "Modeling method to abstract collective behavior of smart iot systems in cps," *Sensors*, no. 13, 2022.

[19] J.-R. Abrial, *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.