Toward a Correct Implementation of LwM2M Client with Event-B

Ines Mouakher[1], Fatma Dhaou[1] and J. Christian Attiogbé[2]
[1]*University of Tunis El Manar, Tunis, Tunisia*
[2]*University of Nantes, Nantes, France*
{*ines.mouakher, fatma.dhaou*}*@fsegt.utm.tn, christian.attiogbe@univ-nantes.fr*

Keywords: IoT, Device Management, OMA LwM2M, Formal Verification, Event-B.

Abstract: Within the Internet of Things (IoT), billions of connected devices can collaborate anytime, anywhere and in any form in various domains of applications. These devices with minimal storage and computational power are based on standards and lightweight protocols. Due to the critical nature of application domains of the IoT systems, the verification of various properties is crucial. To this end, the benefits of using formal methods are widely recognized.

In this paper, we present an approach that integrates modelling and verification techniques, required for the specification of IoT systems, by exploiting the OMA Lightweight M2M (LwM2M) enabler. We propose a formal model of the LwM2M client, which is located in an LwM2M device, by building several mathematical models of discrete transition systems using Event-B. Indeed, we opt for a systematic and refinement-based approach that helps us to model and to verify gradually the specification. The Rodin tool is used to specify and verify the Event-B models. The generated Event-B models allow us to analyze and verify the behavior of LwM2M client that supports the latest LwM2M 1.1 version. Furthermore, it is a first step towards providing formally proven LwM2M client implementations.

1 INTRODUCTION

The IoT finds applications in various areas such as automotive, home automation, smart cities, energy efficiency, industry, agriculture, health, education and others. The IoT extends the internet connection to a diverse range of devices, and everyday things that are equipped with embedded technology, including sensors, actuators, RFID[1] tags, etc. With advancing technologies, these devices have communication and computing capabilities, and they can be remotely monitored and controlled.

Remote device management is a critical issue since the connected devices are diverse and their number is growing rapidly over time. Hence, it is essential to efficiently manage this huge amount of devices, as well as to equip the IoT with open standards to ensure its durability. This calls for abstraction of device management functions that have to hide the complexity, and to be independent of the technology. Such an abstraction can be provided by the OMA[2].

The OMA LwM2M enabler (Open Mobile Alliance, 2018a) is based on a client/server architecture. The device acts as the LwM2M clients, and the platform or the application acts as the LwM2M server. The LwM2M enabler defines the application layer communication protocol between an LwM2M server and an LwM2M client. We investigate only the client side (device). The target devices for this enabler are essentially resource-constrained devices; therefore, this enabler makes use of a light and compact protocol as well as an efficient resource data model. The LwM2M enabler includes the specification of application protocol for device management, and service enablement for LwM2M devices. There are several IoT platforms based on this standard, which explains the various implementations of the LwM2M specification. Some of them implement either the server side or the client side. Others implement both of them (client and server). These implementations support most of the features of the LwM2M protocol.

Our final objective is the correct implementation of the LwM2M client; but we proceed incrementally. Indeed, in this article, we deal with the preliminary steps of modelling and analysing the LwM2M specification. We use formal methods to help in ensuring correctness, consistency, and clarity of the LwM2M specification. Since a well-defined and verified pro-

[1]RFID = Radio-Frequency IDentification.

[2]OMA = Open Mobile Alliance, https://www.omaspecworks.org

tocol specification can reduce the cost for its implementation and maintenance. Modelling and analysis are important steps of the protocol development life cycle from the viewpoint of protocol engineering.

Event-B (Abrial, 2010) is a state-based formal method, hence we can easily describe resources data model associated with client. The Event-B method is dedicated to describe event-driven reactive systems; it allows us to describe interactions between LwM2M clients and servers. We take advantage of the Event-B method refinement, which permits us to represent the LwM2M client at different abstraction levels, and it allow us to generate correct-by-construction code from Event-B models.

In this work, we propose to model the behavior of the LwM2M client as a discrete transition system by building its mathematical models with Event-B. Our models are compliant with the enabler technical specification for LwM2M 1.1. The generated Event-B models allow us to analyze and verify the behavior of the LwM2M client. The Event-B method and its tools support are used to specify, validate and simulate our models. We choose an approach that gradually models a complex behavior of the client using layered refinement, such that in each refinement we focus on modelling and verifying a subset of the enabler specification. The proposed approach is based on the systematic rules to generate the Event-B models. To better describe our models, we use State Machine Diagrams (SMD) to specify protocols involved in the interactions between LwM2M client and servers. These diagrams allow us to summarize the specification in a graphical way, and to better explain some Event-B models, since they are presented as a translation of SMD. Furthermore, we define the rules to translate interaction patterns into Event-B such as request/response and subscribe/notify.

This paper is structured as follows: Sec. 2 is dedicated to a brief presentation of LwM2M enabler and some background notions of the Event-B method, In Sec. 3, we explain our approach of the formal modelling of the LwM2M client. In Sec. 4, we summarize the Event-B development of the LwM2M client.In Sec. 5, we discuss some related work. Finally, Sec. 6 presents our conclusions and future work.

2 BACKGROUND

2.1 Brief Introduction to LwM2M

With LwM2M enabler (Open Mobile Alliance, 2018a; Open Mobile Alliance, 2018b), OMA has responded to the market demand for a common standard to the managing lightweight and low power devices on a variety of networks that is necessary to achieve the potential of the IoT.

A client-server architecture (Fig. 1) is introduced for the LwM2M enabler, where the LwM2M device acts as an LwM2M client and, platform or application acts as the LwM2M server. The LwM2M enabler has two components: an LwM2M server and an LwM2M client.

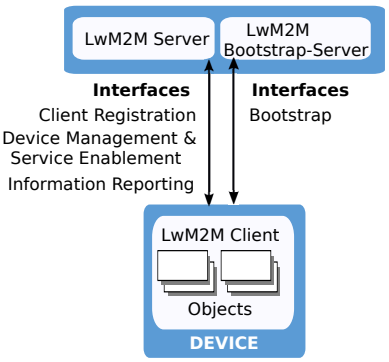


Figure 1: The overall architecture of the LwM2M enabler (Open Mobile Alliance, 2018a).

Four interfaces are specified between the LwM2M client and the LwM2M servers: Bootstrap Interface (I1), Registration Interface (I2) Device Management and Service Enable Interface (I3), and Information Reporting Interface (I4). The operations for the four interfaces can be classified into uplink or downlink operations. The LwM2M operations for each interface are mapped to CoAP Methods. The CoAP protocol provides a request/response interaction model. The LwM2M client must ignore the LwM2M server operations on the *I*3 interface and on the *I*4 interface during a server initiated bootstrap and until it received its registration acknowledgement.

The LwM2M objects are identified with an *Object ID* (i.e. *Security* object ID:0, *Server* object ID:1, *Access Control* object ID:2, *Device* object ID:3, *Connectivity Monitoring* object ID:4, *Firmware Update* ID:5, *Location* object ID:6 and *Connectivity Statistics* object ID:7). The resource data model is extensible, thus any companies can define additional LwM2M objects.

2.2 Brief Introduction to Event-B

Event-B (Abrial, 2010) is a formal method, which is used in numerous industry projects. It is dedicated to the modelling of critical systems. The basis for the mathematical language in Event-B is first logic and a typed set theory. Set-theoretical notation of Event-B

defines different types of relations and functions[3] enhanced by different properties. The Event-B method is supported by the Rodin toolkit which comprises editors, theorem provers, animators and model checkers. Two basic constructs compose Event-B models: contexts and machines. The static part of models is defined in a context. The behavior of a system is modelled as a transition system. A *machine* specification usually defines variables that specify the states of a system and guarded *events* that specify the system's transitions.

3 FORMAL MODELLING OF THE LwM2M CLIENT

The behavior of the LwM2M client is given mainly by the interactions with the servers through four interfaces. It can also have some internal processes. The supported operations are based on different communication models: a request/response interaction model, synchronous or asynchronous model and subscribe/notify model, we define how we model each of them in Event-B (Sec. 3.1).

In addition, the performance of these operations depends on the state of the client and on the protocol required by the interfaces. For example, the LwM2M client must ignore LwM2M server operations on the *I3* interface during a server initiated bootstrap and until it received its registration acknowledgment. For this purpose, we use state transition diagram that allows us, on the one hand, to summarize the description of the client, and on the other hand, to give a preliminary model based on state and transition which structures the translation into Event-B (Sec. 3.2).

3.1 Modelling Interaction Patterns

3.1.1 Request/Response Pattern

The operations of the LwM2M client are based on a request/response interaction pattern. We consider that there is no loss of messages. Then, for each received or sent operation, the client must respectively send or receive its response. For each operation, there are two kinds of response: $failure$ and *success*.

For each operation, we associate one event for the request. For some operations, we associate two events to their response in order to distinguish between $failure$ and *success* responses; for the others, we associate one event that will group the two kinds of response. We use the naming convention as illustrated by the examples in Tab. 1.

Table 1: Naming convention for Event-B events.

Operations	Event-B events
Request is uplink operation from $I1$ interface	*s_Request_I1*, *r_Request_I1_Response*
Discover is downlink operation from $I1$ interface	*r_Discover_I1*, *s_Discover_I1_Response*

We use one variable to synchronize between request and response of each interface. Then we define *partial functions* that map the operations with the corresponding interface. For the $I1$ interface, the client communicates with one bootstrap server. The *I1_State* variable is defined as a partial function that associates *True* to an operation if it waits for a response otherwise it is *False*, i.e. $I1\_State \in OPERATIONS\_I1 \pfun BOOL$.

For the rest of the interfaces $I2$, $I3$ and $I4$, the client can interact with several servers, and can receive several simultaneously requests from different servers. The variable *Ii_State* (with $i \in \{1,2,3\}$) is a partial function that associates an operation with the set of servers waiting for a response for that operation (e.g. $I3\_State \in OPERATIONS\_I3 \pfun \mathbb{P}(dom(ServersState))$[4]).

3.1.2 Subscribe/Notify Pattern

The $I4$ interface allows the server to observe any changes in resource values on the LwM2M client based on subscribe/notify pattern. The server initiates the conversation by sending an *Observe* or *Observe-Composite* operation that expresses interest in receiving notification messages about an object, an object instance or a resource. Then, the client delivers a stream of $Notify$ messages to the server at the place in which data becomes available. When a *Cancel Observation* or a *Cancel Observation-Composite* operation is performed, the observation terminates. Particularly, there are two sequences of operations based on subscribe/notify pattern: *Observe/Notify/Cancel Observation* and *Observe-Composite/Notify/Cancel Observation-Composite*.

For $I4$ interface, in addition to the *I4_State* variable introduced to support request/response pattern, the variable *tokens* is introduced to support subscribe/notify pattern. The variable *tokens* is used as a set of tokens.This variable is used to match the asynchronous notifications with previous *Observe* opera-

[3]Total ($\tfun$) or partial ($\pfun$) functions, total ($\tinj$) or partial ($\pinj$) injections, total ($\tsur$) or partial ($\psur$) surjections and bijections ($\tbij$).

[4]dom: domain, $\mathbb{P}$: powerset.

tions. For each successful *Observe* operation, a token is added to the *tokens* variable and it will be deleted when the client receive *Cancel Observation*.

3.1.3 Synchronous and Asynchronous Communication Pattern

The OMA LwM2M enabler don't specify if the communication is synchronous or asynchronous then the implementation can support both of them. We only consider the context of synchronous communication. In the context of uplink operation, the client can not start another sending operation until it has received the response to the already sent one; but it can receive operations (downlink operation). In the context of downlink operation, the client can receives operations from several servers. Other servers are assumed to communicate in a synchronous manner. Therefore, if the client has received an operation from a server, he can not receive another one until he has sent the response of the first one.

The asynchronous communication can be easily supported by our models by modifying the guards of operations.

3.1.4 Example of Interaction Model Events

To illustrate the rules proposed in the Sec. 3.1.1, Sec. 3.1.2 and Sec. 3.1.3, we give the example of the two Event-B events associated to $Notify$ operation (Fig. 2).

Event s_Notify_I4 ⟨ordinary⟩ $\widehat{=}$
any s
where
grd3: $I2\_State = (OPERATIONS\_I2 \times \{\varnothing\})$
grd4: $I3\_State = (OPERATIONS\_I3 \times \{\varnothing\})$
grd5: $I4\_State = (OPERATIONS\_I4 \times \{\varnothing\})$
grd6: $s \in tokens$
then
act1: $I4\_State(Notify) := I4\_State(Notify) \cup \{s\}$
end
Event s_Notify_I4_Response ⟨ordinary⟩ $\widehat{=}$
any s
where
grd5: $(Notify \in dom(I4\_State)) \Rightarrow (s \in I4\_State(Notify))$
grd6: $s \in tokens$
then
act1: $I4\_State(Notify) := I4\_State(Notify) \setminus \{s\}$
end

Figure 2: Event-B events associated to $Notify$ operation.

- Request/response pattern : the $Notify$ operation is an uplink operation from $I4$ interface, then it has two events: *s_Notify_I4* and *s_Notify_I4_Response*. The variable *I4_State* appears in the *guard* of the two events (*grd*5 in Fig. 2) and it is updated in their action (*act*1 in Fig. 2). The variable *I4_State* allows us to synchronize between request and response of the operation $Notify$.
- Subscribe/notify pattern: the $Notify$ operation can be performed (i.e. the *s_Notify_I4* and *s_Notify_I4_Response* events can be executed) only if the considered server has a token in the *tokens* variable (*grd*6 in Fig. 2)
- Synchronous communication pattern: the event *s_Notify_I4* is sending event and it can be performed only if there is not another received or sent operation waiting for response (*grd*3, *grd*4 and *grd*5

3.2 Modelling Interfaces Protocol

3.2.1 Description of the LwM2M Client with a State Machine Diagram

The abstract behavior of the LwM2M client can be shown as a SMD in Fig. 3. This diagram includes two composite states, "attempting to Bootstrap" and "Bootstrap Success", and one simple state "Bootstrap Failure".

The "Attempting to Bootstrap" state has three nested states: "Attempting to Factory Bootstrap", "Attempting to Bootstrap from Smartcard", and "Attempting Client Initiated Bootstrap". In this composite state, the device sequentially try to bootstrap via *Smartcard* and/or *Factory Bootstrap* mode. If these two steps fail (i.e. the client has not any LwM2M *Server* object instances), the LwM2M client performs the *Client Initiated Bootstrap*. If the *Client Initiated Bootstrap* also fails, then the LwM2M client is in a deadlock state:"Bootstrap Failure". In the "Attempting Client Initiated Bootstrap" state, there is the "Client Initiated Bootstrap" activity. This activity consists of a set of interactions between the client and the bootstrap server through the operations of the $I1$ interface.

In the "Bootstrap Success" state, we introduce the operations of $I2$ interface: $Register$, $Update$ and $De\text{-}register$. After the success of bootstrap, the client must have at least one server account. As soon as the client registers to a server, the client can perform the other operations of $I2$ interface with this server as well as the operations of $I3$ and $I4$ interfaces. Finally, the client can *De-register* and stop interaction with the LwM2M server. In the "Bootstrap Success" state, we can deduce three substates: "Not Registered", "Registered" and "Fail Reg-

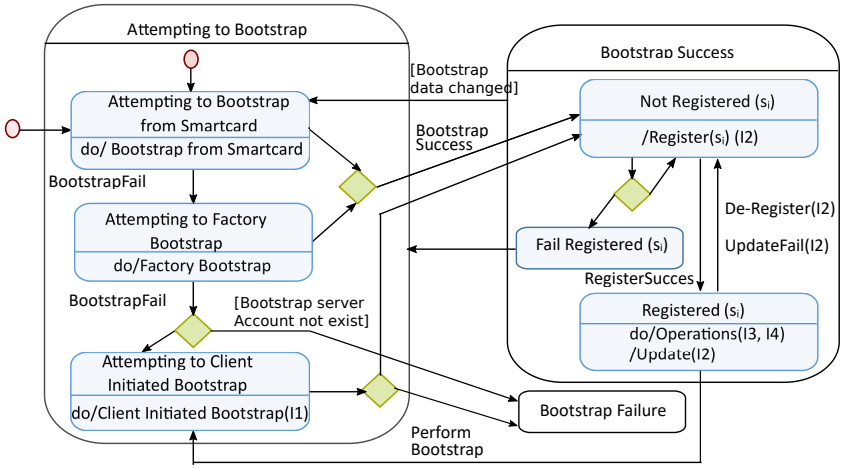


Figure 3: Behavior of an LwM2M client with a State Machine Diagram.

istered". Since the LwM2M client can register to several LwM2M servers, the state of client is defined by the set of LwM2M server's instances. The "Bootstrap Success" state is a composite state; it has three nested states: "Not Registered (s_i)", "Fail Registered (s_i)" and "Registered (s_i)"; each of them refers to a LwM2M server instance. If registration succeeds, the client can interact with the corresponding LwM2M server via operations on $I3$ and $I4$ interfaces. If the registration fails, the client can perform bootstrap or trying to register again.

In the "Not Registered (s_i)" state, the client can perform *Register* operation; upon the client receives a response it can change its state. In the "Registered (s_i)" state, there is an activity composed of interactions between the client and a server s_i. These interactions use $I3$ and $I4$ interfaces, and also *update* and *De-register* operations from $I2$ interface can be performed.

3.2.2 Translation of the State Machine Diagram into Event-B

We propose some rules to translate SMD to an Event-B model associated with the LwM2M client. The translation rules are not exhaustive since we restrict ourselves to the elements used in the SMD of Fig. 3. These rules are applied progressively since we rely on a refinement-based approach. The elements of SMD that will be covered, and its associated translation rules are summarized below.

Translation of a State. We associate a variable with the states (simple state and composite state) of a SMD, e.g. *ClientState*, and we ignore the sub-states. For each composite state, we associate a variable to its direct substates (i.e. it is not contained by any other state), e.g. *AttemptingToBootstrapState* and *ServersState*, and so on. We define two rules for the type of the generated variables.

The first rule applies when the generated variable is an enumerated set, which is generated from the states of the considered states. It is the same rule as the one proposed in (Snook and Butler, 2006) and called enumeration translation in iUML-B tool (Snook, 2014). For the SMD depicted in Fig. 3, we introduce two variables, *ClientState* that represents the three states of the SMD and *AttemptingToBootstrapState* that represents the three nested states of "Attempting to Bootstrap".

The second rule applies when the generated variable is a function that associates to each instance its state. We use another variable that represents the set of the considered instances.

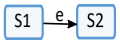
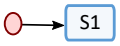
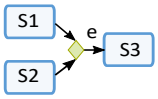
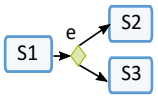
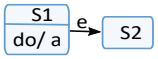
For the SMD depicted in Fig. 3, we introduce two new variables: *Servers* and *ServersState*. The first one represents the servers with which the client can interact. These servers are setting up in the bootstrap phase. The second *ServersState* variable represents the three nested states of "Bootstrap Succes" state and it is defined by the function that associates a state to each server.

$$ServersState \in servers \rightarrow SERVER\_STATE$$

Translation of a Transition. Transitions are translated into events. The Tab. 2 presents some of the rules used to generate Event-B events associated to activities and transitions: simple transition ($R1$), initial transition ($R2$), junctions ($R3$), choice ($R4$), and transition and activity ($R5$).

In these rules, $\langle in\ S1 \rangle$, $\langle initiated\ by\ S1 \rangle$ and $\langle becomes\ S2 \rangle$ depend on the data that represents the state. The rules $R5$ considers that there is an activity and an outcoming transition in the same state.

Table 2: Client' behavior translation rules.

Ri	SMD elements	Transition into Event-B
$R1$		$e \widehat{=}$ *when* $\langle in\ S1 \rangle$ *then* $\langle becomes\ S2 \rangle$
$R2$		*INITIALISATION* *then* $\langle initiated\ by\ S1 \rangle$
$R3$		$e \widehat{=}$ *when* $\langle in\ S1\ or\ S2 \rangle$ *then* $\langle becomes\ S3 \rangle$
$R4$		$e \widehat{=}$ *when* $\langle in\ S1 \rangle$ *then* $\langle become\ S2\ or\ S3 \rangle$ $e \widehat{=}$ *when* $\langle in\ S1\ and\ cond1 \rangle$ *then* $\langle becomes\ S2 \rangle$ $e' \widehat{=}$ *when* $\langle in\ S1\ and\ cond2 \rangle$ *then* $\langle becomes\ S3 \rangle$
$R5$		$a \widehat{=}$ *when* $\langle in\ S1 \wedge$ $ActProcessed = 0 \rangle$ *then* $\langle becomes\ S2$ $ActProcessed := 1 \rangle$ $e \widehat{=}$ *when* $\langle in\ S1 \wedge$ $ActProcessed = 1\ \rangle$ *then* $\langle becomes\ S2$ $ActProcessed := 0 \rangle$

When a state contains activity, triggering the transition marks the end of the activity. We use a variable *ActProcessed* that can take the value 0 or 1 to synchronize between the execution of the activity and the transitions in the same state. **Translation of an Activity.** A state can hold a list of internal actions and activities (do), that are performed while the element is in the state. We associate at least one event with each activity that depends on the description of the activity and the considered level of abstraction.

In the "Attempting to factory Bootstrap" and the "Attempting to Bootstrap from Smartcard" states, there is respectively "Factory Bootstrap" and "Bootstrap from Smartcard" activities. Each of these activities is translated into an Event-B event.

In the "Attempting to client initiated Bootstrap" state, the operations from Bootstrap Interface can be performed. This activity consists of a set of interactions between the client and the bootstrap server through the operations of $I1$ interface: *Request*, *Discover*, *Read*, *Delete*, *Write* and *Finished*. Bootstrap communication starts with *Bootstrap-Request* message and ends with a *Bootstrap-Finish* message. Between these two messages, the bootstrap server may configure the client with the necessary information. In the Event-B model, we introduce the events associated to the operations from $I1$ Interface and the $I1\_State$ variable.

In the "Registered(s_i)" state, the operations from $I3$ and $I4$ interfaces, and $Update$ and *De-register* operations from $I3$ interface can be performed. The events associated to these operations are added. The events associated to $I4$ interface are based on subscribe/notify pattern.

In the "Not registered (si)" state, the client can send *register* request to the server and wait for the response. If it receives *failure* response it can send *register* operation one more time.

4 EVENT-B DEVELOPMENT OF THE CLIENT

We summarize in this section the Event-B development of the LwM2M client.

4.1 Architecture of the Event-B Models

The summary of the hierarchy of specification presented in Sec. 3 is illustrated in Fig. 4. In the Event-B specifications, beside the *ctx* context, which gathers all the used sets, we use the following contexts:

- *ctx_Interfaces:* contains constants that are required for the definition of the client interfaces and their properties.
- *ctx_InterfaceInstances_Properties:* contains the properties on the client interfaces.

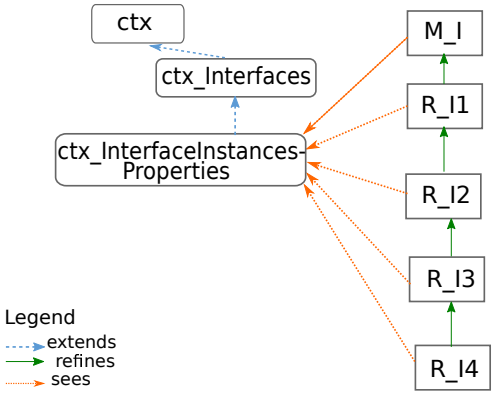


Figure 4: Development hierarchy.

We construct several successive models. In the first following five models, we focus on the modelling of operations associated with the four interfaces. Therefore, we focus more on the protocol aspect (order of execution of operations) which is considered in Event-B by protocol variables that allow synchronization between operations. Then in the last

model we introduce the resource data model. We use the following Event-B *machines*:

- *M_I*: we focus on the bootstrap process by introducing the four modes for bootstrapping: *Factory Bootstrap*, *Bootstrap from Smartcard*, *Client Initiated Bootstrap* and *Server Initiated Bootstrap*; no operation is considered yet.
- *R_I1*: we increase the first model with the description of the operations in *I*1 interface involved in *Client initiated bootstrap* and *Server initiated bootstrap* modes.
- *R_I2*: we refine *R_I1* machine by adding the description of the operations in *I*2 Interface. The client can interact with one or several LwM2M servers.
- *R_I3*: the machine *R_I3* refines the machine *R_I2*, and it contains new events associated with the *I*3 interface.
- *R_I4*: the machine *R_I4* refines the machine *R_I3*, and it is enhanced by the events associated to the *I*4 interface.

4.2 Proof Statistics

The development is formalized and proved using the Rodin platform[5](version 3.4). The verification of the Event-B models is provided via means of proof obligations (PO). The summary of the POs for the development is as follow: *M_I* (9 POs), *R_I*1 (23 POs), *R_I*2 (50 POs), *R_I*3 (68 POs) and *R_I*4 (55 POs). The POs are proved automatically (about 60%) or manually using the theorem provers of Rodin.

5 RELATED WORK

IoT protocols are the most crucial part of the IoT technology. Indeed, without them, hardware would be useless as the IoT protocols allow us to exchange data in a structured and meaningful way. The IoT communication protocols attract numerous researchers (Aziz, 2014), (Che and Maag, 2013), (Schmelzer and Akelbein, 2019), (A. Nikolov and Atanasov, 2016), (Thangavel et al., 2014), (Diwan and D'Souza, 2017), (Snook et al., 2017). These works deal with different issues and each of them is interested in distinct protocols or deal with different issues.

There are few works (Aziz, 2014), (Che and Maag, 2013) that provide formal semantics for some IoT protocols in order to check some communication properties.

Event-B method is already used to provide formal semantics for communication protocols in IoT. In (Diwan and D'Souza, 2017), the authors propose an approach to verify an IoT communication protocol through a framework in Event-B. They present models of MQTT, MQTT-SN and CoAP protocols, and they verify some communication properties.

In (Snook et al., 2017), a general approach is proposed for constructing and analysing security protocols using Event-B. The approach is based on abstraction, refinement, and a systematic modelling method using the UML state and class diagrams of iUML-B (Snook and Butler, 2006; Said et al., 2015). Due to the exponential growth of network endpoint devices, several studies have been made on protocols towards efficient remote device management (de C. Silva et al., 2019). In this paper, we focus on the LwM2M emerging open standard that meets the requirements for managing constrained devices. It is adopted by several manufacturers such as Microsoft and Intel. Several implementations of the LwM2M protocol stack became available in the last few years, such that the open source implementations Wakaama[6] and Leshan[7] that are, widely used.

Consequently LwM2M implementations attract numerous researchers. In (Schmelzer and Akelbein, 2019), the authors propose useful metrics for measuring device management capabilities on constrained nodes based on the LwM2M standard. They present their work as an upfront analysis for selecting an appropriate implementation and they give a comparison between two open source implementations: Wakaama and Leshan.

In (Thramboulidis and Christoulakis, 2016), the main contribution is the definition of a UML profile for the IoT, namely the UML4IoT profile. The authors define a UML profile based on the LwM2M protocol, designed to integrate the smart devices into industrial-level Iot-based systems. UML profile is used to automatically generate code and they propose a prototype implementation of the OMA LwM2M (v1.0) protocol based on meta programming.

To our knowledge, there is one work that provides formal semantics for this standard. In the paper (A. Nikolov and Atanasov, 2016), the authors define a formal approach to the verification of LwM2M server and client behaviour. Behavioral models of LwM2M server and client for connectivity management are proposed. Both models are formally described as labeled transition systems, and

[5]http://www.event-b.org/

[6]http://www.eclipse.org/wakaama/

[7]http://www.eclipse.org/leshan/

they are compliant with *Enabler Test Specification for Lightweight M2M (v1.0)*. Then, it is proved that both models are synchronized using the concept of weak bisimulation. The considered behavioral models of LwM2M server and client are at a very abstract level.

In our work, we provided a formal semantics for the LwM2M protocol by using Event-B method. We analyzed and verified the behavior of LwM2M client with The Rodin tool and the ProB model checker. We intend to generate automatically the code from the Event-B models that we have defined for the LwM2M protocol.

6 CONCLUSION

To better understand and trust the LwM2M enabler, formally derived and verified models should be defined. In this paper, we presented an approach of modelisation and verification for the specification of the IOT systems by exploiting the OMA lightweight M2M enabler. We focused on the behavior of the LwM2M client. We followed a refinement-based approach by building several formal models by using the Event-B method. Indeed, we proposed systematic rules for the translation of the LwM2M enabler toward Event-B. The refinement-based approach allows us to master the complexity of the model and facilitates the proof and the correction of the sub-models. The proof of the correctness of models, and the checking of consistency properties are made with the powerful tools of the Event-B platform.

We are conducting more experimentations on modelling with Event-B. We project to define an approach for providing code generation. A possible perspective consists to consider the modelisation of LwM2M server side and then to consider both sides (client and server). In the latest case, the checking of the compatibility between client and server must be established. Our future work will focus on validation, which includes running scenarios in the form of model acceptance test.

REFERENCES

A. Nikolov, E. P. and Atanasov, I. (2016). Formal verification of connectivity management models in m2m communications. In *2016 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 1–4.

Abrial, J.-R. (2010). *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 1st edition.

Aziz, B. (2014). A formal model and analysis of the mq telemetry transport protocol. In *Ninth International Conference, Availability, Reliability and Security (ARES)*, pages 59–68, Fribourg.

Che, X. and Maag, S. (2013). A passive testing approach for protocols in internet of things. In *Green Computing and Communications (GreenCom), IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 678–684.

de C. Silva, J., Rodrigues, J. J. P. C., Al-Muhtadi, J., Rabêlo, R. A. L., and Furtado, V. (2019). Management platforms and protocols for internet of things: A survey. *Sensors*, 19(3):676.

Diwan, M. and D'Souza, M. (2017). A framework for modeling and verifying iot communication protocols. In *Dependable Software Engineering. Theories, Tools, and Applications, SETTA 2017*, volume 10606 of *LNCS*, pages 266–280. Springer.

Open Mobile Alliance (2018a). Lightweight machine to machine technical specification: Core.

Open Mobile Alliance (2018b). Lightweight machine to machine technical specification: Transport layer.

Said, M. Y., Butler, M., and Snook, C. (2015). A method of refinement in uml-b. *Softw. Syst. Model.*, 14(4):1557–1580.

Schmelzer, P. and Akelbein, J.-P. (2019). Evaluation of hardware requirements for device management of constrained nodes based on the lwm2m standard. In *Proceedings of the 5th Collaborative European Research Conference (CERC 2019)*, pages 103–110.

Snook, C. (2014). iUML-B statemachines. In *Proceedings of the Rodin Workshop 2014*, SE'08, pages 29–30, Toulouse, France.

Snook, C. and Butler, M. (2006). Formal modeling and design aided by UML. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(1):92–122.

Snook, C. F., Hoang, T. S., and Butler, M. J. (2017). Analysing security protocols using refinement in iuml-b. In *NASA Formal Methods - 9th International Symposium, NFM 2017*, volume 10227 of *LNCS*, pages 84–98. Springer.

Thangavel, D., Ma, X., Valera, A., Tan, H., and Tan, C. K. (2014). Performance evaluation of mqtt and coap via a common middleware. In *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6. IEEE Press.

Thramboulidis, K. and Christoulakis, F. (2016). UML4IoT - A UML-based approach to exploit IoT in cyber-physical manufacturing systems. *Computers in Industry*, 82:259–272.