Multi-layered Model for Performance Evaluation of oneM2M-based IoT Solution

Samir Medjiah, Thierry Monteil, Marie-Agnès Peraldi-Frati, Luigi Liquori

Multi-layered Model for Performance Evaluation of oneM2M-based IoT Solution

Samir Medjhah[1,2], Thierry Monteil[1,3],
Marie-Agnès Peraldi-Frati[3], and Luigi Liquori[3]

[1] Université de Toulouse, Université Paul Sabatier, LAAS-CNRS, France
[2] Université de Toulouse, INSA, IRIT, France
[3] Université Côte d'Azur, Inria, France

Abstract. In this paper we evaluate the impact of standards in terms of performance and their applicability in the field of IoT system design and deployment. We focus on the global IoT oneM2M standard. Our objective is to evaluate a oneM2M-based IoT solution regarding different relevant Key Performance Indicators. We propose a multi layered-model of an IoT standardized solution, able to tackle applicative, infrastructure and deployment aspects. Based on this model, we are able to globally evaluate and analyze, through simulation, the adequacy of a deployment with respect to the initial applicative constraints and the chosen oneM2M standard implementation. In our case, the constraints are mix-critical coming from the e-Health remote monitoring of patient by their physician but also the management of the patient in case of vital emergency situation. By tuning the system configuration and parameters of the proposed applicative scenario, we evaluate, by simulation, the KPIs of a oneM2M-based IoT solution by exploiting (1) the different features of the standard, (2) the capabilities of the underlying infrastructure, and (3) the performance of the oneM2M stacks used in the solution. The simulation and performance evaluation are based on two tools developed by the authors. One is a specific profiler for oneM2M open-source stack, whereas the simulation and performance evaluation is build on top of the OMNeT++ discrete event simulator.

Keywords: ETSI · oneM2M Consortium · IoT Use Cases · Performance Evaluation · Meta-Modeling

1 Introduction

In the IoT system design, there is a need for rationalizing and dimensioning the devices-edge-cloud computing infrastructure and deployments, according to different criteria ranging from the amount of data and their velocity by the devices, to their analysis or the decision making located on the edge or the cloud computing parts. This design is highly dependent on the application domain and the specific extra-functional requirements (time, energy, sustainability, scalability, etc.) of the applications. Associated with these constraints, and to move away from siloed IoT application design, an additional strategic constraint is the need for interoperability. This requires the integration of standards in the design process and consequently, the additional choices and possible strategies in the use of these standards and their impact on the infrastructure deployments choices.

IoT standard such as oneM2M [3] provides a communication architecture and associated services that bring together devices, communications networks and IoT applications. This standardizes links between connected devices, gateways, communications networks and cloud infrastructure. oneM2M is highly supported by telecom operators, ICT solution companies, SMEs hardware designer and IoT stakeholders. There is a real need from these parties for methodologies and tools to support the design and planning phase of IoT networks, infrastructures, and behaviours. The ETSI SmartM2M Technical Committee, engaged in IoT standards, is highly supporting work that illustrates the different uses of the oneM2M standard in the different fields, but until now, there has been few work that addresses the problem of evaluating implementations (stacks) of oneM2M standard, from the point of view of performance in these different uses. This subject is not fully covered by standardization bodies, and can constitutes an obstacle for new IoT customers who desire to explore the potential of oneM2M for their specific use cases. The initiative by ETSI SmartM2M TC of a building a specific "task force" on performance evaluation, is a first step for an evaluation of the deployment scenarios of oneM2M-based IoT solutions.

This paper conducted inside the ETSI SmartM2M Technical Committee and hand-to-hand with oneM2M Consortium aims at proposing an assistance in the specification and the design of IoT-standardized systems, taking into account the application edge cloud continuum down to the deployment on a oneM2M infrastructure. We propose a multi layered-model of an IoT standardized system, able to tackle applicative, infrastructure, performance and deployment aspects of the system. Each layer of the model provide the concepts and artefacts that closely matches the specific needs of the different engineering aspects i.e.: the application development, oneM2M logical infrastructure, and the hardware deployment infrastructure. In addition, we propose a tooled-methodology able to discern between different oneM2M stacks and to assist the specification and the design of IoT systems. First, the objective is to measure, through a *profiler*, the relevant Key Performance Indicators (KPI) values resulting on a oneM2M stack execution, that will be injected in an *adhoc simulation library* to globally evaluate and analyze, the conformance of a deployment with respect to the initial applicative constraints and the chosen standard implementation.

Concerning the application domain, we put a special emphasis on the eHealth IoT domain because it gives us the possibility to envisage concrete scenarios characterizing either mix-critical constraints from the applications side but also CPU time and memory space constraints from the oneM2M infrastructure side. All of these constraints are important, sometimes essential to be verified for a conform deployment of an IoT application using oneM2M. We choose ACME [2] as a open-source oneM2M implementation stack, written in Python. By tuning the system configuration and parameters of the proposed applicative scenario, we evaluate, by simulation, the KPIs of an oneM2M-based IoT solution by exploiting (1) the different features of the standard, (2) the capabilities of the underlying infrastructure, and (3) the performance of the oneM2M used stacks.

The paper is organized as follows. Section 2 provides a necessary reminder of interesting works and results in the field of oneM2M standard, network simulation and IoT deployment principles. Section 3 presents our contribution in the specification of oneM2M standardized infrastructure with a multi-layered model that takes into account application, infrastructure, performance and deployment aspects. Section 4 implements the multi-layered model into the OMNeT++ [6], a discrete event network simulator. Section 5 presents an experimental proof of concept of one realistic scenario, using the scalability features of OMNeT++. Finally, Section 6 shows conclusions and future perspectives of this work.

2 Context and state of art

This section explore works in the different fields of IoT standardization, network simulation, as well as current trends and issues in the computing deployment and data storage on the continuum device, edge, cloud w.r.t. applications domains.

2.1 The oneM2M IoT standard

oneM2M [3] is a Consortium including five standardization organizations, bringing over 200 industrial partners and laboratories. This standardization initiative aims to delineate a suite of services, communication patterns and resource structures that enable the establishment of a network of interconnected objects. All technical specifications of the associated functional architecture and the associated services are available online [9]. By doing so, oneM2M facilitates uniform access across different hardware, infrastructures, and datasets.

Several implementations are currently available by private companies and academic laboratories, but only few of them are currently available in the open-source ecosystem: as example, OM2M [1] in Java, ACME [2] in Python, OceanMobius [7] in Javascript and TinyIoT [8] in C, just to mention a few.

oneM2M entities The standard delineates various kind of nodes, each of one running on the equipment that participates in an IoT solution. These nodes are interconnected in a tree-based topology. Starting from the leaves of the tree, these nodes are:
- the **Application Dedicated Node (ADN):** hosts the sensors and actuators of an IoT system and the associated application known as the Application Entity (AE).
- the **Application Service Node (ASN):** is an ADN endowed with the oneM2M Common Service Layer (CSE). It is deployed on objects with robust processing capabilities.
- the **Middle Node (MN):** acts as a gateway that collect from the ADN and ASN the IoT data. A MN hosts the common service entities that allows to collect these data. From a communication point of view, an MN acts as a bridge between local sensor/actuator networks and long-distance networks. Several levels of MN can be connected for scalability and security with the physical systems.
- the **Infrastructure Node (IN):** can be considered as the root of the tree, it serves as a hub for connecting different middle nodes, enabling high-level user

applications to interact with all nodes. Typically deployed in the cloud, the IN node hosts the Common Service Entity (CSE) defined below.

oneM2M resources The oneM2M service layer operates on the RESTful [15] software architectural style, where data and services are structured within a resource tree topology. Within oneM2M, this structure begins with the creation of a Common Service Entity (CSE), serving as the root of the tree. Resources within the tree can encompass various types, with the primary ones including:
- **Application Entity (AE):** This resource signifies a sensor, an actuator, a connected object, or an application. Within an AE resource, data and executable actions are stored.
- **Containers (CNT):** Positioned beneath an AE resource, containers group multiple other resources for storage and organization.
- **Content Instance (CIN):** Typically housed within containers, content instances predominantly represent sensor-collected values or descriptions of potential actions on an actuator.
- **Subscriptions (SUB):** Primarily situated within containers, subscriptions delineate actions to be executed, such as when a content instance is added to the same container. In a broader sense, they enable subscription to any modification within the resource where they are established by implementing a notification mechanism.

A typical operational procedure unfolds as follows for the resources management:
- The Infrastructure Node (IN) is started on the cloud, it creates its representation in own resources tree by the creation of an IN-CSE.
- The Middle Node (MN) is running on a gateway and initiates its registration on the Infrastructure Node (IN), establishing reciprocal links within their respective databases (remote IN/MN CSE).
- The Application Node (AN) hosting the different sensors, actuators and application parts complete their registration on the different MN, generating an Application Entity (AE) that encapsulates application descriptions. Sensor and actuators data are instantiated in the resource tree of this AE but creating Containers (CNT). Each sensor's new value create a new Content Instance (CIN).
- The end-user applications, like a smartphone app or a supervision center, are hosted by ASN nodes and gains access to the IoT application via a call to the different oneM2M services. These accesses can make a request to get the last value of a CIN or the subscription to certain data to receive notifications upon updates. The majority of accesses occur at the IN level. The internal mechanism of re-targeting inside oneM2M architecture allows to access to all CSE running and their resource tree on all equipment through an unique access to the IN node. CSE are able to transmit the request to an other CSE if the resource asked is not on their resources tree.

2.2 IoT systems simulation and performance evaluation

IoT systems have specific characteristics such as the numerous and multiform nature of their communications, different data exchange protocols along with the number of nodes participating in the system and their network topology.

Their modeling and simulation requires a rich environment to capture different protocols, but also a high performance simulation engine able to specify and run large-scale topologies and system's behaviour. In addition and depending on the fine or coarse grain modeling-level of the system under evaluation, the environment must be open to programming, to enable a certain level of accuracy of the behavior of nodes, their interconnections, as well as the performance parameters to be measured. Finally, simulation results must be easily exploitable, which requires open environments for accessing simulation results and, ideally, a open source environment for their comprehension and exhaustive analysis.

Among the many discrete event network simulators, some of them cover partially the previous requirements, NS3 [4] is highly adopted by the network performance evaluation community. As NS3 is network-dedicated, it has some weaknesses in the tuning of nodes behaviors and scalability. SimGrid [5] is also highly network-oriented, with a distributed Grid simulation engine that makes it difficult to get to grips with.

The chosen simulator is OMNeT++ [6], an extensible and modular, component-based C++ simulation library and framework, well-suited for a generic description of node processing and a programmatic style for network description via its NEtwork Description language (NED). Its large open source community has enabled the development of numerous APIs for protocols and interesting features for distributed simulation. We intensively use the component architecture capabilities of OMNeT++ i.e.: the definition components/nodes of the IoT network into C++ specific classes, their assembling into larger components or via their interconnection described in the NED. With such a decomposition, the reusability is an inner characteristics. The simulation kernel (and models) and the associated simulation results can be extracted from events traces, output vector and/or output scalar files to be analysed or plotted in a simulation interface, sequence diagrams or simple text-based files.

2.3 Applications/edge cloud

In IoT systems, the data collection protocols, the data storage locations, their analysis, their computing and the associated algorithms making decisions, are dependent on the real time and of the amount of data and the application concerns in terms of energy, time-sensitivity, security and privacy. A number of studies have been carried on the criteria for distributing and allocating the computational and collection parts of IoT systems according to application constraints.

In particular, the paper [13] proposes a coarse-grained classification of the distribution of computational and storage parts on fog and cloud parts according to application situations as well as their constraints. Authors compare the different computing paradigms of edge, Fog, Cloud Computing in Multi-access Edge-Computing (MEC) according to the features (heterogeneity, distribution, mobility, real-time support, ultra low latency ...) however, if it raises awareness the designer to the different pros and cons of these paradigms, it does not enable a precise reasoning on the scenario under development.

Additional research works, more specific to e-Health applications highlights the essential features of services and infrastructure in the domain of remote

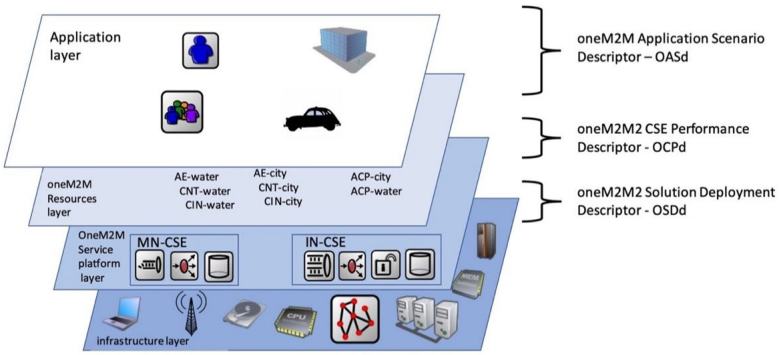

Fig. 1. Multi-layer model for an Standardized-IoT system

healthcare services. In [14], authors analyses such systems in terms of their essential features of scalability, bandwidth supposed to be high as the QoS, latency supposed to be low as the energy consumption. They choose 5G network as their communications medium and they provide an architecture (Cloud-RAN for 5G slicing, Fog, and Cloud Computing Data centers) and a queue-based model and the associated analytical equations for a performance analysis and simulation of KPIs. In their solution, the various aspects of a IoT system, present intricate connections. In their solution, there is a strong intricacy between the different aspects of the system, i.e.: the applicative parts, the communication and hardware infrastructure, intricacy visible on the analytical models, which reveals a *ad hoc* character. Nevertheless, and for this particular use case, interesting results are presented which have fed our own reflection, in particular the application aspects as well as the KPIs evaluated during the performance analysis.

3 Modeling approach for a oneM2M IoT system

We propose a multi layered-model of an oneM2M-based standardized IoT system, able to tackle applicative, infrastructure, and deployment aspects of the system. The detailed description of this model, their different artifacts, and the different performance indicators, are presented in [11]. The model view proposed on Fig. 1 facilitates the decomposition of all the physical, hardware, software, and human elements involved in a IoT system. Each layer interacts with the others, ultimately forming the IoT system and its usage environment.

3.1 A layered vision

The initial top layer on Fig. 1, termed broadly the "application layer", communicates with the physical realm with which the IoT system interacts. This layer encompasses human and environmental interactions with sensors and actuators, as well as the constraints and requirements articulated by application domains for actions in the physical world.

The next layer, termed the "resource layer", embodies the mapping of the IoT system and its specific use case onto resources (in the context of oneM2M).

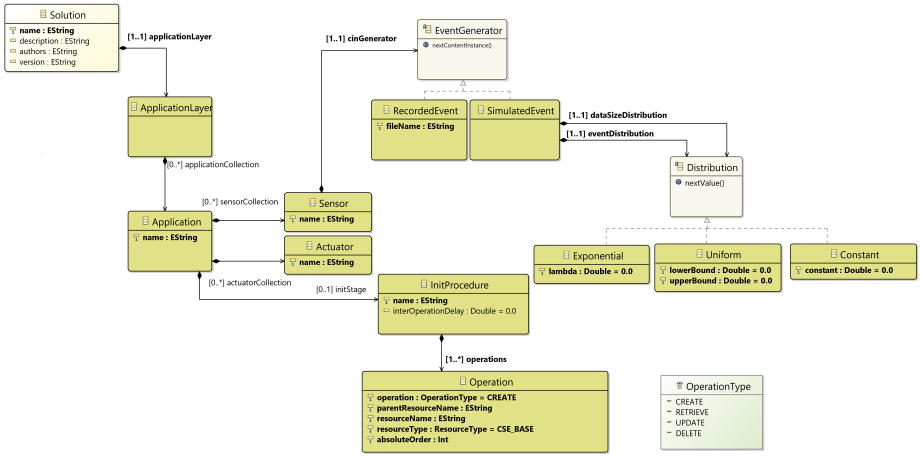

Fig. 2. Eclipse Ecore model for a oneM2M application Scenario Descriptor

This layer delineates the relationships among its resources and their ownership, articulating the business logic on the application side. The environment and resource layers will be detailed in a model referred to as the oneM2M Application Scenario descriptor (OASd).

The next layer, termed the "oneM2M service platform layer", represents the software or middleware system necessary to support the requirements of the above layers. In the oneM2M framework, it embodies the Common Service Entities (CSEs) and their interconnections, as well as modeling the behavior and performance of CSEs based on their configuration and the projection of the above layers onto this service platform layer. This layer will be instantiated in a model known as the oneM2M2 CSE Performance descriptor (OCPd).

The lower layer, termed the "infrastructure layer", encompasses the hardware infrastructure that hosts all preceding layers. It defines the equipment, communication links, and hardware characteristics. The oneM2M Solution Deployment descriptor (OSDd) model will describe this layer.

This framework allows for the dissection and understanding of the complexity of an IoT system, leveraging specialized expertise in each layer. Designing and deploying an IoT system entails implementing and integrating all these layers. From a performance analysis perspective, it enables the characterization of the requirements, parameters, and behavior of each layer, elucidating the connections between them through models.

3.2 A oneM2M Eclipse Ecore Meta Model

In this subsection we present the implementation in the Eclipse Ecore format [16], of our oneM2M system layered-model. The Ecore representation is at the hearth of the Eclipse Modeling Framework (EMF) and describes, through an object-oriented syntax and associated relations, the different concepts to be handle in a

modeling project. In our case, these concepts are more or less the one described in the proposed multi-layered model for a oneM2M infrastructure.

From the Ecore representation, multiple models can be instantiated by manipulating the Ecore inner defined elements. The advantage of such a representation is first to visualize in a standardized format, our reasoning framework, second to play quite easily and for free with the model, and last but not least, to explore the serialization capacity in EMF and to navigate into the modeling elements and to generate code.

The oneM2M Ecore meta model is divided into three parts. The first one, presented in Fig. 2, is dedicated to the IoT application and the capture of its behaviour. The objective of the OASd model is to represent the behavior of the IoT application. We consider a high-level modeling of an IoT application as a set of sensors and actuators endowed with a behavior modelled by event generation (for the sensors). The policy for the generation of sensor data is based on different distribution profiles i.e. constant (equivalent to periodic) or sporadic with different laws. Sensors and actuators have at an initial stage a transient behavior that corresponds to the creation of standardised oneM2M resources.

The second part of the meta model, called OCPd (oneM2M CSE Performance descriptor), is presented on Fig. 3. The OCPd refers to the underlying oneM2M platform and services. It describes the CSEs (IN-CSE, MN-CSE, ASN) and their interconnection onto a tree-based topology. The meta model also considers performance aspects i.e. the impact of the implementation choices of a CSE by a supplier such as the services call and the execution costs in terms of memory and processor, the interconnection protocols characteristics such as the bandwidth, the latency, and all additional costs generated by the data persistence of resources creation, storing and retrieving.

The last part of the meta model called OSDd (oneM2M Solution Deployment descriptor). The associated meta model is presented on Fig. 4. The deployment refers to the allocation of CSEs onto a physical architecture. This hardware platform is composed of servers nodes, gateway nodes, IoT nodes and communication links that hosts the IoT application. This infrastructure is made of multiple nodes of different types, interconnected by a network. The meta model quantifies the capacities (memory storage, processing capabilities, location) of the physical nodes and the underlying communication networks.

4 Implementation of the Meta Model in OMNeT++

OMNeT++ is an extensible, component-based `C++` simulation library and framework IDE, dedicated to the simulation of distributed/networked systems. The OMNeT++ environment is open source and thus, federates a large community who develops different features and libraries for specifying, editing, programming, and simulating networked systems. The simulation engine is a discrete event simulator that can handle multiple nodes and their network topology. Depending on the fine or coarse grain modelling-level of the system under evaluation, the environment can provide either predefined libraries that implements full protocol stacks such as TCP/IP, MQTT, or enables a more or less accurate programming of the communication protocols and nodes behaviour, their

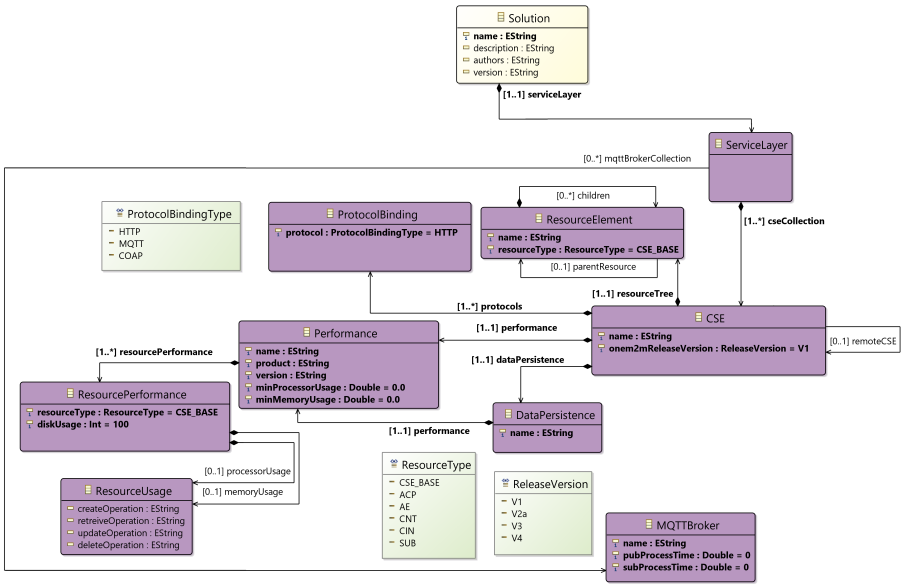

Fig. 3. Eclipse ECore model for a oneM2M CSE Performance Descriptor

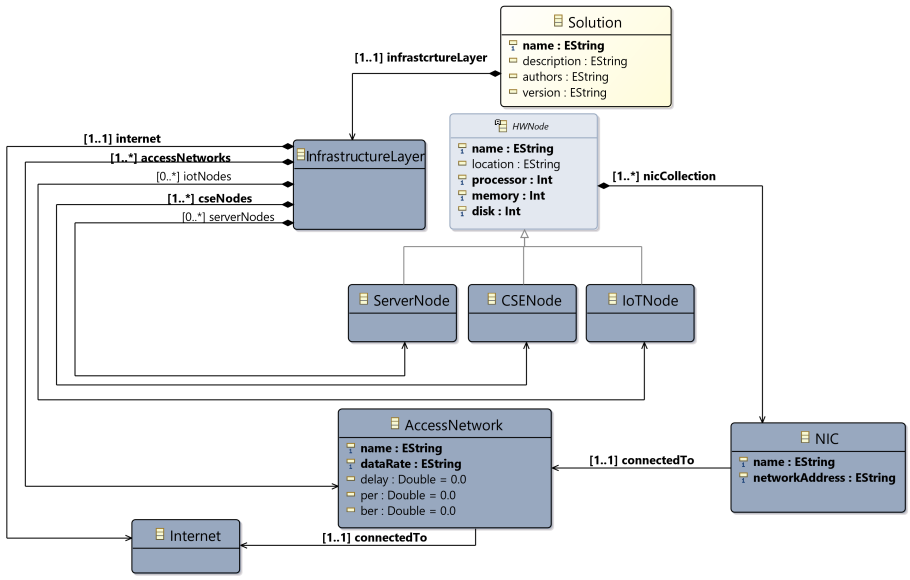

Fig. 4. Eclipse Ecore model for a oneM2M Solution Deployment Descriptor

interconnections, as well as the performance parameters to be measured. The integrated discrete event simulation engine also supports parallel distributed

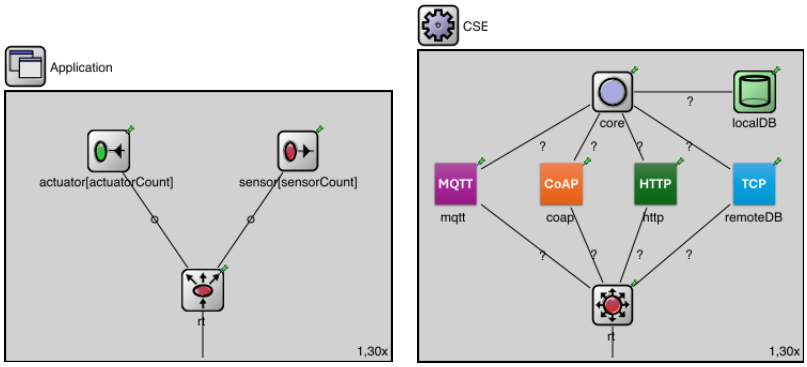

Fig. 5. Application Layer modules and Service layer modules

simulation to run large-scale topologies and system's behaviour. We use intensively the programming features of the IDE for the development of a oneM2M system simulator. The notions of modules (simple or compound), gates, channels, networks, interfaces are the inner elements of an OMNeT++ specification. Each element is associated with a `C++` class with the associated properties, attributes, variables, and behaviour. The element behaviour is developed in two `C++` methods namely: `initialize()` and `handleMessage()`. The method `initialize()` is called at the initial/transitional phase where OMNeT++ elements of these different types are instantiated whereas the `handleMessage()` method is called every simulation step and covers the permanent behaviour of the element.

The `<project-name>.ini` and `<project-name>.ned` files participate in the definition of the network. The `<project-name>.ini` file contains the parameters and variables initialization values for a current simulation whereas in the `<project-name>.ned file`, written with the NED language, allows the description of the topology of an OMNeT++ system. The NED language has a programming syntax closed to an imperative language to define graph topologies of different shapes, such as, e.g tree, ring, mesh, and their size. This is a real advantage of OMNeT++ over its direct competitor NS-3. The OMNeT++ simulator of our oneM2M-standardized IoT system is organized into multiple modules either simple or compound with respect to the meta model presented in Section 3 and fully defined in [11]. The modules are organized into three categories: `ApplicationLayer` (OASd in the meta model), `ServiceLayer` (OCPd in the meta model), and `InfrastructureLayer` (OSDd in the meta model).

4.1 Application Layer

The application layer is represented by a compound module named Application that hosts 3 types of sub-modules as shown in Fig. 5.
- `actuator:Actuator [0..N]`: an *Actuator* is a simple module that receives messages from a remote CSE.
- `sensor:Sensor [0..N]`: a *Sensor* is a simple module that sends messages to a remote CSE. One of the main parameters of this module is a JSON object

representing a *CinGenerator* as defined in [11]. The *CinGenerator* is a description of how oneM2M Content Instances (CIN) messages are being generated by the sensor. The *CinGenerator* includes informations about its generation and its size. Concerning the event generation, the simulator supports periodic generation, stochastic generation (uniform & exponential distribution), and generation following a time-series provided in an external file. Concerning the size of the message, the simulator supports constant size, stochastic size (uniform & normal distributions), and generation following a time-series provided in an external file.
- `rt:AppRouter[1..1]`: an *rt* is a simple module that handles messages routing between sensors/actuators and the underlying communications services. Both `sensor` and `actuator` modules have a parameter `InitializationProcedure` that contains the internal actions and messages sent to the remote CSE at start-up. Examples of these messages include the creation of oneM2M resources such as *ApplicationEntity* (AE), *Containers* (CNT), *Subscription* (SUB) or initial *ContentInstances* (CIN).
Finally, an *Application* can host multiple *Sensor* and *Actuator* modules.

4.2 Service Layer

The service layer is represented by the compound module *CSE* as shown in Fig. 5. Depending on its parameters, the CSE compound module hosts multiple simple modules that implement a feature of a oneM2M CSE. These modules are:
- `core:Core [1..1]`: refers to the oneM2M common services. This mandatory module is responsible for handling CRUD operations received by a CSE through message passing. One of its main parameter is a JSON object that represents a oneM2M CSE Performance Descriptor (OCPd) as defined in [11]. Based on this parameter, the CSE core can simulate the processing cost of a message and its associated operations in terms of computing (CPU) and memory (RAM) resources.
- `mqtt:MQTTBinding [0..1]`: this optional module is responsible for managing the MQTT protocol encapsulation/decapsulation of primitive oneM2M messages (requests & responses).
- `coap:COAPBinding [0..1]`: this optional module is responsible for managing the CoAP protocol encapsulation/decapsulation of primitive oneM2M messages (requests & responses).
- `http:HTTPBinding [0..1]`: this optional module is responsible for managing the HTTP protocol encapsulation/decapsulation of primitive oneM2M messages (requests & responses).
- `remoteDB:TCPBinding [0..1]`: this optional module represents a data persistence service that is available in a remote host, and that needs a TCP communication. It is suitable for representing CSEs that use remote databases.
- `localDB:DataStorage [0..1]`: this optional module represents a data persistence service that is embedded within the CSE. It is suitable for representing CSEs that use embedded databases (either file-based or in-memory database).
- `rt:CSERouter [1..1]`: this is a simple module that handles messages routing between different components of the CSE and the underlying communication services.

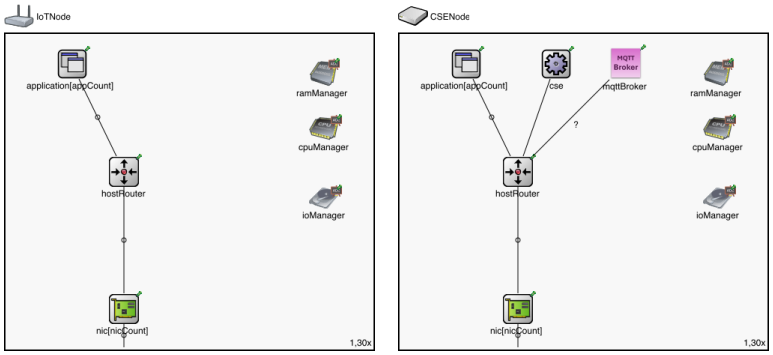

Fig. 6. Infrastructure Hardware nodes

4.3 Infrastructure Layer

The infrastructure layer is represented by modules related to hosting nodes (IoTNode, CSENode, ServerNode) and to networking (WiFiLink, LoRaLink, CellularLink, FiberLink, NetworkElement, InternetNode). First, all hosting nodes derive from one generic node: the `HWNode` one. This compound module hosts the following modules, as shown in Fig. 6.
- `IoTNode`: hosts only IoT applications, manages sensors and/or actuators.
- `CSENode`: extends the generic `HWNode` and hosts one CSE. It can also host an MQTT brooker alongside the CSE.
- `ServerNode`: extends the `HWNode` by hosting an MQTT Broker and/or a database. This node is suitable for representing nodes acting as remote databases for a CSE, an independent MQTT Broker, or any application logic such as a monitoring application.
Inside these 3 nodes, operate different sub modules:
- `application:Application [0..N]`: this compound module represents the IoT applications (cf. Application Layer) that run on the hosting node.
- `hostRouter:HostRouter [1..1]`: this is a simple module that handles messages routing between different applications/services running on the hosting nodes and the underlying communication services (NICs).
- `ramManager:RAMManager [1..1]`: this is a simple module that tracks the overall usage of the processing power of the hosting node (i.e. CPU).
- `cpuManager:CPUManager [1..1]`: this is a simple module that tracks the overall usage of the memory of the hosting node (i.e. RAM).
- `ioManager:IOManager [1..1]`: this is a simple module that tracks the overall usage of the disk usage of the hosting node.
- `nic:NIC [1..N]`: this simple module representing a networking interface available on the hosting node. The networking aspect is represented by two simple modules and four specific channels.
`NetworkElement`: This simple module represents a network router with basic IPv4 routing logic. It can be connected to any hosting node using a specific network channel.
- `Internet`: This is a simple module that is similar to the `NetworkElement`

module since it acts as second hop router and connects all the network elements present in the IoT solution. It could be seen as the core network of telecommunication operators.
Four channels are defined in the simulator. These channels represent the common communication links found in IoT solutions. Each channel defines its own data rate, latency, bit error rate (BER) and packet error rate (PER).

4.4 Simulation of an oneM2M-based Iot Solution

A simulation of our IoT solution following the Meta Model is composed by two main files: first, the physical topology (Infrastructure Layer) of the IoT solution must be defined in a file compliant with the syntax and semantics of OMNeT++'s description language (NED) describing the topology such a graph where vertices are one of the following: `IoTNode, CSENode, ServerNode, NetworkElement`, and `Internet`. Vertices are connected using one of the links defined in the simulator: `WiFiLink, CellularLink, LoRaLink`, and `FiberLink`. The topology can be seen as a tree with 3 levels: Level 0 (root element): `Internet` node, Level 1: nodes of type `NetworkElement`, and Level 2: nodes of type: `IoTNode, CSENode`, or `ServerNode`. A second file is the `omnetpp.ini` that contains values for the relevant parameters of the modules present in the topology file. All these parameters are of basic types such as `String`, `IPAddress`, and `Integer`. Two specific parameters use structured data that can also be references to external files. These parameters are the following (in `C++`):
- **Event Generator:** this is a parameter of the Sensor module. It is provisioned in JSON format. It implements the oneM2M Application Scenario descriptor (OASd) as defined in [11]. It contains information about how a sensor generates oneM2M messages (i.e. CREATE operation of `ContentInstance` resource on the remote CSE) in terms of time instant and message data size.
- **Performance Descriptor:** this is a parameter of both CSE and Storage modules. It is provisioned in JSON format. It implements the oneM2M CSE Performance Descriptor (OCPd) defined in [11]. It contains information about system resources usage in terms of processing, memory, and disk usages for each CRUD operation per resource type.
Measurement probes have been integrated in the source code of the simulator to build KPIs and evaluate the performance of the IoT system based on multiple simulation runs. These indicators include measures such as runtime, memory utilization, volume of data transferred, as well as specific metrics for each CRUD operations in the context of oneM2M resource creation on oneM2M objects. The associated KPIs are defined and described in Clause 5 of [11].

5 Experiments

This section reports on the experimental work carried out to simulate an oneM2M-based IoT solution following the multi-layered model proposed in this paper. It presents the simulation procedure as well as the performance indicators obtained from the simulation. The chosen case study is the deployment of e-Health services

for patient monitoring, requiring varying levels of quality of services. Thanks to the multi-layered model, we are able to offer a high-level specification of these services. Once these specifications are instantiated in the proposed OMNeT++ based simulator, application layer, service layer, and infrastructure layer designers are able to assess, by simulation, the relevance of their choices at a very early stage in the design cycle.

5.1 Use case description

The deployment of e-Health services for a remote monitoring of patients with chronic health disease, have led to a powerful transformation of the traditional medical practices. Due to the inner characteristics of these services, the underlying IoT solution should be capable, at any time, of collecting and storing patient data in real time, identifying abnormal patient situations before they become emergencies, and of course reacting to any emergencies that do occur. The chosen use case illustrates these different situations by considering three main types of actors namely:
- the *patient* with a chronic health disease that uses sensor devices for medical status measurements (heart rate, glycemia, ...).
- the *physician* that follows remotely the pathology of its patient through an application that is able to access patient medical measures.
- the *emergency physician* or *cardiologist* at the hospital who is called in urgency to treat an acute episode (heart attack, diabetic coma,...), who needs near-real-time access to the patient constants to stabilize the patient.
The oneM2M logical infrastructure that will instantiate the various elements of the e-Health monitoring system is shown on Fig. 7. The different sensors devices and applications appear as oneM2M logical entities such as ANs (Application Nodes) for the sensors, ASNs for the patient applications which collects the sensors data and carries the Patient Application Service (PLA). Middle MN-CSE nodes host the collection and detection services associated with the monitoring application from the Physician side and act as a gateway able to manage multiple patients. Finally, an infrastructure IN-CSE node is the root node of the oneM2M e-Health infrastructure, able to interact with practitioners or directly with patients in case of emergency situations. The different use case application flows are represented in plain or dashed lines according to the three situations:
- *Normal flow.* Every X minutes, sensors measure the patient biological constants and send them when possible to the *Patient Local Application (PLA)*. Periodically (e.g. every week) or in case of presential or remote consultation, the last data are deposited on the Physician Data Space and processed.
- *Abnormal flow* with triggering. In case of a abnormal evolution of data detected by the *Physician Server App (PSA)*, an alert with the associated data is sent to the Physician. The Physician triggers an adequate action for a future consultation or new directives in the medical treatment of the patient.
- *Emergency flow.* In case on critical data threshold or value detected by the Patient Local App, an emergency alarm is sent to the hospital emergency service. The emergency doctor need a real-time access to the health data constants of the patient through a *Emergency Server App (ESA)*.

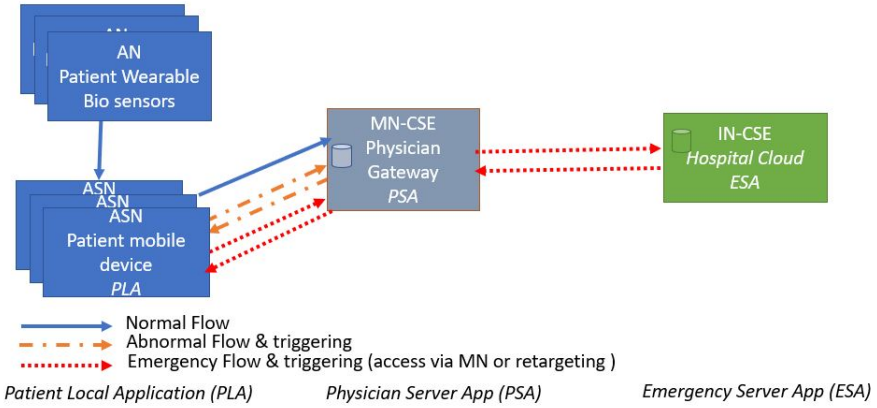

Fig. 7. oneM2M logical e-health monitoring application

5.2 Simulator parameters

The OMNeT++ topology of the use case is pictured on Fig. 8. In order to instantiate this use case, the oneM2M-based IoT solution considers a patient with his/her medical sensor connected to his smartphone or a medical device acting as oneM2M ASN node. This device is connected to the internet through the internet gateway (i.e. Internet Box). Within the patient's home, these devices are assumed to be connected through a WiFi network (green dashed links). The practitioner has data management software connected to an oneM2M CSE (mnNode) that is connected to the Internet. Finally, the emergency physician (at the hospital) has access to a data management software connected to an oneM2M CSE (inNode). At the practitioner's office or at the hospitals, devices are assumed to be connected through wired links such as Fiber (solid black links). The corresponding NED topology file for this use case as well as the corresponding configuration `.ini` file can be found in the git repository [17].

5.3 Simulation results

Multiple parameters can be considered for tuning the simulation. By changing these parameters it is possible to adjust the numbers of devices to be managed by nodes or the quantity of data to be collected and/or to check the need to reinforce the processing and communication infrastructure to absorb the application load. In this way, by simulation, it is possible to dimension and/or verify an IoT system deployment at an early stage of the design. These parameters characterize the application's load (number of devices, rate of data production, etc) in relation to the infrastructure on which the application will be deployed (networks and their bandwidth, computing capacity, RAM on the edge or on the cloud). The parameter we are considering are :
- the numerosity of the applicative elements (sensors, actuators) and their velocity to produce/consume data that characterize the applications' generated traffic

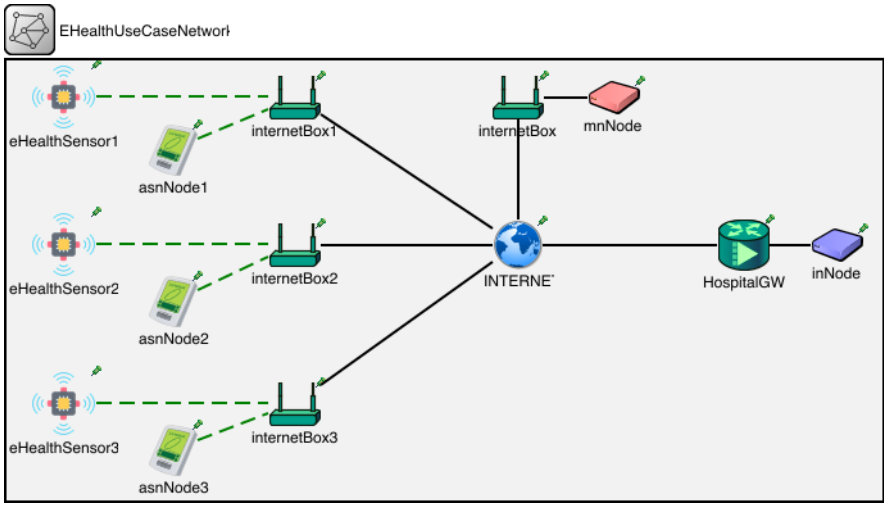

Fig. 8. Topology view of the E-Health Use Case in OMNeT++

(initially expressed through the application descriptor). In the OMNeT++ tool, the sizing and the structure are instanciated in the .ned file whereas the velocity of sensors are stated in the `omnetpp.ini` file.
- the available hardware resources on CSE nodes (amount of RAM in bytes & processing power in terms of instructions per seconds). These information are collected from the performance descriptor of the hardware node, and defined in the `omnetpp.ini` file.
- the characteristics of the communications links (bandwidth, latency, packet/bit error rates) take also part of the `omnetpp.ini` file.
- the performance characteristics of the CSE nodes (expressed through the performance descriptor generated by the oneM2M stack profiler). Here again this information is in the `omnetpp.ini` file.
Thanks to the instrumented code placed inside the simulation modules, multiple KPIs are measured during the simulation. Fig. 9 draws the results of the normal flow simulation and its associated RTT. In this demonstration simulation, ASN nodes are given different processing capabilities (asnNode3 > asnNode2 > asNode1) but IoT sensors attached to these nodes generate similar traffic (periodic messages for the normal flow and exponential traffic for the alternative flow). Also, the IN Node is deliberately under-provisioned in terms of CPU and RAM given the overall traffic it manages.

Due to the different resources available at the ASN nodes, the application layer RTT grows during the simulation. The random peaks correspond to the messages that are sent to the IN nodes (alternative data flow) where this node is always saturated due to its poor resources.

These KPI include information such as (1) application-layer RTT representing time difference between message sending time and acknowledgement recep-

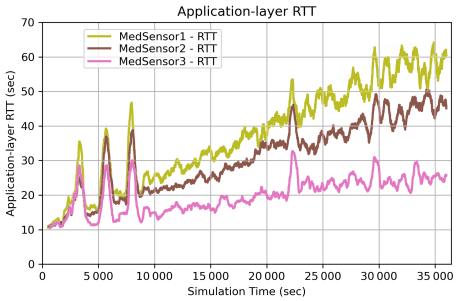

Fig. 9. Application-Layer RTT

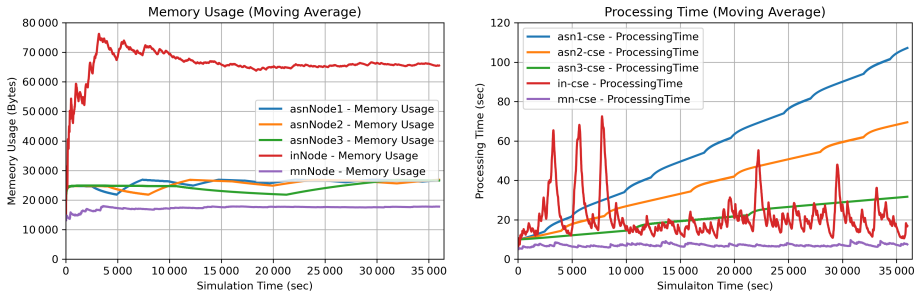

Fig. 10. CSE Memory usage and Processing Time

tion, (2) the message processing time in a CSE, and (3) the message queue occupancy in a CSE.

Fig. 10 shows examples of these two KPIs generated at the end of the simulation. In the left part the (simulated) RAM occupancy grows on ASN Node as messages are queued before being processed by the CSE (as it happens in real CSEs). The right part shows the impact of this phenomenon on the processing time since it is the sum of the waiting time in the queue and the actual processing time (provided in the performance descriptor of the CSE). This figure also shows how the memory usage and the processing time of messages in the case of the IN Node are always high. It is also worth noticing that the MN node is behaving well since the node is well provisioned. These experiments clearly show that the resources allocated to the IN Node are undersized to handle the overall traffic. Therefore, our simulator can be used in order to seek the best deployment plan by tuning the configuration parameters (in particular nodes resources).

6 Conclusions and Further Work

The objective of this paper is to evaluate the impact of standards and more precisely oneM2M in terms of performance and its applicability in the field of IoT system design and deployment. We first proposed a multi-layered meta model that facilitates the decomposition of all the physical, hardware, software, and human elements involved in an IoT-standardized solution. From this meta model and results of the effective performance profiling of oneM2M stacks, we are able

to generate an OMNeT++ simulator of a full oneM2M-based IoT solution. By tuning the system configuration and parameters, we evaluate the KPIs of this system and compare these KPIs with the real-time constraints of an application.

A first original contribution of this work is the possibility to include in the simulation, the effective performance values of oneM2M stacks.

A second interesting contribution is the flexible design of the simulator that can be used to simulate any other use case. Thanks to the modularity of the simulator, it is possible to capture other application scenarios. IoT system designers or deployment engineers, interested in experimenting with these tools can access them from the ETSI lab repository [17].

Some extensions of this work have already been identified such as improving the scalability of simulated systems to large realistic simulations, firstly by developing a adhoc domain specific language and associated automatic OMNeT++ code generation tools to speed up the description phase of the IoT system in all its aspects, and automatically produce the associated simulator. A second enhancement is to explore OMNeT++'s parallel computing features to address the processing limitation of current simulations on a single server for grid-based computing simulations.

References

1. Eclipse OM2M platform M2M communication. https://eclipse.dev/om2m
2. ACME. https://github.com/ankraft/ACME-oneM2M-CSE
3. The global oneM2M IoT standard. https://www.onem2m.org
4. Ns-3 network simulator. https://www.nsnam.org
5. Simulation of distributed computer systems. https://simgrid.org
6. OMNeT++ Discrete Event Simulator. https://omnetpp.org
7. Mobius oneM2M IoT Server Platform. https://github.com/IoTKETI/Mobius
8. TinyIoT. A oneM2M implementation. https://onem2m.org/technical/published-specifications/release-4
9. oneM2M Functional Architecture. Technical Specification TS-0001-V4.20.0. https://onem2m.org/technical/published-specifications/release-4
10. B. Flynn, L. Liquori, M.-A. Peraldi-Frati, S. Medjiah, T. Monteil. SmartM2M: Scenarios for evaluation of oneM2M deployments, ETSI TS 103839, 2023.
11. S. Medjiah, T. Monteil, L. Liquori, M.-A. Peraldi-Frati, B. Flynn. SmartM2M; Model for oneM2M Performance Evaluation, ETSI TS 103840, 2023.
12. S. Medjiah, T. Monteil, L. Liquori, M.-A. Peraldi-Frati, B. Flynn. SmartM2M: oneM2M Performances Evaluation Tool, ETSI TS 103841, 2024.
13. A. Yousef , C. Fung, T. Nguyen, K. Kadiyala and F. Jalali and A. Niakanlahiji and J. Kong and J.P. Jue. All One Needs to Know about Fog Computing and Related Edge Computing. *Journal of Systems*, 2019, *98*, 289-330, Elsevier.
14. S.A. AlQahtani. An Evaluation of e-Health Service Performance through the Integration of 5G IoT, Fog, and Cloud Computing. *Sensors*, 2023, *23*, 5006.
15. R.T. Fielding. Chapt 5 of Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, 2000.
16. The Eclipse meta model https://wiki.eclipse.org/Ecore
17. oneM2M Performance Evaluation Tools : Profiler & Simulator https://labs.etsi.org/rep/iot/smartm2m-onem2m-performance-evaluation