Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 32 (2014) 1079 – 1086

Procedia Computer Science

International Workshop on Recent Advances on Machine-to-Machine Communication

OM2M: Extensible ETSI-compliant M2M service platform with self-configuration capability

M. Ben Alaya[a,b], Y. Banouar[a,b], T. Monteil[a,b], C. Chassot[a,b], K. Drira[a,b]a*

[a]*CNRS, LAAS, 7 avenue du Colonel Roche, F-31400 Toulouse, France*
[b]*INSA, LAAS, Université de Toulouse, F-31000 Toulouse, France*

Abstract

Machine-to-Machine (M2M) concept is one of the main features of Internet of Things (IoT). It promises to interconnect billions of devices in near future covering various domains. However, M2M suffers from a high vertical fragmentation of current M2M markets and lack of standards. To bridge this gap, the European Telecommunications Standards Institute (ETSI) released a set of specifications for a common M2M service platform. In this paper, we propose the open source OM2M project, an autonomic ETSI-compliant M2M service platform. OM2M provides a RESTful API to enhance interoperability. It proposes a modular architecture running on top of an OSGi layer, making it highly extensible via plugins. It enables multiple communication protocols binding, reuse of existing remote devices management mechanisms, and interworking with existing legacy devices. Addressing the M2M complexity issue, a new M2M service based on the autonomic computing paradigm and semantic models is defined to provide dynamic discovery and reconfiguration mechanisms.

Keywords: Internet of Things; Machine-to-Machine; service platform; OSGi plugins; Modularity, Autonomic Computing; Self-configuration; semantic web; RESTful web servcie; ETSI M2M.

1. Introduction

Machine-to-Machine (M2M) is a phenomenon that has been proceeding quietly in the background, and it is coming into the phase, where explosion of usage scenarios in businesses will happen. Sensors, actuators, RFID/NFC tags, vehicles, and intelligent machines all have the ability to communicate. The number of M2M connections is continuously increasing, and it has been predicted to see billions of machines interconnected in a near future. M2M applications provide advantages in various domains from building, energy, healthcare, industrial, transportation, retail, security to environmental services.

* Corresponding authors. M. Ben Alaya
E-mail addresses: ben.alaya@laas.fr (M. Ben Alaya), ybanouar@laas.fr (Y. Banouar), monteil@laas.fr (T. Monteil), chassot@laas.fr (C. Chassot), drira@laas.fr (K. Drira)

doi:10.1016/j.procs.2014.05.536

M2M market is undergoing a massive shift from a set of closed vertical markets involving vendor-specific technologies to a global horizontal M2M platform bringing order to current legacy M2M systems. This revolution is paving the way to seamless interactions between smart applications and heterogeneous devices for large-scale M2M deployment. However, factors such as lack of interoperability, distributed system increasing complexity and lack of standardization are M2M concept challenges.

Several standardization efforts have been done to face the M2M interoperability challenge [1]. The most advanced M2M standard is provided by the European Telecommunications Standards Institute (ETSI). ETSI released several specifications [2, 3, 4] covering M2M service requirements, the functional architecture, communication interfaces, and how to interwork with existing standards and technologies. Other organization addressed the increasing complexity challenge of large scale computing systems. Inspired by the body's autonomic nervous system, IBM proposed the autonomic computing (AC) paradigm. AC aims to provide systems with self-management capabilities to hides intrinsic complexity to administrators and users.

In this work, we propose the OM2M project, an ETSI-compliant platform for M2M interoperability. OM2M proposes a RESTful architecture running on top of an OSGi layer. It is extensible via plugins and supports several protocols and technologies. An additional service based on autonomic computing and semantic is designed to enable dynamic discovery and self-configuration of M2M applications.

The rest of the paper is organized as follows: Section 2 presents an overview of the ETSI M2M standard. Section 3 details OM2M platform building blocks and main plugins design via UML diagrams. Section 4 describes OM2M self-configuration via the autonomic computing service. Section 5 concludes and points out some perspectives.

2. ETSI M2M standardization

According to the Global Standards Collaboration Machine-to-Machine Task Force, more than 140 organizations around the world are involved in M2M standardization**.** A considerable effort is done by ETSI to decrease M2M market fragmentation by defining a horizontal service platform for M2M interoperability. The proposed solution provides a RESTful Service Capability Layer (SCL) [3] accessible via open interfaces to enable developing services and applications independently of the underlying network. Such platform facilitates the deployment of vertical applications and boosts innovation across industries for an effective interoperability.

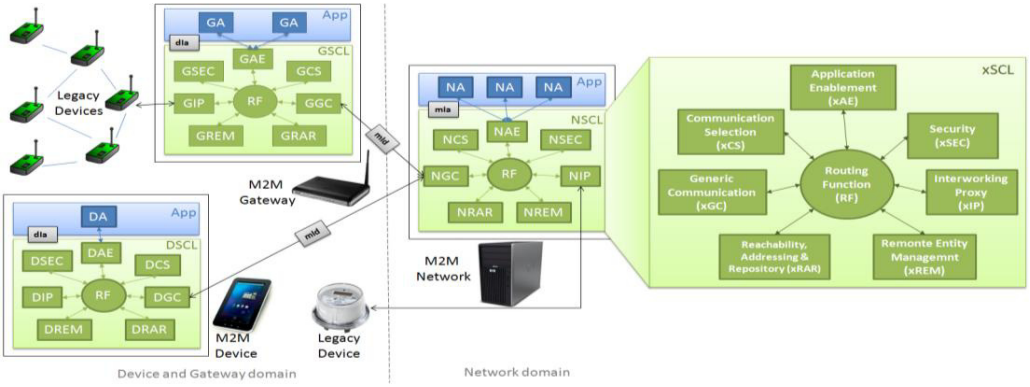


Fig. 1. ETSI M2M functional architecture

An M2M Device runs applications using the SCL. It connects directly to the Network Domain via the Access network and may provide service to other devices connected to it that are hidden from the Network Domain. It can also be connected to the Network Domain via a Gateway through a Local Area Network. A Gateway also runs M2M Applications using the SCL, and can act as a proxy between local devices and the network domain. The access network allows M2M devices and gateways to communicate with the Core Network. The SCL provides functions that can be shared by different Applications. Network Management Functions enables to manage the Access and Core Networks, such as Provisioning,

Supervision, and Fault Management. M2M Management Functions consist of all the functions required to manage the SCL in the Network Domain.

Figure 1 depicts the ETSI M2M functional architecture. An SCL can be deployed on an M2M network (NSCL), a gateway (GSCL), or a device (DSCL). It provides several service capabilities to enable machine registration, synchronous and asynchronous communication, resource discovery, access rights management, group broadcast, etc. Three reference points [4] based on open APIs are specified: mIa, dIa, and mId. The mIa reference point allows a Network Application (NA) to access the NSCL. The dIa allows a Device or Gateway Application (D/GA) to access the D/GSCL. The mId reference point allows a D/GSCL to access the NSCL. These interfaces are defined in a generic way to support a wide range of network technologies and protocols to enhance interoperability.

ETSI M2M adopts a RESTful architecture style. Each SCL contains a standardized resource tree where the information is stored. A resource is uniquely addressable via a Universal Resource Identifier (URI), and has a representation that can be transferred and manipulated with verbs (e.g. retrieve, update, delete, and execute).

An SCL resources tree supports different kind of resources. The “sclBase” resource describes the hosting SCL, and is the root for all other resources within the hosting SCL. The “scl” resource stores information related to distant SCLs, residing on other machines, after successful mutual authentication. The “application” resource stores information about the application after a successful registration on the hosting SCL. The “container” resource acts as a mediator for data buffering to enable data exchange between applications and SCLs. The “contentInstance” resource represents a data instance in the container. The “accessRight” resource manages permissions and permissions holders to limit and protect the access to the resource tree structure. The “group” resource enhances resources tree operations by adding the grouping feature. The “subscription” resource allows subscribers to receive asynchronous notification when an event happens such as the reception of new sensor event or the creation, update, or delete of a resource. The “announced” resource contains a partial representation of a resource in a remote SCL to simplify discovery request on distributed SCLs. The “discovery” resource acts as a search engine for resources. The collection resource groups common resources together.

3. OM2M service platform

3.1. OM2M building blocks

In this study, we propose OM2M: an ETSI-compliant M2M service platform. OM2M provides a RESTful API for XML data exchange through unreliable connections within a highly distributed environment. It offers a modular architecture running on top of an OSGi Equinox runtime [5] as shown in figure 2. OM2M provides a flexible SCL that can be deployed in an M2M network, a gateway, or a device. An SCL is composed of small tightly coupled plugins, each one, offering specific functionalities. A plugin can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot. It can also detect the addition or the removal of services via the service registry and adapt accordingly facilitating the SCL extension.

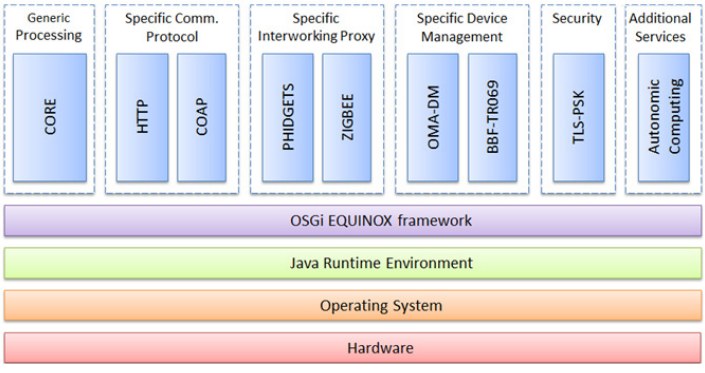


Fig. 2. OM2M building blocks

The CORE is the main plugin that should be deployed in each SCL. It provides a protocol-independent service for handling REST request. Specific communication mapping plugins can be added to support multiple protocol bindings such as HTTP and CoAP. OM2M can be extended with specific device management mapping plugins to perform device firmware updates by reusing existing protocols such as OMA-DM and BBF TR-069. It can be also extended by various interworking proxy plugins to enable seamless communication with legacy devices such as Zigbee and Phidgets technologies. The TLS-PSK protocol is used to secure M2M communications based on pre-shared keys. A new plugin based on the autonomic computing paradigm is designed to enhance OM2M resource discovery and self-configuration.

3.2. OM2M plugins components diagram

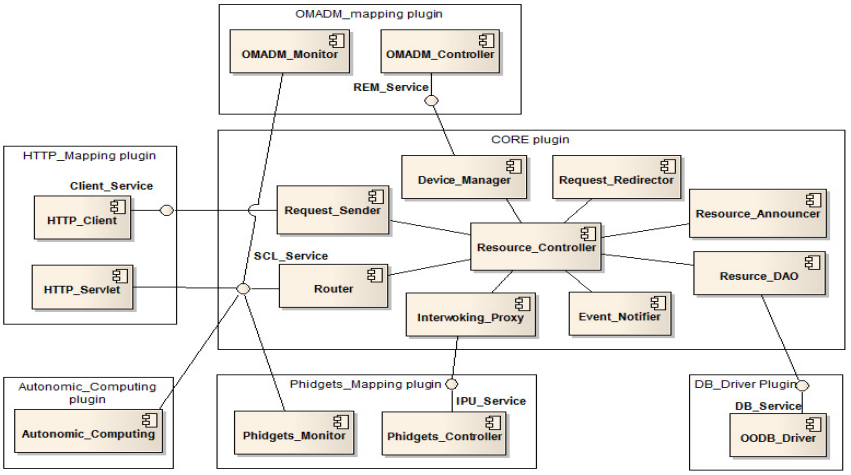


Fig. 3. OM2M component diagrams

The *CORE* plugin implements the *SCL_Service* interface to handle generic RESTful request. It receives a protocol-independent request indication and answers with a protocol-independent response confirm. The *Router* defines a single route to handle every request in a resource controller simply using request URI and method. The *Resource_Controller* implements CRUD methods (Create, Retrieve, Update, and Delete) for each resource. It performs required checking operations such as access right authorization, and resource syntax verification. The *Resource_DAO* provides an abstract interface to encapsulate all access to resource persistent storage without exposing details of the database. The *Event_Notifier s*ends notifications to all interested subscribers when a resource is created, updated or deleted. It performs filtering operations to discard events not of interest to a subscriber. The *Resource_Announcer a*nnounces a resource to a remote SCL to make it more visible and accessible to other machines. It also handles resource de-announcement. The *Request_Sender* holds discovered protocol-specific clients implementing the *Client_Service* interface. It acts as a proxy to send a generic request via the correct communication protocol. The *Interworking_Proxy* holds discovered interworking proxy units (IPUs) implementing the *IPU_Service* interface, and acts as a proxy to call the correct IPU controller. *Device_Manager* holds discovered Remote Entity Managers (REMs) implementing the *REM_Service* interface, and acts as a proxy to call the correct device manager controller.

The *HTTP_Mapping* plugin provides a bidirectional binding to the HTTP protocol. The *HTTP_Servlet* receives and converts an HTTP request into a generic one, and call the *CORE* plugin via the *SCL_Service* interface. The *HTTP_Client* implements the *Client_Service* interface to send a generic request via HTTP. The *Phidgets_Mapping* plugin provides a bidirectional mapping to interwork with legacy Phidgets devices. The *Phidgets_Monitor* discovers Phidgets devices, and calls the *CORE* plugin to create required resources on the SCL. The *Phidgets_Controller* implements the *IPU_Service* interface to seamlessly perform a generic request via the Phidgets API. The *OMADM_Mapping* plugin provides a bidirectional mapping to manage OMA-DM enabled devices. The *OMADM_Monitor* listens to OMA-DM enabled

devices, and calls the *CORE* plugin to create required resources on the SCL. The *OMADM_Controller* implements the *REM_Service* interface to converts generic request into OMA-DM management session. The *DB_Driver* plugin provides an Object Oriented Data Base (OODB) accessible via the *DB_Service* interface. Other plugins can be deployed using the same approach to interwork with additional protocols or to integrate new capabilities such as the *Autonomic_Computing* plugin.

3.3. Application resource creation scenario

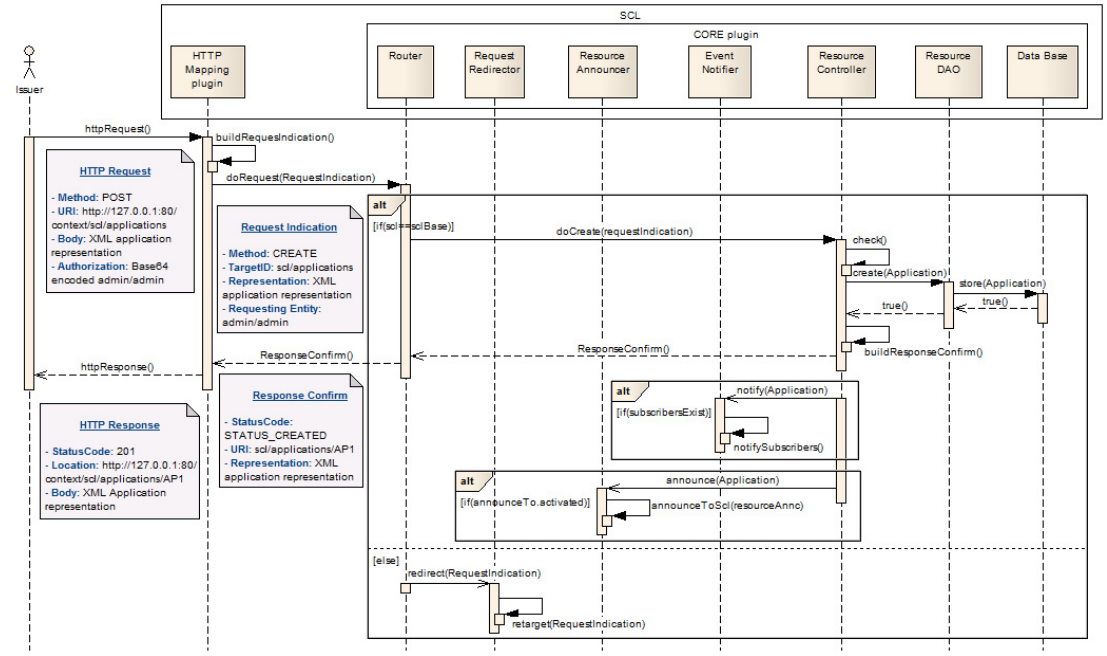


Fig. 4. Resource creation scenario

To create an Application resource, an issuer sends a POST HTTP request with a Base64-encoded *username/password* as authorization header, including the application XML data as body to this URI: *http://ip:port/context/scl/applications*. The *HTTP_Mapping* plugin builds a protocol-independent *RequestIndication* object from the received request, and uses it to call the *Router*. If the target *scl* is equal to the *sclBase*, the *Router* routes the request locally to the *Application_Controller, otherwise it retargets it to the corresponding remote SCL*. The *Application_Controller* checks the issuer access right and validates the application representation. It creates an Application object from the received XML representation, and stores it on the database via the *Application_DAO*. Then, it notifies all interested subscribers via the Event_Notifier, and may announce the application resource to pre-selected remote SCLs via the *Resource_Announcer*.

3.4. Extension to various M2M technologies via interworking proxy plugins

In the following scenario, an issuer will seamlessly handle heterogeneous devices only via REST requests. It monitors a Phidgets temperature sensor, and controls a Zigbee fan actuator accordingly using HTTP. Once the *Phidgets_IPU* plugin discovers the temperature sensor, it creates a *TEM_APP* application resource and a *TEM_DATA* container sub-resource where to store temperature events. In the same way, the *Zigbee_IPU* plugin creates a *FAN_APP* resource on the SCL. The issuer sends a GET HTTP request to discover registered applications, and then sends a POST request to subscribe to *TEM_DATA* contentInstances. As soon as an event is reported by the temperature sensor, the *Phidgets_IPU* creates a new *TEM_DATA* contentInstance sub-resource. The issuer receives an HTTP notification. In the case of

high temperature, it sends a POST HTTP request to switch ON the fan. The *Zigbee_IPU* plugin performs the received request using the Zigbee protocol and returns back the response.

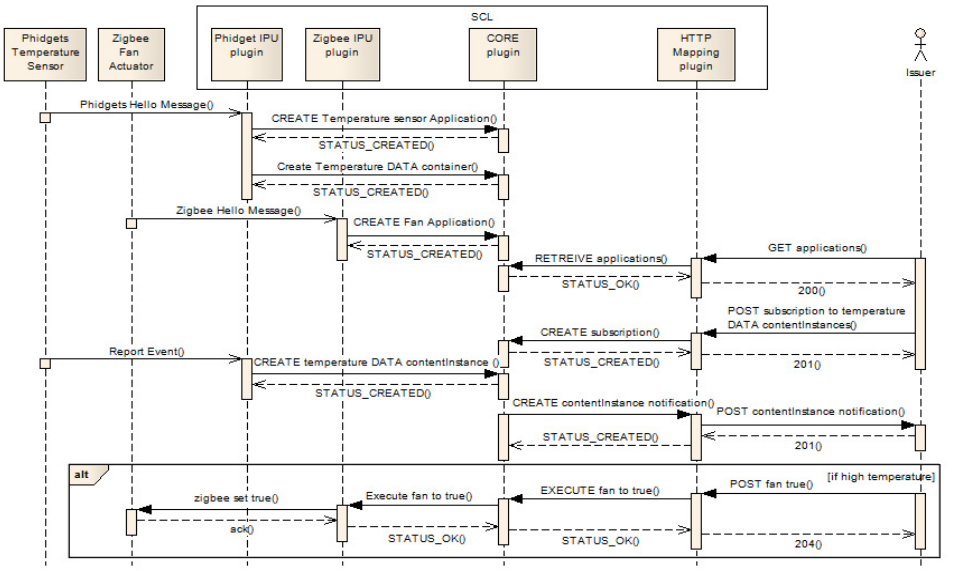


Fig. 5. M2M interoperability through various interworking proxies

Additional plugins can be deployed to interwork with various M2M communication protocols via specific communication mapping plugins for better interoperability. For example, it is possible to add a *CoAP_Mapping* plugin in such a way that an issuer request can seamlessly cross several HTTP and CoAP enabled machines before reaching the target resource.

4. M2M self-configuration via an Autonomic Computing plugin

Autonomic Computing [6, 7] is a paradigm proposed by IBM in 2001. It aims at developing distributed systems capable of self-management to hide intrinsic complexity to administrators and users. An autonomic manager is organized into four main modules: the Monitor, the Analyzer, the Planner and the Executor. These modules share a same knowledge (managed resources details, policies, symptoms, request for change, plans, etc.), exploit policies based on goal and environment awareness, and perform together an active control loop called MAPE-K.

An autonomic computing service called ACS is designed and implemented for the self-configuration of M2M communications according to device profiles and application roles. The proposed ACS provides an autonomic manager that operates as an expert system to emulate the decision-making ability of humans to solve complex problems by reasoning about knowledge.

In our approach, the ACS is deployed as a service capability within the SCL. As an initial step, the ACS subscribes to the applications resource to be aware of new applications registrations. Once an application is created, the ACS receives a notification with the application resource representation including information related to the type, category, location, etc. The ACS stores all discovered applications in an ontology model within a local knowledge base. It reasons based on semantic rules to detect semantic matching between existing applications [8, 9]. Then, it reconfigures the SCL resource tree to interconnect required applications together. Figure 6 illustrates three basic self-reconfiguration scenarios.

Figure 7 depicts the ontology model considered by the ACS to perform the proposed self-configuration scenario as well as SWRL rules [10] applied by the JESS engine to match existing monitor and controller managers with available sensor and controller devises within the M2M system.

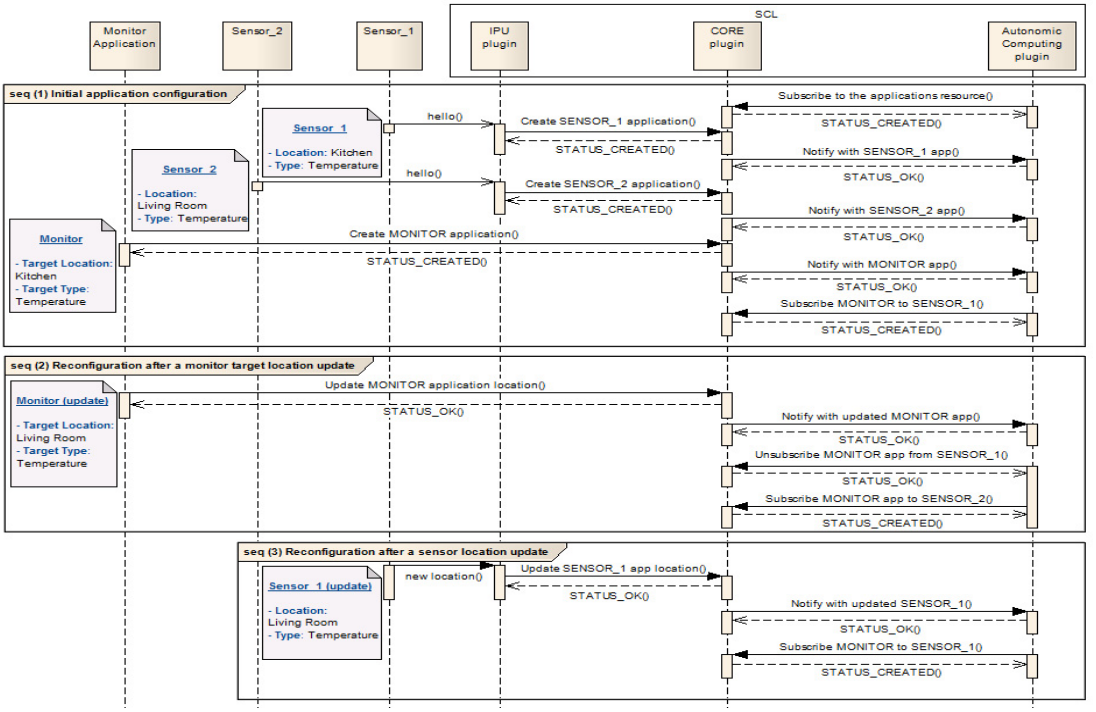


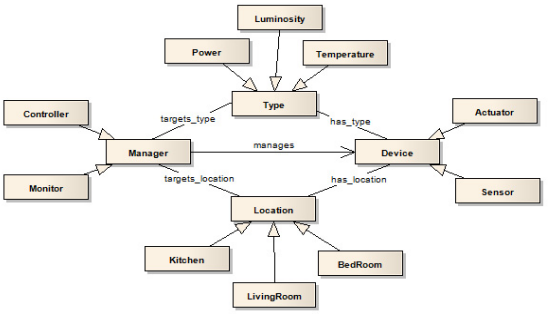
Fig. 6. SCL self-configuration sequence diagram

Sequence (1) illustrates an initial configuration scenario. The ACS monitoring module subscribes to the applications resource and receives the following events: Event_1(Id:Sensor_1, Location:Kitchen, Category:Temperature), Event_2(Id=Sensor_2, Location:LivingRoom, Category:Temperature), and Event_3(Id:Monitor_1, TargetLocation:Kitchen, TargetCategory:Temperature). The ACS monitoring module adds discovered applications to the M2M ontology model described in Figure 7, then infers and sends the following symptoms to the analyzing module: Symptom_1(Type:NewSensor, Id:Sensor_1), Symptom_2(Type:NewSensor, id:Sensor_2), and Symptom_3(Type:NewMonitor, id:Monitor_1). The ACS analyzer applies the two SWRL rules presented in Figure 7 to find correlation between discovered applications. It detects a potential link between Monitor_1 and Sensor_1 based on their descriptions, then generates and sends the following Request for change to the planning module: RFC_1(Goal:Subscribe, Subscriber: Monitor_1, Publisher: Sensor_1). The planning module generates an action plan based on RFC_1 including the following action and sends it to the executer: Action_1(Method:CREATE, TargetID:Sensor_1, Representation: Subscription resource including Monitor_1 Contact URI). The executer module maps Action_1 to the http protocol and sends the following http request to the SCL: HTTP_Request(Method:POST, URI:Sensor_1 subscriptions URI, Body:Subscription XML including Monitor_1 contact URI). Once the subscription resource is created, Monitor_1 starts receiving kitchen temperature events from Sensor_1.

Sequence (2) shows a reconfiguration after a monitor target location update. The monitor application sends an update request to the SCL to toggle its target location to living room. The ACS is notified with the new monitor application description and updates the ontology instance accordingly. Applying SWRL rules enables to detect a potential connection between the monitor and sensor_2. As a first step, the ACS deletes the sensor_1 subscription. As a second step, it creates a new subscription resource to subscribe the monitor to sensor_2 which is initially located on the living room. At this moment, the monitor starts receiving temperature events from the living room instead of the kitchen.

Sequence (3) describes a reconfiguration after a sensor location update. Sensor_1 is now moved to the living room. Consequently, Sensor_1 sends an update request to the SCL to correct its location. The ACS receives the new Sensor_1 description and updates the ontology instance accordingly. Applying SWRL rules results in the detection of a possible connection between the monitor and sensor_1. The ACS creates

a new subscription resource on the SCL to subscribe the monitor to sensor_1. Here, the monitor starts receiving living room temperature events simultaneously from sensor_1 and sensor_2.




```
Monitor(?m) ∧ targets_location(?l)  ∧ targets_type(?t) ∧
    Sensor(?s) ∧ has_location(?l)  ∧ has_type(?t) ∧
     ∧ (non manages(?m,?s)) → sqwrl:select(?m,?s)
```

SWRL Rule 1

```
Controller(?c) ∧ targets_location(?l)  ∧ targets_type(?t) ∧
    Actuator(?a) ∧ has_location(?l)  ∧ has_type(?t) ∧
     ∧ (non manages(?c,?a)) → sqwrl:select(?c,?a)
```

SWRL Rule 2

Fig. 7. M2M ontology model and SWRL rules

5. Conclusion

In this paper, we presented OM2M, an open source M2M service platform in line with ETSI M2M standard. The proposed solution provides a modular OSGi architecture extensible via plugins, and exposes services via a RESTful API for M2M interoperability. OM2M building blocks, components diagram, and main plugins were detailed and a new autonomic service was introduced for M2M applications dynamic discovery and configuration. OM2M was first experimented on a small building mock-up, and then deployed and validated on the real smart building LAAS ADREAM. The OM2M platform is accepted as an open source project by the Eclipse foundation. It is now part of the Eclipse IoT Working group [11].

As future work, we envisage to extend OM2M features by providing mapping for CoAP, MQTT, and Lightweight M2M protocols which are designed for resource-constrained internet devices. The autonomic computing service will be enhanced with the SSN ontology [12] to target more complex scenario. We are also working on a new M2M service to enables information centric routing [13] for better data exchange and cashing to enhance scalability.

References

[1] Boswarthick D, Elloumi O, Hersent O. *M2M Communications: A Systems Approach*. Wiley: Chichester, U.K., 2012. Print.

[2] ETSI TS 102.689 v1.1.1. *Machine-to-Machine communications (M2M); M2M service requirements*. August 2010.

[3] ETSI TS 102.690 v1.1.1. *Machine-to-Machine communications (M2M); Functional architecture*. October 2011.

[4] ETSI TS 102 921 v1.1.1. *Machine-to-Machine communications (M2M); mIa, dIa and mId interfaces*. February 2012.

[5] McAffer J, VanderLei P, Archer S. *OSGi and Equinox: Creating Highly Modular Java Systems*. Addison-Wesley; Upper Saddle River; NJ; 2010.

[6] Parashar M, Hariri S. *Autonomic Computing: Concepts, Infrastructure, and Applications*. CRC/Taylor and Francis: Boca Raton; 2007. Print.

[7] Kephart JO, Chess DM. *The vision of autonomic computing*. Computer 2003; 36(1):41–50. DOI: 10.1109/MC. 003.1160055.4. Dobson S, Sterritt R.

[8] Stojanovic L, Schneider J, Maedche A, Libischer S, Studer R, Lumpp T, Abecker A, Breiter G, Dinger J. *The role of ontologies in autonomic computing systems*. IBM Systems Journal 2004; 43(3):598–616.

[9] M. Ben Alaya, T. Monteil. *FRAMESELF: An ontology-based framework for the self-management of M2M systems*. Concurrency and Computation: Practice and Experience, Wiley, 2013.

[10] Zhang W, Hansen KM. *Towards self-managed pervasive middleware using owl/swrl ontologies*. In Fifth International Workshop on Modeling and Reasoning in Context (MRC 2008), Delft, TheNetherlands; 1–12. June 2008.

[11] OM2M project, http://www.om2m.org.

[12] Compton M, and al. *The SSN Ontology of the W3C Semantic Sensor Network Incubator Group*. Journal of Web Semantics, 17. 25 - 32. 2012. ISSN 1873-7749.

[13] Grieco LA, Ben Alaya M, Monteil T, Drira K. *Architecting Information Centric ETSI-M2M systems*. IEEE International Conference on Pervasive Computing and Communications WIP. Budapest · Hungary · March 24-28 2014.