Secure OM2M Service Platform

Sabrina Sicari, Alessandra Rizzardi, Alberto Coen-Porisini
DISTA, Università degli Studi dell'Insubria
Varese, Italy
{*sabrina.sicari;a.rizzardi;alberto.coenporisini*}*@uninsubria.it*

Luigi Alfredo Grieco
DEI, Politecnico di Bari
Bari, Italy
a.grieco@poliba.it

Thierry Monteil
LAAS-CNRS
Toulouse, France
monteil@laas.fr

***Abstract*—Machine-to-Machine (M2M) paradigm is one of the main concern of Internet of Things (IoT). Its scope is to interconnect billions of heterogeneous devices able to interact in various application domains. Since M2M suffers from a high vertical fragmentation of current M2M markets and lacks of standards, the European Telecommunications Standards Institute (ETSI) released a set of specifications for a common M2M service platform. An ETSI-compliant M2M service platform has been proposed in the context of the open source OM2M project. However such a platform currently only marginally addresses security and privacy issues, which are fundamental requirements for its large-scale adoption. Therefore, an extension of the OM2M platform is proposed, defining a new policy enforcement plugin, which aims to manage the access to the resources provided by the platform itself and to handle any violation attempts of the policies.**

***Keywords*-Internet of Things, OM2M, Security Enforcement**

I. Introduction

Internet of Things (IoT) paradigm has been approaching our lives thanks to the availability of wireless communications (e.g., RFID, WiFi, 4G, IEEE 802.15.x), which have been increasingly employed as technology driver for smart monitoring and control applications [5] [12]. The IoT concept is many-folded, since it embraces many different technologies, services, and standards. IoT deployments may adopt different processing and communication architectures, technologies, and design methodologies, based on the target scenarios. Therefore, a middleware may be neeeded in order to deal with such heterogeneity of devices and communication protocols [10].

In this context, Machine-to-Machine (M2M) market has been spreading, due to the fact that the number of M2M connections is continuously increasing. The advantages of M2M applications range in various application domains from building, energy, healthcare, industrial, transportation, retail, to environmental services. The goal is to shift from the actual vertical and fragmented deployments to a global horizontal M2M platform. In this direction, several standardization efforts have been done to face the M2M interoperability challenge [7]. Among them, a very promising proposal is being contributed by the European Telecommunications Standards Institute (ETSI). ETSI released several specifications [1] [2] [3] covering M2M service requirements, the functional architecture, communication interfaces, and how to interwork with existing standards and technologies. Moreover, in [4], the OM2M project is proposed, consisting of an ETSI-compliant platform aiming at facilitating the interoperability among M2M applications and devices. Such an architecture is extensible via plugins and supports several protocols and technologies.

However, the actual OM2M platform only marginally addresses the security and privacy requirements. In fact, such a high level of heterogeneity of involved technologies and protocols makes such an architecure as object of multiple security and privacy attacks. Traditional security countermeasures and privacy solutions cannot be directly applied to IoT technologies [14] (e.g., limited power resources, scalability issues). Furthermore, privacy and security issues should be treated with a high degree of flexibility and adaptation to the target environment [6] [8]. Note that, in order to reach a full acceptance by users it is mandatory to define valid security and privacy mechanisms suitable for IoT as well as M2M applications [9] [13] [12] [15]. More in details, confidentiality and integrity have to be guaranteed, as well as authentication and authorization mechanisms in order to prevent unauthorized users (i.e., humans and devices) to access the system. Whereas, concerning privacy requirement, both data protection, anonymity and users personal information confidentiality have to be ensured, since devices may manage sensitive information (e.g., user habits). It is important to remember that in IoT and M2M contexts the number of violation attempts is high. So, it is fundamental to define and develop some enforcement mechanisms. Therefore, our contribution lies in the extension of OM2M platform with a policy enforcement layer, able to deal with security and privacy application-specific requirements and violation attempts, in order to increase the robustness of the actual architecture.

The rest of the paper is organized as follows. Section II describes the OM2M standard platform with the involved plugins and functionalities, then our security extension is discussed. Section III analyzes the behavior of the extended platform. Section IV presents an application example, while V ends the paper and provides some hints for future works.

II. OM2M Secure Platform

A. OM2M standard platform

OM2M project has been proposed as an ETSI-compliant platform for M2M interoperability [4]. OM2M provides a horizontal service platform which facilitates the deployment of vertical applications and leads innovation towards an effective interoperability. It provides a RESTful Service Capability Layer (SCL) accessible via open interfaces to enable the development of services and applications independently of the underlying network. RESTful API are provided for XML data exchange through unreliable connections within a highly distributed environment. 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 methods (e.g., retrieve, update, delete, execute). An SCL resources tree supports different kind of resources, as described in the following. 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, for example residing on other machines, after a 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 "registration" resource allows subscribers to receive asynchronous notification when an event happens such as the reception of new 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.

Moreover, the SCL works as an interface between the network access and the application domain. Each device has to register the resources in the OM2M standard platform which in turn can be accessed from the applications from OM2M platform in a seamless way. More in details, 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. Fig. 1 presents an high level representation of OM2M architecture.

An M2M Device runs applications using the SCL (i.e., DSCL). It can connect directly to the network domain via the access network and may provide service to other devices connected to it. 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 (i.e.,GSCL), and can act as a proxy between local devices and the network domain. The Network Access allows M2M devices and gateways to communicate with the Core Network. The SCL provides functions that can be shared by different applications. Well-defined network management functions enables to manage the Access and Core Networks. They consist of all the functions required to manage the SCL in the network domain.

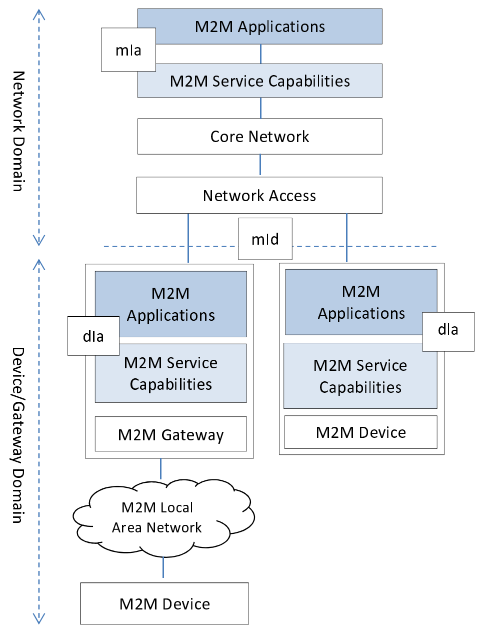


Figure 1. OM2M high level architecture

Three reference points 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 in order to enhance interoperability. Note that OM2M platform is accepted as an open source project by the Eclipse foundation and it is part of the Eclipse IoT Working group [1].

Summarizing, the building blocks of an ETSI M2M system are: devices, gateways, and networks. A device is a machine equipped with a set of resources/services that can be made accessible to the rest of the system. Many devices may also bind to the same gateway in order to make their resources available outside their local domain. Finally,

[1]http://www.om2m.org

the resources available at many gateways and devices are exposed at a wide area scope through an ETSI M2M network.

B. OM2M secure extended platform

As just described, OM2M provides a flexible SCL, which can be deployed in an M2M network, a gateway, or a device. An SCL is composed of small tightly coupled plugins, each one offers 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 and adapt to the changes, thus facilitating SCL extension. Thus, OM2M platform is extensible via plugins and is able to support several protocols and technologies. The OM2M layers are shown in Fig. 2.

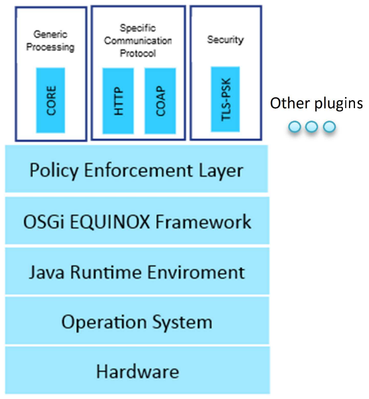


Figure 2. OM2M building layers

In particular, OM2M proposes a modular architecture running on top of an OSGi Equinox runtime [11]. The CORE is the main plugin that should be deployed in each SCL. It provides a protocol-independent service for handling RESTful requests. Specific communication mapping plugins can be added to support multiple protocol bindings, such as HTTP and CoAP. In fact, we can easily use HTTPS with the jetty plugin. As regards security, the TLS-PSK protocol is used, which aims at securing M2M communications on the basis of pre-shared keys. For TLS-PSK we need to use an other plugin that is not in the open-source version because there is a problem of license. Nevertheless, such a secure plugin only refers to securing the communications among the involved modules. In fact, OM2M currently only marginally addresses security issues: no attention is paid to the definition of security and privacy policies for the management of services and the filtering of data among the requesting applications. Therefore, this contribution is focused on adding a secure policy enforcement layer. It provides the same interfaces of the services, but before proceeding with the processing operations by the CORE plugin, verifies the compliance with the security and privacy policies associated to the requested services.

More in details, as shown in the component diagram in Fig. 3, 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 each request in a resource controller simply using request URI and the required method. In fact, the *Resource_Controller* implements CRUD methods (i.e., create, retrieve, update, 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 any details of the database. The *Event_Notifier* sends notifications to all interested subscribers when a resource is created, updated or deleted. It executes filtering operations to discard events not of interest to a subscriber. The *Resource_Announcer* announces a resource to a remote SCL to make it 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. More details about these components and their interactions are available in [4].

Other plugins can be deployed using the same approach in order to interwork with other protocols or to integrate new capabilities. In our contribution, a new component plugin, named *Policy_Enforcement* plugin, is added and provides an interface towards the CORE plugin. Within the *Policy_Enforcement* plugin, the *Policy_Enforcement* component is responsible of handling the requests, while the *Policy_Manager* manages the security and privacy policies defined for the M2M services and data. Note that, for each resource request or event notification, the *Policy_Enforcement* component queries the *Policy_Manager* component, in order to take a decision on the basis of well-defined enforcement mechanisms (i.e., a set of policies specified for each kind of resource). In this way, the *Policy_Enforcement* plugin acts as a layer (Fig. 2) between the CORE plugin and the other interacting plugins.

III. Behavioral Aspects

In a typical ETSI M2M scenario, firstly the gateway and the device mutually authenticate to the NSCL. In order to add the proposed *Policy_Enforcement* plugin to the existing OM2M platform implementation (available at [2]), it has

[2]http://wiki.eclipse.org/OM2M/Download

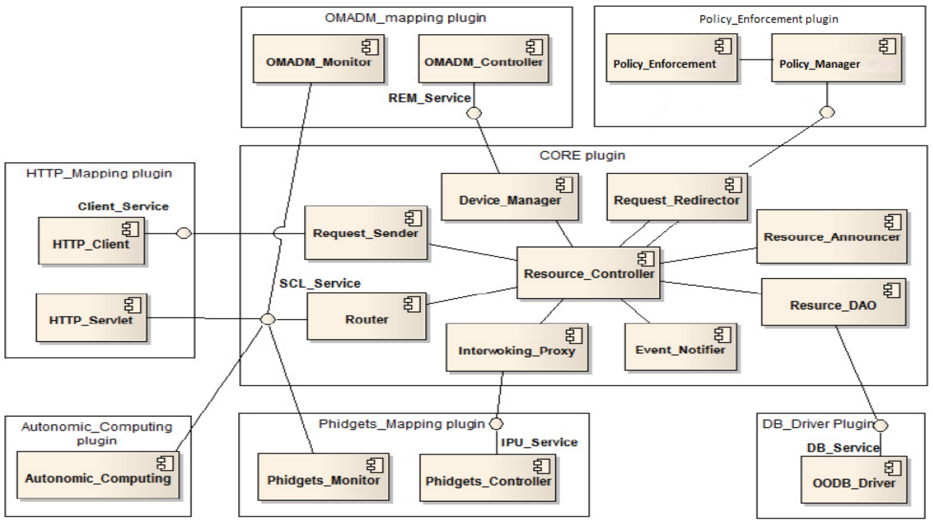


Figure 3. OM2M component diagram

to be registered to the D/G/NSCL, which in turn create a description container, where descriptive information are stored, and a data container, where the data are stored. Such data regards, in the case of the *Policy_Enforcement* plugin, the set of policies related to the resources handled by the OM2M platform. As shown in Fig. 4, such a plugin provides an application for handling the service/data requests (i.e., *POLICY_ENFORCEMENT_REQ*) and an application for handling the service/data responses (i.e., *POLICY_ENFORCEMENT_RESP*). Such applications represent the interfaces exposed to the CORE plugin, in order to allow the interactions between the two components.

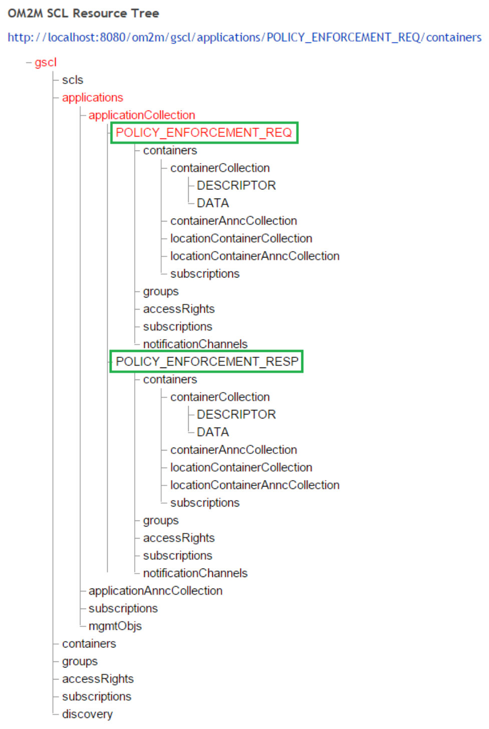
Note that the *Policy_Enforcement* plugin can be activated or not within the OM2M platform at the different levels (i.e., D/G/NSCL), depending on the desired level of security and the importance of the compliance of the defined security and privacy policies with the services/data disclosure.

In the following section, an application example is presented, in order to clarify the effectiveness of the proposed policy enforcement mechanism.

IV. Application Example

In the following application scenario, an issuer wants to handle, via RESTful requests, a set of GPS location information regarding the vehicles in a particular area. Supposing that communications happen by means of HTTP protocol, the sequence diagram in Fig. 5 illustrates the case study, in which a device reports the GPS values and the issuer receives the relative notifications (i.e., the presence of a particular vehicle on which the GPS device is installed).

Once the GPS IPU plugin discovers the device which sends the GPS information, it creates on the SCL the corresponding GPS application and data container, where to store the location events (i.e., transactions 1-7 in Fig. 5).



OM2M SCL Resource Tree

http://localhost:8080/om2m/gscl/applications/POLICY_ENFORCEMENT_REQ/containers

```
- gscl
  scls
  applications
    applicationCollection
      POLICY_ENFORCEMENT_REQ
        containers
          containerCollection
            DESCRIPTOR
            DATA
          containerAnncCollection
          locationContainerCollection
          locationContainerAnncCollection
          subscriptions
        groups
        accessRights
        subscriptions
        notificationChannels
      POLICY_ENFORCEMENT_RESP
        containers
          containerCollection
            DESCRIPTOR
            DATA
          containerAnncCollection
          locationContainerCollection
          locationContainerAnncCollection
          subscriptions
        groups
        accessRights
        subscriptions
        notificationChannels
    applicationAnncCollection
    subscriptions
    mgmtObjs
  containers
  groups
  accessRights
  subscriptions
  discovery
```

Figure 4. OM2M enforcement plugin

Note that, during this phase, the *Policy_Enforcement* plugin is informed about the new resource and, as a consequence, asks to the CORE plugin what kind of security and privacy policies it has to apply for the future requests of the data provided by the application just created (i.e., transactions 8-11 in Fig. 5). For example, a policy to be applied to the location information is that only issuers registered as

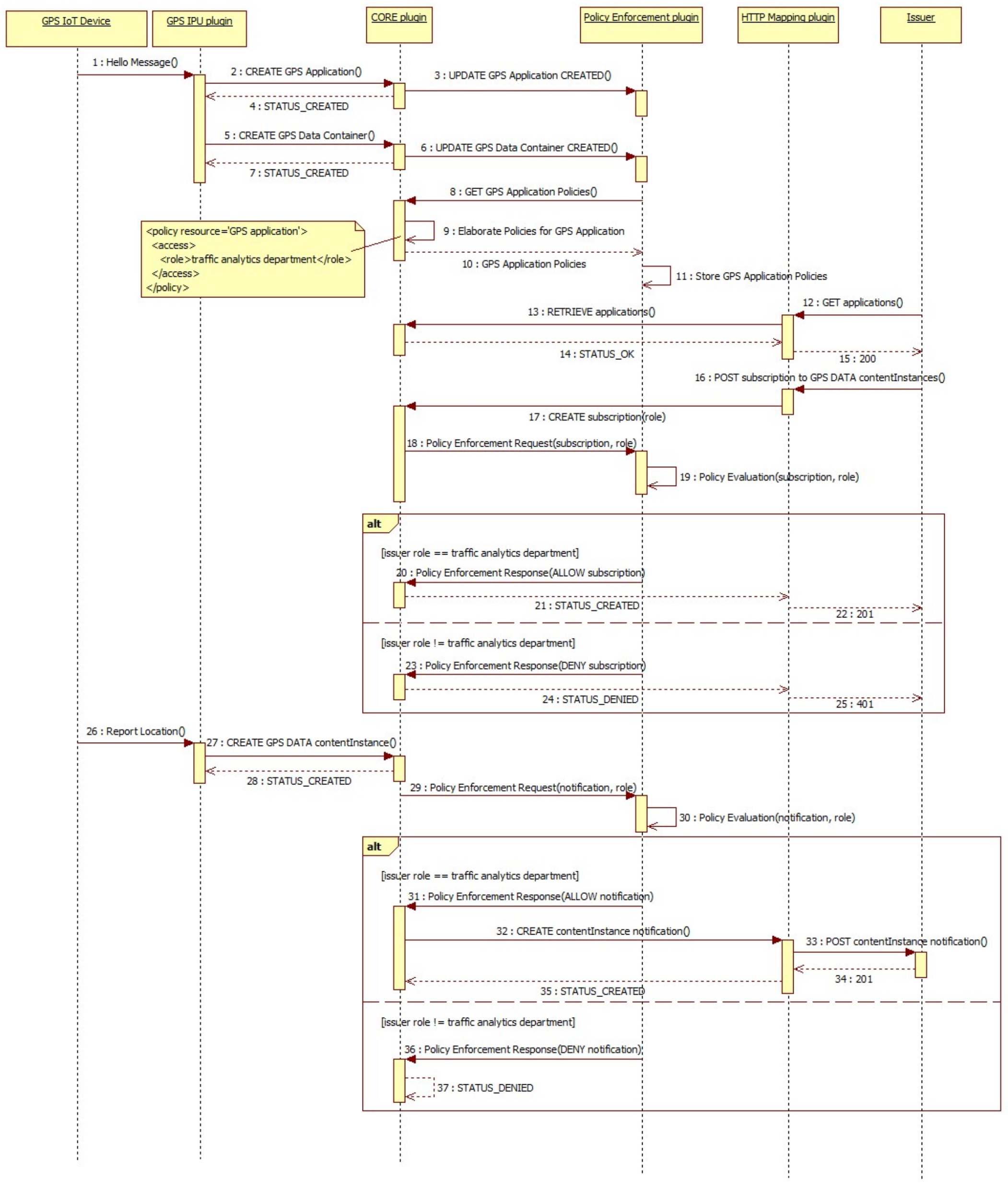

GPS IoT Device
GPS IPU plugin
CORE plugin
Policy Enforcement plugin
HTTP Mapping plugin
Issuer
1 : Hello Message()
2 : CREATE GPS Application()
3 : UPDATE GPS Application CREATED()
4 : STATUS_CREATED
5 : CREATE GPS Data Container()
6 : UPDATE GPS Data Container CREATED()
7 : STATUS_CREATED
8 : GET GPS Application Policies()
<policy resource='GPS application'>
<access>
<role>traffic analytics department</role>
</access>
</policy>
9 : Elaborate Policies for GPS Application
10 : GPS Application Policies
11 : Store GPS Application Policies
12 : GET applications()
13 : RETRIEVE applications()
14 : STATUS_OK
15 : 200
16 : POST subscription to GPS DATA contentInstances()
17 : CREATE subscription(role)
18 : Policy Enforcement Request(subscription, role)
19 : Policy Evaluation(subscription, role)
alt
[issuer role == traffic analytics department]
20 : Policy Enforcement Response(ALLOW subscription)
21 : STATUS_CREATED
22 : 201
[issuer role != traffic analytics department]
23 : Policy Enforcement Response(DENY subscription)
24 : STATUS_DENIED
25 : 401
26 : Report Location()
27 : CREATE GPS DATA contentInstance()
28 : STATUS_CREATED
29 : Policy Enforcement Request(notification, role)
30 : Policy Evaluation(notification, role)
alt
[issuer role == traffic analytics department]
31 : Policy Enforcement Response(ALLOW notification)
32 : CREATE contentInstance notification()
33 : POST contentInstance notification()
34 : 201
35 : STATUS_CREATED
[issuer role != traffic analytics department]
36 : Policy Enforcement Response(DENY notification)
37 : STATUS_DENIED

Figure 5. OM2M interactions with policy enforcement plugin

members of the ”traffic analytics department” can access to the relative data. Such a policy may be expressed in XML syntax, following the SCL RESTful API (Section II), as presented in Listing 1. Of course, such policies can be set by a system administrator or can be based on rules established according to an ontology of the data provided by the different applications, which interact with the CORE plugin.

```
<policy resource='GPS application'>
  <access>
     <role>traffic analytics department</role>
  </access>
</policy>
```

Listing 1. XML policy

The issuer sends a GET HTTP request in order to discover the registered applications (i.e., transactions 11-15 in Fig. 5); therefore, it sends a POST request to subscribe to the GPS application, in order to receive the corresponding *contentInstances* resources (i.e., transactions 16-17). Before giving the consent to the subscription, a *POLICY_ENFORCEMENT_REQ* as to be sent by the CORE plugin to the *Policy_Enforcement* plugin (i.e., transaction 18), as presented in Section III). The *Policy_Enforcement* plugin has to evaluate each request from subscribers and decide wheter to approve or not the request itself on the basis of the stored security and privacy policies (i.e., *POLICY_ENFORCEMENT_RESP* in Section III, transactions 19-25 in Fig. 5). More in details, if the requesting issuer is registered as a member of the "traffic analytics department", the response if the *Policy_Enforcement* plugin will be "STATUS CREATED", otherwise the subscription is denied and the response sent to the issuer is "STATUS DENIED".

As soon as an event is reported by the IoT device, the GPS IPU plugin creates a new *contentInstances* resource, which is notified to the issuer only if he is authorized by the *Policy_Enforcement* plugin (i.e., transactions 26-37 in Fig. 5). In particular, the *Policy_Enforcement* plugin verifies the correspondance of the issuer role with the policies associated to the requested data; in case of positive outcome, the notification is allowed and a POST request is sent to the subscribed issuer, otherwise the notification is denied. Note that *Policy_Enforcement* plugin traces the transactions both from outside and from the inside of the SCL, as a firewall, in order to prevent misbehaving activities. Hence, it is important to point out the definition of enforcement mechanisms, which allow to deal with violation attempts.

The presence of the *Policy_Enforcement* plugin generates an overhead of communications with respect to the standard OM2M platform, but it allows to integrate the system with a policy management and enforcement point, in which security and privacy policies can change without the need to reconfigure the CORE plugin, exploiting the XML language potentiality. In particular, regarding the example investigated in this section, there is an overhead in the initial phase of the device discovering (the transactions 3, 6, 8-11 in Fig. 5); such an overhead occurs only once, as well as the overhead due to the subscription requested by the issuer (the transactions 18, 19, 20, 23 in Fig. 5). Finally, at each reported event, there is an overhead due to the verification of policies (the transactions 29, 30 in Fig. 5) and, possibly, the notification to the issuer (the transactions 31, 36 in Fig. 5). Note that such a distribution of the overhead can be applied to other case studies.

V. Conclusion

In this paper, an extension of the open source OM2M platform has been presented, in order to add new functionalities related to the security and privacy management of resources. To this end, a new policy enforcement plugin has been integrated in the actual OM2M architecture, communicating through RESTful requests. In such a way, the plugin is able to trace the transactions both from outside and from the inside of the system, thus filtering the event notifications and preventing violation attempts. In the next future, we plan to investigate a real case-study in order to verify the effectiveness of the proposed solution in a context in which well-defined policies are defined for the provided resources.

VI. Conclusion

References

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

[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] M. Ben Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira. OM2M: Extensible ETSI-compliant M2M service platform with self-configuration capability. *Procedia Computer Science*, 32(0):1079 – 1086, 2014.

[5] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Comput. Netw.*, 54(15):2787–2805, October 2010.

[6] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta. A survey of middleware for internet of things. In *Third International Conferences, WiMo 2011 and CoNeCo 2011*, pages 288–296, Ankara, Turkey, June 2011.

[7] D. Boswarthick, O. Elloumi, and O. Hersent. *M2M Communications: A Systems Approach.* John Wiley & Sons, Ltd, 2012.

[8] M. A. Chaqfeh and N. Mohamed. Challenges in middleware solutions for the internet of things. In *2012 International Conference on Collaboration Technologies and Systems (CTS)*, pages 21–26, Denver, CO, May 2012.

[9] H. Feng and W. Fu. Study of recent development about privacy and security of the internet of things. In *2010 International Conference on Web Information Systems and Mining (WISM)*, pages 91–95, Sanya, October 2010.

[10] L. A. Grieco, M. B. Alaya, T. Monteil, and K. K. Drira. Architecting information centric ETSI-M2M systems. In *IEEE PerCom*, 2014.

[11] S. Archer J. McAffer, P. VanderLei. *OSGi and Equinox: Creating Highly Modular Java Systems.* Addison-Wesley Professional, 2010.

[12] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Survey internet of things: Vision, applications and research challenges. *Ad Hoc Netw.*, 10(7):1497–1516, September 2012.

[13] R. Roman, J. Zhou, and J. Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, July 2013.

[14] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76:146–164, 2015.

[15] Rolf H. Weber. Internet of things - new security and privacy challenges. *Computer Law & Security Review*, 26(1):23–30, January 2010.