2013 IEEE 37th Annual Computer Software and Applications Conference

Through the Internet of Things - a Management by Delegation Smart Object Aware System (MbDSAS)

Marcelo Antonio Marotta, Felipe José Carbone, José Jair Cardoso de Santanna,
Liane Margarida Rochenbach Tarouco
Institute of Informatics – Federal University of Rio Grande do Sul (UFRGS)
Postal Code 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
e-mail:{mamarotta, fjcarbon, jjcsantanna, liane}@inf.ufrgs.br

***Abstract*—The management of smart objects (SObjs) is an important task because they are huge in number and applications. Such huge number of SObjs may lead the Internet of Things (IoT) to face severe network conditions, in terms of network congestion and large delays. Thus, the management of SObjs is fundamental to avoid future IoT network problems. In such a management, network boxes, also called gateways, have been configured to manage SObjs with software updates or reconfiguration followed by a warm start. However, gateways configuration become soon outdated because SObjs join and leave the network quite frequently. Therefore, we propose an approach called MbDSAS to reconfigure gateways without the need of a software updating or patching to manage and detect SObjs and deal with the dynamicity of the IoT network. An evaluation of MbDSAS was performed through an airport modeled scenario. In addition, MbDSAS was experimentally tested to be qualified as a management solution to IoT scenarios and to determine the best performance combination of technologies to implement MbDSAS.**

***Index Terms*—Internet of Things, network management, Management by Delegation, smart objects, network architecture**

I. Introduction

Traffic lights, surveillance cameras, home appliances, and mobile phones are objects of everyday that, when equipped with network interfaces, have been classified as Smart Objects (SObjs) [1]. The Internet has been gradually incorporated these SObjs into its environment, leading to the so called Internet of Things (IoT). Because the extremely large number of everyday objects, the future IoT is naturally expected to be composed of billions of SObjs. [2] Predictions foresee that the IoT will accommodate 50 to 100 billion of SObjs in 2020. As a consequence, the huge number of SObjs can potentially lead the IoT to severe network conditions in terms of congestion and large delays because of the network traffic generated by SObjs.

The management of SObjs is fundamental to avoid severe network conditions of the future of the IoT. In such a management, traditional devices as computer servers, set-top boxes, and routers can potentially incorporate to role of management stations (or simply managers) that access information from SObjs to both monitor and configure them. Managers and SObjs exchange information according to two models: direct and indirect communications. In direct communication [3], occasional intermediate devices between managers and SObjs (e.g., gateways, firewalls, NAT boxes) do not change the communication semantics and perception. In the indirect communication, however, intermediate devices expose to managers a different, possibly enhanced view of SObjs [4]. For example, gateways can cache SObjs management information to improve the perceived availability of SObjs from the manager point-of-view [5].

Usually, SObjs have insufficient hardware resources to implement more sophisticated management features found in traditional devices. That includes, for example, supporting robust security mechanisms or even timely replying to bursts of manager requests. In fact, the usual lack of resources in SObjs leads solution designers to prefer the indirect communication model because it offers the opportunity to handle the SObjs constraints at intermediate devices. Routers, switches, access points, and other network boxes can perform proper intermediation between managers and SObjs [6][4], and, in this case, are referred to as management gateways in this paper.

Currently, management gateways already available tend to be shipped with a management code that is not expected to be replaced, unless as a result of a software update or patch. In the IoT, however, management gateways need to cope with the typical dynamicity of the network, where new SObjs join and leave the IoT quite frequently. It means that management gateways must be able to intermediate a varying number and types of SObjs without passing through the typical process of a software update or reconfiguration (usually followed by a warm start). In summary, today's management gateways tend not to offer dynamic reconfiguration features, which represents a critical problem when intermediating the management of SObjs in the IoT. Therefore, to address this critical problem, we propose an architecture with SObjs dynamic behavior awareness feature in combination with reconfiguration mechanisms.

In this paper we propose an approach called Management by Delegation SObj Aware System (MbDSAS). In such an approach, gateways detect SObjs dynamic behavior by taking advantage of discovery protocols, for example, Link Layer Discovery Protocol (LLDP) or Universal Plug and Play protocol (UPnP). In addition, MbDSAS allows the creation of lists based on the detected behavior of SObjs to later be obtained by managers through Web Services (WS). Afterwards, managers can use Management by Delegation (MbD) concepts to delegate the management of SObjs tasks to gateways. Such

0730-3157/13 $26.00 © 2013 IEEE
DOI 10.1109/COMPSAC.2013.122

delegation is carried out by the use of WS services in combination with the IETF Script MIB [7] or Open Service Gateway initiative (OSGi) [8], which enables gateways reconfiguration without the need of patches or updates followed by a warm start. A prototype of MbDSAS was created and deployed in a modeled airport scenario for the assessment of MbDSAS performance. Finally, results were experimentally collected in terms of response time, network traffic, management delay, CPU and memory usage.

The remainder of this paper is organized as follows. In Section II, we present a background study and the state-of-the-art on management solutions for the IoT context. In Section III, we describe our solution to configure gateways to manage SObjs. A proof of concept of our solution is presented in Section IV. In Section V, a case study and a scenario are defined to evaluate our solution, whereas in Section VI the achieved results are discussed. Finally, in Section VII, we conclude this paper presenting final remarks and future work.

II. Background & Related Work

Through this section we discuss fundamental work related to our solution. We present the underlying concepts of our proposal and review the IETF's Script MIB. In addition, we explain how OSGi can be used as a mechanism to reconfigure gateways. Finally, we discuss how Web Services may be used in IoT.

Traditional centralized management approaches are usually insufficient to manage large networks [9]. This management approaches use a single management station to control various types of nodes of a network domain, however, with the increasing size of modern networks, the management become complex and station becomes overloaded with data. Thus, centralized approaches had to be replaced by distributed management solutions to avoid such overload.

The distribution of network management can be accomplished with the Management by Delegation (MbD) model [9]. This model is based on a hierarchy of three entities: (*i*) Top-Level Managers (TLMs) that are stations responsible for the management tasks, such as manage each network node of a domain or providing summarized reports about a domain; (*ii*) Mid-Level Managers (MLMs) that are network nodes capable of executing mundane actions, such as execute a delegated management tasks and gather local network information; and (*iii*) Managed Devices (MDs) that are the final target of the management tasks. Scripts contain the code describing the management tasks, which are sent by TLMs to MLMs. The execution of these scripts allows MLMs to manage closer MDs. The communication between TLMs and MLMs is independent of the delegated management scripts, whereas the communication between MLMs and MDs depends on the management interfaces exposed by MDs.

In distributed MbD systems, the overall management task is distributed among almost all network entities. Such a distribution enables MbD systems to achieve the most important characteristics for this work: scalability and flexibility [10].

- Scalability: Accordingly to Schonwalder *et al.* [10], there are three reasons for MbD systems reach scalability. First, a TLM management task will be distributed through different network entities, minimizing its workload. Second, the network overhead will be minimized because MLMs will exchange management tasks with others that are closer to MDs, avoiding distant communications. In addition, scripts sources will be communicated between MLMs rather than raw data. Afterwards, MLMs can summarize collected data to minimize the communication traffic with distant entities. Finally, MLMs need less storage resource by maintaining only active scripts stored instead of raw data.
- Flexibility: As soon as MbD systems configurations become outdated, managers may create new scripts to reconfigure MbD systems on the fly [11]. Afterwards, these MbD systems can spread the new script among its internal nodes and MbD systems neighbors. Therefore, management tasks can be freely reassigned dinamically without the need of a software update usually followed by a warm start.

In the IoT context, we argue that MbD may be used in large scenarios to delegate management tasks to IoT gateways operating as MLMs. These gateways are then reconfigured to manage closer SObjs. However, because MbD is an abstract model, it has to be carefully adapted to be properly applied in IoT management. Thus, we review two important technologies that can be used to realize MbD.

The first technology is the Script MIB [7], introduced by the IETF's Distributed Management (DISMAN) group. The Script MIB is a Simple Network Management Protocol (SNMP) Management Information Base (MIB) supported by MLMs. The Script MIB defines neither a specific language for coding of management scripts nor their objectives, for example, a management script may be designed to create firewall rules or to collect informations from an MD, both are managed but not defined by Script MIB. Therefore, Script MIB provides a way to remotely manage scripts life-cycles, *i.e.,* scripts installation, launch, execution, and termination.

Granville *et al.* [12] present a WS based alternative to the Script MIB. Such alternative was designed to offer original Script MIB services as WS operations accessed through the Simple Object Access Protocol (SOAP) [13]. The WS-based solution presents an improved performance in comparison to the Script MIB because it reduces the number of messages exchanged in the network. The important consequence of that work is the observation that MbD services can consume less network resources if MbD is realized through carefully defined WS operations, instead of having a traditional Script MIB implementation in place.

WS applications may present different architectures, *e.g.,* Service Oriented Architecture (SOA) [13] and Resource Oriented Architecture (ROA) [14]. SOA is coupled to the client/server model that uses Uniform Resource Identifier (URI) to access different services, *i.e.* endpoint. In general, this approach is implemented through the W3C standards,

also called WS-*. Each WS based on SOA requires a Web Services Description Language (WSDL) document to provide communications interfaces, *i.e.,* description of services, message formats, and data to be communicated. In addition, a SOA WS requires the use of SOAP [13] to perform communication among services and clients. SOAP messages are described in eXtensible Markup Language (XML) and must be serialized before transmitted over the network.

ROA [14] is a loosely coupled approach to client/server model that uses URI to directly access resources of a WS, also known as RESTful services. In general, this approach follows the architectural style called Representational State Transfer (REST). This style defines HTTP as the only application protocol and standardizes access interface for its methods (*i.e.,* GET, PUT, POST, and DELETE). Each message of REST is loosely coupled and represents a state of the accessed resource, *i.e.,* the current collection of meaningful information, such as network parameters and hardware configurations. REST states can be described by XML as SOA or also by JavaScript Object Notation (JSON), a lightweight description language.

The research of Pautasso *et al.* [15] shows that ROA presents almost the same features of SOA, but is lighter in terms of network overhead. However, Granville *et al.* [12] presented only SOA as their main WS architecture to provide an alternative to Script MIB. Therefore, the use of ROA as a WS architecture to provide such alternative remains a research question.

Another technology that may be used to realize MbD is the OSGi framework [8] [6]. Through OSGi, Gama *et al.* [4] use gateways that are reconfigured to manage SObjs. Such reconfiguration is carried out by external managers, which delegates management tasks through scripts to OSGi enabled gateways. These scripts are programs called bundles. Such bundles are installed and managed by a single instance of the OSGi as a part of it. However, OSGi does not present SObjs awareness mechanisms, therefore, it cannot be used solely to detect, manage, and monitor SObjs automatically, without external intervention of human managers.

In summary, Script MIB and OSGi may be used to delegate management tasks to gateways located closer to SObjs in the IoT. This delegation enables the management of SObjs with different characteristics. For example, SObjs may be created to measure temperatures periodically or to record videos to be sent constantly over the network. These two different SObjs may be managed by the same gateway. Thus, the Script MIB and OSGi may be used to reconfigure the gateway to manage both types of SObjs. Unfortunately, both the Script MIB and OSGi do not offer a SObjs awareness feature to address the dynamicity of IoT scenarios, where SObjs join and leave the network frequently. This awareness is important to avoid gateways to contact departed nodes and to inform management stations about SObjs in range. It means that the Script MIB and OSGi remain constrained to the traditional network context. Therefore, we proposed a WS new approach, based on MbD concepts, using either the Script MIB or OSGi in combination with WS architectures, SOA and ROA, for IoT management.

III. MbDSAS Architecture

In this section we introduce MbDSAS conceptual architecture and its internal details. From top to bottom, TLM, MLM, and MD are depicted in Figure 1. In addition, a *Script repository* that hosts management scripts can be seen in the right side of Figure 1. White boxes present different applications on both TLM and MLM. Finally, dashed gray boxes present technologies that can be used by our architecture, but not developed by us.

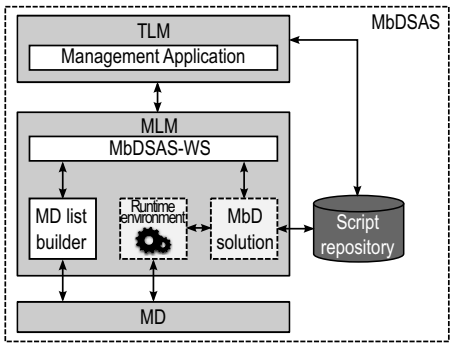

Figure 1. MbDSAS architecture to manage SObjs

Original MbD entities can be properly mapped to IoT objects, *i.e.,* TLMs are mapped to management stations, MLMs are mapped to gateways, and MDs are mapped to SObjs. TLMs are deployed as a *Management Application* to perform specific management tasks, better explained as follows:

- Checking MLMs about discovered MDs: The *Management Application* may send messages to communicate a MLM to verify possible new SObjs in the range of this MLM. When these SObjs are detected, the *Management Application* may delegate to MLM a specific routine to manage these SObjs. Another important feature is the possibility to detect departures and arrivals of SObjs.
- Sending management scripts to MLMs: The *Management Application* may send management scripts to MLMs to manage MDs delegating the management task to network entities closer to MD. This management scripts can also be more than just network management scripts, they can be used as network services exposing WS interfaces to be accessed by network users that are interested in their use as it is depicted in Gama *et al.* [4] by the use of OSGi
- Receiving notifications about new management scripts available from other TLMs: The *Management Application* may be notified about new management scripts available from another TLMs or by the *Script Repository*. When a new notification arrives, the *Management Application* will add this notification to TLM's database organizing accordingly with specific SObjs characteristics and the management script URL. Thus, the *Management Application* may send updated scripts to MLMs.
- Notifying sub-domains about the creation or update of management scripts: The *Management application* may

send notification to sub-domain TLMs about arrived notifications.

The communication among TLMs and MLMs is performed through the network, which may vary in context between a Local Area Network (LAN) or the Internet. Such variability requires that MbDSAS presents a specific protocol and a defined access interface to perform the communication between TLMs and MLMs over both network contexts, whereas the network overhead remains low. Therefore, taking advantage of Granville *et al.* research [12], MbDSAS will be developed with WS to address both requirements stated above.

MbDSAS through MbDSAS-WS may offer interfaces based on WS architectures, such as SOA and ROA. These interfaces are accessed by TLMs to consume services offered by MbDSAS-WS. Such services execute management tasks through information exchange with other internal applications of MLMs (*e.g., MD list builder* and *MbD solution*). In addition, these services may perform other tasks, such as the creation of management sessions or management of the life-cycle of a script.

A subset of MbDSAS-WS services is dedicated to manage an application, which is called *MD list builder*. Such application was created to add SObjs awareness to MLMs. *MD list builder* monitors network interfaces available on MLM to extract relevant information about received messages from available network nodes (*e.g.,* IP, MAC address, or used transport layer protocol). In addition, the *MD list builder* can execute this monitoring taking advantages of discovery protocols, such as the Link Layer Discovery Protocol (LLDP) or the Universal Plug and Play Protocol (UPnP). This monitoring allows the creation of MD Discovery Lists (MDLs). These MDLs are organized in rows based on MD entries presenting their extracted information as columns. Every information about an MD may require the creation or edition of a specific management script. Therefore, TLMs may obtain MDLs to delegate specific management tasks to MLM, avoiding a possible overload on MLM by sending only a set of management scripts to manage SObjs in the range.

MLMs need to install the appropriate management scripts to handle discovered SObjs, that might leave the network at anytime. By the leaving of SObjs, the set of scripts loaded at a given MLM may become soon outdated. To avoid waste of resources in MLMs, a TLM may send an abort command to stop an outdated script from being executed. However, the awareness of an MD behavior depends exclusively on MLMs and this awareness is not covered by current MbD concepts. Therefore, MD list builder is designed to provide a cached information about the last contact with an MD. TLMs can use cached information and a threshold to determine when a management script become outdated and should be terminated or perpetuated.

MbD concepts are designed through services by MbDSAS-WS. For example, a MbDSAS-WS service may be used to receive a management script being delegated by a TLM. These services perform sets of commands to *MbD solutions* (*e.g.,* OSGi or Script MIB). Such solutions have among their functions: pull scripts from external repositories and manage a life-cycle of a scripts.

MbD solutions use runtime environments to execute management scripts and manage their life-cycle. Each *MbD solution* may support one or more runtime environments, such as JAVA, C, and TCL. In such an environment, functions of running scripts are constantly executed to manage MDs. All scripts executions are accomplished without reboots or operational system installations of updates and patches on MLMs. Also, during scripts executions, informations can be returned to TLMs or cached in MLMs.

Finally, the *Script Repository* may be settled as a web server being deployed over a local computer, a cluster of computers, or a cloud system. The deployment of this server shall vary accordingly with the network domain characteristics, such as size or security needs. For example, a huge domain should use a cluster or a cloud system to enable the *Script Repository* to support a huge number of scripts requests and downloads from TLMs. In addition, MbDSAS may take advantage of Domain Name Systems to improve the reachability of a Script Repository. However, in contrast to the former case, an hospital network has high security needs, thus, a local deployed Script Repository may provide a better control over which scripts will be published and will be offered to which network node of the hospital.

A *Script Repository* hosts management scripts. These scripts may be developed and published by managers that are interested in the management of new SObjs. The development of these scripts must be performed considering the supported runtime engines of each MbD infrastructure of an MLM. When such scripts are published on *Script Repositories*, a notification informing the publishing is sent to every known TLM from the Repository or requested by TLMs.

IV. Prototype

The MbDSAS architecture of Section III was prototyped and described bellow. We mainly focused on the design of three components: *Management Application*, *MbDSAS-WS*, and *MD list builder*.

The *Management Application* installed on a TLM was developed in JAVA. This application aims to provide a human-friendly interface to managers in order to send management scripts remotely to MLMs. In addition, the *Management Application* is a service that automatically sends and notifies about updates and creation of management scripts to MLMs and sub-domain TLMs. This sending requires that the *Management Application* communicates *MbDSAS-WS* placed on MLMs; therefore, all necessary communication is based on the specification of *MbDSAS*.

MbDSAS-WS component was developed in PHP to expose services based on WS architectures, *i.e.*, SOA [13] and ROA [14]. These services were developed taking advantage of Granville *et al.* research [12]. However, different from that research, we explored ROA and OSGi as a reconfiguring mechanism in MbDSAS, offering the same management services provided by Script MIB. In addition, we added different

services in *MbDSAS-WS* that are divided in two groups: (*i*) *MbD services*, which implement the concepts of MbD [12], and (*ii*) *MDL services* to acquire and edit MDLs from MLMs. *MbDSAS-WS* services are summarized in Table I.

MbD services		
Service	Parameter	Return
beginMbD	$username, password, solution$	$sessionID$
endMbD	$sessionID$	
scriptPull	$sessionID, url, store, lang$	$scriptID$
scriptRun	$sessionID, scriptIndex, args, block$	$runID$
getResults	$sessionID, runID$	$runResult$
scriptPullAndRun	$sessionID, url, args, lang, store, block$	$runResult$
scriptRemove	$sessionID, scriptID$	
MDL services		
Service	Parameter	Return
getMDL	$sessionID$	MDL
updateMDL	$sessionID, MDEntry, params$	MDL

Table I
WS APPLICATION SERVICES

According to Table I, when a TLM uses the service *beginMbD*, the *MbDSAS-WS* creates a delegation session, validating the parameters *username*, *password*, and a reconfiguration solution to be used (*solution*). With the created session, such TLM is authorized to use other services, for example, TLM may end a session through messages accessing the service endMbD and informing a valid session identification (*sessionID*). Furthermore, a TLM may send a script using the *scriptPull* service to make an MLM to pull a script from an external repository. This service receives from the TLM a valid *sessionID*, a URL to some script stored on the script repository (*url*), information about how the script will be stored on an MLM (*store*), and the specific script language (*lang*), such as JAVA or C. In return, *scriptPull* sends a valid launch identification to trigger the execution of the script (*scriptID*).

A TLM can use a valid launch identification (*scriptID*) with the *scriptRun* service to launch and run a script, informing some arguments to its execution (*args*). In return, *scriptRun* sends a valid script execution identification (*runID*). Afterwards, TLM may use the returned *runID* to retrieve results from the script execution with the *getResults* service, or suspend this execution informing a parameter (*block=1*) with the *scriptRun* service. Moreover, a TLM can use the service *scriptPullAndRun* to summarize the *scriptPull*, *scriptRun*, and *getResults* management services, just informing the same services parameters. Finally, the manager can delete a script by accessing the service *scriptRemove*, informing a valid session identification (*sessionID*) and a script identification (*scriptID*).

MDL services are presented in Table I as well. These services are exposed to simplify the retrieving of MDLs from the *MD list builder* component. Such component was developed in JAVA in combination with the Link Layer Discovery Protocol (LLDP) to monitor network interfaces and to create MDLs. Moreover, the MD list builder component exchanges MDLs by CLI commands with *MbDSAS-WS*, which, in turn, provides MDLs following a set of steps. First, through ROA or SOA based messages, the TLM starts a session accessing the *beginMbD* service, informing a valid *username* and *password*, and then receiving a valid *sessionID*. Afterwards, this TLM may retrieve an MDL through the *getMDL* service informing a valid *sessionID*. When some entry on an MDL must be edited, a TLM can use the *updateMDL* to do so, specifying a valid *sessionID*, one MD identification (*MDEntry*), and the parameters to be changed (*params*). A new MDL is generated and returned to TLM.

The development of a MbDSAS prototype allows to verify how this architecture may be applied in IoT scenarios. In addition, we are interested in analyze MbDSAS performance against a system where all the management scripts are already installed to manage SObjs, henceforth referred to as traditional management. Therefore, we developed a JAVA prototype of the traditional management and analyzed a case study where both prototypes may be deployed for further assessment.

V. CASE STUDY

Through this section, we describe a generic scenario where MbDSAS may be deployed. Afterwards, we apply this scenario in a airport modeled scenario. Finally, metrics are defined for the assessment of the performance of MbDSAS trying to solve different research questions.

A. Scenario

MbDSAS must be evaluated to be classified as a management solution to IoT scenarios. To evaluate MbDSAS we have to define a generic scenario to understand how MbDSAS must be applied in real IoT scenarios, *e.g.,* an airport station, a train station, or a city neighborhood [1]. Therefore, a scenario based on management stations that are interested in the management of SObjs would be the most generic scenario where MbDSAS would be applied. However, to manage these SObjs, management stations must rely on gateways. The components and features of such scenario can be seen in Figure 2 and are described as follows.

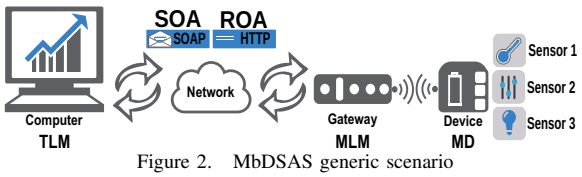

Figure 2. MbDSAS generic scenario

Management station: This element is usually a computer that is also classified as a TLM, which is installed with a *Management Application*. This *Management Application* may be developed to inject workload, monitor experiments, collect results, delegate management, or manage different MLMs through messages based on different access approaches, such as SOA and ROA.

Gateway: The gateway is a network node classified as an MLM that is connected with a network, which usually is a LAN. Through this network, the MLM receives scripts from TLMs or provide MDLs. These scripts are received by a MbDSAS-WS installed on a MLM that exposes services to be consumed based on SOA and ROA. An MLM can also presents one of the covered *MbD Solutions* from this work (the Script MIB or OSGi) to manage scripts. These scripts manage SObjs through communications realized with different

technologies, *e.g.,* IEEE 802.15.4, ZigBee, and Bluetooth. Finally, MLMs are installed with an *MDListBuilder* that may monitor network interfaces or use discovery mechanisms, *e.g.,* LLDP or UPnP, to create MDLs.

Device: The device is characterized as a real example of a SObjs and may be classified as an MD. One of the main features of these devices is that they have constrained resources in relation a traditional computers, such as few memory available, low processing power, and battery based power. In addition, these devices may have a set of sensors, *e.g.,* movement, humidity, and luminosity, integrated into them. Devices can also expose informations regarding their operational characteristics, *e.g.,* memory, clock, power level, and I/O status. Such informations may be accessed through many different approaches, for example SOA, ROA, or others.

Analyzing the generic scenario of MbDSAS allows to find a traditional IoT scenario where it fits. For example, an airport station could be interested in provide flight schedules to SObjs that can handle their service, therefore, a management station uses gateways to provide these schedules to SObjs. However, SObjs may use different technologies (*e.g.,* IEEE 802.15.4, ZigBee, or Bluetooth) to communicate or have particular configuration needs, such as support for communication over layer 2 only or other. In addition, SObjs in an airport are always arriving and departing in anytime. Therefore, through MbDSAS the airport station may be mapped to TLM (management station), MLM (gateway), and MDs (SObjs) as a reconfigurable management system to manage SObjs of many kinds and be aware of their arrivals and departures. We deployed MbDSAS prototype from Section IV to be evaluated in an airport station scenario. We do not have permission to use a real airport station to evaluate MbDSAS, however, we may model its people arrivals and departures on a computer system [16][17].

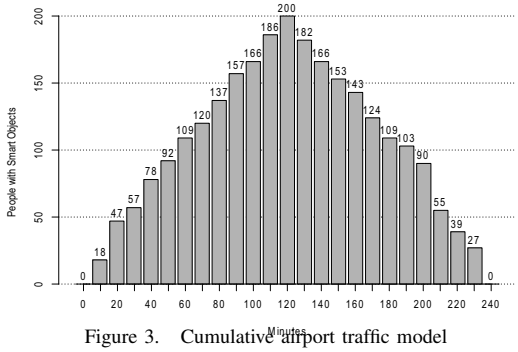

Figure 3. Cumulative airport traffic model

A computer system may replicate a people traffic airport through a poisson distribution with $\lambda = 0.6$, as it is depicted in Figure 3 [17]. In the horizontal axis, four hours are shown in minutes with highlights about each ten minutes. Also, in the vertical axis, columns represent cumulative arrivals and departures of two hundred persons with SObj. In the people traffic replication, people that arrive are summed and when they depart they are subtracted. The flow of arrivals and departures have different values as may be seen in Figure 3.

MbDSAS may be evaluated by experiments using the airport modeled scenario with a TLM and an MLM that has to discover all new MDs in its range and manage them. In addition, MbDSAS has to install and remove management scripts on an MLM to handle the arrival and departure of MDs. During the experiments, the performance of MbDSAS must be measured, therefore, we defined five questions to guide MbDSAS performance assessment and five metrics to solve this questions, which are defined bellow.

- How much time is expended by MbDSAS to perform a management delegation?
- How much network traffic is generated for MbDSAS to perform a management delegation?
- Which is the best combination of technologies to implement MbDSAS?
- How does MbDSAS behave within a dynamic IoT scenario, where SObjs join and leave very frequently?
- How many computational resources of a gateway are needed to execute MbDSAS in terms of memory and CPU utilization?

B. Metrics and Experiments

To answer each of the defined questions above, we used metrics to measure MbDSAS performance during experiments. These experiments take in concern the airport modeled scenario. Through five metrics, we evaluate the best performance technologies available to implement MbDSAS. In addition, MbDSAS is evaluated as a management solution to IoT scenarios. These metrics definitions, methodology, unit, goals, and workload used during experiment are described below.

1) *Response time of MbDSAS*: The response time is the time spent from a complete management script lifecycle. Such time impacts the user experience with the system, *i.e.*, the larger time, the longer waiting for an answer from the script execution. Therefore, the response time is used as a metric to measure the efficiency of the access approaches (SOA and ROA) based on each MbD solution (the Script MIB or OSGi), answering the first and third question defined above. In addition, the standard access approach to execute management delegation, *i.e.*, SNMP, may be compared with MbDSAS-WS as another access approach, therefore, a script delegation based on SNMP messages was realized with the Script MIB to be used as a baseline to measure MbDSAS efficiency. However, the same could not be realized to the OSGi, because no standard was defined so far to provide SNMP access to such solution. In this case, we performed experiments based on the response time (in seconds) to determine MbDSAS efficiency. TLM uses a workload based on *scriptPullAndRun* tasks to send scripts to an MLM. These scripts perform just a time waiting varying from zero, five, ten, and fifteen seconds. This way, both access approaches can be compared when different performance scripts are executed.
2) *Network traffic of MbDSAS*: The network traffic is the amount of bytes transferred over the network during

a TLM and MLM communication. Such communication impacts the network with overhead, *i.e.,* the larger amount of bytes transferred, the greater overhead caused on the network. Therefore, the network traffic was adopted in this work as a metric to measure the efficiency of the access approaches (SOA and ROA) based on each MbD solution (the Script MIB and OSGi), answering the second and third questions defined above. In addition, a SNMP script delegation was also measured in terms of network traffic to be used as a baseline. In this case, we performed experiments based on the network traffic (in KB) to determine MbDSAS efficiency. These experiments were conducted with the same workload of the time response metric.

3) *Management delay*: The management delay is the amount of time spent since an MD comes to the range of an MLM and starts to be managed. This delay impacts the quality management (QoM), *i.e.,* the performance of a management system to achieve its main objective, in this case, the management of all SObjs in an airport scenario. Therefore, the management delay was adopted in this work as a metric to measure the efficiency of MbDSAS against a traditional management prototype. In this case, we performed measurements during the experiments based on the management delay (in seconds) to determine such efficiency, answering the fourth question defined above. This experiments were conducted with a workload based on the modeled airport scenario with a TLM consuming *scriptPullAndRun*, *scriptRemove*, and *getMDL* services. This TLM become updated about discovered MDs from an MLM to send or remove management scripts and to start or stop a management script execution as well.
4) *CPU utilization*: The CPU utilization is the amount of CPU processing capacity used by an MLM during the execution of MbDSAS. This CPU utilization impacts the performance of an MLM, *i.e.,* the larger amount of CPU used the lesser amount of remaining processing power an MLM will have, decreasing the number of process an MLM can support. Therefore, the CPU utilization was adopted in this work as a metric to measure the efficiency of MbDSAS against the traditional management. In this case, we performed we performed measurements during the experiments based on the CPU utilization (in percent) to determine such efficiency, answering partially the fifth question defined above. These experiments were conducted with the same workload as the management delay experiments.
5) *Memory utilization*: The memory utilization is the amount of memory spent by an MLM during the execution of MbDSAS. This memory utilization impacts the performance of an MLM, *i.e.,* the larger amount of memory consumed the lesser amount of remaining memory an MLM will have. Therefore, the memory utilization was adopted in this work as a metric to measure the efficiency of MbDSAS against the traditional management prototype. In this case, we performed measurements during the experiments based on the memory utilization (in percent) to determine such efficiency, completing the answer of the fifth question defined above. The experiments evolving the memory utilization were conducted with the same workload as the management delay experiments.

In the next section, experimental results collected considering the proposed scenario and the presented methodology are shown and discussed.

VI. Results and Analysis

The experiments were based on a test environment defined bellow. Afterwards, we shown results obtained during the execution of these experiments. Finally, we analyzed each obtained result and presented our considerations.

A. Test environment

The test environment was composed of three main entities: (*i*) a personal computer as TLM, that running the Management Application software, developed in Java to delegate scripts and collect MDLs of MLMs; (*ii*) two reconfigurable management solutions as MLMs, *i.e.,* Script MIB and OSGi, that were instantiated in virtual machines with an MbDSAS-WS and an MD List Builder component; and (*iii*) two hundred MDs instantiated in virtual machines to replicate the behavior of the airport scenario described in previous section. The option of using virtual machines was because of the huge number of devices involved and to simplify the migration of an MLM through other network domains. In addition, each virtual machine environment used Kernel-based Virtual Machines (KVM), libvirt, and Virt-Manager application inspired in the research of Wickboldt *et al.* [18].

In Table II, all the hardware and software specifications of involved entities are summarized.

Specification	TLM	MLM - Script MIB	MLM - OSGi	MD
Processor Power (MHz)	2000 × 4	500	500	100
RAM Memory (MB)	8000	128	128	8
Hard Disk (MB)	650000	1000	1000	8
Network interface	Ethernet	Ethernet	Ethernet	Ethernet
Operational System	Arch 3.0.40	Debian 3.0	Arch 3.0.40	OpenWRT
Script MIB	-	JASMIN 0.96	-	-
OSGi	-	-	Apache Felix 4.0	-
Web Server	-	Apache 2.2.22-4	Apache 2.2.22-4	-
SNMP	-	net-snmp 4.2	net-snmp 4.2	-
JDK	7.1	1.1 / 1.2 / 6.0	7.1	-
PHP	-	5.0	5.0	-
Sqlite	-	2.4.7	2.4.7	-
TCL	-	8.2	-	-
LLDPD	-	0.5.7	0.5.7	0.5.7

Table II
Hardware and software specifications

The specified hardwares and softwares depicted above were used during experiments based on the defined metrics of the Subsection V-B. In addition, all of these metrics were collected 30 times and summarized in averages. For each average, the confidence interval of 95% is showed for each sample plotted graphically in next subsections.

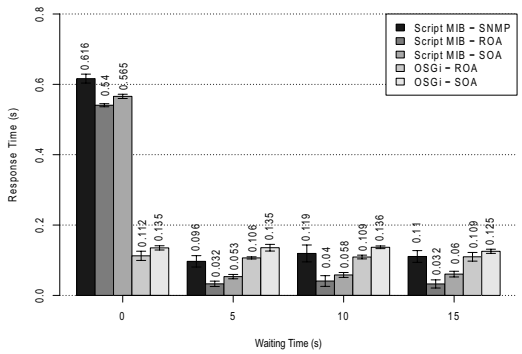

Figure 4. Response time of MbDSAS using *scriptPullAndRun*

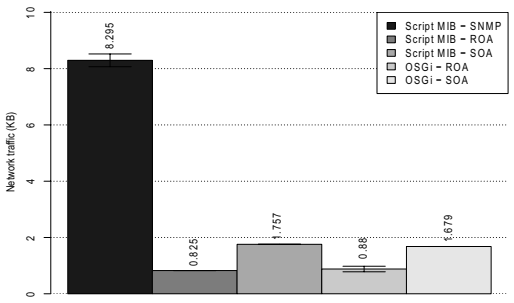

Figure 5. Network traffic of MbDSAS with zero waiting time

B. MbDSAS response time analysis

In Figure 4, management delegation solutions was measured through the textitscriptPullAndRun task performance to determine the response time of the MbDSAS for each access architecture (ROA and SOA) combined with a *MbD solution* (Script MIB and OSGi). In addition, the same management delegation based on SNMP is performed and measured to be used as a comparison baseline. Such measurements are presented in seconds by the vertical axis being represented by columns. At the top of each column the confidence interval is shown. In the horizontal axis, each measurement is related to a combination between access architecture and a MbD solution, that was performed in different waiting times, *i.e.,* 0, 5, 10, and 15 seconds.

As can be seen in Figure 4, without waiting time, *i.e.,* equal to 0, Script MIB results has a different behavior from others, it occurs because management delegation process requires a polling process based on SNMP messages to be performed [12]. This polling process insert an overhead to retrieve information from scripts execution that perform in few seconds. However, this overhead is not replicated through scripts that require more time to be executed, *i.e.,* for waiting times 5, 10, and 15.

Comparing *MbD solution* to be used with MbDSAS, Script MIB presents the best response time being at least 30% faster than the OSGi at the best case (waiting time 5). However, Script MIB has an unstable behavior with different performance scripts, being at least 4 times slower than OSGi in the worst case (waiting time 0). Therefore, because OSGi is less unstable, we prefer it over SNMP as the *MbD solution* to be used in MbDSAS.

Additionally, in Figure 4, the average response time of ROA approach is at least 10% faster than the average response time of the SOA approach, and 14% faster than the SNMP baseline. Through this results ROA access architecture is possible to evaluate which is the best

C. MbDSAS network traffic

Similar to the response time, in Figure 5, a script delegation using the $scriptPullAndRun$ task was evaluated to determine the network traffic of MbDSAS for ROA and SOA combined with Script MIB or OSGi. Such measurements are presented in averages in KB by the vertical axis being represented by columns. At the top of each column the confidence interval is shown. Differently from the response time, these measurements were performed only for scripts with no processing time, *i.e.,* scripts that have waiting time equals to zero. A different measurement would be unfair to compare with the SNMP approach, which presents a polling strategy to communicate with Script MIB.

In Figure 5, ROA access architecture presents almost 2 times less network traffic than SOA and 10 times less than SNMP. This difference of network traffic is particular related to the lightness of JSON based messages from ROA against the excessive verbosity of XML messages from SOA and the polling strategy adopted with SNMP. However, network traffic results present a confidence interval that intercept each other. Therefore, ROA network traffic for Script MIB and OSGi is quite similar and cannot be used to determine the best *MbD solution* to be used by MbDSAS.

In summary, answering the first, second and third research question of Section V, MbDSAS performed better with the ROA access architecture in response time (0.032 seconds) and network traffic (0.825 KB). Analyzing this architecture results, OSGi is preferred as the most stable solution against the Script MIB that has the best performance in MbDSAS. Therefore, we used MbDSAS based on the combination of OSGi and ROA approach in the next experiments to evaluate the management performance of MbDSAS against the traditional management.

D. MbDSAS management delay

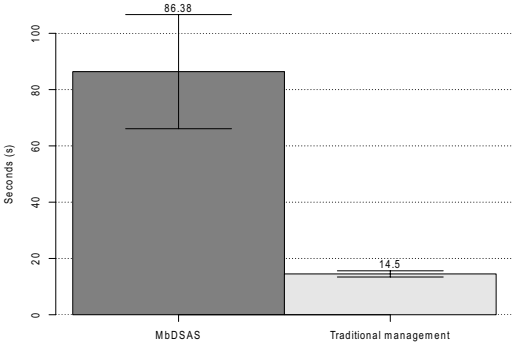

Figure 6. Management delay of MbDSAS (OSGi-ROA)

MbDSAS and the traditional management are compared in Figure 6. MbDSAS reconfigure dynamically the management scripts while traditional management has all management scripts previously installed. The airport modeled scenario, described in Section V, was used during four hours of experiment. The measurements are presented in averages in seconds by the vertical axis being represented by columns. At the top of each column the confidence interval is shown. In the horizontal axis, the management approach adopted labels each column.

As can be seen in Figure 6, MbDSAS presents a difference in management delay against the traditional management approach of at least 50 seconds. This difference was expected because MbDSAS has to install a management script on the fly to start the management of a SObjs, discovering SObjs and reconfiguring the MLM two hundred times during the experiments to manage every SObjs of the airport modeled scenario. However, in a real scenario, is preferable that management solutions have flexibility to interact dinamically with new SObjs than traditional management systems that not support it. Therefore, a trade-off may be observed, where 50 seconds of management delay may be assumed as a price to pay to add reconfiguration and discovery features to a management system.

E. MbDSAS CPU utilization

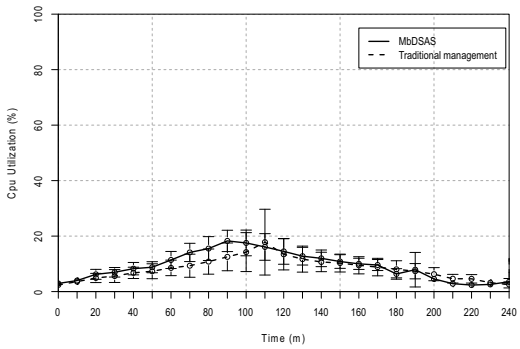

Figure 7. CPU utilization of MbDSAS (OSGi-ROA)

In parallel with the management delay measurement, MbDSAS and the traditional management was used to manage the airport modeled scenario, as depicted in Figure 7. During this experiment we measured their CPU utilization. Such measurement is presented in averages in percents by the vertical axis being represented by lines. In the horizontal axis, the current experiment time is expressed by minutes and highlighted about each ten minutes with its current confidence interval.

Surprisingly, the CPU utilization behavior of MbDSAS and the traditional management are similar, despite of MbDSAS install scripts, discover SObjs and reconfiguring the MLM two hundred times. Both of then have their higher CPU utilization near two hours and their lower use at the beginning and ending of the experiment (0 and 240 minutes). This behavior is explained by the airport modeled scenario, where at the beginning SObjs are arriving and becoming managed, near two hours the maximum number of SObjs being managed is achieved, and till the end every SObjs are departing and their management is deleted or wait in background.

Almost all the confidence intervals of MbDSAS and the traditional management intercept each other, therefore, they are similar and cannot be compared in terms of CPU utilization. However, this is good for MbDSAS because indicates that even with new features, management delegation and SObjs awareness, the MLM is not particularly influenced in CPU utilization more or less then the expected of a traditional management solution, answering partially the fifth question of Section V.

F. MbDSAS memory utilization

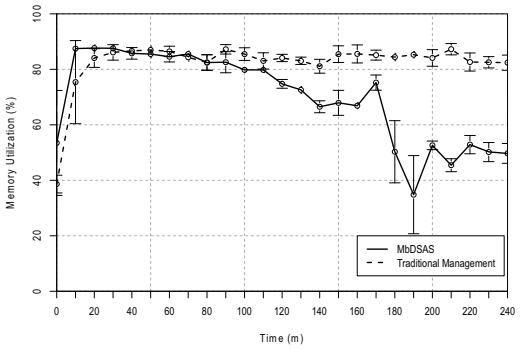

Figure 8. Memory utilization of MbDSAS (OSGi-ROA)

Similar to the CPU utilization MbDSAS and the traditional management was used to manage the airport modeled scenario and during this experiment we measured their memory utilization as can be seen in Figure 8. Such measurement is presented in averages in percents by the vertical axis being represented by lines. In the horizontal axis, the current spent time is expressed by minutes and highlighted about each ten minutes. The confidence interval is shown accordingly with this highlights.

As can be seen in Figure 8, MbDSAS and the traditional management behave similar at the beginning of the experiment using the available memory and reaching the maximum available near tow hours of experiment. However, different from the CPU utilization, the memory used by MbDSAS drops drastically near the end of the experiment. This drop of memory utilization is related to the departure of SObjs that, when detected by the TLM, the management script responsible by the management of this departed SObjs is stopped and after removed. Therefore, the memory of the MLM is freed and the resource is better used by MbDSAS.

Almost all the confidence interval of MbDSAS and the traditional management measurement intercept each other near the beginning indicating that they are similar. However, with the changing of behavior of MbDSAS near the ending, the confidence interval of the measures stop to intercept each other and allow to determine that MbDSAS uses better the memory than a traditional management. Answering the forth question created at Section V, MbDSAS uses better the resources of a MLM than a traditional management.

VII. Conclusion & Future Work

In this paper, we presented Management by Delegation (MbD) concepts in the IoT with the goal of managing SObjs. We used these concepts to propose MbDSAS, an architecture for gateway reconfiguration through an MbD Solution, *i.e.,* the Script MIB or OSGi. Such reconfiguration is carried out by developed scripts that, when executed, perform the management of SObjs. These scripts are sent by management stations to gateways close to SObjs. Such gateways are reconfigured to manage SObjs, performing part of the management of an entire network in a distributed way.

MbDSAS also includes a MD list builder component that is in charge of adding SObjs awareness to gateways. Such awareness is provided by SObjs entries in a MDL created by the MD list builder. This MDL, which is not found in traditional MbD solutions, is key to manage dynamic IoT scenarios. As such, we combine the MD list builder with a MbD solution to expose services to external managers. That allows managers to be aware of what kinds of objects are available in gateways and which scripts may be delegated to manage them. As a result, MbDSAS can be used to manage dynamic IoT scenarios.

We conducted an experimental evaluation to observe most appropriate access methods and MbD solutions to be used with MbDSAS. Results showed that MbDSAS implemented through the ROA access architecture performs better in terms of response time and network traffic. We also observed that the combination of ROA and OSGi results in less variation of the response time when services are accessed. The combination of ROA and the Script MIB, however, presented better response times in MbDSAS.

The performance of MbDSAS as an IoT management approach was also experimented through a modeled airport scenario. In such a scenario, we compared the management delay, CPU utilization, and memory utilization of MbDSAS against a traditional management system, *i.e.,* a management station that has all necessarily software to manage every SObjs present at an airport scenario. Results shown that MbDSAS has a small management delay compared to the traditional management. Afterwards, we showed that the CPU utilization of both approaches are very similar. However, during its execution, MbDSAS continuously freed memory resource for other processes to execute, proving its superiority in saving resource compared to a traditional management. In summary, MbDSAS added reconfiguration features to a management station without great overheads to manage SObjs, showing to be qualified to manage real IoT scenarios. In addition, MbDSAS behave better in an dynamic scenario than a traditional management system.

As a future work, we will investigate the use of MbDSAS in Mashups systems [19]. We believe that in Mashups, the delegation of scripts and the access of MDLs can assist managers to develop complex network management system to dynamic scenarios of IoT.

References

[1] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Vision and challenges for realising the internet of things," *Cluster of European Research Projects on the Internet of Things, European Commision*, 2010.

[2] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. Jubert, M. Mazura, M. Harrison, M. Eisenhauer10 *et al.*, "Internet of things strategic research roadmap," *Aerospace Technologies and Applications for Dual Use*, p. 9, 2008.

[3] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the web of things," in *Internet of Things (IOT), 2010*. IEEE, 2010, pp. 1–8.

[4] K. Gama, L. Touseau, and D. Donsez, "Combining heterogeneous service technologies for building an internet of things middleware," *Computer Communications*, vol. 35, no. 4, pp. 405 – 417, 2012.

[5] H. Viswanathan, B. Chen, and D. Pompili, "Research challenges in computation, communication, and context awareness for ubiquitous healthcare," *Communications Magazine, IEEE*, vol. 50, no. 5, pp. 92 –99, may 2012.

[6] J. Rellermeyer, M. Duller, K. Gilmer, D. Maragkos, D. Papageorgiou, and G. Alonso, "The software fabric for the internet of things," in *Proceedings of the 1st international conference on The internet of things*. Springer-Verlag, 2008, pp. 87–104.

[7] D. Levi and J. Schoenwaelder, "Definitions of managed objects for the delegation of management scripts - rfc3165," *RFCs and Standards, IETF*, 2001.

[8] OSGi Alliance, "Osgi - the dynamic module system for java," http://www.osgi.org/Main/HomePage, 2012, [ONLINE. Last access at august, 22].

[9] G. Goldszmidt, Y. Yemini, and S. Yemini, "Network management by delegation: the mad approach," in *CASCON First Decade High Impact Papers*, ser. CASCON '10. New York, NY, USA: ACM, 2010, pp. 78–92.

[10] J. Schonwalder, J. Quittek, and C. Kappler, "Building distributed management applications with the ietf script mib," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 5, pp. 702–714, may 2000.

[11] J.-P. Martin-Flatin, S. Znaty, and J.-P. Hubaux, "A survey of distributed enterprise network and systems management paradigms," *Journal of Network and Systems Management*, vol. 7, no. 1, pp. 9–26, 1999.

[12] L. Z. Granville, R. Neisse, R. L. Vianna, and T. Fioreze, *Handbook of Research on Telecommunications Planning and Management for Business*, I. Lee, Ed. IGI Global, Mar. 2009.

[13] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (SOAP) 1.1," 2000.

[14] R. Fielding and R. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.

[15] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. "big"' web services: making the right architectural decision," in *Proceeding of the 17th international conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 805–814.

[16] Y. Park and S. B. Ahn, "Optimal assignment for check-in counters based on passenger arrival behaviour at an airport," *Transportation Planning and Technology*, vol. 26, no. 5, pp. 397–416, 2003.

[17] S. Solak, J.-P. B. Clarke, and E. L. Johnson, "Airport terminal capacity planning," *Transportation Research Part B: Methodological*, vol. 43, no. 6, pp. 659–676, 2009.

[18] J. A. Wickboldt, L. Z. Granville, F. Schneider, D. Dudkowski, and M. Brunner, "A new approach to the design of flexible cloud management platforms," in *8th International Conference on Network and Service Management (CNSM)*, Las Vegas, USA, October 2012, pp. 155–158.

[19] C. R. P. dos Santos, R. Bezerra, J. Ceron, L. Z. Granville, and L. M. R. Tarouco, "On using mashups for composing network management applications," *Communications Magazine, IEEE*, vol. 48, no. 12, pp. 112 –122, december 2010.