UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MARCELO ANTONIO MAROTTA

A Management by Delegation Smart Object Aware System for the Internet of Things

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof[a]. Dr[a]. Liane Margarida Rockenbach Tarouco
Advisor

Porto Alegre, June 2013

CIP – CATALOGING-IN-PUBLICATION

Marotta, Marcelo Antonio

A Management by Delegation Smart Object Aware System for the Internet of Things / Marcelo Antonio Marotta. – Porto Alegre: PPGC da UFRGS, 2013.

73 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2013. Advisor: Liane Margarida Rockenbach Tarouco.

1. Internet of things. 2. Network management. 3. Smart objects. 4. Management by delegation. 5. OSGi. 6. Script MIB. I. Tarouco, Liane Margarida Rockenbach. II. Título.

“No fact is too pure
to be definitive”
— GONÇALO M. TAVARES

ACKNOWLEDGMENTS

I would like to express my gratitude to my mother Maria Aparecida Sousa Gay Marotta and also my sister Samantha Marotta. They always provided me all the financial and emotional support that I needed during my whole life. They are my life examples, my idols, and also my goals. In addition, I would like to thank my stepfather, brother in law, nephew, and niece, Antonio Donizete Silva Santos, Nesken Diniz, Yago Marotta Diniz, and Maria Paula Marotta Diniz, respectivelly, by all the love that was given to me.

I am grateful to my girlfriend, Renata Sayuri Muranaka, who spent the last five years by my side. She always listened carefully to my problems and showed her intention to help. Also, she laughed and cried with me, however, always offered me her smile, even when wrapped in tears. My companion, my friend, my love, and my happiness.

I would like to express my gratitude to those that I cannot say it in personally, my father Antonio Ernesto Marotta and my grandmother Maria da Penha Sousa Gay. Both of them are not among of us anymore, however, they are still alive inside of me, in my dreams and thoughts.

I am grateful to my family, which always took care of me with love, providing me happiness. In particular, I would like to thank my grandfather, Sebastião Milton Gay, by his affection and patience with me.

I would like to express my gratitude to my advisor, Liane Margarida Rockenbach Tarouco, by always encouraging me and guide me to be a better researcher. In addition, I would like to thank my professor, Lisandro Zambenedetti Granville, for his good advices and time, always trying to polish my inner researcher. Also, I am thankful to my professors that participates on my masters, professors Ingrid Eleonora Schreiber Jansch Pôrto, Luciano Paschoal Gaspary, Marcelo Pimenta, Marinho Pilla Barcellos, and Juergen Rochol.

I am grateful to all of my friends, mainly, Anderson Rocha Tavares, Camila Wrasse, Carlos Raniery P. dos Santos, Cristiano B. Both, Grasiela Alves de Castro, José Felipe Carbone, Isabel Flor, José Jair C. Santanna, Juliano A. Wickboldt, Leonardo Faganello, Lucas Bondan, Luiz Otávio V. B. Oliveira, Renan Amaral, Sérgio Montazolli Silva, and Wanderson J. Paim. All of them were always supporting me by giving their best effort to help me or by just being there and making me happy.

I would like to thank also the rest of my friends, colleagues, professors, and staff of the Informatics Institute from UFRGS, by their good work and assistance. Mainly, Luís Otavio and Carlos Alberto (Cabeto), by their support during the accomplishment of this work.

Finally, I would like to express my gratitude to those people that influenced me directly or indirectly during the masters and also my life. Thank you all.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application and Programming Interface
CLI	Command Line Interface
GaaS	Gateway as a Service
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
JASMIN	Java Script-MIB Implementation
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LLDP	Link Layer Discovery Protocol
MbD	Management by Delegation
MbDSAS	Management by Delegation Smart Object Aware System
MbDSAS-WS	Management by Delegation Smart Object Aware System - Web Service
MD	Managed Device
MIB	Management Information Base
MLM	Mid-Level Manager
NAT	Network Address Translation
OSGi	Open Service Gateway initiative
P2P	Peer-to-Peer
REST	Representational State Transfer
RFC	Request for Comments
ROA	Resource Oriented Architecture
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol

SObj	Smart Objects
TCP	Transmission Control Protocol
TLM	Top-Level Manager
UFRGS	Federal University of Rio Grande do Sul
UPnP	Universal Plug and Play Protocol
W3C	World Wide Web Consortium
WS	Web Service
WSDL	Web Service Description Language
XML	eXtensible Markup Language

LIST OF FIGURES

LIST OF TABLES

Through the Internet of Things - A Management by Delegation Smart Object Aware System

RESUMO

Os objetos inteligentes (SObjs) são numerosos e irão comunicar-se diretamente através da Internet das Coisas (IoT). Esse grande número de SObjs, pode levar a IoT a enfrentar severas condições de rede, em termos de congestionamento na rede e atrasos na comunicação fim-a-fim. Assim, o gerenciamento de SObjs torna-se fundamental para evitar futuros problemas da IoT. Em tal gerenciamento, *switches*, *network-boxes* e roteadores, também chamados de *gateways*, são configurados para gerenciar SObjs através de reconfigurações e atualizações de software seguidas por reinicializações nos *gateways*. Entretanto, a dinamicidade da IoT, causa a necessidade de se reconfigurar os *gateways* frequentemente, *i.e.,* a configuração vigente dos *gateways* torna-se rapidamente desatualizadas por lidar a todo momento com a entrada e saída de novos SObjs da rede. Assim, propõe-se uma abordagem chamada *Management by Delegation Smart Object Aware System* (MbDSAS), para reconfigurar *gateways* e gerenciar SObjs, sem a necessidade de uma atualização de *software* ou aplicação de *patches*, lidando com a dinamicidade da rede IoT. MbDSAS foi prototipado e uma avaliação foi realizada, através de um cenário, onde o fluxo de dispositivos existentes se assemelha com o de um aeroporto. Em adição, MbDSAS foi testado experimentalmente, em termos de desempenho, para avaliar sua qualificação como uma solução de gerenciamento para cenários da IoT e determinar a melhor combinação de tecnologias para implementar essa solução. Os resultados encontrados mostram que MbDSAS desempenha melhor quando desenvolvida com uma arquitetura de acesso baseada em recursos e através do uso de um módulo *Open Service Gateway initiative* (OSGi). MbDSAS quando comparada a um sistema de gerenciamento tradicional, se mostra superior em termos de consumo de memória e processamento, se classificando como uma importante solução para o gerenciamento de SObjs da IoT.

Palavras-chave: Internet das coisas, gerenciamento de redes, objetos inteligentes, gerenciamento por delegação, script MIB, OSGi.

ABSTRACT

The smart objects (SObjs) are numerous and will communicate directly through the Internet of Things (IoT). Such huge number of SObjs may lead the 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. The results shown that MbDSAS has its performance improved when developed with an architecture based on resources and through the use of a module textit Open Service Gateway Initiative (OSGi). MbDSAS when compared to a traditional management system shows superior in terms of memory consumption and processing, being classified as an important solution for the managing of SObjs from the IoT.

Keywords: Internet of things, network management, smart objects, management by delegation, OSGi, Script MIB.

1 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) (Dunkels e Vasseur, 2008). The Internet has been gradually incorporating 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. Predictions (Sundmaeker et al., 2010) 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 the 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 (Guinard et al., 2010), 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 (Gama et al., 2012). For example, gateways can cache SObjs management information to improve the perceived availability of SObjs from the manager point-of-view (Viswanathan et al., 2012).

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 (Bottazzi et al., 2006)(Vermesan et al., 2008)(Mukhtar et al., 2008). Routers, switches, access points, and other network boxes can perform proper intermediation between managers and SObjs (Rellermeyer et al., 2008)(Gama et al., 2012), and, in this case, are referred to as management gateways in this dissertation.

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 often reconfigured to be able to intermediate a varying number and types of SObjs without passing through the typical process of a software

update usually followed by a warm start. However, currently management gateways tend not to offer dynamic reconfiguration features to avoid warm starts, remaining constrained to the traditional network of computers.

Gateways management codes are limited to manage a finite number of SObjs. These codes must be precisely designed to avoid unnecessary software processing. In the IoT, SObjs may join and leave the network in an unspecific time, leading management gateways to process unnecessary computer routines. In addition, new SObjs that join the network may be left without management because they are not covered by the current management code. Therefore, gateways must offer SObjs awareness solutions to determine which are the SObjs in range to adjust the management code more precisely. In summary, today's management gateways tend neither to offer dynamic reconfiguration features nor SObjs awareness solutions, which represents a critical problem when intermediating the management of SObjs in the IoT. Therefore, we designed an architecture that present both features, SObjs awareness and dynamic reconfiguration.

In this dissertation we described 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, such as Link Layer Discovery Protocol (LLDP) (Congdon, 2002) or Universal Plug and Play (UPnP) (Reynolds, 2006). In addition, MbDSAS allows the creation of lists based on the detected behavior of SObjs to be later obtained by managers through Web Services (WS). Afterwards, managers can use Management by Delegation (MbD) (Fioreze et al., 2005; Granville et al., 2009; Goldszmidt et al., 2010) concepts to delegate the management of SObjs tasks to gateways. Such delegation is carried out by the use of WS in combination with the IETF Script MIB (Levi e Schoenwaelder, 2001) and Open Service Gateway initiative (OSGi) (OSGi Alliance, 2012), which enables gateways reconfiguration without the need of patches or updates followed by a warm start.

A prototype of MbDSAS was designed to explore and evaluate the combination of different technologies. For example, MbDSAS prototype was deployed with different WS architectures, *i.e.,* Service Oriented Architecture (SOA) (Box et al., 2000) and Resource Oriented Architecture (ROA) (Fielding, 2000). Both of these architectures were explored to seek best performance with WS that, when consumed by managers, provide the same functionalities of the Script MIB and OSGi. To qualify MbDSAS as a IoT management solution, its prototype was deployed in a typical IoT scenario, more precisely, into an airport station (Sundmaeker et al., 2010) modeled through poisson distributions to the assessment of MbDSAS. Finally, results were experimentally collected in terms of response time, network traffic, management delay, CPU, and memory utilization to address the following research questions.

- How to adapt gateways to offer dynamic reconfiguration and Smart Objects awareness features to cope with the IoT dinamicity?
- How much cost in terms of hardware resources for a gateway to provide dynamic reconfiguration and Smart Objects awareness features?
- Which are the benefits and trade-offs acquired from a new solution for IoT management based on reconfiguration and Smart Objects awareness features?

The remainder of this dissertation is organized as follows. In Chapter 2, we present a background and the state-of-the-art on management solutions in the IoT. In Chapter 3, we

describe our solution to configure gateways to manage SObjs. A proof of concept of our solution is presented in Chapter 4. In Chapter 5, a case study and a scenario are defined to evaluate our solution. In addition, the achieved results are discussed. Finally, in Chapter 6, we conclude this dissertation presenting final remarks and future work.

2 BACKGROUND & RELATED WORK

Through this chapter we discuss fundamental work related to our solution. We present the underlying concepts of our work reviewing the distributed management approach called Management by Delegation. Afterwards, the IETF's Script MIB is described to enable the understandability of how an architecture should apply MbD to reconfigure gateways. In addition, OSGi is described as a mechanism to reconfigure gateways as well as Script MIB. Finally, we discuss how Web Services and SObjs awareness mechanisms have to be used in combination to manage the IoT, reviewing current projects that use such a combination.

2.1 Management by Delegation

Traditional centralized management approaches are usually insufficient to manage large networks (Goldszmidt et al., 2010). In such a management, a single station manage all variety of network nodes presented in one network domain. However, with the increasing size of modern networks, a single manager becomes easily overloaded with management information (Goldszmidt et al., 2010). Thus, the centralized management is expected to be replaced by distributed management approaches to avoid such overload.

Distributed network management systems are typically composed by managers, agents, and dual-role entities. Managers are characterized by entities, which perform management functions that includes monitoring an network domain or creating summarized reports about domains characteristics. An agent, in its turn, is typically a network node that performs mundane actions, such as gathering, caching or providing access to data. Finally, dual-role entities are network nodes that assume both roles as managers and agents, making the definition of management tasks and agent tasks to blur (Schonwalder et al., 2000).

The distribution of network management can be accomplished with the Management by Delegation (MbD) model (Goldszmidt et al., 2010). This model is based on delegation, *i.e.,* the process of transferring power, authority, accountability and responsibility for a specific task among network entities (Martin-Flatin et al., 1999). Such entities are hierarchically organized in a top-down model: (*i*) Top-Level Managers (TLMs) are stations responsible for the management of an entire network domain; (*ii*) Mid-Level Managers (MLMs) are network nodes capable of executing delegated management tasks; and (*iii*) Managed Devices (MDs) are the final target of the management tasks (Granville et al., 2009). Scripts contain the code describing the management tasks, which are sent from 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 may depend on the

management interfaces exposed by MDs and the routine implemented in the management script.

Martin-Flatin *et al.,* (Martin-Flatin et al., 1999) defines a distribution level that classifies MbD systems accordingly with their management organization, ranging from centralized, via distributed, and cooperative management. On one hand, centralized management and via distributed systems presents an hierarchical management organization, in which MLMs are typically constrained to mundane functions. On other hand, in the via-distributes and cooperative management, MLMs assume both roles as TLM and MLM, *i.e.,* a dual-role entity. These MLMs communicate other MLMs, delegating management tasks and sharing information. With the increasing number of MLMs, a management system become more elastic, *i.e.,* a system with entities that perform or share management tasks with others dynamically, creating an enhanced local autonomy (Goldszmidt et al., 2010).

In Schonwalder *et al.,* (Schonwalder et al., 2000), an MbD system is classified as centralized, weakly distributed, strongly distributed, and cooperative management accordingly with the number of TLM and MLMs in a system. In addition, in each classification, MLMs can play both roles, as agent or dual-role entity. The total number of TLMs and MLMs in an MbD system may be equated by μ, whereas η represents the total number of elements in an MbD system. Therefore, accordingly with Schonwalder *et al.,* (Schonwalder et al., 2000):

$\mu = 1$	centralized management;
$1 < \mu \ll \eta$	weakly distributed management;
$1 \ll \mu < \eta$	strongly distributed management;
$\mu \approx \eta$	cooperative management;

Management distributed classifications defined above may be better understood by Figure 2.1. In this figure, centralized, weakly, strongly distributed, and cooperative management are presented. White circles represent TLMs, gray ellipses are MLMs, and black ellipses are MDs. In addition, each black line represents a possible communication interaction among entities. A centralized management is depicted in Figure 2.1.*a*, where a single TLM is handling all the management of an entire network domain ($\mu = 1$). For each MD, a TLM uses computer routines based on direct network interactions to perform monitoring, controlling, and management. However, when the number of MDs raises, the number of management tasks also raises, leading TLM to become overloaded. Thus, distributing the management task among intermediary entities may avoid problems related to overload.

A weakly distributed management presents few MLMs and TLMs ($1 < \mu \ll \eta$), as can be seen in Figure 2.1.*b*. In such a management, TLMs usually use MLMs to just perform mundane functions, the level of distribution remains low (Schonwalder et al., 2000). It means that TLMs still being overloaded by the overall management task because MLMs has a minimalistic participation in the management task, requiring TLMs to manage it.

In the strongly distributed management and cooperative management, the number of TLMs and MLMs grows as depicted in Figures 2.1.*c* and *d*. The overall task management is distributed among almost all network entities. The strongly distributed management presents some TLMs, whereas, the cooperative management is nearly composed solely by dual-role entities MLMs and some few MDs. Such a high-level of distribution enables MbD based systems to achieve the most important characteristics for this dissertation: scalability and flexibility (Schonwalder et al., 2000).

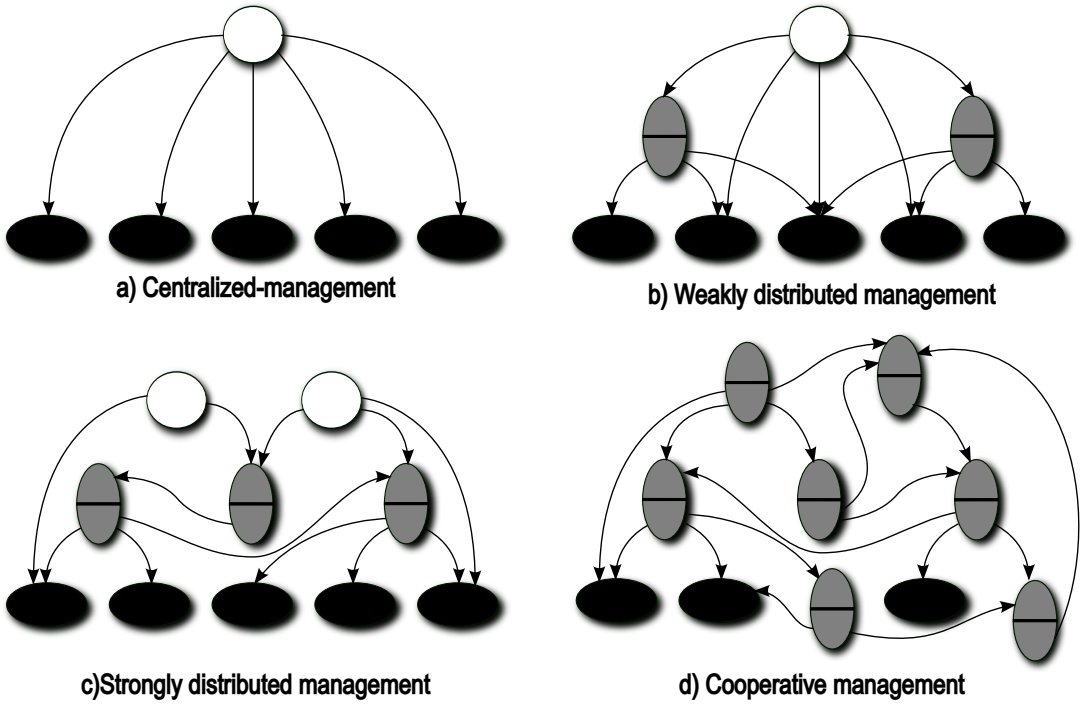

Figure 2.1: Distribution level classification

- Scalability: Accordingly to Schonwalder *et al.,* (Schonwalder et al., 2000), 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. Afterwards, these MbD systems can spread the new script among its internal nodes and neighbors MbD systems dynamically. Therefore, management and agent tasks can be freely reassigned dinamically without the need of a software update usually followed by a warm start.

The distribution level of an MbD system must be adapted accordingly with the given management task (Schonwalder et al., 2000). A detailed discussion of management task characteristics that can exercise some influence in the distribution level choice is discussed in Meyer *et al.,* (Meyer et al., 1995).

There are other distributed network management approaches, for example, Santos *et al.,* (dos Santos et al., 2008) developed a P2P distributed network management approach, which rely on overlay peers communication to distribute their informations among other peers to monitor, control, and manage a network domain. However, caching information among peers presents a problem when P2P distributed network management is applied to scenarios with security concerns, such as military, healthcare, and banking. In these scenarios, source and destiny are the only peers allowed to understand each other's messages

and cache than. Therefore, an attack to an intermediary peer that had cached these messages become a risk to the security communication. Through MbD, the communication is reconfigured at anytime by a TLM to allow messages exchanging only among defined network nodes without sharing of messages.

In an MbD based system, increasing the level of distribution enable huge network scenarios to be managed with a lower network traffic because of scalability and elasticity properties that an MbD system may achieve (Schonwalder et al., 2000)(Goldszmidt et al., 2010). In the IoT, 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 through TLM scripts delegation that 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 using different scripts. However, because MbD is an abstract model, it has to be carefully adapted to be properly applied in IoT management. In this case, we review two important technologies that can be used to realize MbD in Section 2.2 and 2.3.

2.2 Script MIB

The Script MIB is an MbD based architecture introduced by the IETF's Distributed Management (DISMAN) group (Levi e Schoenwaelder, 2001). The Script MIB was designed as a Management Information Base (MIB) to be used with a Simple Network Management Protocol (SNMP) agent. Managers from the network can communicate SNMP agents to retrieve informations from management objects defined in these MIBs. These objects may be used to monitor or control different features from the Script MIB, for example, ending a script execution or informing a URL to retrieve one script from an external repository. In addition, 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 launched, executed and managed but not defined by Script MIB. In summary, Script MIB provides a way to remotely manage scripts installation, launch, execution, and termination.

An overview about Script MIB structure, characteristics, and features is provided bellow. This overview will cover only specific details about Script MIB that are important for this dissertation. Therefore, the documentation of Schownwalder *et al.,* (Levi e Schoenwaelder, 2001) can be consulted for any further details about Script MIB.

SNMP management objects of Script MIB were all accommodated into the SNMP group *smObjects*, as can be seen in Figure 2.2. In such a group, there are two sub-groups, *smScriptObjects* and *smRunObjects*. In addition, there are table objects that will carefully describe different features of Script MIB. Each of these sub-groups and table objects are better explained bellow.

- *smLangTable:* The Script MIB defines an object table, henceforth referred to as table, called *smLangTable* that accommodates objects to describe MLM's supported environment programming languages to execute scripts, for example, JAVA, C, CPP, or Python. In such a table, a programming environment is described accordingly to a unique index, name, distributor or vendor, current revision, and a description.

- *smExtsnTable:* When a language entry is added to *smLangTable*, usually one or

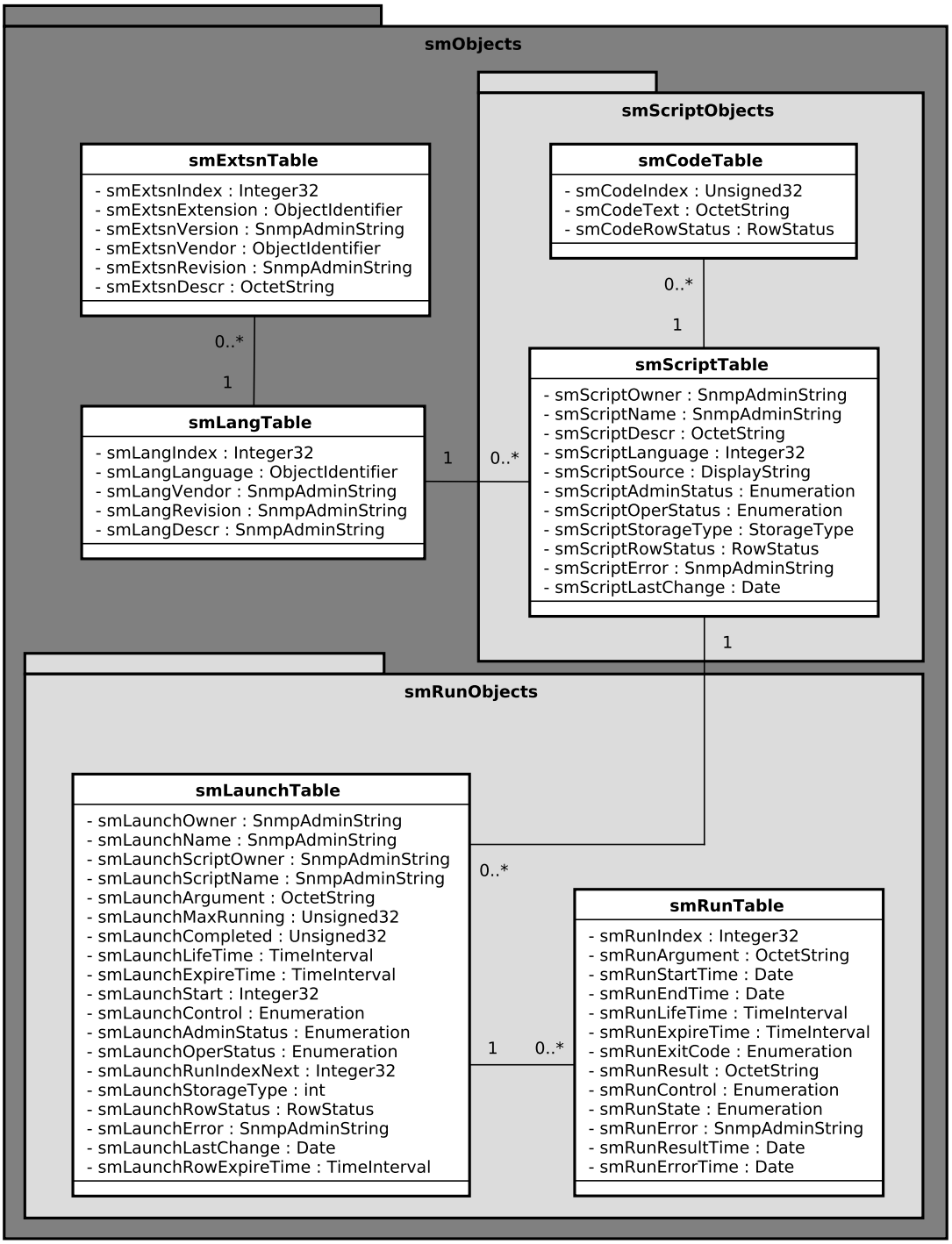

Figure 2.2: Script MIB relational diagram

more extensions are associated whit these languages, such as .jar, .exe, .cpp, or .py. Each extension is presented as an entry of table *smExtsnTable*. A common entry of *smExtsnTable* will be composed by a unique index, name, current version, revision, and a description.

- *smScriptObjects:* The *smScriptObjects* sub-group accommodates all objects needed to install, store, and describe a management script.
  - *smScriptTable:* In the *smScriptObjects* sub-group there is the table *smScript-*

Table, which lists scripts installed on MLMs. These scripts are installed by the creation of an entry on the *smScriptTable* with at least informations about the script owner, name, group, programming language, and a URL to retrieve the script. Each entry contains a column that indicates the status of the script, which, when it is changed, informs the MLM that the script is ready to be pulled from an external repository. In addition, the pulled script can be used volatile, stored directly over the memory RAM, or locally, stored on hard disk.

  - *smCodeTable:* The second table of the *smScriptObjects* named *smCodeTable* stores information about the location and identification of local scripts. In addition, the *smCodeTable* can be used by TLMs to push scripts through SNMP messages, however, the feature of pushing scripts is optionally adopted by a Script MIB implementation (Schonwalder et al., 2000).

- *smRunObjects:* Each entry in the *smScriptTable* represents an installed script on an MLM that can be launched and executed, these both activities can be monitored and controlled by a TLM accordingly with *smRunObjects* sub-group objects.

  - *smLaunchTable:* The *smRunObjects* presents the table *smLaunchTable*, which lists scripts ready to be launched. Each script launch has one entry in *smLaunchTable* and presents at least informations about who owns this launching, a launch name identification, a script name to be launched, and the script owner. Through these informations, a TLM will change the value of *smLaunchStart* column informing an MLM to launch the script. Different scripts launch can be started sequentially or in parallel by the same TLM or from other managers from the network. The capabilities and system permissions of scripts are the same of their owner and group.

  - *smRunTable:* A script launched from the table *smLaunchTable* starts to be executed passing to the state of running, being added as an entry in *smRunTable*. During the script execution, TLMs can monitor scripts being executed acquiring objects from *smRunTable* through SNMP messages. In addition, script executions can be controlled changing the values of the *smRunTable* column objects, terminating, pausing, or continuing a script execution. Finally, TLMs can use *smRunTable* objects to obtain the results of a script execution.

One of the main concerns about the Script MIB is the complexity and network overhead related to the great number of SNMP messages needed to just install and execute a management script (Granville et al., 2009). Fioreze *et al.,* (Fioreze et al., 2005) present a WS based alternative to the Script MIB. In such an alternative, an improved performance in comparison to the Script MIB was achieved by reducing the number of messages exchanged in the network to just one WS message. Therefore, Fioreze *et al.,* proved that a WS approach can perform MbD functionalities better than a traditional Script MIB implementation in place.

Script MIB is one technology that implements MbD concepts in traditional networks, where TLMs are interested in reconfigure MLMs to manage the network substrate and known MDs. In the IoT networks, MLMs have to manage an unknown number of SObjs that stay active over the network for uncertain time. However, the Script MIB does not present awareness solutions to both acquire knowledge about a new SObjs nor to detect a SObj joining or leaving from the network. In addition, one of the main problems of

the Script MIB is that by design it lacks of mechanisms to check script dependencies, correlation, and services availability.

2.3 Open Services Gateway initiative

Another technology that may be used to realize MbD is the OSGi framework (OSGi Alliance, 2012) (Rellermeyer et al., 2008). Through OSGi, Gama *et al.,* (Gama et al., 2012) 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. Different from the Script MIB, OSGi presents a mechanism to check bundle dependencies and enables link between bundles. Both differences are checked and created during the bundle installation. Another difference between Script MIB and OSGi is that the OSGi threats bundles as network services exposing an enhanced view of them, for example, providing Web Services Description Language (WSDL) or eXtensible Markup Language (XML) documents to describe bundles.

The OSGi framework provides all the subsides needed to handle bundles in terms of software. In addition, this framework provides also a uniform interface that enables the access to the hardware substrate. The OSGi architecture may be better explained by the layered architecture presented in Figure 2.3.

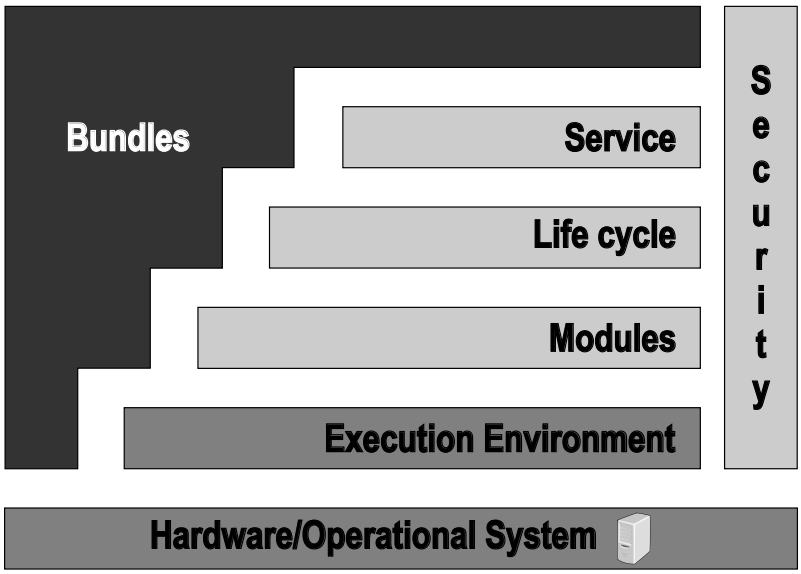

Figure 2.3: OSGi layered architecture

In Figure 2.3, the basic infrastructure necessarily to execute the OSGi Framework is presented by underlie layers composed by *Hardware*, *Operational System* and *Execution Environment* layers. Over this infrastructure, bundles are designed taking advantage of OSGi layers, such as *Security*, *Modules*, *Life Cycle*, *Service Registry*, and *Services* better described as follows.

- *Security* layer: The *Security* layer is an optional layer that underlies all the OSGi framework, it was designed based on the Java 2 security specification. This layer can provide different security features, for example, code authentication by signer

or location. However, the security layer itself does not define an API to control applications. Thus the management of the security layer is left to the life cycle layer.

- *Modules* layer: The modules layer defines a generic and standardized module package to store and publish scripts, called bundle. Bundles must contain all the resources that a script may need to execute, such as help files, icons, text files, and other specific resources that a script may need and is not provided by default. In addition, a bundle must present a manifest document that describes the bundle and its dependencies to be checked by OSGi. Once a bundle is started, its functionalities are exposed to other bundles as a service.

- *Life Cycle* layer: The *Life Cycle* layer is based on the *Modules* and *Security* layers. This layer provides an API that covers the installation, starting, stopping, updating, uninstallation, and monitoring of bundles in a fine-grained secure environment (security is optional) (OSGi Alliance, 2012). In addition, the Life Cycle layer must provide an API that enables the remote management of the OSGi framework.

- *Service* layer: The *Service* layer defines a publish, find and bind model that is highly integrated with the *Life Cycle* layer. This layer enables bundles to publish, find, and bind each other's services without having a priori knowledge of those bundles. In addition, this layer enables a bundle to restrict its service access and also track services across the framework.

Different from Script MIB, OSGi was not developed to implement MbD concepts by design (OSGi Alliance, 2012). However, the MbD principles can be observed through similarities presented between Script MIB and OSGi. For example, Script MIB enables network managers to delegate scripts remotely through SNMP. OSGi, in its turn, may be remotely accessed by network managers through web applications or remote terminal solutions to install new scripts. Both solutions enable the reconfiguration of network nodes without the need of a warm start or system reboots. In addition, Script MIB enables security by adjusting scripts permissions accordingly with user and group authentication, whereas OSGi rely on code authentication through signature and certificate based mechanisms that allow the same association among user and group permissions to a script.

Rellermeyer *et al.,* (Rellermeyer et al., 2008) improves OSGi to be programming language independent, providing different forms to embed the OSGi into SObj. Such improvement allows SObjs to pull scripts from configured remote repositories or URLs delegated by managers over different technologies, such as IEEE 802.15.1 or IEEE 802.15.4. However, even the improved OSGi does not present a decoupled way to install new scripts by default. Thus, OSGi remains coupled to specific applications to install new scripts, such as CLI, Eclipse IDE, or Netbeans IDE.

In the IoT, gateways with OSGi tend to not offer a SObjs awareness mechanisms by default, therefore, it cannot be used solely to detect, manage, and monitor SObjs automatically, without external intervention of human managers. In addition, as the OSGi was not design to be an MbD solution, it does not present all interfaces necessarily to support MbD concepts presented in Section 2.1. Therefore, WS solutions may be explored to address the lack of awareness mechanisms and MbD interfaces of OSGi.

2.4 Web Services

Granville *et al.,* (Granville et al., 2009) present a WS 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) (Box et al., 2000). The important consequence of that work is the observation that a WS solution can provide interfaces necessarily to implement MbD concepts. In addition, 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 (Fioreze et al., 2005). Therefore, we argue that WS can be combined with OSGi as well as the Script MIB to provide MbD functionalities leading to an improved experience to install scripts and manage them.

WS applications may present different architectures, *e.g.,* Service Oriented Architecture (SOA) (Box et al., 2000) and Resource Oriented Architecture (ROA) (Fielding e Taylor, 2002). 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 communication interfaces, *i.e.,* description of services, message formats, and data to be communicated. In addition, a SOA WS requires the use of SOAP (Box et al., 2000) to perform communication among services and clients. SOAP messages are described in XML and must be serialized before transmitted over the network.

ROA (Fielding e Taylor, 2002) 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.

Pautasso *et al.,* (Pautasso et al., 2008) show that ROA presents almost the same features of SOA, however, it is lighter in terms of network overhead. In Granville *et al.,* (Granville et al., 2009), SOA was presented as the only WS architecture to provide an alternative to Script MIB. Therefore, the use of ROA as a WS architecture to provide such alternative remains an open research question.

In summary, as Granville *et al.,* present a WS application to map Script MIB functions, we argue that the same can be done with the OSGi. In this case, a WS approach is designed to provide MbD functionalities using both, Script MIB or OSGi. However, just provide interfaces to support MbD concepts is not enough to manage IoT because MbD solutions solely cannot define which are the scripts to be delegated nor when they must be started or stopped to avoid unnecessary processing by gateways, *i.e.,* MbD is not enough to detect which are the SObjs nor which are on-line or off-line to be managed. Therefore, SObjs awareness mechanisms should be took in concern to enable MbD systems to manage IoT networks.

2.5 Smart Objects Awareness Mechanisms

SObjs awareness describes the capability of gateways to sense SObjs in its covered area. It means that gateways have features to discover new SObjs, detect when they are joining or leaving the network. In addition, gateways regards on awareness mechanisms to acquire important information about SObjs, for example, IP, system name, and machine address. These features are usually presented as standard network protocols, which are examples LLDP (Congdon, 2002) and UPnP (Reynolds, 2006).

LLDP is a protocol designed to run over all IEEE 802 networked devices. This protocol is media independent enabling the contact of LLDP agents with adjacent devices to learn higher layer management reachability and connection endpoint informations. In addition, LLDP runs over the data-link layer exclusively (OSI model), allowing devices with different high level protocol stacks to learn about each other. The main disadvantage of such protocol is that LLDP does not cover SObjs service discovery.

UPnP, in its turn, defines an architecture, formerly known as the Device Control Protocol (DCP) Framework, usually presented as a set of protocols standardized by UPnP forum[1], such as IP, TCP, UDP, HTTP, SOAP, and XML. This architecture tries to provide seamlessly connectivity to ad-hoc SObjs being media independent. It means that UPnP can be deployed over any operational system and computational language. Gama *et al.,* (Gama et al., 2012) states that UPnP is the *, the facto,* protocol for plug-and-play SObjs. In addition, UPnP can be used to add discovery, description, control, eventing, and presentation capabilities to devices. However, UPnP uses high level layers (OSI model) to be deployed, these layers cannot be fully supported by some constrained SObjs (*e.g.,* RFID tags).

SObjs awareness mechanism can be used to detect devices that, when combined with MbD solutions, Script MIB and OSGi, enables gateways to detect new SObjs and to receive scripts to manage them. However, awareness mechanisms and MbD solutions are not combined by default to manage IoT. It means that even when a gateway has installed an MbD solution and an awareness mechanism, by default, they are not related and have to be combined by some other solution. Therefore, solutions that provide such combination of features should be explored to achieve a better management of the IoT.

2.6 Middleware

Most of current solutions to manage the IoT are designed as a middleware (Teixeira et al., 2011), *i.e.,* an intermediary software layer that offers an enhanced view of the lower layers usually composed by operational systems applications, hardware drivers, or network services. This enhanced view is usually provided by WS to improve interoperability among other available middlewares and network clients. In addition, middlewares adopts SOA in order to support the IoT network topology that is both unknown and dynamic (Teixeira et al., 2011). These middlewares usually are developed over OSGi and may use UPnP technologies to provide a discovery mechanism to handle IoT dynamicity.

Middlewares designed to manage the IoT are based in different strategies, which main examples are devices as services and data/information extraction. On the first strategy, middlewares create a WS for each SObj discovered. This WS will have its consumption translated to SObjs calls. Such calls can be responded directly by SObjs or indirectly by gateways that has SObjs cached information to provide. There are current projects

[1] UPnP Forum - www.upnp.org

that use the stategy of devices as a service, such as LinkSmart formerly known as to HYDRA[2] (Eisenhauer et al., 2009)(Eisenhauer et al., 2010), SENSEI (Presser et al., 2009), SOCRADES (Cannata et al., 2008), and COBIS (Decker et al., 2007).

The information extraction strategy is based on data analysis and ontology detection. This strategy tend to provide an macro overview of the IoT, where middlewares are more interested in process the information provided by SObjs rather than how to obtain it. It means that these middlewares are bound to semantic web, extracting meaningful information through ontology approaches, and use database models to provide a better summary of informations in an easier way. Some projects use this middleware strategy, such as Smart-M3 (Honkola et al., 2010), SATware (Massaguer et al., 2009), and Global Sensor Networks GSN (Aberer et al., 2007).

The main disadvantages of current middlewares is that they tend to apply one strategy rather than other, furthermore, middlewares are actually creating an heterogeneity among themselves. It means that there is no agreement among middlewares about the strategy adopted, letting managers to decide which is the best strategy to their network domains. MbD based solutions can be used to simplify the use of both strategies because one strategy can be easily replaced by delegating a script that supports one strategy rather than other.

Current systems based on middlewares tend not to support one role entity solely anymore *e.g.,* LinkSmart, SENSEI, and Smart-M3. It means that there is no clear division between TLM and MLM roles and there are only dual-role entities. In a macro overview of the IoT, to manage it plentifully, a fully distributed system composed solely by dual-role entities fits better than a less distributed solution because minimize network traffic and achieve high level of scalability. However, in a micro universe of the IoT, a fully distributed systems may actually lead some scenarios to face new problems. For example, an hospital network that handles private patient informations installs a fully distributed management system, where every node gather information and share among each other. Afterwards, one of these nodes may share private informations by receiving external share requests from other systems that has the same distributed solution, presenting an information leakage. Such a leakage may be avoided with a small distribution level, where MLMs share information only with their specified TLMs. In addition, as stated before at Section 2.1, a distributed level should be correctly adapted accordingly with a network domain management task (Schonwalder et al., 2000). Therefore, an architecture that supports different levels of distribution, such as an MbD based architecture, should be more suitable to the IoT rather than a full distributed solution, as the current middlewares present.

Observing the need of an solution based on MbD concepts, WS, and SObjs awareness mechanisms, in this dissertation, we described a solution called Management by Delegation Smart Object Aware System (MbDSAS) (Marotta et al., 2013). MbDSAS was design based on MbD concepts, using either the Script MIB and OSGi, in combination with a WS architecture, SOA and ROA, and an awareness mechanisms, UPnP and LLDP, to manage IoT SObjs. Therefore, MbDSAS was carefully described in the next sections.

[2]HYDRA homepage - http://www.hydramiddleware.eu/news.php

3 MBDSAS ARCHITECTURE

In this chapter, we introduce MbDSAS conceptual architecture and its internal details. From top to bottom, TLM, MLM, and MD are depicted in Figure 3.1. In addition, a *Script repository* that hosts management scripts can be seen in the right side of Figure 3.1. 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.

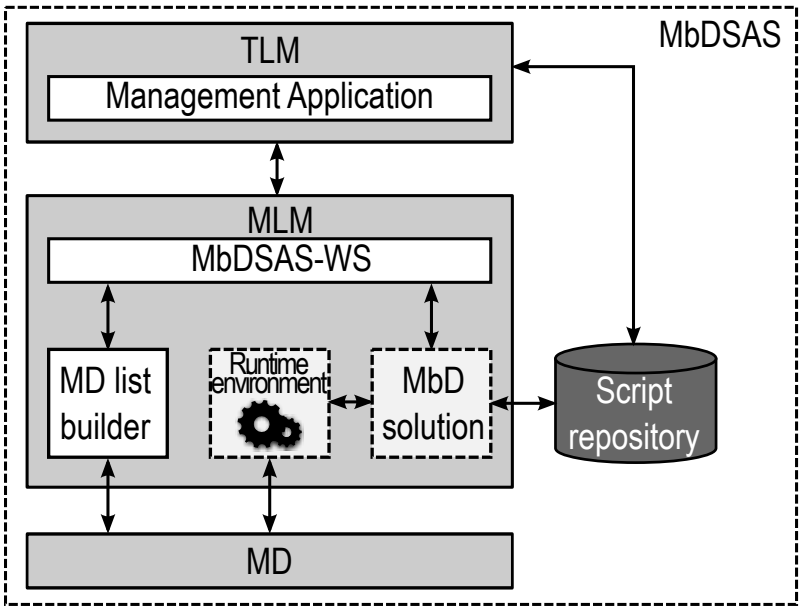

Figure 3.1: MbDSAS conceptual architecture

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. In addition, accordingly with the distribution level adopted by an MbDSAS system, TLM and MLM may be deployed at the same network node, which will be classified as a dual-role entity. TLMs run 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 new available 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 detection of SObjs joining and leaving from the network. The detection is carried

out by collecting information from MLMs about their detected SObjs through a list created by the *MD list builder* module, called Managed Device List (MDL). Afterwards, a comparison among recent collected MDLs with previous versions enables TLM to identify joining ans leaving of SObjs from the network.

- Sending management scripts to MLMs: The *Management Application* may send management scripts to MLMs to manage MDs. It means, that a station delegates the management task to a network entity closer to a SObj. Management scripts can also be more than just network management routines, they can be used as network services being exposed through WS interfaces to be accessed by users that are interested in their consume as it is outlined in Gama *et al.,* (Gama et al., 2012) by the use of OSGi.

- Requesting or receiving notifications about new management scripts available from other TLMs and Script Repositories: The *Management Application* may request or be notified about new management scripts available from other TLMs or *Script Repositories*. When a new notification is detected, the *Management Application* will add this notification to TLM's database organizing accordingly with specific SObjs characteristics and the management script URL. Afterwards, 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. These sub-domain TLMs are settled during the creation of a TLM, later added by manual setup, or added automatically by other TLMs notification.

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 among TLMs and MLMs over both contexts, whereas the network overhead remains low. Therefore, taking advantage of Granville *et al.,* research (Granville et al., 2009), MbDSAS was designed based on WS to address both requirements stated above through the MbDSAS-WS module.

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 components 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 the management of the life cycle of a script.

A subset of MbDSAS-WS services is dedicated to manage an component, which is called *MD list builder*. Such component was created to add SObjs awareness to MLM. It means that *MD list builder* monitors network interfaces tracking SObjs that surrounds MLMs by extracting relevant information of received messages (*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 LLDP or UPnP. This monitoring allows the creation of 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. TLM 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, an 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 life cycles of 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 receive and store new management scripts. These scripts may be developed 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*.

The *Script Repository* is defined through the network domain needs. It means that each specific network domain may change definitions of the *Script Repositories*, how they will behave, and how they will gather and spread new scripts. For example, a local computer may become the *Script Repository* of a private network domain such as an hospital network, an police station network, or a small enterprise network. This repository will be locally established to simplify the administration of privacy and the security maintenance by being locally controlled through publishing only private and carefully defined scripts without external influences.

The needs of huge network domains such as global enterprise or federative networks may change the *Script Repositories* definitions to cloud computers and clusters. Such needs are related to a large number of TLMs requesting scripts updates that shall be handled without failing. Therefore, cloud computer and clusters may address these requests rather than a single computer that may not provide computational resources enough to support all these requests. In addition, these cloud computers and clusters may take advantage of recommender systems to establish which are the best scripts to be provided, *i.e.,* users given opinion about a useful script are used to determine the most popular scripts that are preferred to be delegated rather than unpopular ones. Afterwards, domain name servers (DNS) may be used to simplify the reachability of these *Script Repositories*.

MbDSAS architecture was designed to be human independent and autonomic. It means that MbDSAS can be deployed in systems intended to manage SObjs and that can also manage themselves. Such systems become autonomic by sensing their network neighbors, acquiring new updated scripts notifications automatically from repositories,

spreading these updates among known TLMs, and feeding themselves and MLMs with management scripts. These scripts can be designed to manage SObjs, but also, gateways, computers, and whatever device connected to the network that is able to be communicated. A better understandability of MbDSAS features and capabilities can be achieved by analyzing each possible communication among MbDSAS entities as outlined in the next section.

3.1 MbDSAS use case

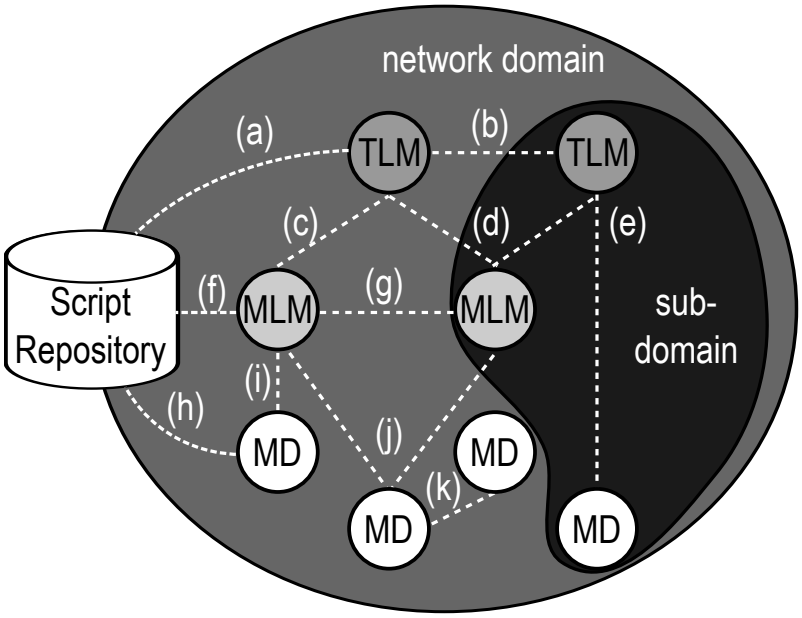

Figure 3.2: MbD entities in a communication graph

A communication graph may be used to cover possible interactions among MbDSAS entities, as depicted in Figure 3.2. Each interaction of different entities is a communication being presented by a dashed line highlighted with a letter from (*a*) to (*k*) in Figure 3.2. These communications may be involved with management task controlling, execution, monitoring, or delegation. In addition, each interaction may differ accordingly with the distribution level of MbDSAS. One important observation is that each communication is described accordingly to MbDSAS concepts, which does not interfere in other possible interactions, being better explained bellow:

(*a*) Communication between TLM and Script Repository: This communication enables TLM to request or be notified about updates stored in *Script Repositories*. These repositories will behave accordingly with the network domain needs, as explained before. Therefore, notifications or requests about script updates and creation are handled by implementation.

(*b*) Communication among TLMs: This communication may be presented as a notification exchanged among TLMs and sub-domain TLMs, reporting the creation or updating of a management script. Such a communication become usual when the distribution level of a system raises.

(*c*) Communication between TLM and MLM: A TLM will communicate an MLM to delegate, monitor, or control a management task, pushing or acquiring informations from MLM. In addition, TLMs may request MLMs to obtain or update MDLs. For example, a TLM will obtain an MDL from an MLM to be checked. Afterwards, TLM will send management scripts to manage all MDs found into that MDL. When a TLM have no script to manage an MD, thus, the MLM receives no management script. In addition, a TLM can verify if an MD left the network, afterwards, this TLM will cease the management of that MD by sending a message to the MLM to stop its management script execution.

(*d*) Communication among two or more TLMs and an MLM: This communication will happens when different TLMs will contact the same MLM to delegate, monitor, or control management tasks as usual. However, diverse TLMs imply in different permissions and capabilities to control or monitor management tasks. Therefore, MLMs will receive management scripts from different TLMs and will execute them accordingly with each other capabilities and permissions. In addition, the system distribution level raises as well as the frequency of this communication.

(*e*) Communication between TLM and MD: This kind of communication happens only when a TLM is also an MLM. In another words, a network node will behave as a TLM whenever notified about a new updated management script and also will behave as MLM when a management script was delegated to be executed. This communication number raises as well as the system distribution level.

(*f*) Communication between MLM and Script Repository: The *Script Repository* may be communicated at anytime by an MLM when it is seeking for a script source. This script source will be usually downloaded from a *Script Repository* accordingly with an architecture previously established such as ROA by downloading directly from a URL as a resource or SOA by consuming a service that will provide the script source. This particular communication is not affected by the distribution level of a system, therefore, it will just raise with the number of new MLMs.

(*g*) Communication between MLM and MLM: An MLM will always manage an MD, even if an MD is an MLM. For example, an MLM receives from a TLM, a management script to manage or exchange information with other MLM and no matter if the managed MLM is from different network domain, it will be managed by the first MLM as a usual MD. In addition, interactions among MLMs will raise accordingly with the system distribution level.

(*h*) Communication between MD and Script Repository: This communication is not supported by MbDSAS by default. It means that any communication among MD and a *Script Repository* may happen by different circumstances that do not involve MbDSAS architecture directly. For example, an MD can contact a *Script Repository* because a management script configured it to do so, however, this kind of configuration may not necessarily happens, therefore, it is not directly supported by common functionalities of an MbDSAS architecture.

(*i*) Communication between MLM and MD: An MLM will communicate an MD whenever a management script executes such communication. This communication is not affected by the distribution level of the system. In addition, communication

among MLMs and MDs may be related to the discovery mechanism adopted or protocol (*e.g.,* UPnP or LLDP), being supported by MbDSAS. However, this communication is not supported by MbD concepts by default.

(*j*) Communication among two or more MLMs and an MD: An MD may be managed by different MLMs, even if they are from different network domains without MbDSAS concepts violation. This communication is not affected by the distribution level of a system.

(*k*) Communication between two or more MDs: The communication among MDs may be used by management scripts to communicate with other MDs belonging to multihop sensor networks or grids. However, this communication depends on the management script implementation. Therefore, MbDSAS concepts do not cover this communication explicitly.

Combining communications enable MbDSAS to manage many different SObjs and reconfigure network nodes. Therefore, analyzing the described communications enable to define how MbDSAS should behave theoretically as described in the next section.

3.2 MbDSAS operations

MbDSAS's operation was plotted as a sequence diagram presented in Figure 3.3. In this diagram, an MD becomes online by joining a local network (1 in Figure 3.3). When this MD joins the network, it sends an acknowledgment multicast message (2 in Figure 3.3) that contains informations about itself accordingly with the discovery mechanism adopted. For example, the discovery protocol LLDP will offer MD informations which includes an IP address, operational system version, and a system name. In addition, MLMs can reply this message offering the same informations about themselves to the contacted MD. These messages exchanges can also be sent by an MLM through periodic acknowledgment multicast messages that are collected by an MD.

In Figure 3.3, an MLM detected an MD collecting meaningful informations about MDs. These informations must be stored to be collected latter by interested TLMs, therefore, an MDL is created and fulfilled with MD informations by the *MD list builder* module (3 in Figure 3.3). In addition, MDL entries are added with a timestamp that allows the detection of MDs departure from the network.

TLMs *Management Application* may contact an MLM to acquire their MDL (4 in Figure 3.3) through MbDSAS-WS. This contact is carried out by WS technologies, such as SOA and ROA. It means that an MLM may be contacted at any time disregarding the previous contacts and also can be contacted in parallel, *i.e.,* asynchronously. Therefore, MbDSAS-WS has to constrain its service to be executed and finished in a single contact. Such constrain imply that MbDSAS-WS secure mechanisms has to authenticate TLMs at each communication established providing at least a valid ID, a system group, and a valid password. However, TLM may apply polling mechanism that will establish many communications in varied times that may imply in network overhead, when authenticating each possible communication. Thus, different authentication mechanism may be explored to minimize such an overhead, for example, the creation of session keys or thrust mechanisms based on computer addresses.

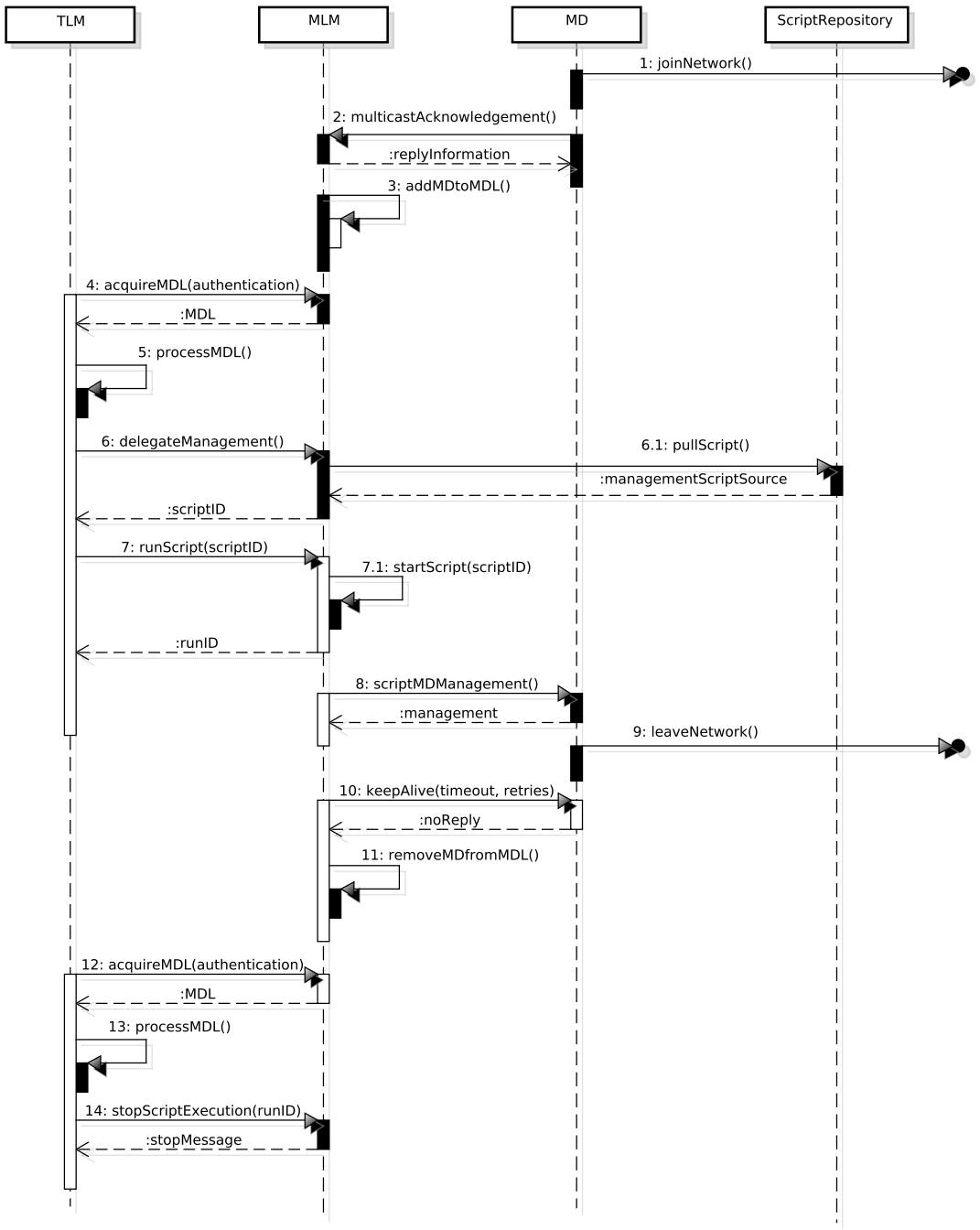

Figure 3.3: MbDSAS sequence diagram

A TLM authenticated can obtain an MLM MDL to be processed (5 in Figure 3.3). This process enables TLM to know which MDs are next to an MLM. It means that an TLM may choose which is the management task to delegate to an MLM. Such task can be very specific to a single MD or for many generic MDs accordingly with available management scripts, managers configurations, and needs. Defined the script, a communication is established with MLM to delegate it (6 in Figure 3.3).

During the script delegation, TLM provides to MLM informations about the management script delegated, at least a URL to download the script and ownership informations, such as owner and group names. The MLM will contact a *Script Repository* specified by the URL informed to pull the delegated script (6.1 in Figure 3.3). Afterwards, the downloaded script will receive a *scriptID* that will be sent back to TLM as a reply indicating the success of the script delegation.

Each script delegated to an MLM has a unique identification (*scriptID*) that enables TLMs to manage the script life cycle. A TLM can obtain one identification during the script delegation as explained before or by requesting the script identifications installed on an MLM. Afterwards, a TLM can use these identifications to run a script at the MLM (7 in Figure 3.3). It means that the script is launched in an MLM using script arguments provided by the TLM and a block parameter. Such a block parameter enables TLM to wait the script execution ending and to receive a result reply. Otherwise, the MLM launch the script in background letting TLMs to manage it latter through a launch identification *runID*. This identification is returned to TLM as a success message of the script launch or being requested by TLMs (7.1 in Figure 3.3).

During a script running, different computer routines are executed enabling MLM to manage surrounding MDs (8 in Figure 3.3). The objective of an MD management can be so specific such as an information request about a luminosity sensor in an MD or generic such as a keep alive message. This objective will be carried out by designed scripts and TLM needs. It means that a TLM has to define which are the scripts to be delegated to manage MLM's surrounding MDs. Such definition may be involved with other parameters, such as script availability, description, or objective.

An MD can leave the network at anytime (9 in Figure 3.3). When an MD leaves the network, MLM discovery mechanism detects this leaving usually through a discovery process (10 in Figure 3.3). For example, there is a discovery process that uses keep alive messages to detect an MD status. This process is usually based on a number of retries and a timeout. It means that MLM sends a message to an MD and waits for a reply till it reaches a timeout. Afterwards, the process is repeated till it reaches a number of retries. Finally by not receiving any reply from the MD, the MLM discovery mechanism states that the MD left the network.

As soon as the MLM discovery mechanism detects the departure of an MD, the *MD list builder* will compare the current timestamps with the last contact stored from its MDL. This comparison is particularly important for the IoT, because many SObjs use sleeping strategies to green its battery life by becoming unable to reply discovery messages. Therefore a different mechanism to detect this departure associated with the *MD list builder* took place to set a larger timeout to analyze a SObj departure. However, if the SObj does not reply messages, surpassing even the timeout from the *MD list builder*, thus, the MD will be removed from the MDL characterizing its departure (11 in Figure 3.3).

A TLM will repeat its management routine by acquiring MDLs from MLMs (12 in Figure 3.3). However, a TLM will notice that the MLM detected an MD departure by processing the received MDL (13 in Figure 3.3). Therefore, a message is sent to MLM

to cease the script execution that manages the departed MD. Finally, MLM replies with a message composed by the stop message generated from the ceasing of the script execution.

The sequence diagram presented in Figure 3.3 presents a usual routine of MbDSAS. However, no technology was defined and presented, making the sequence diagram to expose a theoretical use of MbDSAS. Therefore, we designed a prototype based on MbDSAS as a proof of concepts and also to test MbDSAS in a real scenario, being better described in the next chapters.

4 PROTOTYPE

The MbDSAS architecture of Chapter 3 was prototyped and described bellow. We mainly focused on the design of three components: *Management Application*, *MbDSAS-WS*, and *MD list builder*.

4.1 Management Application

The MbDSAS prototype was designed to have a TLM installed with a *Management Application*. This *Management Application* was developed in JAVA, aiming to provide a human-friendly interface to managers in order to send management scripts remotely to MLMs manually. In addition, in such an interface, human managers can manually set new *Script Repositories* by adding a URL to the repository list contained in the *Management Application*. This list was designed as a JSON document, which have entries that define a specific supported MD with a script URL for downloading.

The *Management Application* is one important module to a MbDSAS system reach autonomy. In such an autonomy, *Management Application* sends and notifies MLMs and sub-domain TLMs automatically about new and updated management scripts. In addition, autonomy includes self management, therefore, the *Management Application* must include the local host as an MD when communicating a script Repository to obtain new scripts to manage itself. However, TLMs require that the *Management Applications* communicate *MbDSAS-WS* placed on MLMs to delegate scripts and retrieve MDLs. Therefore, all necessary communication is based on the specification of *MbDSAS-WS* better described in the next section.

4.2 MbDSAS-WS

MbDSAS-WS component was developed in PHP to expose services based on WS architectures, *i.e.*, SOA (Box et al., 2000) and ROA (Fielding e Taylor, 2002). *MbDSAS-WS* services were developed taking advantage of Granville *et al.,* research (Granville et al., 2009). However, different from that research, we explored ROA architecture 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 (Granville et al., 2009), and (*ii*) *MDL services* to acquire and edit MDLs from MLMs. *MbDSAS-WS* services are summarized in Table 4.1.

According to Table 4.1, services will represent a different operation specified by the MbDSAS architecture. In this table, column *Service* will present all covered services of

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 4.1: MbDSAS-WS services

MbDSAS-WS. In addition, each information needed to be provided to consume a service is presented at column *Parameter*. Afterwards, all the expected results from a successful consume of a service from the first column are depicted at column *Return*. Finally, all covered services, parameters and results from Table 4.1 are better described in the following subsections.

4.2.1 BeginMbD service

A *Management Application* will be interested in the access of different services from MbDSAS-WS placed at an MLM. However, this access will require that the TLM validates itself through the *BeginMbD* service. It means that *BeginMbD* enables TLMs to use other services from *MbDSAS-WS*, creating a delegation session by validating parameters, which includes at least a *username* and a *password*. Created the delegation session, a TLM become authorized to use other services by receiving a valid *sessionID*. Without a valid *sessionID* a TLM cannot use any of the other available services at an MbDSAS-WS placed on an MLM.

```
1  {
2  "passwd":"secret",
3  "owner":"mbdsas-manager",
4  "community":"adm",
5  "solution" : "OSGi"
6  }
```

```
1  {
2  "status":"accepted",
3  "sessionID":"2f80de1c3e812477196c0859f531c4f295ecb8a5"
4  }
```

Listing 4.1: BeginMbD JSON message exchange

A usual *BeginMbD* consumption is performed through messages exchanges among TLMs and MLMs as exemplified in the JSON code presented in Listing 4.1. The first message is a common consumption of *BeginMbD* by a TLM that is composed of different parameters, such as a user name (*owner*), a password (*passwd*), a group name (*group*), and

the *MbD solution* to be used (*solution*). Afterwards, an MLM will validate each parameter and, in case of acceptance, the MLM will reply the first message with a operation status as accepted (*status*) and a valid unique session identification (*sessionID*). Otherwise, a simple message will be replied indicating the failure attempt with a description containing an error message. Finally, a TLM authenticated during *BeginMbD* consumption will be henceforth referred to as validated TLM.

4.2.2 EndMbD service

A validated TLM may use different services at an MLM such as described in next subsections. However, a TLM may want to end a started session to start a new one with another user, capabilities, *MbD Solution*, or permissions. Therefore, a *EndMbD* service is designed to perform a session close in an MLM. This service is usually performed through a message exchange between a TLM and an MLM. A typical example of messages related to the *EndMbD* service can be seen at Listing 4.2.

```
1    {
2    "sessionID":"2f80de1c3e812477196c0859f531c4f295ecb8a5"
3    }
```

```
1    {
2    "status":"destroyed",
3    "sessionID":"2f80de1c3e812477196c0859f531c4f295ecb8a5"
4    }
```

Listing 4.2: EndMbD JSON message exchange

In Listing 4.2, a TLM sends a message to a URL that points to the *EndMbD* service on an MLM. This message contains solely a valid session identification (*sessionID*) to be destroyed. Afterwards, the contacted MLM process this message and destroys the session defined by the given session identification. Finally, when the session is successfully destroyed, the contacted MLM reply with a message containing the status of the operation (*status*) and the session identification (*sessionID*) of the destroyed session. Otherwise, a message containing a status of "none" is replied indicating that the session identification provided is wrong. In addition, a status containing "error" may be returned when related to a TLM that is waiting for a service to be executed, therefore, the session is locked and cannot be destroyed.

4.2.3 ScriptPull service

A TLM is interested in sending and installing scripts on MLMs. Therefore, a validated TLM can consume *ScriptPull* services on MLMs to make then pull a script from an external repository. This pulling of scripts is performed directly by the chosen *MbD Solution* defined during the consumption of *BeginMbD* service. Therefore, MbDSAS-WS calls the chosen *MbD Solution* informing different parameters to perform the script pulling. These parameters are provided by TLM consumption of *ScriptPull* service and can be better observed in the JSON message contained in Listing 4.3.

Through two simple messages between a TLM and an MLM as showed in Listing 4.3, different management tasks can be delegated using pulling of scripts from external repositories. The MLM will receive the first message of Listing 4.3 with a valid session identification (*sessionID*) and a URL to some script stored in an external repository (*url*). In addition, the message contains information about how the script will be stored in an

MLM (*store*), for example, the script can be stored volatile in the MLM memory RAM or non-volatile in the MLM hard disk. The message also have a parameter defining the specific script language (*lang*), such as JAVA or C. In return, MLM sends a valid script identification to trigger the execution of the script (*scriptID*) and a status description of the pulled script. Finally, any possible errors occurred during the script pulling are replied through an error field (*errors*).

```
{
"sessionID":"2f80de1c3e812477196c0859f531c4f295ecb8a5",
"url":"http://Script_Repository_IP/bundles/wait.jar"
"store":"1",
"language":"1",
}
```

```
{
"status":"ready",
"scriptID":"adm.mbdsas-manager.wait.jar",
"errors":"none"
}
```

Listing 4.3: ScriptPull JSON message exchange

A TLM can also delete a script by consuming the service *ScriptPull*, in the case of ROA, through HTTP DELETE method. This consumption will require that TLM informs a valid session identification (*sessionID*) and a script identification (*scriptID*) to be removed. The MLM will reply this consumption with a message composed by a removed script identification (*scriptID*) and a error status about the operation execution. In addition, *ScriptPull* service will stop every execution instance of the removed script, making a list containing each terminated execution launch identification.

4.2.4 ScriptRun service

```
{
"sessionID":"2f80de1c3e812477196c0859f531c4f295ecb8a5",
"scriptID":"adm.mbdsas-manager.wait.jar",
"args":"sleep=5",
"block":"0"
}
```

```
{
"scriptRunID":"adm.mbdsas-manager.wait.jar.01",
"status":"running",
"result":"",
"errors":"none"
}
```

Listing 4.4: ScriptRun JSON message exchange

MLM can hold different scripts that are supposed to be executed. However, the execution of a script must be launched by a TLM. Therefore, TLMs can use the *ScriptRun* service to launch scripts executions. In addition, different TLMs can be interested in the launching of the same script, thus, two different executions of the same script can be performed. Although, each execution will have the same capabilities and permissions of the TLM that launched it. Such launching is performed with different parameters provided

by a TLM, as outlined in Listing 4.4.

A validated TLM has to inform a script identification (*scriptID*) to the *ScriptRun* service to launch and run a script, informing some arguments to its execution (*args*). For example, a script that executes polling to some device should receive a valid timeout as one argument to be executed. In addition, the parameter *block* makes MLM to change its behavior. When the *block* parameter is equal to 0, MLM sends to TLM a message with a valid script execution identification (*runID*) to be managed and the current script status of execution. Otherwise, in the case of *block* equals to 1, MLM holds the communication with the TLM and then waits the script execution to finish to retrieve a result (*result*) and, after, sends it back to TLM. Finally, errors that occurred during the script execution are also replied (*errors*).

The *ScriptRun* service can also be used to edit or terminate a script execution. It means that a TLM can send a message informing a parameters to edit a script execution, for example, informing a script running identification with additional informations to pause a script execution or change its identification number. In addition, a script running can be terminated by a TLM by sending a message to *ScriptRun* service, in the case of ROA, through HTTP DELETE, informing a session and a script execution identification (*sessionID* and *runID*).

4.2.5 GetResults service

During a script execution, partial results can be generated and collected by a TLM through the *GetResults* service. This service will be used to obtain results from scripts executions that were launched in background (*ScriptRun* with *block* parameter equals to zero). In addition, when one script execution identification points to a terminated script execution, the TLM retrives the last result generated during this script execution. Therefore, TLM must exchange messages with MLM as outlined below in Listing 4.5.

```
1  {
2  "sessionID":"2f80de1c3e812477196c0859f531c4f295ecb8a5",
3  "scriptRunID":"adm.mbdsas-manager.wait.jar.01",
4  }
```

```
1  {
2  "scriptRunID":"adm.mbdsas-manager.wait.jar.01",
3  "status":"running",
4  "result":"sleeping"
5  }
```

Listing 4.5: GetResults JSON message exchange

The *GetResult* service must receive from a validated TLM a session identification (*sessionID*). In addition, TLM must inform a script launch identification to MLM, which performs a check on the script being executed in an *MbD Solution* to gather partial results. When these partial results are gathered by MbDSAS-WS, they are send back to the TLM with a script launch identification (*scriptRunID*), a current status of the script execution (*status*), and partial results (*result*), as shown by the second message in Listing 4.5.

4.2.6 ScriptPullAndRun service

The consume of *scriptPull*, *scriptRun*, and *getResults* can be summarized by the consume of *ScriptPullAndRun*. This service was particular important to Granville *et al.,*

research (Granville et al., 2009) because improved the delegation process of TLMs to MLMs to a single message over the network, minimizing drastically the network traffic consume. Therefore, MbDSAS-WS was also designed to support such a service through two simple message exchanges between TLM and MLM, such as outlined in Listing 4.6.

```
{
"sessionID":"2f80de1c3e812477196c0859f531c4f295ecb8a5",
"url":"http://Script_Repository_IP/bundles/wait.jar",
"store":"1",
"language":"1",
"args":"sleep=5",
"block":"1"
}
```

```
{
"scriptID":"adm.mbdsas-manager.wait.jar",
"scriptRunID":"adm.mbdsas-manager.wait.jar.02",
"result":"Done",
"errors":"none"
}
```

Listing 4.6: ScriptPullAndRun JSON message exchange

The *ScriptPullAndRun* service will receive informations from the TLM that summarizes the parameters of *ScriptPull* and *ScriptRun* services. These parameters are a session identification from a validated TLM (*sessionID*), a URL to download the delegated script from a *Script Repository*, a information defining if a script will be volatile or non-volatile stored (*store*), and the programming language of the script (*language*). In addition, arguments must be provided to *ScriptPullAndRun* service to launch the delegated script (*args*) and if the TLM will wait for results from the script execution (*block*).

An MLM can reply a TLM consumption of *ScriptPullAndRun* with results that summarizes the results from *ScriptPull* and *ScriptRUN* services. These results are outlined in Listing 4.6 by the second message, being presented by a script identification created during the script pulling and installation (*scriptID*), a launch identification created during the start of the script execution (*scriptRunID*), and a information about the results generated during the execution (*result*). In addition, an error information are returned as well, to inform about any possible error during the service *ScriptPullAndRun* execution.

4.2.7 ScriptReplace service

Scripts can become soon outdated with the IoT dynamicity. Therefore, these outdated scripts shall be removed and replaced with new updated scripts to manage new SObjs. TLM may use the *ScriptReplace* service to replace these outdated scripts with new ones through only one message exchange with MLMs as outlined bellow.

```
{
"sessionID":"2f80de1c3e812477196c0859f531c4f295ecb8a5",
"scriptID":"adm.mbdsas-manager.wait.jar",
"url":"http://Script_Repository_IP/bundles/new_wait.jar",
"store":"1",
"language":"1",
"args":"sleep=5",
"block":"0",
"start":"1"
}
```

```
{
"scriptID":"adm.mbdsas-manager.new_wait.jar",
"scriptRunID":"adm.mbdsas-manager.new_wait.01.jar",
"scriptsRunTerminated":"adm.mbdsas-manager.wait.jar.01, adm2.
   mbdsas-manager.wait.jar.02",
"result":"replaced, running",
"errors":"none"
}
```

Listing 4.7: ScriptReplace JSON message exchange

The first message contained in Listing 4.7 is composed almost by the same parameters as the *ScriptPullAndRun* service except by the addition of a script identification (*scriptID*) and a start flag (*start*). The *ScriptReplace* service uses the *scriptID* provided to replace the script pointed by this identification for a new script pulled from the URL provided (*url*). In addition, the start flag defines if the new script pulled will be also launched. All the other parameters will be used to execute the same functionalities of *ScriptPullAndRun*.

An MLM may also reply the *ScriptReplace* consumption with a message composed by a script identification of the new installed script installed(*scriptID*) and a execution identification (*scriptRunID*) in the case of the start flag from the first message be equals to one. In addition, ScriptReplace will stop every execution instance of the removed script, making a list containing each terminated execution launch identification (scriptsRunTerminated). A result field (*result*) will inform any gathered partial results from the script execution and the replacement status of the outdated script. Finally, an error field will indicate any occurred error during all the *scriptReplace* consumption.

4.2.8 GetMDL service

MDL services are presented in Table 4.1 as well. These services are exposed to simplify the retrieving of MDLs from the *MD list builder* component. Such retrieval is accomplished through messages exchanges between TLM and MLM as outlined in Listing 4.8.

```
{
"sessionID":"2f80de1c3e812477196c0859f531c4f295ecb8a5"
}
```

```
{
"49300.2.1":{
        "lldpRemChassisIdSubtype":"4",
        "lldpRemChassisId":null,
        "lldpRemPortIdSubtype":"3",
        "lldpRemPortId":null,
        "lldpRemPortDesc":"vnet0",
        "lldpRemSysName":"SObj01.vnet1",
        "lldpRemSysDesc":"Arch Linux Linux 3.8.10-1-ARCH x86_64
            ",
        "lldpRemSysCapSupported":"8",
        "lldpRemSysCapEnabled":"8",
        "lldpRemManAddrEntry":"200.123.568.98"
        }
...
"49300.2.5":{
            ...
        }
}
```

Listing 4.8: GetMDL JSON message exchange

The TLM starts a session accessing the *beginMbD* service at an MLM and then receiving a valid *sessionID*. Afterwards, this TLM may retrieve an MDL from the MLM through the *GetMDL* service informing a valid *sessionID* as outlined in the first part of Listing 4.8. In addition, TLMs may inform a valid set of SObjs identifications number, retrieving information only about the chosen SObjs from this list rather than a complete MDL.

An MLM may reply the consume of *GetMDL* with a message containing an MDL as outlined in the second part of Listing 4.8. This MDL is composed by different entries with a unique identification (*MDEntry* presented as a number *49300.2.1*) that represents the MLM detected SObjs. For each SObj entry, there are fields that where selected accordingly to the LLDP MIB defined in Congdon *et al.,* (Congdon, 2002). These fields enable TLM to obtain important informations about a SObj and how to contact it. For example, an IP address (*lldpRemManAddrEntry*), a system name (*lldpRemSysName*), and the network interface that was communicated (*lldpRemPortDesc*).

4.2.9 UpdateMDL service

The *MD List Builder* may present an MDL with an entry that have errors or must be edited. Such an MDL may be edited by consuming the *UpdateMDL* service. This service is consumed by TLMs exchanging messages with MLMs as outlined in Listing 4.9.

```
{
"sessionID":"2f80de1c3e812477196c0859f531c4f295ecb8a5",
"49300.2.1":{
    "lldpRemPortDesc":"eth0",
    "lldpRemManAddrEntry":"192.168.1.12"
    }
}
```

```
{
"49300.2.1":{
        "lldpRemChassisIdSubtype":"4",
        "lldpRemChassisId":null,
        "lldpRemPortIdSubtype":"3",
        "lldpRemPortId":null,
        "lldpRemPortDesc":"eth0",
        "lldpRemSysName":"SObj01.vnet1",
        "lldpRemSysDesc":"Arch Linux Linux 3.8.10-1-ARCH x86_64
            ",
        "lldpRemSysCapSupported":"8",
        "lldpRemSysCapEnabled":"8",
        "lldpRemManAddrEntry":"192.168.1.12"
        },
"errors":"none"
}
```

Listing 4.9: UpdateMDL JSON message exchange

A TLM can consume the *updateMDL* by specifying a valid *sessionID*, one MD identification (*MDEntry*), and the parameters to be changed (*params*). A new MDL is generated and returned to TLM containing just the MDL entry that was edited and an error field containing possible error messages (*errors*).

4.2.10 Script Delegation

An example of a script delegation algorithm may be summarized in two simple requests, as outlined in Algorithm 1.

1: $sessionID \leftarrow HTTP\_Put(owner, passwd, mbdsol, startSessionURL)$
2: $scriptResult \leftarrow HTTP\_Put(sessionID, name, scrURL, arguments, lang, block, scriptPullRunURL)$

Algorithm 1: Script delegation and execution based on ROA approach

Using the ROA architecture, the Algorithm 1 executes a simple script delegation from the TLM to MLM with HTTP calls. These calls consume resources exposed by two different URLs contained in *startSessionURL* and *scriptPullRunURL* variables. The first variable is addressing the service *beginMbD* exposed by *MbDSAS-WS* on an MLM, *e.g.,* "https://MLM_IP/control/MbD/beginMbD". The calls are based on the PUT method from HTTP because this service creates a session. During the creation of a session, an identification is returned and stored in *sessionID*. In the second command, the *scriptPullRunURL* variable contains the address of the resource exposed by *MbDSAS-WS* named *scriptPullAndRun*. Through the second service, a script will be sent specifying relevant information to its installation and launch, such as a URL of the management script stored on a repository to be pulled (*scrURL*), the script name (*name*), arguments to be used during the launch of the script (*args*), a parameter to wait the end of a script execution (*block*), a

specification of how the script will be stored (*store*) and the language of the script (*lang*).

4.3 MD List Builder

A component designed to support discovery mechanisms and to create standard MDLs was called *MD List Builder*. This component was prototyped in JAVA in combination with LLDP to monitor network interfaces and to create MDLs. Moreover, the *MD list builder* exchanges MDLs by CLI commands with *MbDSAS-WS*, which, in turn, provides MDLs to TLMs through the network following a set of steps outlined in the previous section.

The *MD list builder* was designed to support different filters. These filters are described in XML and act as drivers that enables the *MD list builder* to support other discovery protocols, such as UPnP and LLDP. In addition, *MD list Builder* provides tools to monitor different network interfaces. It means that *MD List Builder* can be used to monitor each network communication performed by a local network interface to discover new SObjs. Therefore, this module enhances the capabilities of an MLM by enabling the support of different protocols and technologies to find available SObjs.

Filters from the *MD list builder* can be delegated to MLMs to be installed through *MbD solutions*. For example, a TLM can delegate to an MLM a program code (*e.g.,* a bundle or a script) containing filters and other needed files to be installed at the *MD list builder* component. This installation will be managed by a defined *MbD Solution* (*e.g.,* OSGi or Script MIB).

The development of a MbDSAS prototype allows to verify how this architecture may be applied to 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 as to 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.

5 CASE STUDY, RESULTS, AND ANALYSIS

Through this chapter, we describe a generic scenario where MbDSAS may be deployed. Afterwards, we apply the generic scenario to an airport station. In addition, metrics are defined for the assessment of the performance of MbDSAS trying to solve different research questions. For the purpose of addressing these questions, experiments were carried out in a carefully described test environment. Finally, results obtained are shown and analyzed presenting our considerations.

5.1 Scenario

MbDSAS must be evaluated to be classified as a management solution to IoT scenarios. For the properly evaluation of 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 (Sundmaeker et al., 2010). 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 5.1 and are described as follows.

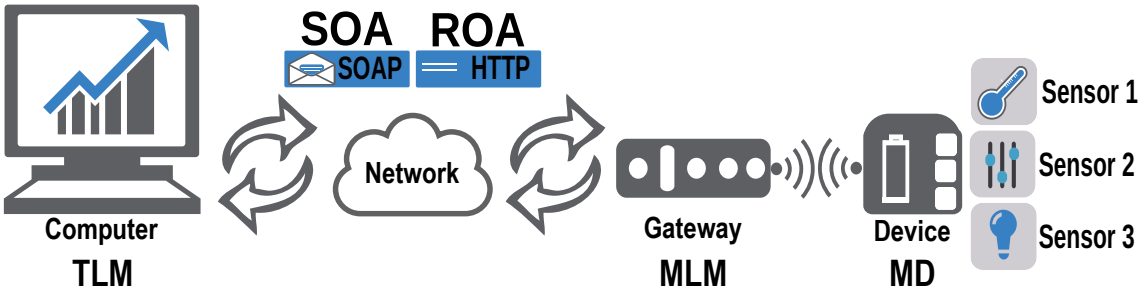

Figure 5.1: MbDSAS generic scenario

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 an MbDSAS-WS installed

on an 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 dissertation (the Script MIB or OSGi) to manage scripts. These scripts may 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 to a traditional computer, such as few memory availability, low processing power, and battery supply 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 to be aware of their arrivals and departures.

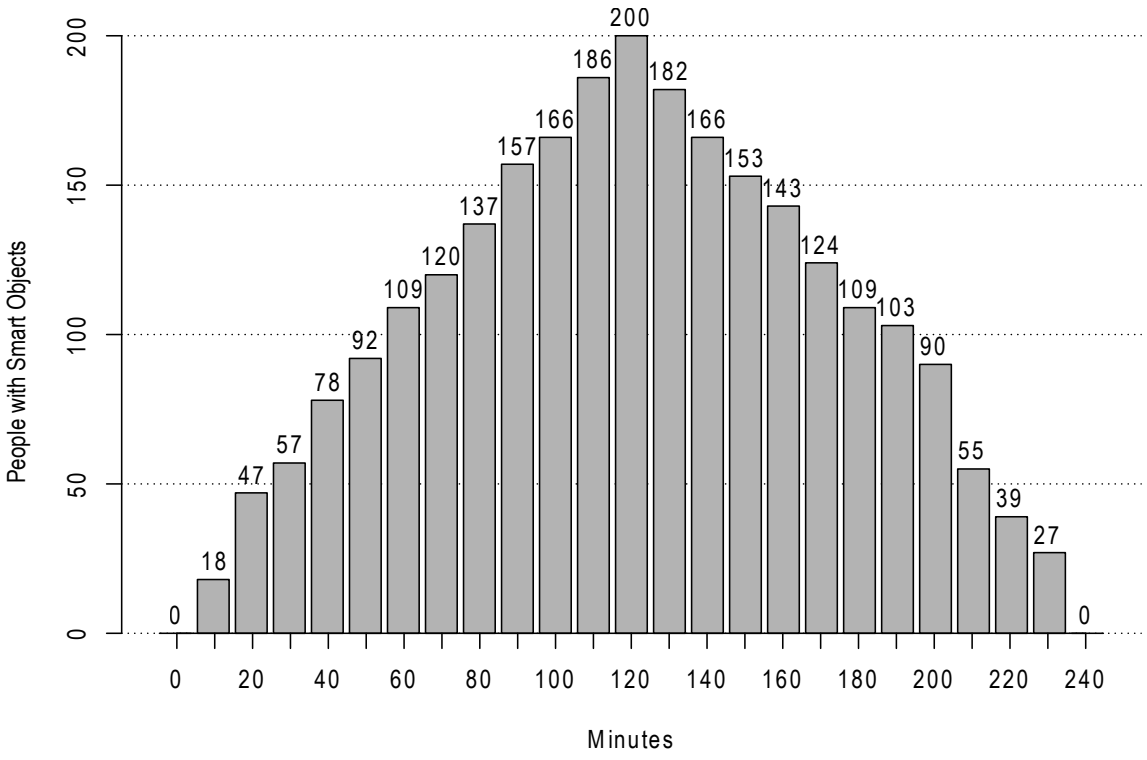

Figure 5.2: Cumulative airport traffic model

We deployed MbDSAS prototype from Chapter 4 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 (Park e Ahn, 2003)(Solak et al., 2009).

A computer system may replicate a people traffic airport through a poisson distribution with $\lambda = 0.6$, as it is depicted in Figure 5.2 (Solak et al., 2009). 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 people with SObj. In the airport 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 5.2.

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?

5.2 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 generic scenario as well as 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 life cycle. 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 (Script MIB or OSGi), answering the first and third question defined above. In addition, the standard to execute management delegations, *i.e.,* SNMP, may be compared with MbDSAS-WS, 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 replicated 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 waiting time 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 dissertation 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 script delegations, in which, these scripts has no waiting time for the sake of comparison among the access approaches, *i.e.,* SNMP, SOA, and ROA. It means that a different waiting time for the scripts would make SNMP to consume a larger amount of network traffic, being unfair to be compared with the other approaches, mainly, because of the polling strategy to manage the script life cycle (Granville et al., 2009).

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 in the quality of 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 dissertation 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 that an MLM can support. Therefore, the CPU utilization was adopted in this dissertation as a metric to measure the efficiency of MbDSAS against the traditional management. In this case, 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 dissertation 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 experiment.

Each of the metrics stated above were used to guide which information should be collected during the experiments executions. These experiments were based on a test environment better explained in the next section.

5.2.1 Test environment

The test environment was composed of three main entities: (*i*) a personal computer that was mapped to a TLM, which runs the Management Application software, developed in Java to delegate scripts and collect MDLs from MLMs; (*ii*) two gateways mapped as MLMs, each one installed with one *MbD Solution*, *i.e.,* Script MIB and OSGi, that were instantiated as virtual machines with an MbDSAS-WS as well as an *MD List Builder* module installed; 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.,* (Wickboldt et al., 2012).

In Table 5.1, 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 5.1: Hardware and software specifications

The specified hardwares and softwares depicted above were used during experiments based on the defined metrics of the Section 5.2. In addition, all of these metrics were measured 30 times and summarized in averages. For each average, the confidence interval of 95% is showed for each sample plotted graphically in next subsections.

5.2.2 MbDSAS response time analysis

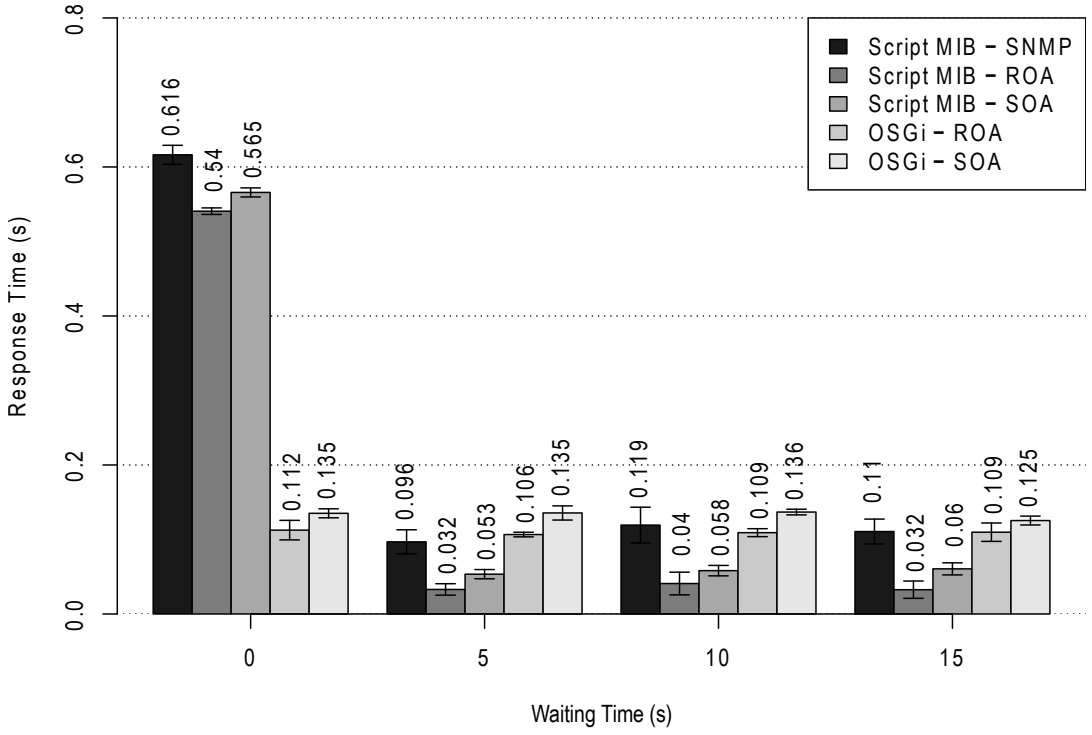

Figure 5.3: Response time of MbDSAS using *scriptPullAndRun*

In Figure 5.3, a management script delegation was performed through the *scriptPullAndRun* service to determine the response time of MbDSAS for each access architecture (ROA and SOA) combined with an *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 an *MbD solution* that was performed in different waiting times, *i.e.,* 0, 5, 10, and 15 seconds.

As can be seen in Figure 5.3, 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 (Granville et al., 2009). 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 Script MIB as the *MbD solution* to be used in MbDSAS.

Additionally, in Figure 5.3, 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 may be defined as the best

access architecture to implement MbDSAS, answering the first question stated in Section 5.1.

5.2.3 MbDSAS network traffic

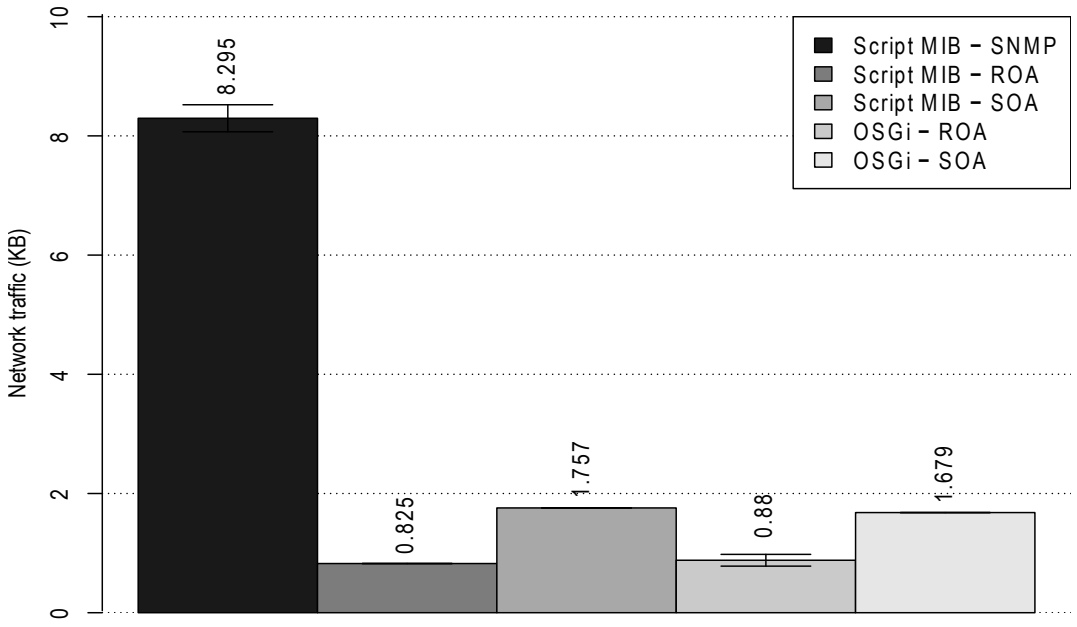

Figure 5.4: Network traffic of MbDSAS with zero waiting time

Similar to the response time, in Figure 5.4, 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 5.4, 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 implement MbDSAS.

In summary, answering the first, second, and third research question of Section 5.1, MbDSAS performed better with the ROA access architecture in response time (0.032 seconds) and network traffic (0.825 KB). In addition, by analyzing results from this access architecture, we defined that the OSGi is preferred as the most stable solution against the Script MIB that is unstable. 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 a traditional management prototype.

5.2.4 MbDSAS management delay

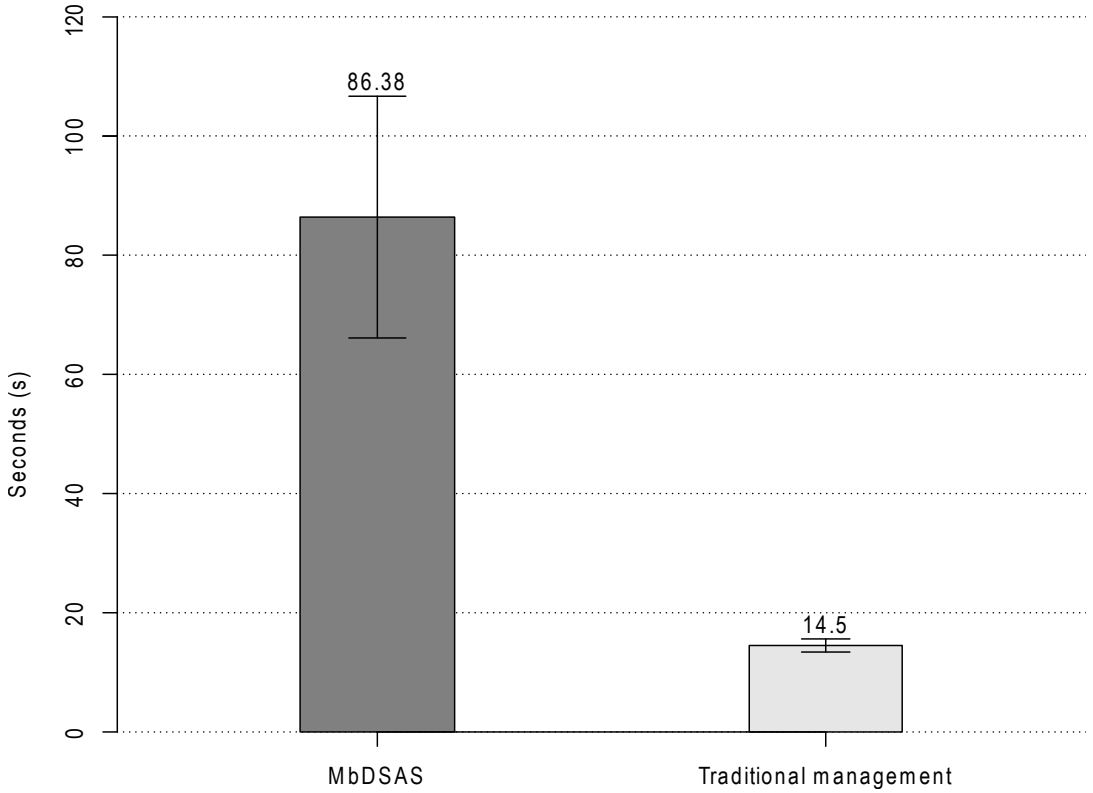

Figure 5.5: Management delay of MbDSAS (OSGi-ROA)

MbDSAS and the traditional management are compared in Figure 5.5. MbDSAS reconfigure dynamically the management scripts while the traditional management has all management scripts previously installed. The airport modeled scenario, described in Section 5.1 was used during four hours of experiment for each replication. The measurements are presented in averages of 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 5.5, 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 at least during the experiments to manage every SObjs of the airport modeled scenario. However, in an IoT scenario, is preferable that a management solution have flexibility to interact dinamically with new SObjs than a traditional management system that does 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. Answering partially the fourth question defined in Section 5.1.

5.2.5 MbDSAS CPU utilization

In parallel with the management delay measurement, MbDSAS and the traditional management was used to manage the airport modeled scenario, as depicted in Figure 5.6. 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

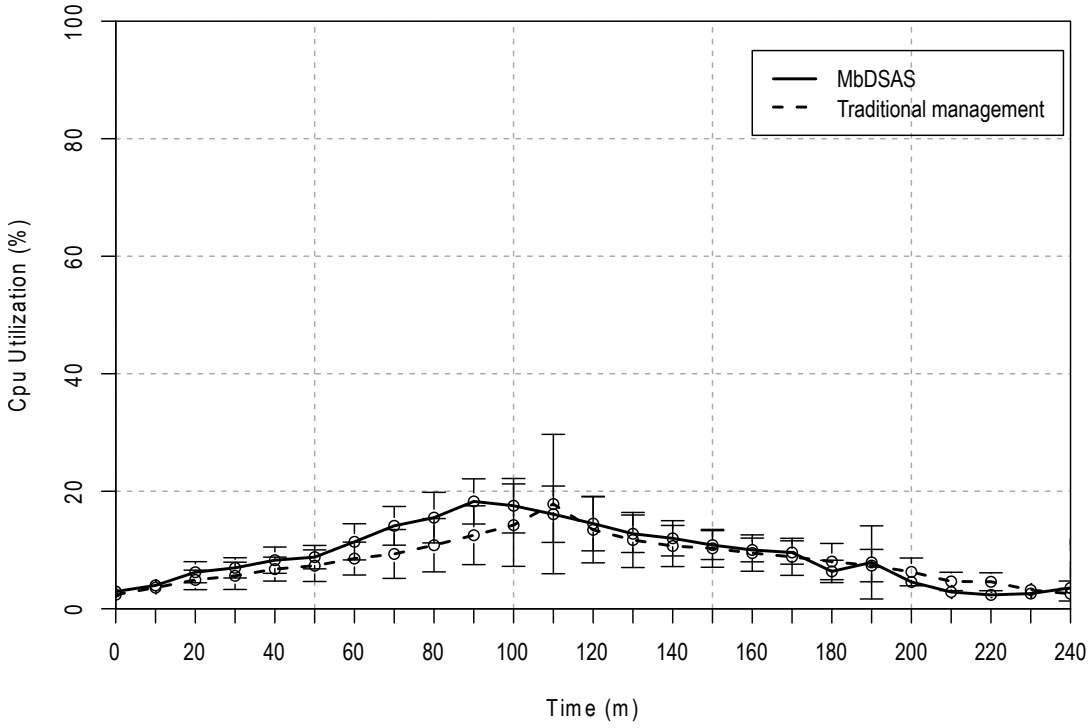

Figure 5.6: CPU utilization of MbDSAS (OSGi-ROA)

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 quite similar, despite of MbDSAS scripts installations, SObjs discoveries, and two hundred MLM reconfigurations. Both of then have their higher CPU utilization near two hours (120 minutes) of experiment 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 two hundred devices join the network, during the first 120 minutes, and then start to leave the network in the next 120 minutes.

Almost all the confidence intervals of MbDSAS and the traditional management intercept each other, therefore, they are quite similar and cannot be compared in terms of CPU utilization. However, this is good for MbDSAS because indicates that even with new features, *i.e.,* reconfigurability 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 fourth and fifth question of Section 5.1.

5.2.6 MbDSAS memory utilization

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 5.7. 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 5.7, MbDSAS and the traditional management behave similar at the beginning of the experiment using the available memory and reaching the maximum available near two hours of experiment. However, different from the CPU utilization, the

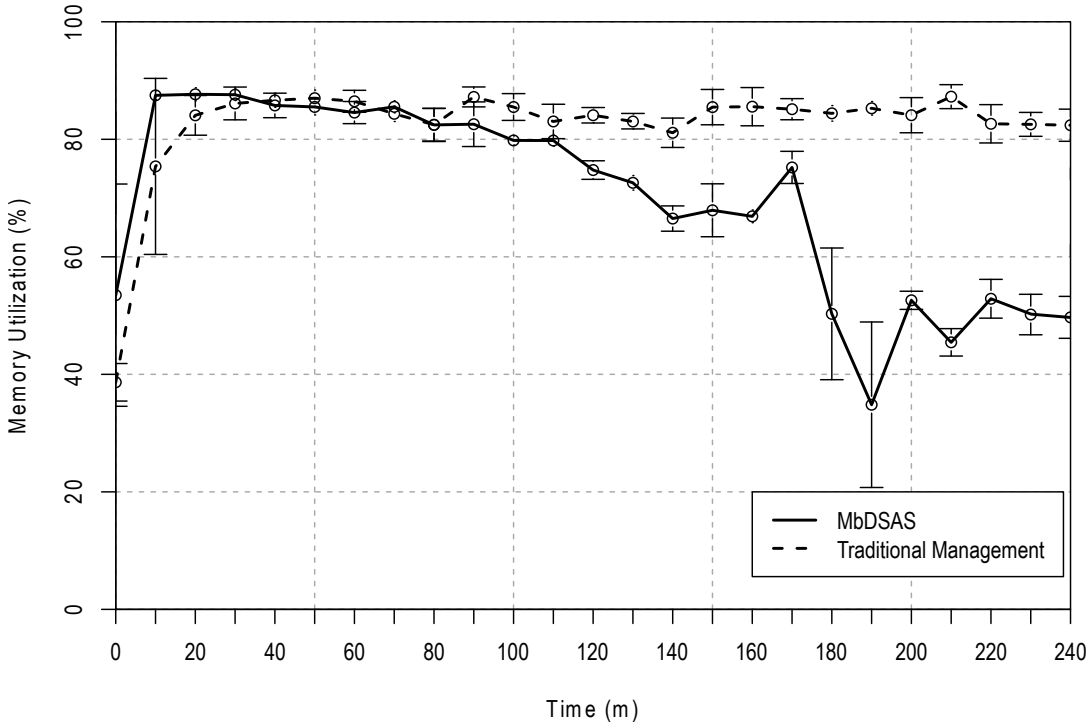

Figure 5.7: Memory utilization of MbDSAS (OSGi-ROA)

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, have their management script placed on the MLM 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 results 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. These results enable to answer the fifth question created at Section 5.1, MbDSAS uses better the computational resources available in an MLM than a traditional management.

In summary, MbDSAS offers a trade-off when managing an IoT scenario, where 50 seconds of management delay most be payed to add reconfiguration and SObj awareness to an MLM. However, MbDSAS offer less memory utilization without consuming more CPU resources. Such a trade-off is the answer for the fourth question of Section 5.1.

6 CONCLUSIONS & FUTURE WORK

In this dissertation, 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 designed 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 an MD list builder component that is in charge of adding SObjs awareness to gateways. Such awareness is provided by SObjs entries in an MDL created by the MD list builder. This MDL, which is not found in traditional MbD solutions, is the key to manage dynamic IoT scenarios. As such, we combine the MD list builder with an MbD solution to expose services to external managers. The consumption of these services enable managers to be aware of what kinds of smart 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.

A prototype was designed as a proof of concepts to evaluate MbDSAS performance. This prototype enabled the conducing of an experimental evaluation to observe which is the most appropriate access architecture, ROA or SOA, and with which MbD solution, Script MIB or OSGi. The first experiment evaluated was the response time measurement where a simple script delegation is performed by MbDSAS. This experiment involved a station installed with a management application that communicates MbDSAS-WS placed at a gateway to delegate scripts. Results showed that MbDSAS implemented through the ROA access architecture performs better in terms of response time. In addition, both MbD solutions, Script MIB and OSGi, were evaluated and compared between each other. When both solutions were faced, the Script MIB showed a better response time rather than the OSGi. However, the Script MIB performance vary accordingly to the script execution time, performing poorly when the script execution is nearby zero.

The second experiment using the MbDSAS prototype was the network traffic measurement. During this experiment, MbDSAS was evaluated accordingly with different access methods, ROA and SOA, in combine with MbD solutions, Script MIB and OSGi. Results showed that ROA enabled MbDSAS to perform better in terms of network traffic rather than SOA. 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. However, 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.

Through the first experiment, we observed that the combination of ROA and OSGi results in less variation in response time during a script delegation. In addition, the second

experiment enabled to determine that ROA is definitely the best access architecture to use in MbDSAS. However, the second experiment analysis yielded no significant results regarding the performance of Script MIB and OSGi. Therefore, the combination of ROA and OSGi is preferred as the best set of technologies to implement MbDSAS accordingly with the first experiment.

The performance of MbDSAS as an IoT management approach was also experimented through a modeled airport scenario. In such a scenario, two hundred SObjs join the network accordingly to a poisson distribution that controls the number of SObjs that joins the network per minute in the first two hours. Afterwards, the same two hundred SObjs start to leave the network randomly by also following a poisson distribution in another two hours. In addition, both poisson distributions present different values from each other, trying to achieve a true airport station flow of SObjs. This airport scenario was used as a base to the execution of all further experiments involving the assessment of MbDSAS performance.

The assessment of the performance of MbDSAS lead us to design an opposite management system to be compared with MbDSAS. This opposite management system was called traditional management system, *i.e.,* a gateway that has all necessarily software to manage every SObjs present at an airport scenario. This traditional management enabled the better understandability of the gathered results because it provided a baseline to compare MbDSAS.

In contrast to the traditional management system in the airport scenario, MbDSAS starts to manage SObjs from a clean state. It means that MbDSAS has no former configured management code to manage any SObj that surrounds it. However, when a SObj joins the network MbDSAS manage to discover their presence, creating MDLs. Afterwards, a manager retrieves these MDLs and delegates management scripts to a MbDSAS-WS placed on a gateway that starts to manage discovered SObjs from the airport scenario. Finally, when SObjs start to leave the network, MbDSAS detects their leaving and removes SObjs entries from its MDLs. As soon as one TLM detects the leaving of SObjs by retrieving a updated MDL, it sends a removal order to MbDSAS to cease the management of these departed SObjs. Both the traditional management system and MbDSAS were compared with results gathered from the airport scenario experiments.

During the experiments, we compared the management delay, CPU, and memory utilization of MbDSAS against a traditional management system. Results shown that MbDSAS has a larger management delay compared to the traditional management. This delay can be assumed as a tradeoff to add dynamic and awareness mechanisms to a network management approach. Therefore, this delay can be assumed as a price to pay for managing a dynamic scenario of the IoT.

Afterwards, we collected results in terms of CPU utilization of both approaches, MbDSAS and the traditional management, during the airport scenario experiment. Results shown that both approaches have similar behavior. These results present their calculated error always intercepting each other. Therefore, both managements approaches can be considered similar in terms of CPU utilization.

The lasts results were measurements of memory RAM utilization during the execution of the airport scenario experiment. These results shown that MbDSAS continuously freed memory resource for other processes to execute, proving its superiority in terms of saving resource compared to a traditional management. In summary, MbDSAS added reconfiguration features to a gateway without great overheads to manage SObjs, showing to be qualified as a potential approach for the management of IoT scenarios. In addition, Mb-

DSAS behave better in an dynamic scenario than a traditional management system. This dissertation's major contribution is a dynamic solution called MbDSAS to adapt gateways from the IoT to manage surrounding SObjs. One of its greatest features is the capability to dynamic receive management routines when a SObj joins the network and cease its management when this SObj leaves the network. In addition, results shown that MbDSAS is more interesting as a management solution than a traditional one, mainly in terms of memory and processing power consumption. Finally, MbDSAS fits as one important solution for the management of the IoT, being based on an MbD distributed architecture that can adapt to many different scenarios from the IoT.

As a future work, we will investigate the use of MbDSAS in Mashups systems (dos Santos et al., 2010). The Mashups systems enable non-profit human managers to build their own management systems with a building block interface. 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 IoT scenarios. Therefore, MbDSAS services and entities, TLM, MLM, and MD, can be mapped to building blocks that will be combined accordingly to network domain needs. These needs will be different accordingly to IoT scenarios, such as plantations, airports, trains, and bus stations. For example, SObjs from a plantation will be mapped to Mashups blocks accordingly with retrieved MDLs. These blocks can be combined with external WS blocks, *e.g.,* Google Maps, Weather, and Bing Maps. Afterwards, managers may define how a grid of SObjs from the plantation can be better managed by delegating specific management scripts, accordingly with specific characteristics from the area, like elevation and weather informations gathered from the external WS.

In the near future, MbDSAS can also be used to implement Gateways as a Service (GaaS). The main idea of GaaS is to enable users to choose their gateways network services that will be provided at the local network. For example, these gateways will be owned by a person registered at an external repository. This repository will offer a user friendly interface where people will authenticate and registry their own gateways. Thus, an authenticated person become enabled to choose potential services to be installed at their home gateways. Afterwards, the external repositories notifies gateways about selected scripts to be delegated. These scripts contains the selected service that will be pushed, installed, and managed by MbDSAS. Finally, local connected users can make use of their new installed services without hard configurations nor installations carried out through verbose command line commands.

MbDSAS can also be applied to healthcare scenarios, such as home patients telemonitoring and hospitals networks (Jara et al., 2010). In these particular scenarios, SObjs collect private informations about patients(Viswanathan et al., 2012). These SObjs have to be carefully managed to avoid any privacy or security flaws (Tarouco et al., 2012). Therefore, MbDSAS may be investigated as a management system approach to manage healthcare scenarios. In this investigation, tests may show which are the security and privacy aspects that MbDSAS succeed or fail to address in these healthcare scenarios.

REFERENCES

ABERER, K., HAUSWIRTH, M., E SALEHI, A. 2007. Infrastructure for data processing in large-scale interconnected sensor networks. In *Mobile Data Management, 2007 International Conference on*. 198–205.

BOTTAZZI, D., CORRADI, A., E MONTANARI, R. 2006. Context-aware middleware solutions for anytime and anywhere emergency assistance to elderly people. *Communications Magazine, IEEE 44,* 4 (april), 82 – 90.

BOX, D., EHNEBUSKE, D., KAKIVAYA, G., LAYMAN, A., MENDELSOHN, N., NIELSEN, H., THATTE, S., E WINER, D. 2000. Simple object access protocol (SOAP) 1.1.

CANNATA, A., GEROSA, M., E TAISCH, M. 2008. Socrades: A framework for developing intelligent systems in manufacturing. In *Industrial Engineering and Engineering Management, 2008. IEEM 2008. IEEE International Conference on*. 1904–1908.

CONGDON, P. 2002. Link layer discovery protocol and MIB. *V1. 0 May 20. 2002*, 1–20. [ONLINE] `http://www.ieee802.org/1/files/public/docs2002/lldp-protocol-02.pdf`. Last access in june, 2013.

DECKER, C., RIEDEL, T., BEIGL, M., DE SOUZA, L., SPIESS, P., MULLER, J., E HALLER, S. 2007. Collaborative business items. In *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on*. 40–47.

DOS SANTOS, C. R. P., BEZERRA, R., CERON, J., GRANVILLE, L. Z., E TAROUCO, L. M. R. 2010. On using mashups for composing network management applications. *Communications Magazine, IEEE 48,* 12 (december), 112 –122.

DOS SANTOS, C. R. P., SANTA, L. F. D., MARQUEZAN, C. C., CECHIN, S. L., SALVADOR, E. M., GRANVILLE, L. Z., ALMEIDA, M. J. B., E TAROUCO, L. M. R. 2008. On the design and performance evaluation of notification support for p2p-based network management. In *Proceedings of the 2008 ACM symposium on Applied computing*. SAC '08. ACM, New York, NY, USA, 2057–2062.

DUNKELS, A. E VASSEUR, J. 2008. IP for smart objects. *IPSO alliance white paper*. [ONLINE] `http://www.ipso-alliance.org/`. Last access in june, 2013.

EISENHAUER, M., ROSENGREN, P., E ANTOLIN, P. 2009. A development platform for integrating wireless devices and sensors into ambient intelligence systems. In *Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009. SECON Workshops '09. 6th Annual IEEE Communications Society Conference on*. 1–3.

EISENHAUER, M., ROSENGREN, P., E ANTOLIN, P. 2010. Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems. In *The Internet of Things*, D. Giusto, A. Iera, G. Morabito, e L. Atzori, Eds. Springer New York, 367–373.

FIELDING, R. 2000. Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California.

FIELDING, R. E TAYLOR, R. 2002. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT) 2,* 2, 115–150.

FIOREZE, T., GRANVILLE, L., ALMEIDA, M., E TAROUCO, L. 2005. Comparing web services with snmp in a management by delegation environment. In *International Symposium on Integrated Network Management, 2005. IM 2005. 2005 9th IFIP/IEEE*. 601 – 614.

GAMA, K., TOUSEAU, L., E DONSEZ, D. 2012. Combining heterogeneous service technologies for building an internet of things middleware. *Elsevier, Computer Communications 35,* 4, 405 – 417.

GOLDSZMIDT, G., YEMINI, Y., E YEMINI, S. 2010. Network management by delegation: the mad approach. In *CASCON First Decade High Impact Papers*. CASCON '10. ACM, New York, NY, USA, 78–92.

GRANVILLE, L. Z., NEISSE, R., VIANNA, R. L., E FIOREZE, T. 2009. *Handbook of Research on Telecommunications Planning and Management for Business*. IGI Global.

GUINARD, D., TRIFA, V., E WILDE, E. 2010. A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*. IEEE, 1–8.

HONKOLA, J., LAINE, H., BROWN, R., E TYRKKO, O. 2010. Smart-m3 information sharing platform. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*. 1041–1046.

JARA, A., ZAMORA, M., E SKARMETA, A. 2010. An architecture based on internet of things to support mobility and security in medical environments. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*. IEEE, 1–5.

LEVI, D. E SCHOENWAELDER, J. 2001. Definitions of managed objects for the delegation of management scripts - rfc3165. *RFCs and Standards, IETF*.

MAROTTA, M. A., CARBONE, J. F., SANTANNA, J. J. C., E TAROUCO, L. M. R. 2013. Through the Internet of Things - a Management by Delegation Smart Object Aware System (MbDSAS). In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*.

MARTIN-FLATIN, J.-P., ZNATY, S., E HUBAUX, J.-P. 1999. A survey of distributed enterprise network and systems management paradigms. *Journal of Network and Systems Management 7,* 1, 9–26.

MASSAGUER, D., HORE, B., DIALLO, M. H., MEHROTRA, S., E VENKATASUBRAMANIAN, N. 2009. Middleware for pervasive spaces: balancing privacy and utility. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. Middleware '09. Springer-Verlag New York, Inc., New York, NY, USA, 13:1–13:20.

MEYER, K., ERLINGER, M., BETSER, J., SUNSHINE, C., GOLDSZMIDT, G., E YEMINI, Y. 1995. Decentralizing control and intelligence in network management. In *Proceedings of the fourth international symposium on Integrated network management IV*. Citeseer, 4–16.

MUKHTAR, H., KANG-MYO, K., CHAUDHRY, S., AKBAR, A., KI-HYUNG, K., E YOO, S.-W. 2008. LNMP- Management architecture for IPv6 based low-power wireless Personal Area Networks (6LoWPAN). In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*. 417 –424.

OSGI ALLIANCE. 2012. OSGi - The Dynamic Module System for Java. [ONLINE] `http://www.osgi.org/Main/HomePage`. Last access at august, 2012.

PARK, Y. E AHN, S. B. 2003. Optimal assignment for check-in counters based on passenger arrival behaviour at an airport. *Transportation Planning and Technology 26,* 5, 397–416.

PAUTASSO, C., ZIMMERMANN, O., E LEYMANN, F. 2008. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceeding of the 17th international conference on World Wide Web*. WWW '08. ACM, New York, NY, USA, 805–814.

PRESSER, M., BARNAGHI, P., EURICH, M., E VILLALONGA, C. 2009. The sensei project: integrating the physical world with the digital world of the network of the future. *Communications Magazine, IEEE 47,* 4, 1–4.

RELLERMEYER, J., DULLER, M., GILMER, K., MARAGKOS, D., PAPAGEORGIOU, D., E ALONSO, G. 2008. The software fabric for the internet of things. In *Proceedings of the 1st international conference on The internet of things*. Springer-Verlag, 87–104.

REYNOLDS, F. 2006. The ubiquitous web, UPnP and smart homes. *UPnP forum white paper*. [ONLINE] `http://www.w3.org/2006/02/reynolds-paper.pdf`. Last access in june, 2013.

SCHONWALDER, J., QUITTEK, J., E KAPPLER, C. 2000. Building distributed management applications with the ietf script mib. *Selected Areas in Communications, IEEE Journal on 18,* 5 (may), 702–714.

SOLAK, S., CLARKE, J.-P. B., E JOHNSON, E. L. 2009. Airport terminal capacity planning. *Transportation Research Part B: Methodological 43,* 6, 659–676.

SUNDMAEKER, H., GUILLEMIN, P., FRIESS, P., E WOELFFLÉ, S. 2010. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*.

TAROUCO, L. M. R., BERTHOLDO, L., GRANVILLE, L. Z., ARBIZA, L., CARBONE, F., MAROTTA, M. A., E SANTANA, J. J. 2012. Internet of things in healthcare : Interoperability and security issues. *International Workshop on Mobile Consumer Health Care Networks, Systems and Services, IEEE 1*, 1–6.

TEIXEIRA, T., HACHEM, S., ISSARNY, V., E GEORGANTAS, N. 2011. Service oriented middleware for the internet of things: A perspective. *Towards a Service-Based Internet, Springer*, 220–229.

VERMESAN, O., FRIESS, P., GUILLEMIN, P., GUSMEROLI, S., SUNDMAEKER, H., BASSI, A., JUBERT, I., MAZURA, M., HARRISON, M., EISENHAUER10, M., ET AL. 2008. Internet of things strategic research roadmap. *Aerospace Technologies and Applications for Dual Use*, 9.

VISWANATHAN, H., CHEN, B., E POMPILI, D. 2012. Research challenges in computation, communication, and context awareness for ubiquitous healthcare. *Communications Magazine, IEEE 50,* 5 (may), 92 –99.

WICKBOLDT, J. A., GRANVILLE, L. Z., SCHNEIDER, F., DUDKOWSKI, D., E BRUNNER, M. 2012. A new approach to the design of flexible cloud management platforms. In *8th International Conference on Network and Service Management (CNSM)*. Las Vegas, USA, 155–158.

APPENDIX A

This appendix presents one article "Through the Internet of Things - A Management by Delegation Smart Object Aware System". This article is a paper result of the second year of the Masters and presents the use of Management by Delegation concepts, WS, and discovery mechanisms in IoT network management. This paper also presents an architecture for the theme, called MbDSAS, shown in details in this dissertation, which enables management systems to reconfigure themselves to better support the dynamicity of IoT. Finally, the article presented a performance evaluation between MbDSAS and a traditional management system.

- Title: Through the Internet of Things - A Management by Delegation Smart Object Aware System
- Name: IEEE Computer Software and Applications Conference (COMPSAC 2013)
- URL: http://compsac.cs.iastate.edu/
- Date: July 22-26, 2013
- Local: Kyoto Terrsa, Japan

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

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.

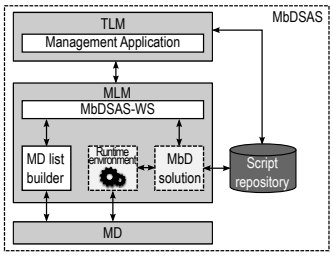

Figure 1. MbDSAS architecture to manage SObjs

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.

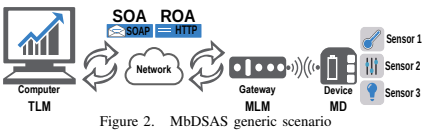

Figure 2. MbDSAS generic scenario

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].

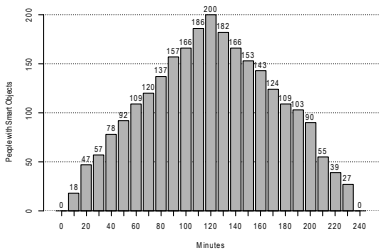

Figure 3. Cumulative airport traffic model

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.

B. MbDSAS response time analysis

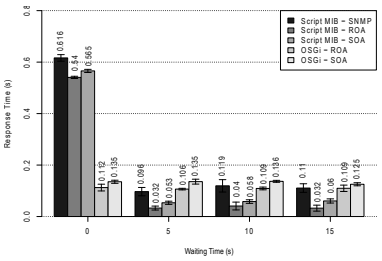

Figure 4. Response time of MbDSAS using *scriptPullAndRun*

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

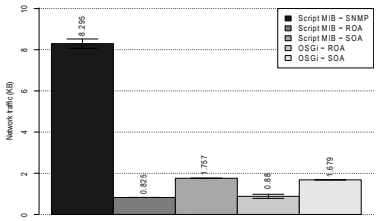

Figure 5. Network traffic of MbDSAS with zero waiting time

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

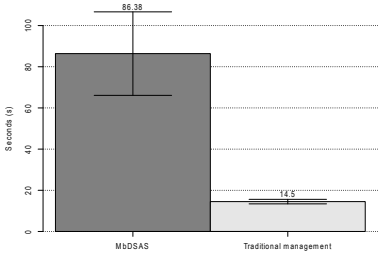

Figure 6. Management delay of MbDSAS (OSGi-ROA)

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

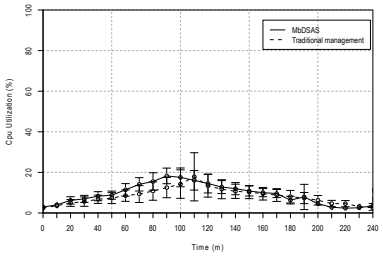

Figure 7. CPU utilization of MbDSAS (OSGi-ROA)

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

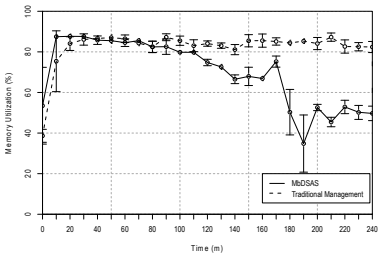

Figure 8. Memory utilization of MbDSAS (OSGi-ROA)

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.