View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Aaltodoc Publication Archive

Aalto University

School of Science

Degree Programme in Service Design and Engineering

Roman Filippov

Security model for the Open Messaging Interface (O-MI) Protocol

Master's Thesis

Espoo, June 17, 2016

Supervisor:	Professor Kary Främling
Advisor:	MSc. Andrea Buda

Aalto University
School of Science
Degree Programme in Service Design and Engineering

Aalto University
School of Science

ABSTRACT OF MASTER'S THESIS

Author:	Roman Filippov		
Title:	Security model for the Open Messaging Interface (O-MI) Protocol		
Date:	June 17, 2016	**Pages:**	vii + 70
Major:	Service Design and Engineering	**Code:**	T-86
Supervisor:	Professor Kary Främling		
Advisor:	MSc. Andrea Buda		

The continuous improvements of computing, networking, storage and sensing technologies together with diffusion of "always-on" internet connectivity is boosting the development of the so-called Internet of Things (IoT).

The number of IoT vendors is also rapidly growing, providing solutions for all levels of the IoT stack. Despite the universal agreement on the need of a standardized technology stack (following the model of the world-wide-web), there is a proliferation of industry-driven domain specific standards that are hindering the development of a single IoT ecosystem. An attempt to solve this situation is the introduction of O-MI (Open Messaging Interface) and O-DF (Open Data Format), two domain independent standards published by the Open Group. The standards do not define any specific security model, and this thesis work tries to define suitable access control and authentication mechanisms that can regulate the rights of different principals and operations defined in these standards. First, an introduction of the O-MI and O-DF standards, including a comparison with existing standards is provided. Then, envisioned security model is presented, together with the implementation details of the plug-in module developed for the O-MI and O-DF reference implementation.

Keywords:	Internet of Things, O-MI, O-DF, Intelligent Products, Messaging Standards, User Authentication, Access Management, Security Policies.
Language:	English

Acknowledgements

This Master's thesis was written as a partial fulfillment for the Master of Science degree of the degree program in Service Design and Engineering in Aalto University.

Besides my own efforts, motivation and support from many others have helped me a lot in success of this thorough work. I take the opportunity to express my personal gratitude to all people who have been encouraging and guiding me through the process and played a significant role in a completion of this project. In particular, I would like to thank my supervisor, Professor Kary Främling for his priceless support and constructive feedback during my research work. I was writing a thesis that meets European standards for the first time so I would like so thank everyone who helped me to do everything right. I would also like to express my gratitude to my advisor, MSc. Andrea Buda for his professional guidance, experience sharing and countless priceless advices during my research work. I would also thank all my teammates - Manik, Tuomas, Jussi and others. It was a great time and a big pleasure of working together on such an interesting and important project.

At last I would like to give my sincere gratitude to my parents for their love, support and motivation. My special thanks are to my near and dear ones for their patience, support and encouragement.

Espoo, June 17, 2016

Roman Filippov

Contents

List of Figures

Chapter 1

Introduction

This chapter is an introduction to the thesis. It starts with the reasoning part (motivation), then provides some background information about the field of research. After that the main research questions are asked and research framework is introduced. At last, the structure of this thesis is presented.

1.1 Motivation

The Internet was originally developed for the military [1] and security has been a concern since very first appearance. The first network, ARPANET [2] was established in 1969 with the only two members, but it grew rapidly to 213 by 1981 [3, 4], mainly adding the main universities across the United States. The rapid diffusion of the personal computer, together with the widespread adoption of standards such as TCP/IP (Transmission Control Protocol/Internet Protocol), HTTP (Hypertext Transfer Protocol) and HTML (Hypertext Markup Language), lead to the creation of a worldwide, standardized network commonly known as the WWW (World Wide Web).

During the last 20 years the Internet has grown exponentially reaching over one-third of the world population [5, 6] and has become an essential part of our daily life. The web has become the primary mean to access to information, news, instant communication, shopping and entertainment. It has transformed our everyday life activities so radically that it has been defined by some as the “8th continent” [7, 8]. However, the more we rely on the web to store our data and support our daily activities, the more we become vulnerable to potential security attacks. For example, online shopping provide the amazing opportunity to purchase anything from anywhere in the world, but banks and online payment systems are constantly fighting a cyber war with hackers trying to compromise their systems [9, 10]. These attacks are not limited to large service providers, but they also affect ordinary users. Phishing is a big threat for naive or inexperienced Internet. Although the attention and education on cyber-security is rising, new challenges are on the horizon. An increasing number of physical devices, ranging from light bulbs, home appliances, heartbeat monitors to more sensitive ones such as fire alarms, ventilation and heating systems, locks [11, 12, 13] are getting connected to the Internet. In the past, the worst consequence of a cyber-attack could be evaluated in term of monetary losses; while the potential hack of some of these devices can translate into real physical danger affecting human lives.

The Internet of things is quickly becoming reality, and security should be the number one priority of any new connected product or installation. However this is seldom the case. The fear of losing business opportunities creates great pressure to bring the so-called Minimum Viable Product (MVP) as fast as possible to market. Obviously in these conditions, security is rarely paramount and securing IoT devices can be quite complicated and expensive. It is reasonable to think that ransomware (a type of malware that restricts access to the infected system in some way, and demands that the user pay a ransom to the malware operators to remove the restriction) will soon move from computers to IoT devices.

1.2 Research Objective and Framework

The motivations described above highlight how IoT security is a very relevant, very vast and complex topic. The complexity in part comes from the fact that the IoT protocol landscape is still pretty fragmented. Partly reusing and extending the OSI model (Open Systems Interconnection model), figure 1.1 tries to depict the current IoT standards and protocols landscape. It is quite

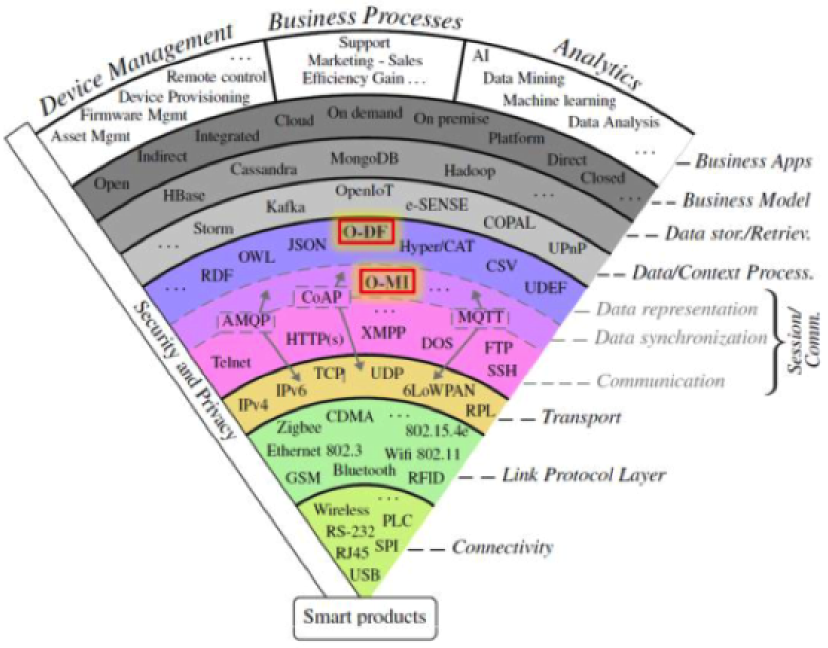


Figure 1.1: IoT standards/protocols landscape. Adapted from [14]

evident how the current status is much more dynamic and messy compared to the web (or "traditional Internet").

Many protocols has been designed and used for specific use cases. This allows having fast and successful one-shoot applications but in turns is undermining the development of a true unified IoT technology stack. A clear solution to the proliferation of vertical solutions and custom APIs is obviously standardization. The vision for a seamless IoT requires protocols to be domain independent yet capable to adapt to different standards and environment. An attempt to solve this issues (at least at the application level) comes from the recently published (under The Open Group) standards Open Messaging Interface (O-MI) and Open Data Format (O-DF), whose potential impact for the IoT is similar to the impact of HTTP and HTML for the World Wide Web when they were published.

However, these standards do not define (on purpose) any specific security model. They clearly state how suitable security mechanism can be applied "on-top" of these standards. This thesis focuses on the development of a security model for these promising IoT standards, and tries to give an answer to the following research questions:

- How a common messaging standard can help to solve the problem of the communication between the different devices in the Internet of Things?
- What are O-MI and O-DF standards and how well they satisfy the requirements of the Internet of Things? How they are different in comparison with already existing standards and solutions?
- How can these standards be implemented and integrated into arbitrary application together with the access control and security mechanisms?

The questions above rose because of the need to have a generalized standard that could satisfy the requirements of the Internet of Things. Furthermore, the reference implementation of the O-MI and O-DF standards has to be presented and analyzed. Thus, there are several main guidelines that lead this work:

- **Background topic discussion.** Mainly based on the review of various literature in Internet of Things topic. The review was performed to provide a brief introduction to the industry, its main needs and possible problems. This study goes along with the description of the messaging protocols for the Internet of Things and the process of understanding the requirements for the implementation.
- **Reference implementation overview.** The reference implementation of the O-MI and O-DF standards is studied together with the examples of the possible use cases. That part includes also a discussion about existing security mechanisms and the need of implementing the access control and authentication system for the reference implementation.

- **Security module design and implementation.** The main part of the thesis that describes the whole process from the formulation of the security requirements to the design and implementation of the security module.

1.3 Structure of the Thesis

This thesis consists of six chapters: The chapter following this introduction provides a literature review of the IoT domain, discussing needs and issues of the current state of art. It later dives deeper into existing network protocols that are suitable IoT messaging protocols. In Chapter 3 the description of the core concepts and design principles of O-MI and O-DF standards are provided, justifying its selection as primary application level messaging protocols for the IoT. Chapter 4 describes the current O-MI and O-DF reference implementation and demonstrates how these standards can be used in a particular application. Chapter 5 provides the requirements for the security and access control modules that has been developed during this thesis, followed by the core implementation decisions. Finally, chapter 6 provides summary, conclusions and future research directions of the work performed.

Chapter 2

Internet of Things

2.1 Introduction

Connecting physical devices to Internet allows retrieving data and remote interaction with physical spaces. Combining these types of information, with existing one, such as weather forecast provided by external web services or OpenData dataset, open up new opportunities to create new type of products and services that are far beyond the capability of any given "isolated" system [15].

IoT is an emerging trend that is expanding very rapidly, and it has the potential of impacting our life as much as the Internet itself. In comparison with fixed Internet wave in 1990s that has connected about 1 billion users and mobile Internet wave in 2000s that has connected 2 billion users, the IoT has a potential to connect 28 billion "things" to the Internet by 2020 [16]. Communication not only between human beings, but human-to-machine and even machine-no-machine is possible, which lead to the next generation of intelligent and highly distributed systems. These systems could help us to increase our living standards significantly [17]. Humanity is going to connect almost everything together. Smart watches and glasses, action cameras, cars, even people's homes (lights, relays, temperature and humidity sensors etc.). Smart cities with intelligent traffic lights, parking spaces interconnected into wireless sensor networks will become the normality in the near future [16, 18]. Besides that, IoT is planned to use in Industrial Internet to connect the machines that perform manufacturing processes, get real-time analytics and perform optimizations [19].

One of the main drivers of this rapid evolution is the significant decline in cost of hardware (sensors and processing power) and network connectivity. The cost of bandwidth has been declining year by year and for the last 10 years it has dropped dramatically by a factor of nearly 40x. The processing power did the same but by a factor of 60x [16]. These processes have quite positive consequences for the development of IoT.

Large enterprises, which have already impacted our daily life (e.g. Google, Apple, Samsung etc.) play a major role in the development of the entire IoT ecosystem. They have all the necessary resources (money, technologies and professional staff) to influence current and future development. Moreover, they could decide to introduce radically new products and technologies while providing integration and support in their existing ones (e.g. iOS or Android ecosystem).

On the other hand, research institutes and SMEs working in specific narrow domains are often the ones that bring true innovations, leveraging on the possibility of taking bigger risks, leaner management and bureaucracy. The business model of some these companies often culminates in the acquisition of their technologies by some of the major players mentioned above. Sadly the evolution of those technologies often dies out due to the lack of vision on how they can embed them into their current product portfolio. Every large player is essentially trying to establish "a winning platform", which in many cases is supposed to be open but often contains some proprietary technologies which give control power to the vendor. It is very true in the mobile market, in which is possible to distinguish different devices with different approaches and technologies used. These technologies are often closed to the public and incompatible with each other [20], further hindering interoperability.

However it is possible to witness a very different situation if we consider the World Wide Web, which is a brilliant example of what can be achieved with the wide adoption of a set open standards and technologies [21]. It is imperative to apply the same logic for the IoT, otherwise there is the concrete risk of creating industry verticals in which few vendors have the business interest in keeping the ecosystem fragmented to defend their dominant position.

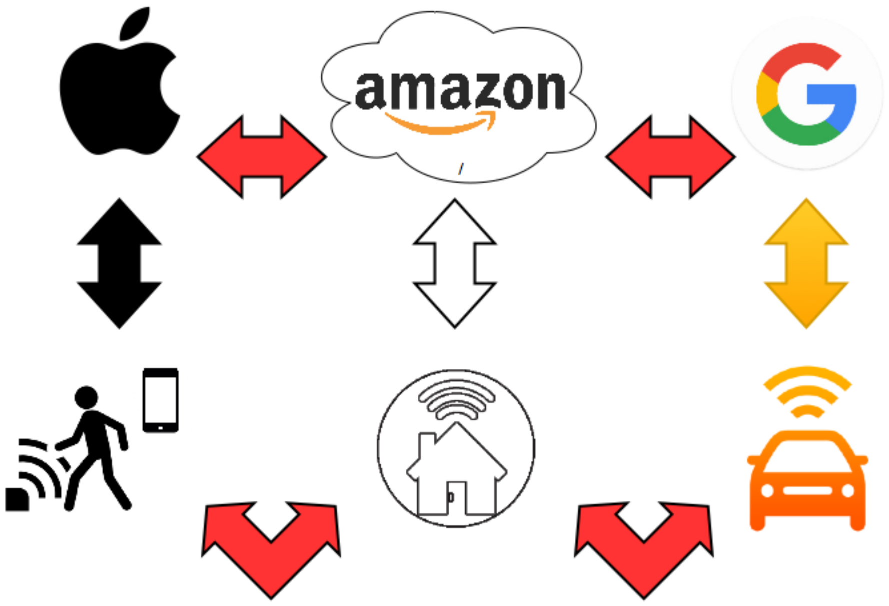


Figure 2.1: Illustration of vertical and horizontal interaction between systems

Fortunately, more than one standardization body is working to prevent this danger. One of these organization is The Open Group, which has published two standards (Open Messaging Interface and Open Data Format) to address the challenges mentioned above. Chapter 3 will provide in depth explanation of these standards and the current reference implementation that provides a "sandbox" infrastructure to showcase and promote the standards.

Before introducing these standards details, this chapter will addressed some of the existing protocols and messaging standards that are commonly used in IoT and describe the set of requirement that a trigger the creation of O-MI and O-DF.

2.2 Issues and concerns

Giving the fact that the IoT will potentially connect billions of devices, there is an understandable concern regarding network capacity and suitable communication patterns and protocols: [22] describes how it would be possible to give every device a unique address using IPv6 that is being integrated already, [23] highlights how the current state-of-the-art requires application-layer gateways both in software and hardware that provide application-specific connectivity to IoT devices. In much the same way that it would be difficult to imagine requiring a new web browser for each website, it is hard to imagine our current approach to IoT connectivity scaling to support the IoT vision. Similar concern are expressed in [24], which states that “The future is not going to be just people talking to people or people accessing information. It’s going to be about using machines to talk to other machines on behalf of people with totally new communication pattern arising between people to machines and machines to machines.

The current situation is an everyday problem for IoT practitioners, dealing with hundreds of different manufacturers with thousands types of different sensors, software, protocols, message structures etc. Without a clear standardization between different organizations and countries the expansion of truly worldwide Internet of Things can become difficult to achieve. It is quite obvious that IoT community must follow the example of the World Wide Web, in which TCP/IP, HTTP/HTML helped the Internet to spread across the world [25]. Now the real question to answer is how it can be achieved. Finding convergence at every level of protocols stack will still take few years and perhaps (at least for some of the lower level) we accept the fact that different protocols serving particular requirements will co-exist. What is already pretty clear is that every level of stack must implements some sort of security mechanism that would fit with the upper and lower levels. This thesis focuses on this aspect and in particular with the application level security.

2.2.1 Why IoT security is so important?

Security is often the hottest point of the discussion between IoT developers and practitioners; this is due to realization that the level of connectivity and involvement of IoT in people's everyday life is much deeper. The information that people share with their devices and sensors is often sensitive and is supposed to provide a background and context-awareness for the new types of services. Therefore it should be guaranteed that this information would not go anywhere else [26, 27]. If everything is connected to the one huge network then every single client of the network could automatically become the starting point of an attack to the whole network and that is the thing that will most likely happen. Especially when talking about publicly available objects such as bus stops and so on. Despite cyber security these objects can be hacked physically and since they do not belong to any particular person it can be hard to protect them. Moreover, there are already many different scenarios that were demonstrated on public - getting the control of a car remotely or stopping its engine when you are riding on a highway [28], hacking a babysitter radio [29] and so on. People's houses, smart watches with the information about the health, traffic lights or public ventilation and life supporting systems can be potentially hacked. These things may scare the society away from the using of the IoT and stop the whole industry from the development. Thus, the security aspects should be carefully designed and worked out. The privacy, authentication, access control mechanisms, the way the sensitive data is stored should satisfy the security requirements and stand the possible attacks in the worst cases.

To sum up, there are several challenges related with the Internet of Things. Some of them are relatively easy to solve by applying existing technologies, but the security problem requires thorough elaboration in a close collaboration with the manufacturers, standardization groups, software developers and the users.

2.3 Messaging protocols

The number of internet-connected Humans and "Things" is on the rise and new types of communication are arising: the usual human-to-device, but also device-to-device, device-to-server, server-to-server communications are going to become widely used [30]. The traditional request-reply communication is only one of the many that are necessary to satisfy the needs of future users and services. A wide range of messaging patterns are already implemented by a number of messaging protocols, this section will describe the communication protocols that have been typically used in IoT applications.

2.3.1 MQTT

MQTT stands for Message Queue Telemetry Transport. It was originally designed by IBM and with the purpose of use in unreliable networks with high latency and low bandwidth [31]. Its main purpose was to collect the data from various devices in the network in one centralized place and make it available for the IT infrastructure [30, 31, 32]. So in IoT terms it can be many devices connected to the cloud server and providing the data (figure 2.2). That can be useful for example to support the pipelines infrastructure and monitor it for acts of leaks or vandalism using thousands of sensors. It runs on the top of TCP providing the reliable stream to deliver the messages from the devices to the data concentrators [30]. The devices can subscribe to certain data updates as well as publish its own data on a real-time basis.

Although MQTT is a quite lightweight, it requires the TCP/IP support that can cause unnecessary complexity and can be difficult to provide using very small and computationally ineffective devices to build sensor networks for example [33]. The publish-subscribe model requires an intermediate actor, the message broker that processes and forwards all the messages. The message broker can become a bottleneck and a single point of failure of the system. Also

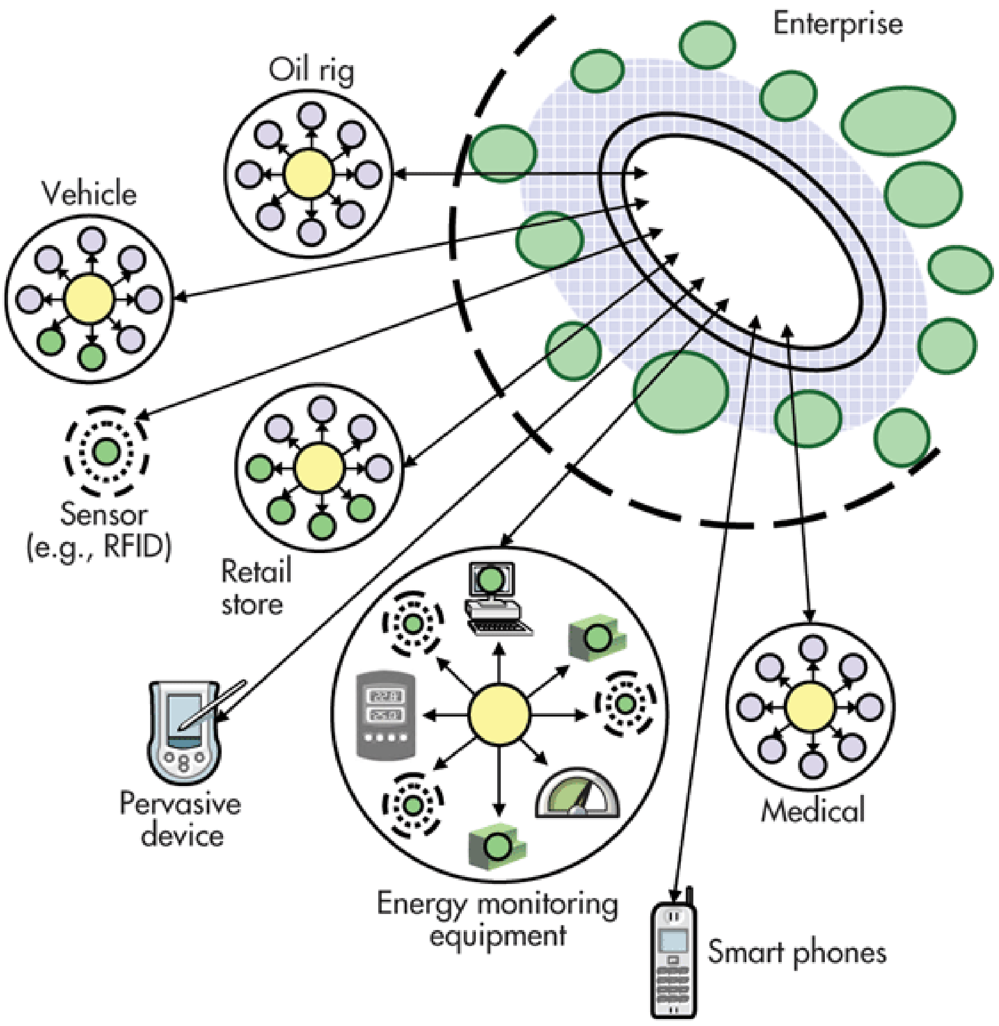


Figure 2.2: MQTT communication model. Source: ibm.com

in the IoT it's not given for granted that the devices always have an Internet or network connection. Thus the direct interaction from machine to machine including publish-subscribe feature would be preferred. The broker can persist the messages, but protocol does not support the persisted messages to backup inside the server [34]. In security terms brokers can require credentials from the clients as well as the connection can be secured with SSL/TLS [34, 35].

2.3.2 CoAP

CoAP or Constrained Application Protocol was intended to use in small electronic devices such as sensors, switches and so on. It was designed with the purpose of easy integration with HTTP, has support for multicast packages, low overhead and quite simple structure [36, 37]. Its main purpose was to provide efficient machine-to-machine communication in restricted environments with a small amount of ROM and RAM memory available. It runs over UDP with QoS support to enable decent level of reliability. The interaction model is server/client where the client sends request and gets response from the server [36]. The subscription feature is supported to some extent - client can observe certain information by setting a special flag in GET request and it will start receiving the notifications afterwards. The notifications are delivered at "best-effort". The protocol itself since it works over UDP is designed as eventually consistent: If the device is not undergo to a new change in state, all subscribers will eventually receive its final status at some time [38]. Even though the QoS provides some sort of reliability for messages to be "confirmed" (by sending an acknowledgment packet from the receiving party) and "non-confirmed", it could cause some traffic overflow in case of resending of the data and waiting for an acknowledgment because UDP has no delivery control mechanism embedded. CoAP supports service discovery using multicast packets (which can be a problem for some networks though) and URI for objects and data [36]. Since it runs over UDP the TLS/SSL can't be supported, but the protocol is still secured by DTLS [38] (although DTLS does not work for group communication using multicast messages and can also add some complexity because of additional control over UDP related to dealing with unreliable environments). It worth mentioning that in an official specification of CoAP there is a chapter about security consideration saying that due to the lack of handshake mechanism in UDP the IP spoofing attacks are possible to a single endpoint, group of endpoints or event the whole network for example by spoofing acknowledge, multicast or observe messages [36].

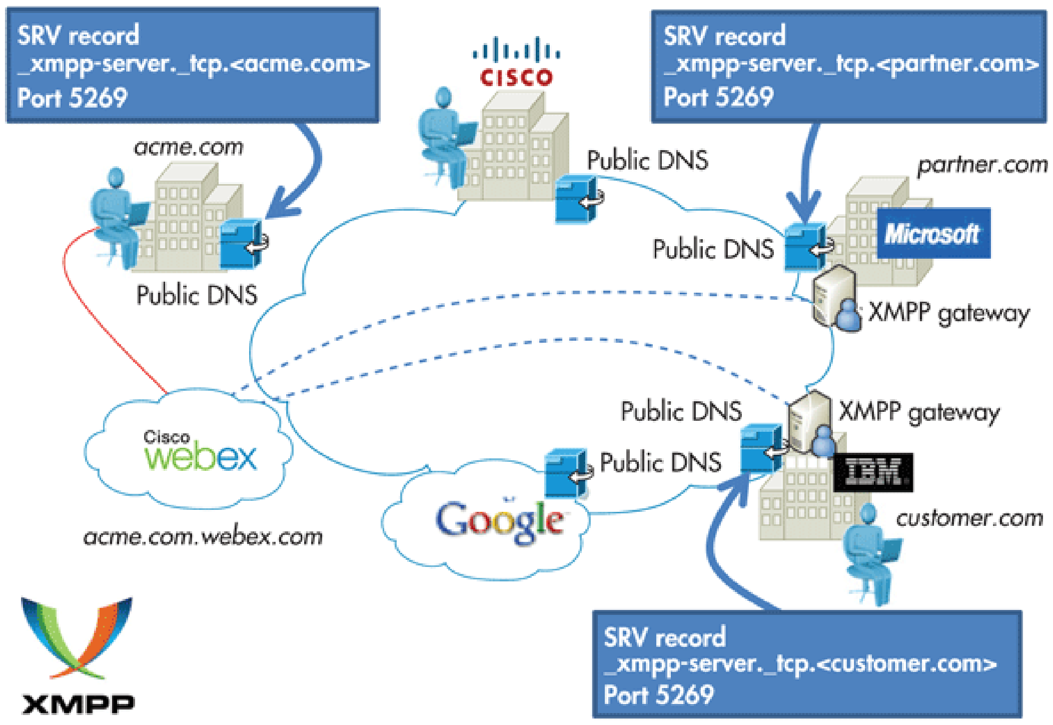


Figure 2.3: XMPP communication model. Source: electronicdesign.com

2.3.3 XMPP

XMPP stands for Extensible Messaging and Presence Protocol, was originally called "Jabber". It was developed to connect people to other people via instant messaging (figure 2.3). It runs on top of HTTP that is on top of TCP. It's not intended to be fast, most implementations even support the polling or update on demand. Thus it provides a reliable, but slow service [39, 40]. Its main strength is in addressing since it uses human-readable user@domain.com names. On the other hand there are several extensions for the protocol that make it fast and applicable for different areas such as VoIP and online gaming. It uses XML as the data format, but it rather uses XML stream than documents basis. It has a decentralized architecture where a client and server interact using the request/response principle, transferring the state (it's called stanzas) through XML. The server has an ability to manage long-lived TCP connection to exchange XML states several times using the same connection. The connection can be also secured using SSL/TLS mechanism [40]. The only format supported is XML, which may introduce significant overhead that can be crucial when using on small embedded devices.

2.3.4 AMQP

AMQP stands for Advanced Message Queuing Protocol. It provides services for the middleware that is a message-oriented. It takes care of the queuing, routing, orientation, reliability and security of the messages (figure 2.4). The protocol supports publish-subscribe paradigm as well [41]. It was developed mainly because of a need from a banking industry and similar ones where there was a lack of reliable and secure protocols that can deliver thousands of highly important transactions. The communication between the endpoints, publishers and subscribers runs over TCP and provides a strictly reliable stream. The QoS provides at-most-once, at-least-once and exactly-once delivery options [42]. Authentication is supported by SASL or TLS technologies under the hood [43]. The data format is defined by a self-describing encoding scheme that has an exact description in a standard document.

The data can include also the application specific information that can be encoded in any way and contain it's own structure [44]. Despite all the advantages, the protocol is quite heavy and is mainly used for server-to-server interaction to exchange the data for example for an analysis or the other purpose even if the target servers are deployed with a completely different hardware and software platforms. Thus its main advantage is interoperability, but the heaviness of the protocol makes its usage on the end devices hard.

To sum up, there are several protocols that can potentially fit a niche in the specific application of the IoT. But each of them has its disadvantages such as a dead lock to the one specific data format, lack of the "back up" feature or

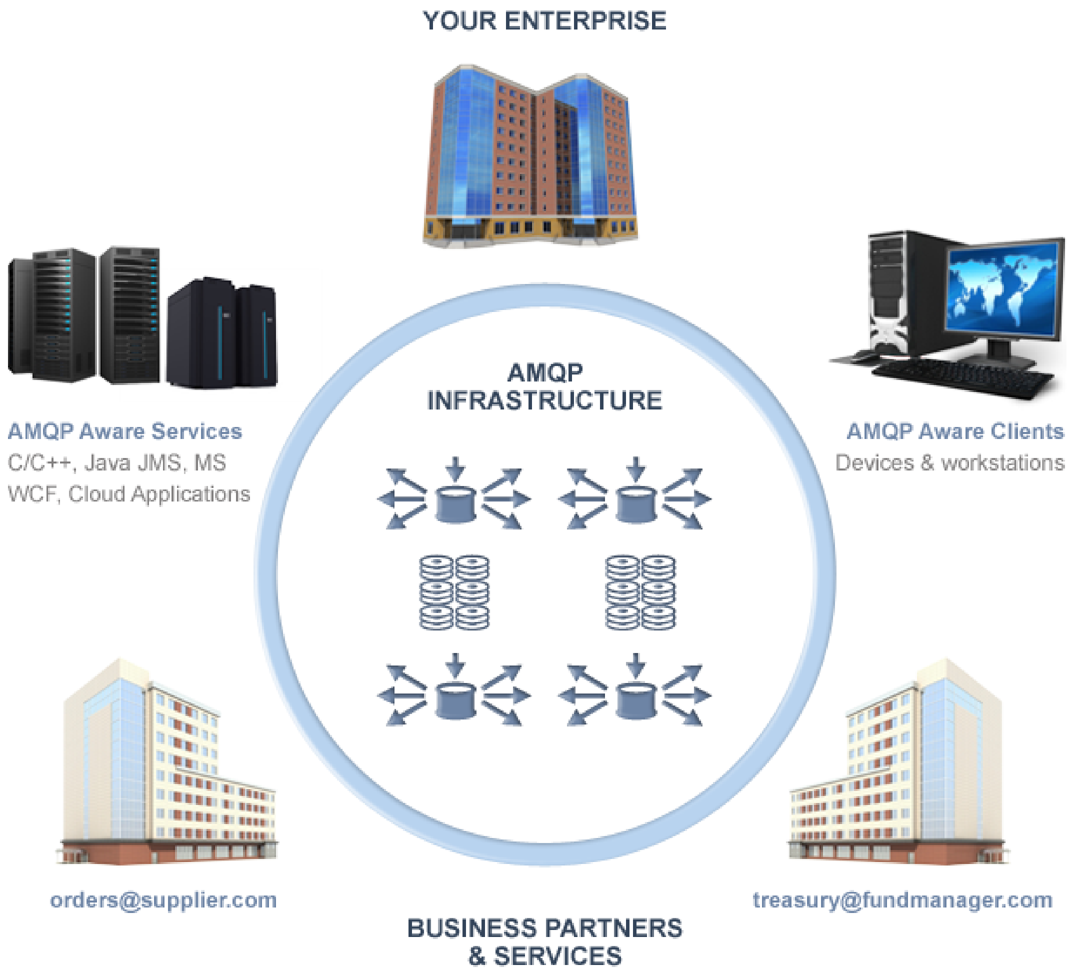


Figure 2.4: AMQP communication model. Source: amqp.org

the mechanism to retrieve the data for a specific period of time. Some of the protocols are suitable for server-to-server or machine-to-machine messaging but then there is a need to invent some intermediate protocols to interact between different levels and that can lead to undesirable increase in the complexity level of the system. To better understand the differences between the protocols and make conclusions they will be analyzed more thoroughly in the next section.

2.4 Suitable messaging protocol for the IoT

In this section the protocols that were described above will be analyzed using comparison framework introduced in [45]. First the framework is briefly described and its main comparison points are outlined, then the comparison between the protocols is performed and some conclusions presented.

2.4.1 Comparison framework

The comparison framework consists of several criteria that are semantically divided into three categories. These categories among with the criteria are introduced below.

Message delivery model. This category defines the criteria that are related to the delivery of the message from the sender to the receiver. This category is the biggest one and makes the basis of the framework. It consists of the following criteria:

1. *Messaging API*: defines whether the messaging interface protocol is application-specific or independent;

2. *Initiation*: defines the way how the transmission of the data is being initiated. It can be either server-initiated (i.e. push mechanism) or client-initiated (i.e. pull mechanism). In the first case the client always listens for the notification and the data from the server. In the second case the client requests new data when it needed ("pulls" it).

3. *Intermediation*: specifies whether the protocol relies on the intermediate parties to complete the operations (or at least offers that possibility).

4. *Persistence*: specifies whether the protocol offers the data persistence feature. That can be split into two modes - persistent and transient. Persistent, means that the data is stored in the system and can be retrieved at any time. Transient - the data will be stored and valid as long as the Time-To-Live (TTL) parameter specifies etc.

5. *Subscription*: specifies whether the protocol supports the subscription

mechanism. In this case two kinds of subscriptions are defined - Interval-based and event-based.

6. *Self-contained*: means whether the message contains all the necessary information to be understandable for the receiver in the right context, meaning if it contains for example a time-to-live parameter, an operation to be performed and so on.

7. *Protocol agnostic*: specifies whether the protocol supports several underlying "low-level" transportation protocols that can be changed without rewriting the core of the protocol itself. It means the possibility to use the protocols such as HTTP, SOAP, SMTP and so on to transfer the data, or even physical transfer using copy to USB sticks.

8. *Synchronicity*: simply defines the support of synchronous and asynchronous operations in the protocol.

9. *Delivery-guarantee*: specifies whether the protocol provides a support for guaranteed message delivery (by sending acknowledge responses for example).

10. *Piggy backing*: specifies the policy to allow piggy backing a new request with the response (i.e. without dissociation of the new request from the response). That property can be very important in real-time applications or applications that are located behind a firewall.

11. *Multiple payloads*: defines whether the protocol supports various formats of the payload data.

Message processing model. This category describes the processing of the received messages by the receiver, how the communication process is defined for the processing the data and sending the response back to the requester. There are two criteria for that:

13. *Processing result*: defines a way, which the processed data is returned to the requester. There are three different options. First single return value - for every request the receiver generates only one single response. Another option is single integrated return value, which means the response that contains the requested value integrated with the data. The last option is a set of individual return values - when response is sent at different intervals.

14. *Communication*: specifies the way that two nodes use to communicate with each other after the message was received from the requester. There are two possible options: separate message or callback address. The first one means sending the response as a separate message to the query, the second one means sending the data as a separate request to the specified callback address that listens for the data.

Message failure model. This is the last category; it defines the actions or the rules that the protocol follows in case of delivery failures or other disruptions. There is only one criterion:

15. *Failure notification*: specifies a way that protocol uses to react to the failures i.e. the way the protocol use to send the failure notifications. There are three possible ways that are usually used and implemented: Timeout of acknowledgement, Reply with error message or Exception.

2.4.2 Comparison between existing protocols

The four existing protocols that are introduced and briefly described are compared with each other and with O-MI / O-DF standard using the comparison framework from [45] that was introduced earlier. The results are in Table 2.1.

It's highlighted that AMQP and O-MI / O-DF standards cover the most of the properties and subproperties and often provide the possibility to choose

Table 2.1: Messaging protocol comparison based on 14 criteria

Property	**Sub-property**	**MQTT**	**CoAP**	**XMPP**	**AMQP**	**O-MI**
Messaging API	Application-specific		✓	✓		
	Application-independent	✓			✓	✓
Initiation	Push	✓	✓	✓	✓	✓
	Pull		✓	✓	✓	✓
Intermediation		✓	✓	✓	✓	✓
Persistence	Transient	✓	✓		✓	✓
	Persistent	✓	✓		✓	✓
Subscription	Interval-based					✓
	Event-based	✓	✓	✓	✓	✓
Self-contained					✓	✓
Protocol agnostic						✓
Synchronicity	Synchronous	✓		✓	✓	✓
	Asynchronous	✓	✓	✓	✓	✓
Delivery-guarantee		✓	✓		✓	✓
Piggy backing			✓			✓
Multiple payloads		✓	✓	✓	✓	✓
Processing Result	Single return value	✓	✓	✓	✓	✓
	Single integrated return value	✓				✓
	Set of individual return value	✓	✓		✓	✓
Communication	Separate message	✓	✓	✓	✓	✓
	Callback address					✓
Failure Notification	Timeout of Acknowledgment	✓	✓		✓	✓
	Reply with error message	✓	✓	✓	✓	✓
	Exception				✓	

between several variants.

The O-MI and O-DF standards are the only ones who provide the possibility to make interval-based subscriptions, which can be important in some real-time applications. All the protocols support usage of intermediate parties and nodes to complete the transactions, but only O-MI / O-DF offers it as an optional (not-necessary) feature. Self-contained message types are important as well because in different systems same field names can have dramatically different actions or meanings and it's crucial for every system to be able to understand from the message what is the semantic of the information that is provided. AMQP and O-MI / O-DF offer that by using special namespaces and links to the field descriptions that can be loaded by the processing software. O-MI is the only standard that provides full protocol agnostic communication and thus can be easily integrated to use different underlying transport protocols. CoAP has somewhat similar in the meaning of easy integration with RESTful HTTP services, but it still uses unreliable UDP as the transmission protocol.

Piggy backing property is also a very important one. In the modern conditions it can be difficult for companies to provide every device with a fixed IP address (e.g. for mobile dynamic environments). In addition to that, companies are tend to secure their data and often establish their workflow using internal local networks that are located behind firewalls. Together with other proactive measures it is used to protect the security, confidentiality and integrity of the data. It leads to challenging conflicts between availability or usability and data security. Here the piggy backing technology plays a big role. It overcomes that problem and still allows establishing bidirectional information exchange with the devices without fixed IP address that are located behind the firewall. For real-time systems related to control and maintenance activities that is a quite important feature. The table above shows that this functionality is covered by CoAP and O-MI / O-DF only.

Starting from different standards of different systems that need to interact with each other it worth to mention the necessity to support different payload formats. Together with the self-contained property of the messages it makes possible for different systems to understand how to deal with the data that was received in the payload. It becomes vital when the complexity of the systems grows as well as the number of actors interacting with each other. In this regard, all the protocols considered supporting this feature.

The callback feature can be useful when doing with the subscriptions and allows specifying a particular URL where the data or the events will be sent. Again when dealing with the actors located behind the firewalls it can help to specify the main processing point that has a fixed IP address and processes the data that is related to several different actors. O-MI / O-DF standards are the only ones who provides the feature.

It's also worth to mention that CoAP and MQTT protocols were designed mainly to work in local environments so they need to be modified to be able to communicate on a global level (e.g. wrapped into a websocket).

To sum up, during the comparison process five different protocols were compared using a special framework. As a result O-MI / O-DF standards seemed to be the most suitable solution to provide a portable and interoperable communication infrastructure for a huge number of actors from device to the systems of devices. Using O-MI and O-DF they can interact easily and thus the problem of using different standards can be solved. These standards are described in a more detail in a section 3.

Chapter 3

O-MI and O-DF

3.1 Introduction

The Open Group is a global consortium that consists of more than 500 members and aims to develop new IT standards that can help individuals and companies to achieve their business goals. The Open Group IoT work group has a pretty clear and ambitious vision: Whereas the Web uses the HTTP protocol for transmitting HTML-formatted information which are rendered in the browser for human consumption, the IoT will use O-MI for transmitting O-DF payloads which will be mainly consumed by information systems [46]. In other words, these standards will enable different manufacturers, individuals and enterprises to connect different kinds of "things" to the Internet and integrate them into their enterprise network and systems on the fly, as it would be if they were developed by the only one vendor. Many manufactures are already gathering information about their products throughout its lifecycle, usually to improve some internal or service processes such as maintenance and get higher level of safety or reliability at lower costs. It can be also used to track the manufacturing process e.g. state and health of the machines, measure energy consumption etc. The collected data can be used for analytics and help to improve manufacturing process, optimize costs and increase product quality. This chapter provides detailed description of both O-MI and O-DF standards, together with real-world examples on how they been used.

3.1.1 O-DF

The Open Data Format is defined as a standard for representing the payload in IoT application. O-DF is specified as an extensible XML Schema. It is structured as a hierarchy with an "Objects" element as its top element. The "Objects" element can contain any number of "Object" sub-elements. It is intentionally defined in a similar way as data structures in object-oriented programming [47]. O-DF addresses the problem of publishing information from various data sources. In IoT these data sources (such as devices, machines, server-based systems etc.) must be able to publish their data, provide the access to consumers, use secure mechanisms to ensure authenticity and integrity of the data and also provide the possibility to filter the information according to the consumer's needs: requester's identity, context, request parameters etc. [47]. O-DF makes it easy for information providers to publish the data using ordinary URL (Uniform Resource Locator) address. Moreover, O-DF provides a way for various number of systems to create, manage and exchange the information about "things" in a standardized, understandable and universal way.

Possible structure of O-DF tree is shown on the figure 3.1. The tree always starts with "Objects" element as the root node. "Objects" element contains arbitrary number of "Object" sub-elements. Each "Object" is supposed to have at least one "id" sub-element as an identifier and optional "description" sub-element. Moreover, each "Object" may have properties that are declared by "InfoItem" sub-element. "InfoItem" element in its turn can have "MetaData" sub-objects and values that are the values from the context of "InfoItem" (i.e. temperature sensor values). "Meta-

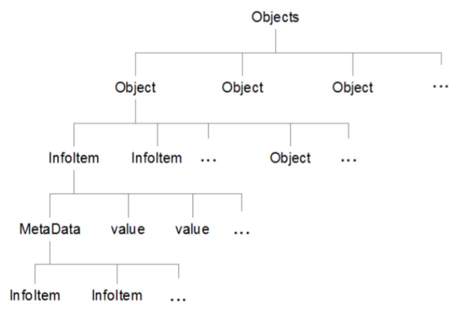


Figure 3.1: O-DF element hierarchy

Data" has the same structure as "InfoItem", but its semantic is different - it contains the description of "InfoItem" object. It can be useful in case the system tries to retrieve unknown "InfoItem". "Object" elements can also have sub-objects of the same "Object" type as itself. So the result is usually a tree that contains arbitrary number of levels with different objects from predefined set of types [47]. The example of a particular XML structure is shown on the figure 3.2.

Because O-DF format has a hierarchical nature by default, it's possible to query for particular elements using a technique called URL mapping. Using this technique, the client requesting information using special URL, including object's id or property name. For example the XML structure that is shown above can be accessed using "**wget <URL>/REST/Objects**" query. If the client wants to access only SmartHouse data or just get (or set) current humidity level in the kitchen it can be done using following commands:

```
wget <URL>/REST/Objects/SmartHouse
wget <URL>/REST/Objects/SmartHouse/Kitchen/Humidity
```

To sum up, O-DF was developed to provide a way to represent information entities about various objects - humans, services, devices etc. in a general way, independently from application or context. The transportation mechanism is not a part of O-DF standard. O-DF encoded information can be transmitted through the network by various low-level protocols or even copied to and from USB storage drive manually. However, the main purpose of O-DF was to use it with messaging interfaces like O-MI as a query and response format [47]. The description of O-MI is provided below.

3.1.2 O-MI

The Open Messaging Interface was created with the same purpose that HTTP was for the Internet. Using O-MI different devices and sensors can in-

```
<omi:omiEnvelope ttl="1.0" version="1.0"
    xmlns="odf.xsd"
    xmlns:omi="omi.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <omi:response>
      <omi:result msgformat="odf">
        <omi:return returnCode="200"></omi:return>
          <omi:msg>
            <Objects>
              <Object>
                <id>SmartHouse</id>
                <Object>
                  <id>Kitchen</id>
                  <InfoItem name="Humidity">
                    <value unixTime="1459348542"
                    dateTime="2016-03-30T17:35:42.413+03:00"
                    type="xs:double">
                    0.5663739287794303
                    </value>
                  </InfoItem>
                </Object>
                <Object>
                  <id>WorkHouse</id>
                  <InfoItem name="FrontDoor">
                    <value unixTime="1460970721"
                    dateTime="2016-04-18T12:12:01.063+03:00"
                    type="xs:double">
                    30.967197215931222
                    </value>
                  </InfoItem>
                </Object>
              </Object>
            </Objects>
          </omi:msg>
      </omi:result>
    </omi:response>
</omi:omiEnvelope>
```

Figure 3.2: Infoitem with multiple values

teract with each other. Since in IoT the "thing" can be almost anything O-MI was designed to be as general as possible. For product lifecycle applications O-MI is supposed to provide a way for communication between products and distributed information systems that consume and publish information on a real-time basis. Despite the context or specific use case, it should be possible to apply O-MI to the "lifecycles of anything" i.e. humans, services, projects, etc. [48]. O-MI can be used as a transportation mechanism for any payload, exchanging information between different O-MI Nodes across the network. The information can be encoded using most of the widely used formats such as XML, JSON and CSV. Since O-MI Nodes do not have predefined roles, the interaction is performed on "peer-to-peer" basis where every node can act both as a "server" and as a "client" with the other O-MI nodes or with other systems [48]. The key functional requirements reported below are based on the needs of real-life closed-loop lifecycle [49, 50, 51] management applications. As no existing standards could be identified that would fulfill those requirements without extensive modification or extensions, they original designer decided to standardize them to provide a more general solution for large-scale scalable IoT systems.

O-MI key functional requirements

1. **"Low-level" transportation ability.** O-MI must be able to utilize the usage of "low-level" protocols to transmit its messages. Protocols don't mean only network access protocols. Besides HTTP, SOAP, SMTP it must be possible to transmit messages by just copying them to and from USB stick or other storage media or just texting them using a mobile phone.

2. **Three main operations - read, write and cancel.** Read operation is used for immediate or deferred retrieval (also known as subscription). Write operation writes the data from "Things" into the system. Cancel operation is used to cancel subscriptions.

   Immediate Read: O-MI Nodes can ask about particular values of particular moment in time or just inquire for the latest ones and they have to get this information immediately.

   Deferred Read (Subscription): Is a type of read request with specified interval rate and (optional) callback URL. After subscription request is made, the requester will start receiving data according to the specified intervals. The data is sent to the callback URL. If no callback URL specified, the data can be "polled" by read request by specifying subscription ID (given in the response of the subscription request). Polling can be useful when using firewalls or NATs that do not allow response to reach the callback URL.

3. **Write operation availability:** O-MI Nodes might need to send any kind of information that became available to the other nodes and they have to be able to do it at anytime.

4. **Different payload formats:** Even though O-DF (XML) is the preferred payload, it must be allowed to use any text-based encoding format that can be embedded into XML message as a payload.

5. **Request and responses have to have TTL.** Time-to-Live parameter specifies the time for which the request is being valid to transmit, forward or reply. If it expires, the request has to be removed and error message returned.

6. **Synchronous communication between nodes.** In response it must be possible to specify the request without sending additional query. There must be also a possibility for clients to initiate the connection/communication to the nodes that are behind the firewall/NAT.

7. **Data sources, services and metadata discovery and publication.** Data providers must be able to publish their data using write operation. It must be also possible discover such data using simple HTTP queries (URL mapping can be used) or even search engines.

8. **List of target O-MI Nodes in request.** It must be possible to specify particular O-MI Nodes that have to receive, process and answer the request. Requesting nodes then are responsible for forwarding the request to the target nodes and error handling.

To sum up, O-MI is general mechanism of delivering messages between nodes. It can be used for different kinds of information like physical products, documents, repositories etc. The variety of operations is not limited to simple read/write, but also allows subscription with data delivery on demand, on change or on specified interval basis. The subscription concept is one of the core concepts of O-MI.

Chapter 4

Reference Implementation

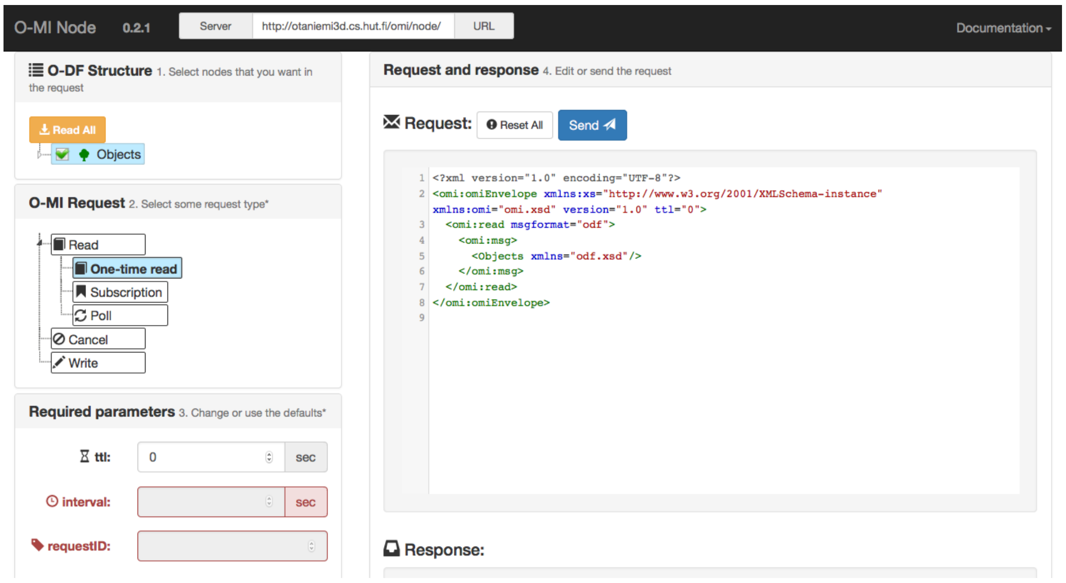
4.1 Introduction

In recent years, standard and technology adoption has often been motivated by the available resources, both in terms of documentation and reference implementation. It is imperative to associate standards with an implementation and a “sandbox environment” in which developers can understand and learn the standards specifications.

The reference implementation acts as some sort of executable documentation, with request and response examples covering essentially every aspect of the standards. In this way developers can jump straight into action in understanding what the protocols supposed to do in reality and not only on paper. The current reference implementation, developed at Aalto University – School of Science – Department of Computer Science, consists of several modules:

O-MI Node Server. The server implements all O-MI basic operations. It maintains a database where the information about O-DF data model, consisting of Object(s) and InfoItem, is stored. Currently the only underlying transport protocol supported is HTTP, but in the future there is a plan to support websockets as well. Therefore any O-MI operation is transported using an HTTP POST. In many regards the reference implementation behave like a normal REST endpoint, except for the subscription mechanism that allows interval or event based responses to a list of subscribers. Moreover, subscribers can specify a callback address (basically a URL different from one that has sent the original request) that will receive the messages during an active subscription session. Without the callback URL the data can be polled using read request with subscription ID. The current implementation also supports the "cancel" to stop and delete active subscriptions.

Webclient. This module provides a graphical interface for users and developers. The main purpose is to demonstrate is automate the creation of correct O-MI/O-DF messages, just by clicking instead of writing XML and HTTP queries by hand.


O-MI Node
0.2.1
Server
http://otaniemi3d.cs.hut.fi/omi/node/
URL
Documentation
O-DF Structure 1. Select nodes that you want in the request
Read All
Objects
O-MI Request 2. Select some request type*
Read
One-time read
Subscription
Poll
Cancel
Write
Required parameters 3. Change or use the defaults*
ttl:
0
sec
interval:
sec
requestID:
Request and response 4. Edit or send the request
Request:
Reset All
Send
Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xmlns:omi="omi.xsd" version="1.0" ttl="0">
  <omi:read msgformat="odf">
    <omi:msg>
      <Objects xmlns="odf.xsd"/>
    </omi:msg>
  </omi:read>
</omi:omiEnvelope>
```

Figure 4.1: Illustration of O-MI Node Webclient UI.

The User Interface depicted on Figure 4.1 (full resolution image can be found in Appendix A) has been designed to guide the user like a numbered step-by-step tutorial. The main interaction happens of left panel, with numbered subsections. In the first one, by clicking on the ReadAll button, the webclient forwards to the server a request to retrieve all objects. An example of an O-DF tree is depicted on Figure 4.2.

The objects can have an arbitrary number of sub-objects or InfoItems (e.g. BackDoor is an "InfoItem", BedRoom is a sub-object). Users can choose the item(s) they are interested in by just clicking on them.

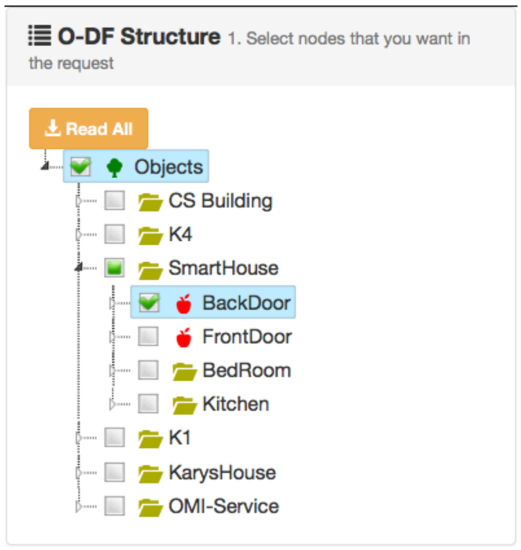


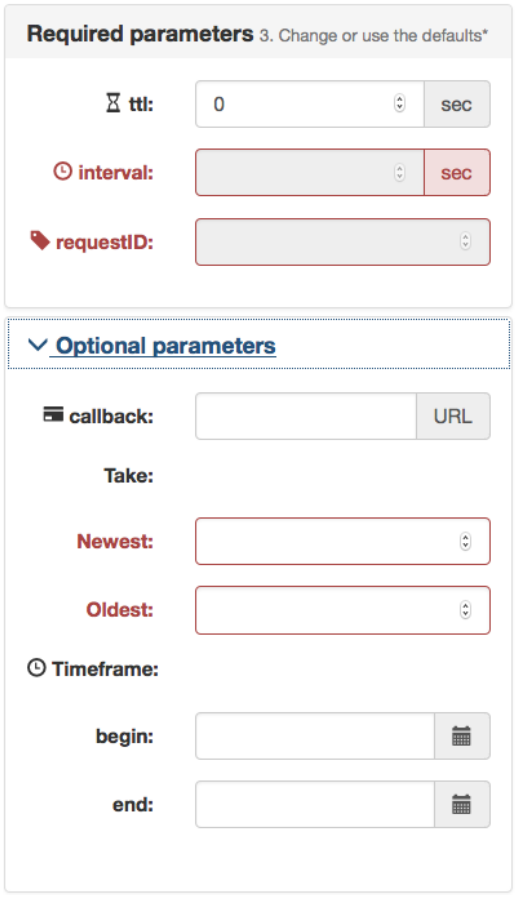
Figure 4.2: O-DF Structure.

When the user toggle a flag on a specific item, the webclient takes care to translate it into the corresponding XML request, including all the necessary parameters according to the selected type of the query. The type can be chosen on the next panel called O-MI Request (see Figure 4.1). When the type is chosen the user can fill the required and optional parameters in the special fields. The number of available inputs depends on the type of the query was chosen earlier. The fully expanded UI for parameters is shown on Figure 4.3. The TTL parameter specified the time for which a request remains valid (time for processing and reply). The Interval, requestID and callback URL parameters are used only for subscription. It is also possible to specify the "n" newest or oldest values or specify particular timeframe from which the values will be queried. All these user selections are automatically translated by the webclient on the corresponding XML, which is available for inspection of the right panel of the UI. The raw XML is also user editable; in this way developers/advanced user can further understand how the protocol behave and what a correct O-MI/O-DF request looks like.

It is possible to submit the generated request to the server simply by clicking on the Send button. The reply from the server is also available for inspection, but of course in this case is not user editable.

Agents. The agent subsystem provides a mechanism to interact programmatically with the core of an O-MI node. In general an agent is a separate worker thread that fetches data from specific data sources using specific protocols and transform them into objects that are understandable by the core. In general agents are used as an intermediary between the lower level hardware (sensors/actuators/any IoT device) and the O-MI node. For example, an agent can be a driver for a Z-wave or Plug-wise plug on/off switch.

The current implementation defines two types of agents: internal and external. Internal agents are implemented as Scala Actors [52]. They run on the same Java Virtual Machine as the O-MI Node Server and they are dynamically loaded at runtime. While external agents run as independent processes from the O-MI Node, and they can be located either on the same machine or different machine. The interaction between external agents and the O-MI node use a plain TCP socket for the communication. One advantage of using external agents is that they can be implemented in any language that supports a TCP/IP library. The security requirements that the different type of agents must implement are described in detail in chapter 5.



Required parameters 3. Change or use the defaults*

ttl: 0 sec

interval: sec

requestID:

Optional parameters

callback: URL

Take:

Newest:

Oldest:

Timeframe:

begin:

end:

Figure 4.3: Additional request parameters.

4.2 Security model for the reference implementation

The current O-MI Reference Implementation does not include any security mechanism. There used to be a very primitive IP whitelist from which it was possible to perform the O-MI write operation while reading was allowed to anyone. This can be hardly called a security and access control model, as it is not possible to specify any roles and associated policies for the users and groups. Configuring ".htaccess" file on a (Apache) web server running the node, solve this issue only partially. One of the problems with this type of authentication is that it is solely based on passwords that are stored on the server (.htpasswd files in Apache) and essentially is useful to define directory access policies. Mapping directory policies with O-DF objects (in our case – Objects and InfoItems from O-DF Tree) is theoretically possible but is more like a patch then a proper solution. Besides being web-server dependent, another drawback of this solution is the fact that is impossible to distinguish between different types of O-MI requests (read/write/delete).

Therefore it has been decided that a dedicated security module has to be developed from scratch to enforce access management policies based on the O-MI verbs and the O-DF data model. The next chapter will provide an in-depth description of the requirements and the development process of the access control and authentication modules developed during this thesis work.

Chapter 5

Security module

5.1 Introduction

For any real usage of the O-MI node reference implementation authentication and access control are fundamental. O-MI node administrators must be able to specify the roles and the permissions for the every O-MI operation and O-DF Object(s) and InfoItem(s). This chapter first introduces a discussion regarding security requirements of different use cases in different network configurations. After that it describes in detail the design and implementation process of the authentication and access control module.

5.2 Different levels of security requirements for O-MI Nodes

In IoT, confidentiality of the information and users' privacy depend on a variety of aspects, such as system configurations, usage scenarios, etc. As suggested in [53], the following paragraphs will classify security requirements according to two main perspectives: 1) network configuration perspective, 2) use case examples.

5.2.1 Security requirements based on network configuration

There are mainly four different scenarios that depend on network configuration or more precisely on the location (in term on belonging to the same or different network) of agents and clients communicating with a given O-MI node.

- **Local machine.** The agent and/or the client are located on the same machine with the server. In this case they can be considered trusted-software, as an administrator allowed their installation on the server, therefore and advanced security mechanisms such as query encryption or server-client certificates can become an unnecessary processing burden. In this scenario the security mainly concern the protection of the server from the external treats.

- **Same subnet, different machines.** In this scenario agents and/or clients are located in the same subnet as the O-MI node server, but on the different machine. Because they are on the same subnet, the server can interact with the other party directly using its internal IP address without a need of going through gateways and other intermediate nodes. In this case the network is like a closed community, with higher level of trust compared to any other node on the Internet. However this does not exclude potential attacks (e.g. a Node in the network has been compromise and start to send confidential data to 3rd parties). Therefore, in this scenario the main security threats will come from the external attacks or if an attacker has physical access to a node in the network.

- **Same network, different subnet.** Typical example of such network structure is a University, where different departments have different subnets, but they are all still connected to one main University network, getting access to shared resources by forwarding queries through gateways and other internal proxy nodes. Again, if the network is isolated from the Internet, then the security requirements from previous paragraph are applied. But in addition to that, the nodes of one subnet don't know much about the nodes from the other subnet, they usually interact through some gateway that might implement certain security restrictions. Potential attackers could override some of these policies, sniffing and spoofing data transferred between nodes of different subnets. In this scenario the authentication and encryption of traffic might be necessary if any of nodes of the set of subnet cannot be fully trusted.

- **Different network (Internet).** In this scenario no node can be trusted and the maximum level of security must be implemented. Anyone can be seen as a potential attacker, therefore any agent/client must be authenticated by the server and the communication must be encrypted.

5.2.2 Security requirements based on product use cases

Besides the network configuration the security requirements often depend on the specific situation or use case. This perspective mostly involves user's view on the situation and consists of his/her personal requirements and concerns. There are several levels of security, outlined here according to the [53]

- **No security requirements.** There are several scenarios where querying and updating of the information does not require any security requirements. For example a smart fridge might provide information about the expiration date of different foods it contains. At first sight, this might seems as not-so-sensitive information, of course a user might be not want that other people knows what he is eating but it does not have any security implication. However, being IoT bonded with real life objects, it is possible to infer that maybe there are no item in fridge, because the house is empty, which is clearly a sensitive information. In general lower security measures increase the usability and potential of any IT system, however in IoT understating if an information is truly sensitive or not might be not so straightforward. Therefore in IoT no-security requirements is a scenario that can rarely applied.

- **Data encryption and integrity check is necessary.** Encryption is necessary, whenever an attacker can gain access to the data exchanged over a given communication channel. Encryption includes integrity check (but not vice versa), so the information cannot be read by a man-in-the-middle and also cannot be tampered without notice. In the World Wide Web, data encryption and integrity check are widely supported via HTTPS. This protocol is recommended and supported by pretty much all the major web-browsers or other user agents. Companies exposing their services though a given server under a given domain simply need to purchase a signed security certificate and maintain it up to date.

- **Authentication of all interacting parties is necessary.** If HTTPS is used for encryption and integrity check and if the server has a valid signed certificate, then client is aware of server's true identity. In this scenario only one of two parties is actually authenticated, however many scenarios require that both parties are authenticated. In the Internet this is usually accomplished using password that the client has to enter to demonstrate its identity over an encrypted channel. Another option is to use client side certificates as authentication mechanism: HTTPS supports this feature out of the box and in general it is safer than easy-to-guess password. However, one of the drawbacks of this solution is that it increases the complexity of the security mechanism, requiring special web-server configuration changes that for some enterprises might be complicated to impose. In addition, even though it is much easier to guess a password than a certificate, the distribution of such certificates and their protection on the client is at risk. Last but not least is usability, most users might not be familiar with client side certificates and when their browser pops up the information box regarding accepting this kind of authentication method they might simply decline or ignore it.

  To sum up, this level of security is more demanding than the previous one encompassing the authentication of all participants in the communication over an encrypted channel.

- **User-based access control is necessary.** After the user is authenticated using one of the methods described above, it is often necessary to assign them specific right over specific data objects. Similarly to what happens in any modern operating system, where a user is allowed to perform only a given set of operations of given files or directories, the same mechanism can be implemented for the resources of a network endpoint.

The last level of security described above is lacking in the current O-MI reference implementation. Therefore, it has been decided that a flexible module to manage users/groups and their access levels has to be designed and implemented. The following section states the requirements for such module.

5.3 Modules requirements

One of the main requirements for the security module was to impact as little as possible the current core implementation while providing the desired functionalities. Therefore it has been decided to create a separate self-contained module which can plugged into the existing implementation. Beside this architectural choice, there is a list of requirements to fulfill:

- **Prevent unauthorized access to the resources.** Only users who have required access can perform given operation over certain data objects.

- **Group based rules.** All users must belong to a group. Access rules are set for groups, never for single users. Every user can be a member of one or many groups.

- **Differentiate permissions according to the O-MI verbs.** Essentially this requirement can be translated into the well-known distinction between simple read and read-write permissions.

- **Minimize server-side account maintenance.** Users must be able to register and login to the service minimizing the maintenance of such account on the server. For this reason, OAuth2 (Facebook login in particular) authentication model was chosen.

- **Recursive permissions mechanism.** Permissions can be inherited from parent object (same as in modern file systems) as well as overridden for particular children.

- **Customizable default permissions.** There should be a way to set the set of rules that apply for every user by default.

- **Universal authentication mechanism.** Target user is not necessarily a human (e.g. Things/Machine). The authentication mechanism has to be able to treat all users of the system in a universal way and should work in the same way if the user is human being using web-browser or a device.

- **Rules management interface.** The system administrator must be able to control access policies through a centralized interface. The interface must implement the following features:

  - View the whole list of groups on the O-MI Node
  - View the list of users for every group
  - View the Access Control Tree for every group
  - Modify the Access Control Tree for every group
  - Modify group information such as name and list of members
  - Add groups
  - Delete groups
  - Register new users

It is quite clear how that requirements are pretty common in every security mechanism. Therefore, it has been decided to mimic as much as possible well-known security models such as the one implemented by the Unix File System. Its simplicity and effectiveness remains to be matched.

5.4 General design decisions

Based on the above listed set of requirements, it has been possible to select certain technologies, design an overall architecture and define the features that needed to be implemented. The main design decisions are summarized in the following list:

- **Two main submodules.** The security module consists of two main parts: Registration/Authentication submodule and the Access Control submodule. The first one is responsible for handling the registration of new users and their information. It will also manage the authentication process and session handling. The latter module consists of two essential parts: administrator console and access control middleware. The first one is a tool for system administrators to manage user groups and policies. The second one processes and authorizes requests made by users.

- **Separate Database.** To satisfy the requirement of changing as little as possible the existing core implementation, it has been decided to manage users/groups and the related policies in a separate database. Obviously this choice has negative effect in terms of memory and overall performance, but it ensures code modularity and drastically simplifies code management.

- **Servlet based.** The module has been written using Java using Servlet technology, this choice is mostly due to the existing knowledge and familiarly with this particular framework.

- **Facebook as OAuth2 service provider.** The feature for registration/authentication module that will support the login to the service using Facebook credentials.

- **Certificates Extension.** Since the "Things” does not have dedicated Facebook profile, these devices use client side certificates containing the necessary credentials verifiable by the server.

Database Schema At the core of the security module there is database, which is used to store and manage users, groups and the associated access policies. The Entity-Relationship diagram depicting the schema of the security module database is illustrated in figure 5.1. (Full resolution can be found in Appendix B).

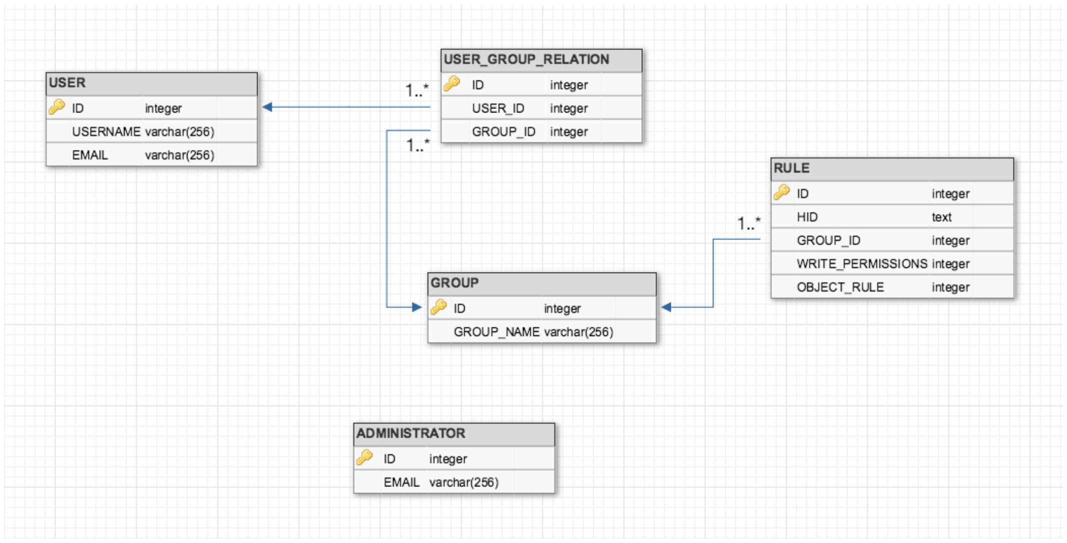


Figure 5.1: Entity-Relationship diagram of the security module database schema.

The database contains 3 main tables: **user**, **group** and **rule**. When registering new users, the module obtains username and email only. Users belong to a "default group” after their registration. The default behavior and access policy for the "default group” can be customized by the O-MI node administrator. The **group** table contains the list of groups. Users and groups have many-to-many relationship that is implemented using a helper table named USER_GROUP_RELATION. Access rules are stored in the **rule** table, is arguably the most interesting for the implementation of the security module, as it maintains the association between a group, a data object (*hid*), and which kind of operation is allowed to perform (Read/ReadWrite). The meaning of the *hid* in the rule table, might be not self-evident to the casual reader, therefore it deserves an explanation.

The *hierarchy_id* is a specific path in the O-DF XML. Figure 5.2 provides an example of it: InfoItem with name equals to *"Temperature"*, has *hierarchy_-id* equals to *"Objects/Kitchen/Temperature"*. Basically it's a path from the Root element (Objects) to the current element including all intermediate parent nodes.

```
<Objects>
   <Object>
        <id>Kitchen</id>
            <InfoItem name="Temperature">
                <value>5</value>
            </InfoItem>
    </Object>
</Objects>
```

Figure 5.2: XML example for hierarchy ID

The *Group id* column of the rule table is related to the *id* property in GROUP table. The *Write_Permissions* column is a Boolean flag (0=R (read) or 1=RW (read+write)). Finally the *Object_Rule* column indicates if the rule is applied to an Object or an InfoItem. The InfoItem is a type of node that contains data (e.g. Temperature Sensor Reading), while an Object (e.g. Temperature Sensor), from a security perspective is similar to a folder in file system, it might contains other objects and/or InfoItems. The rules for a given Object must also apply recursively to the Objects and InfoItems it contains. Like in a normal file system, an Administrator can also break rule inheritance and manually override the rules for a particular node. Finally the *Administrator* table is a helper table that maintains a list of user with admin rights and it is filled manually by the server administrators.

5.5 Interaction principles

Given the decision of developing the security module as a separate plug-in for the O-MI reference implementation, the interaction between these two software has to be planned: Two main interaction types were identified 1) user registration 2) authentication and access control granting.

5.5.1 User registration or authentication scenario

When a user lands for the first time in an O-MI webclient window (or in case his/her session has expired), the system redirects the user to perform an authentication process. Because OAuth2 has been used for reasons explained above, the credentials are obtained from a 3rd party website without the need for the user to type them manually. The interaction scheme is shown in the Figure 5.3.

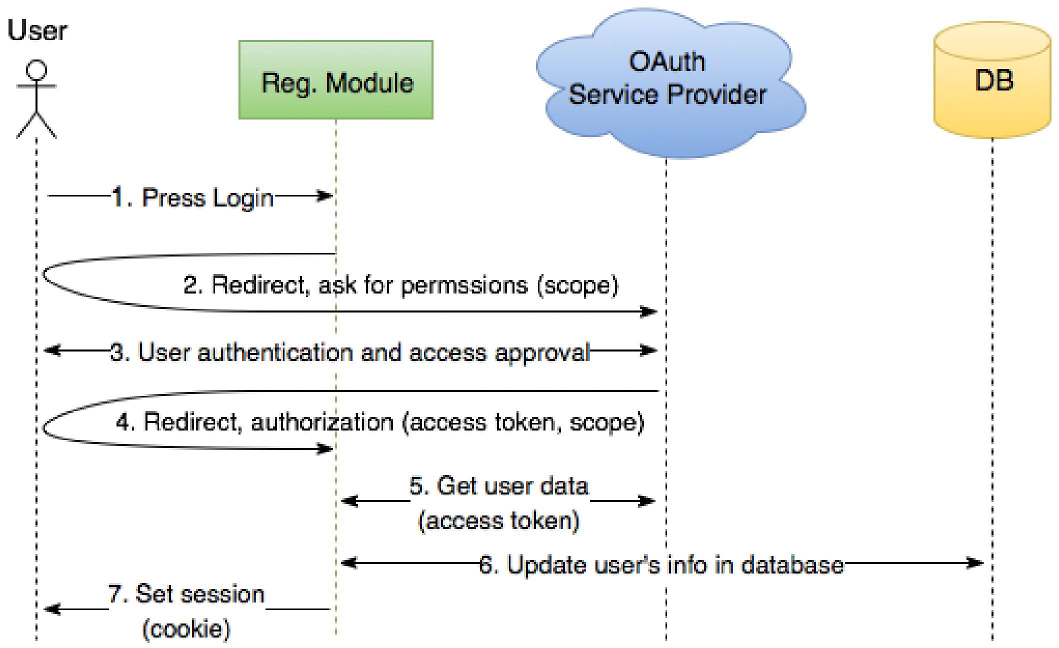

User
Reg. Module
OAuth Service Provider
DB
1. Press Login
2. Redirect, ask for permssions (scope)
3. User authentication and access approval
4. Redirect, authorization (access token, scope)
5. Get user data (access token)
6. Update user's info in database
7. Set session (cookie)

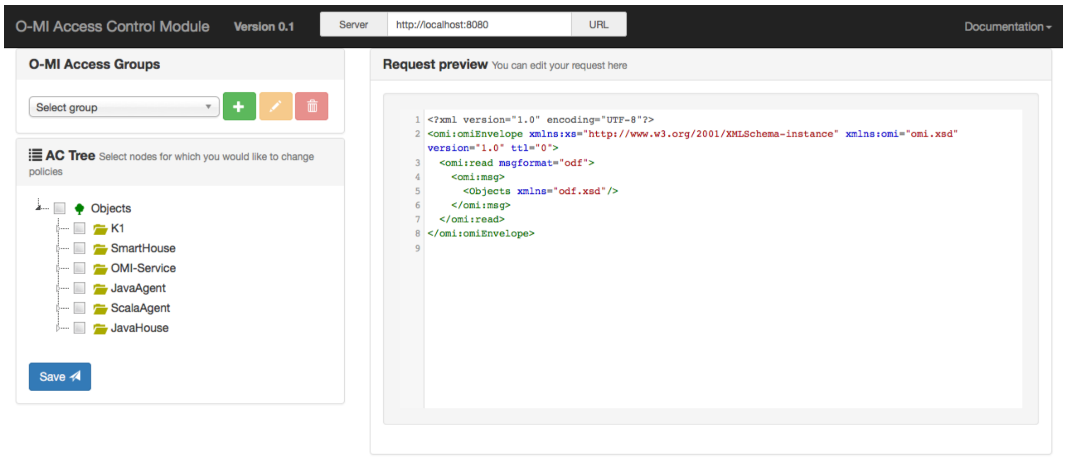
Figure 5.3: User registration/authentication interaction map.

In this scenario the user interacts with the access control (AC) module through the web-browser. On the login webpage the user selects the registration using an OAuth provider (e.g. Facebook). AC Module redirects the user to the service provider's webpage where he/she is asked for a permission of using his personal data. After the user agrees, the service provider redirects him/her back together with the access token, which is forwarded to the AC Module. After that the module is able to get user's personal data by making HTTP queries to the service provider calling vendor-specific APIs (in this case Facebook). After getting the data, the AC Module checks if the user is already registered; if not, user's info are stored in the database presented in the previous paragraph. Then if no errors occur the module sets the session cookies in the user's browser. Finally the user is authenticated and is able to perform queries to the O-MI Node Server and using the webclient.

5.5.2 Access control

Once a user is registered to the system, it is possible to associate this account to particular groups restricting/granting the access to particular data objects. For this purpose a dedicated user interface, called the access management tool, has been developed (see Figure 5.4, additional images of UI can be found in Appendix C). This user interface interacts with the access control module in the backend, which essentially manages and stores access rules on the database described above.

Access management tool: The user interface of this tool partially resembles the O-MI Node, extending its functionality. This tool is supposed to be used by the administrative staff in charge of the O-MI node. In the user interface, administrators can manage groups and users and set special rules for them. They can create, modify or delete groups and add or remove particular users to given group(s). Mimicking the same user interface used in the reference implementation webclient, it is possible to retrieve all the objects available in



```xml
<?xml version="1.0" encoding="UTF-8"?>
<omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd"
version="1.0" ttl="0">
  <omi:read msgformat="odf">
    <omi:msg>
      <Objects xmlns="odf.xsd"/>
    </omi:msg>
  </omi:read>
</omi:omiEnvelope>
```

Figure 5.4: Access control module User Interface.

the O-MI Node (ReadAll operation) and set access policies for every node of the O-DF tree. The policies are simple [no-access/read/read-write] flag, which are associated with every node in O-DF tree. These rules are forwarded (by pressing the "Save" button) to AC module on the backend which is in charge of storing them in the database. When a certain group is selected from the dropdown list (see Figure 5.4), the system automatically loads the rules for that group from the database and shows them in the tree.

Backend: The access control backend, besides storing the configuration set in access management tool in the database, has the fundamental function of interacting with the O-MI node core. The interaction is extremely simple: essentially the O-MI node asks: "Is the access to the resource X for the user Y and request type Z allowed?", and the module replies "Yes or No" based on rules that have been set by the administrators. The module's interaction scheme is shown on the Figure 5.5.

This scenario usually starts after the authentication process is completed and the user has a valid session cookie. After being authenticated, the user can start to interact with the O-MI Node Server via the webclient.

When the user sends a request (e.g. read request for a particular object or set of objects) to the O-MI a session cookie is also forwarded. The O-MI Node

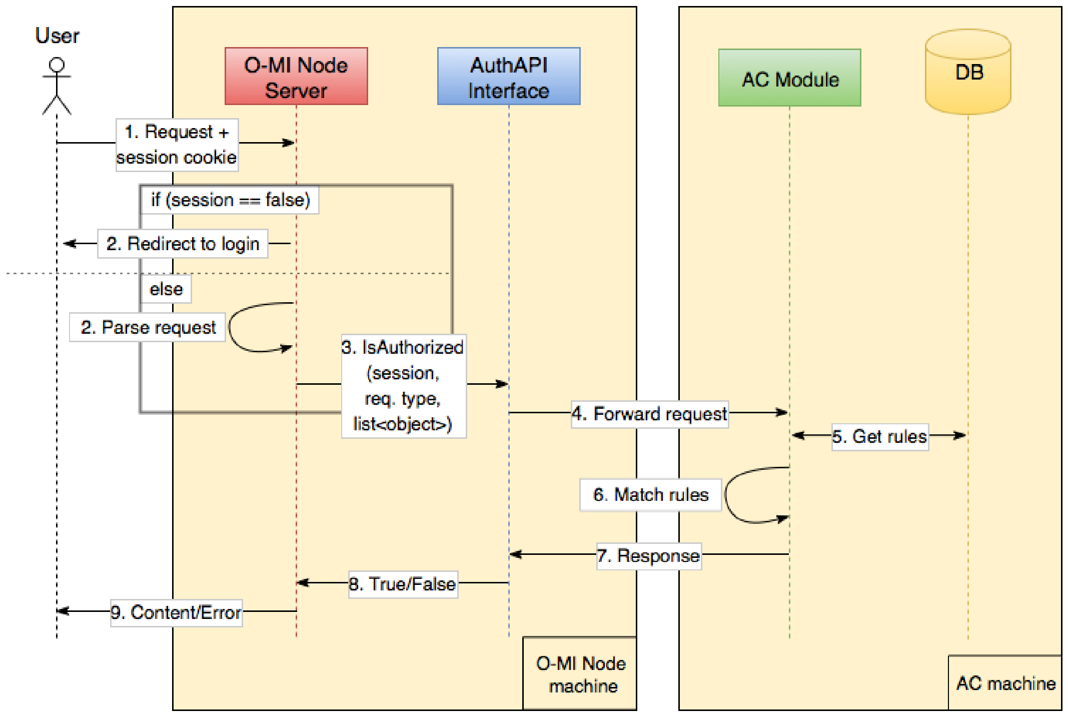


Figure 5.5: Access control interaction map.

receives and parses the request, it then invokes one of the AuthAPI methods, (which is part of the O-MI core implementation), passing as parameters the list of objects, described in term of *hierarchy id*, that user is trying to access and the user session. The invoked method forwards the request to the AC Module, by performing an HTTP POST to the service running in localhost. The AC Module service selects the appropriate rules from its database, matching them with the request permissions. If the user has appropriate access right on every item of the *hierarchy_id* list, the service replies True back to Auth API. In case 1 or more item violate access rules stored in the database, the service replies False scrapping the entire request. Once the Auth API has received the True/False answer from the AC service, the O-MI Node finally replies to the user with either the requested data or an access denied error.

5.6 Implementation

In the previous paragraph the overall design of the module has been presented, while this paragraph focuses on the technologies and implementation details of the developed module. Despite the fact that the O-MI Node Server is written in Scala it was decided to implement security module in Java. This decision is mostly due to the familiarly with the language and the available server-side framework. In addition it is worth to notice that it is possible to execute Java code from Scala applications as they are both compiled as byte-code for the same Java Virtual Machine. Therefore there was no real concern if a tighter integration between the modules is required.

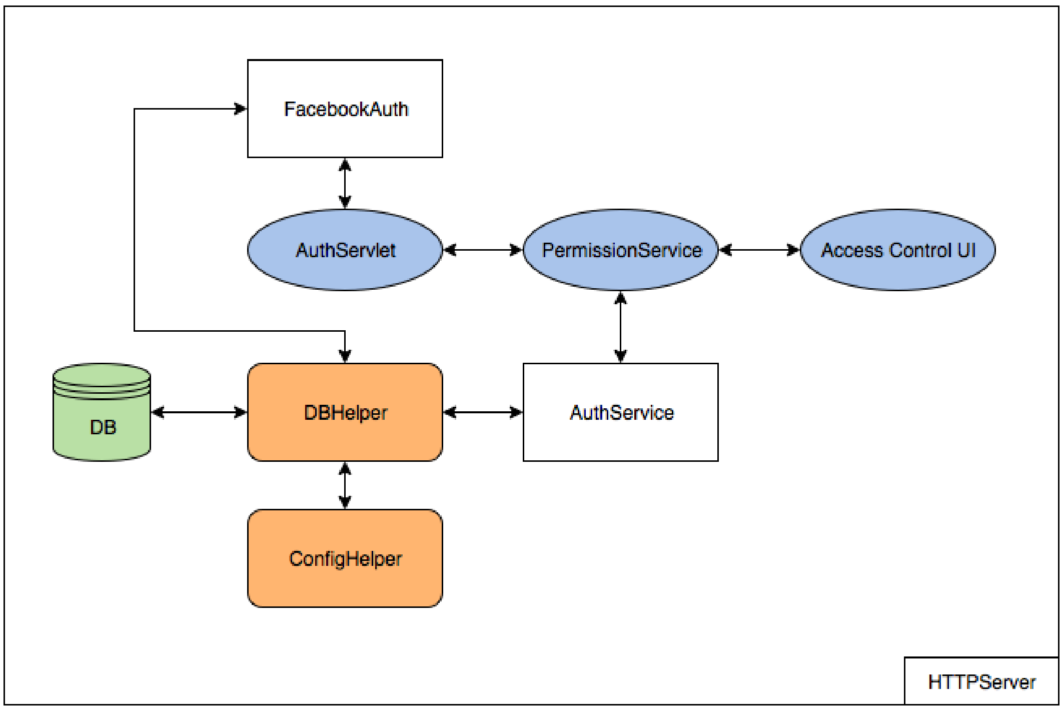


Figure 5.6: Module class diagram.

Since the module has been designed to be standalone, an embedded Jetty Servlet container was used to implement the communication (http) with the O-MI Node.

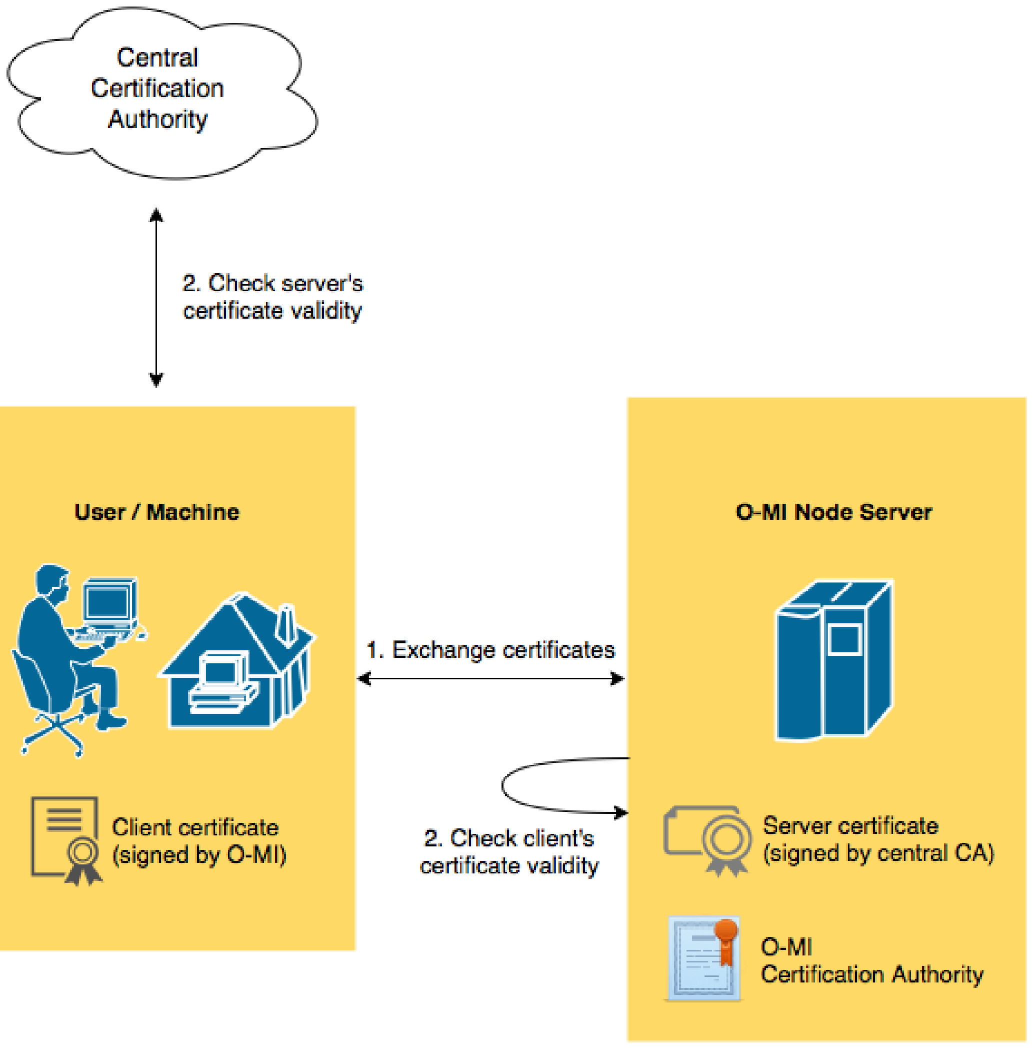
Figure 5.6 depicts the overall code organization. The blue ovals represent the servlets: the AuthServlet is an authentication servlet that handles user authentication and sets up a session. The PermissionService is the core servlet of the module. It is responsible for the majority of functions implemented by the AC module; such as: the backend service for access management UI tool and enforcing access control on behalf of the O-MI Node.

Additional classes include: DBHelper which is a wrapper class for managing table structure and entities in for the SQLite database. The ConfigHelper class contains basic configuration parameters, such as the database name and server URL. The AuthService is an intermediary class responsible for writing object permissions from the O-DF tree structure (using the access management UI tool) into the database. Finally, the last component interacting with the AC module is the AuthAPI. Essentially it is an external class, which is now included in the O-MI Node implementation, providing an abstracted and uniform way to perform authentication and authorization, hiding the implementation details regarding how this functions are performed. This allows future updates to the AC module which will be completely transparent to the O-MI Node.

5.7 SSL certificates extension

OAuth is the ideal choice when the entity interacting with the system is a human being utilizing a web-browser. If the "user-agent" is something else than a web-browser, such as an IoT device running its own firmware, the OAuth sessions mechanism is not a suitable method.

One of the real-world uses in which the O-MI Node and AC Module has been applied for testing was a smart home installation. Essentially it consists of a number of different sensors connected together to central gateway. The gateway connects the house with the internet and the ISP (Internet Service Provider) assigns a dynamic IP which might change over time. The usage of dynamic IPs is very common and that was one of the reason why the previous authorization method based on IP whitelist was abandoned. However, in this scenario also OAuth won't work because the user agent is not a browser and it does not make sense to authorize a physical device (the home gateway) using a Facebook account. To address this issue, it has been decided to use client SSL certificates.



Central Certification Authority
2. Check server's certificate validity
User / Machine
Client certificate (signed by O-MI)
1. Exchange certificates
2. Check client's certificate validity
O-MI Node Server
Server certificate (signed by central CA)
O-MI Certification Authority

Figure 5.7: Certificates exchange diagram.

Normally, when using HTTPS protocol, the server buys a certificate from an authorized Certificate Authority. When the client connects to the server through HTTPS it receives the server certificate and checks within the CA if the certificate is valid [54]. For the O-MI Node, mutual authentication is needed, meaning the gateway need to sure that it talks to the correct server, and the O-MI server needs to be sure that it is talking to the right client (See figure 5.7). In practice the O-MI Node creates a certificate and signs it (with the server private key). Network distribution of such certificate is also possible, if a trusted software is already running on the target machine, otherwise the certificate has to be physically installed in the device. At this point when the home gateway establishes a normal HTTPS connection with the O-MI Node, it sends its certificate to the O-MI Node. The O-MI node is able to verify (using its public key) that the received certificate has been signed by the server itself, and it can finally authenticate the device.

Obviously, the identity of the device and its access rights must be also configured beforehand using the AC management UI tool. In this case an invented e-mail address has been used as *"user-id"*, which is stored upon registration in AC module database. This e-mail address is one metadata of the client certificate, which is extracted by the O-MI Node and forwarded as *"user-id"* to the AC module that can finally check if the requests the device is performing comply with policies stored in the AC database. It is worth to notice that a real manufacturer could use the product serial number, instead of an email, as *"device/user-id"*, or even better a globally unique identifier, such as the ID@URI concept proposed by [55]. Ideally, the URI is the internet domain of the manufacturer (e.g. samsung.com) whose uniqueness is guaranteed by the DNS [Oat Systems & MIT Auto-ID Center, 2002], while the ID part can be only locally (inside the address space of the URI) unique identifier such as a product serial number or it a Global Trade Identification Number (GTIN).

As final note, it is worth to notice that the same authentication mechanism could be used also for human being; however even if the major web-browsers currently support client side certificate, the casual user is usually not aware of this authentication scheme and (at least for the moment) might confuse him/her which ultimately lead to leaving the website. Considering the fact that the main purpose of the O-MI reference implementation is to promote and divulgate the standards; it is paramount to attract as many users as possible, that's supporting well know authentication method like OAuth is still necessary.

Chapter 6

Conclusion

6.1 Summary of findings

The main focus of this research was to develop a suitable security model for the Open Messaging Interface (O-MI) and Open Data Format (O-DF) standards. In the first part of the thesis a general introduction to IoT was provided, highlighting its disruptive potential and the importance of overcoming current vertical information silos to realize a truly unified IoT. A set of requirements for achieving this IoT grand vision was discussed. Subsequently, several existing messaging standards were presented and compared with O-MI and O-DF. This comparative analysis highlighted how current standards address only partially the requirements necessary for achieving a unified IoT and identify O-MI and O-DF as one potential enabler for this goal.

The second part of the thesis describes in detail the core concepts behind O-MI and O-DF, together with their application in real-world usage scenarios. The set of characteristics that sets O-MI apart from existing standards (transport protocol independency, subscription, support of different payload formats) were also highlighted. After that, the description of the current O-MI reference implementation, and how it can be integrated into any arbitrary data provider/consumer application scenario, has been presented. Its main components, such as the O-MI Node Server, the webclient and agents system were also described and usage examples were provided. Next, the support for access control and authentication for the reference implementation was discussed. Since the existing mechanisms were limited to IP-whitelisting for given O-MI operation, it was decided to develop an external security plug-in module that could provide authentication and access control to the existing O-MI reference implementation.

The third and final part of the thesis describes the design and implementation choices of the module that was developed. First, the requirements and possible scenarios were presented. Three main functionalities needed to be addressed: authentication, access control and administration management. Based on the stated requirements, core design decisions were discussed, such as a platform/language selection, code architecture etc. Finally, the integration with the existing O-MI reference implementation was presented using a smart-home scenario as testbed.

6.2 Implications of the research

The thesis had two main objectives. The first one was to present O-MI and O-DF standards and show that they can help to enable seamless communication between heterogeneous devices in IoT, acting as aggregator nodes and ensuring interoperability at a system-to-system level. The second one was to develop an access control mechanism that could be plugged-in into the existing reference implementation. It was demonstrated how it is possible to set the rules for different data elements and semantic entities and apply the rules to the real IoT system in a general manner. Legal and privacy issues, related with the IoT data exchange, were not covered by this thesis.

6.3 Reliability and validity of the research

The security module developed can be only partially reused as such for the integration with others systems than the O-MI reference implementation. However, the requirements, the core design decisions and the code structure has been conceived to be generally applicable to other systems, providing a solid foundation for a further abstraction and generality of the used approach.

6.4 Future work

This research leads to the conclusion that the O-MI and O-DF standards represent a step towards a unified vision for the IoT, providing better integration capability compared to existing protocols, systems or standards.

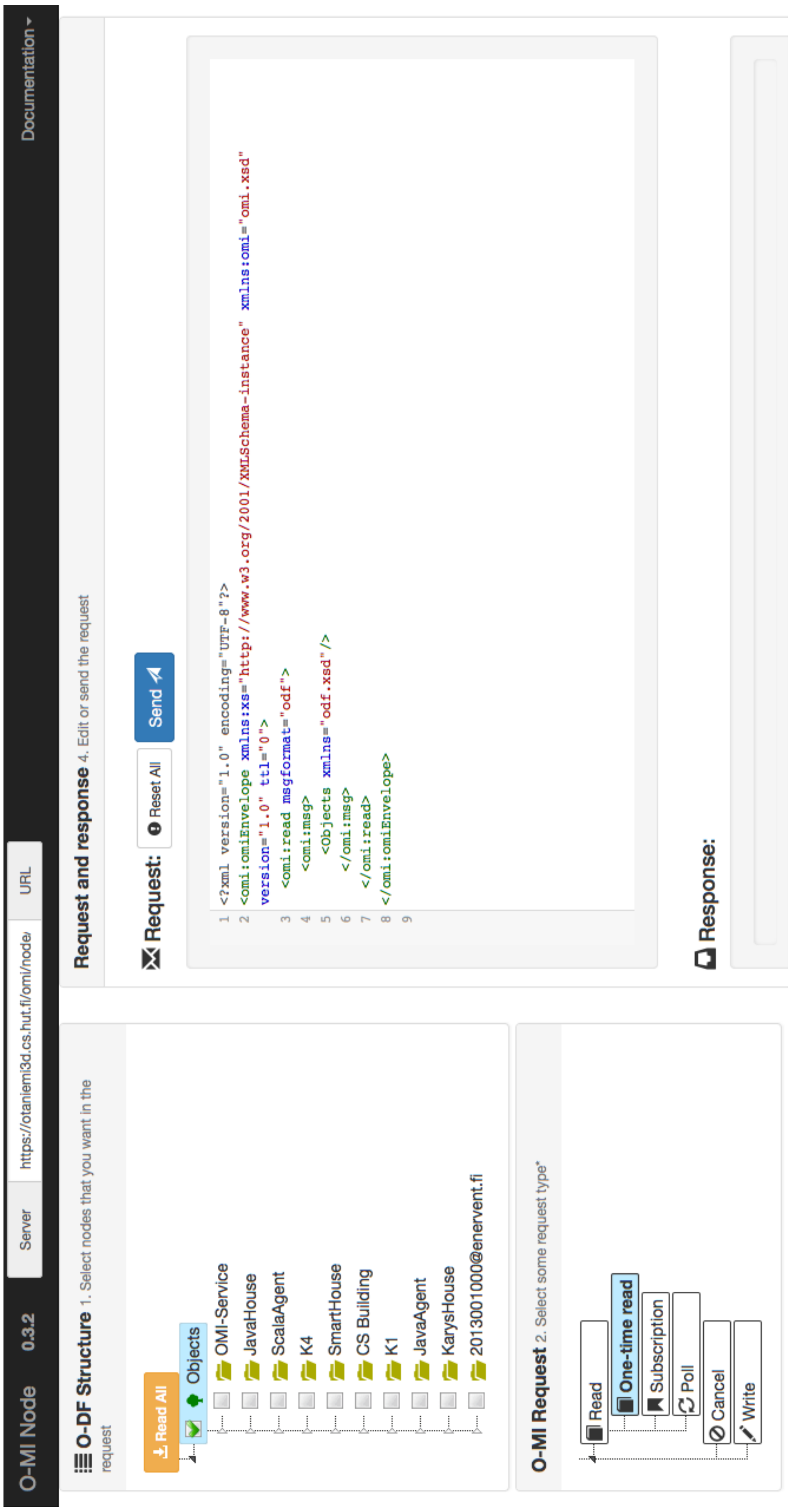
However, there are still some areas that need to be addressed. In the introduction it was argued how security is already one of major IoT concerns, for users, developers and other stakeholders. The O-MI and O-DF standards deliberately do not include any particular security model. However, it would be extremely useful to advice third parties, who might want to develop their own implementation of the standards, the recommended authentication and authorization model. The authentication model presented in this thesis is based on client side SSL certificates and it is truly applicable to pretty much any scenario. Certificates are a far superior authentication method than traditional device ID and password, however their safe distribution over the internet might pose additional challenges. However, it is worth noticing that if this authentication method will become the dominant one, a device manufacturer could embed the certificate directly into their product even before the device is sold to the public. Hardware manufacturers could also develop methods for making it impossible to alter or steal these certificates, even with physical access to the device. On the downside one common problem with certificates is that they have an expiration date, updating certificates manually on potentially millions of device is definitely not an option. Of course an alternative is to make the client register itself by generating a private and public key pair. The public key is sent to the server as a part of certificate request. This request also contains additional information about the target user such as email, organization name and so on. The server then builds the certificate, signs it with the client's public key and sends back to the client that decrypts it. The connection between the server and the client is secure and using encryption mechanism (such as HTTPS). Clearly this would work if and only if the server can be sure that it is talking with the right client.

Currently there are plenty of options on how IoT authentication and authorization can be implemented. A single master thesis can barely scratch the surface of this vast topic, however this research has highlighted the importance of the convergence to one unified IoT technology stack. In conclusion this thesis reject the argument that security must be use case dependent, and advocates for a set of well-known shared technologies that every IoT installation should comply to.

Appendix A

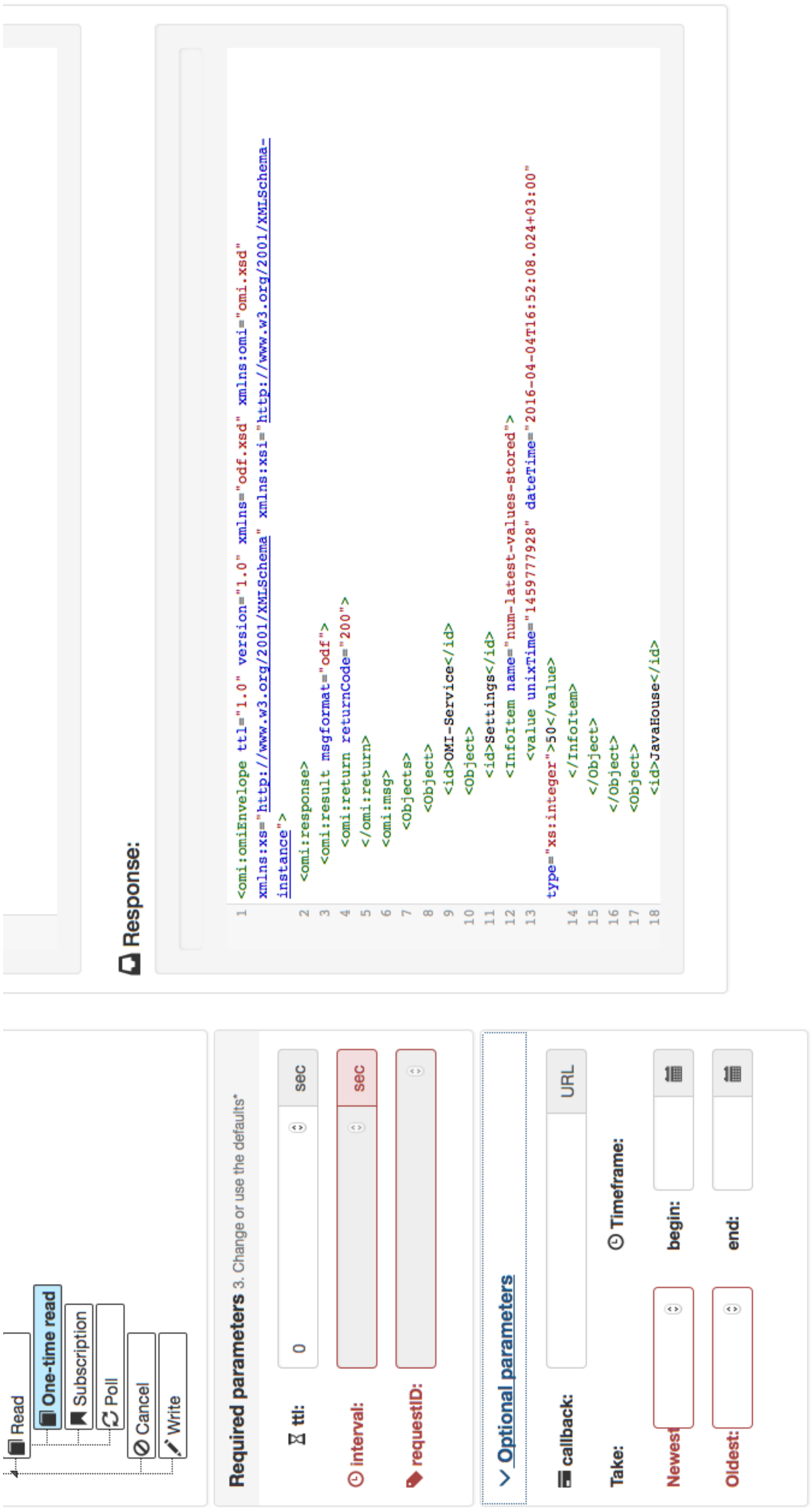
O-MI Webclient

This shows the user interface that was developed for O-MI Reference Implementation and already existed by the time when this thesis started. Main windows and parameters and the overall interface composition are presented.



```
<?xml version="1.0" encoding="UTF-8"?>
<omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd"
version="1.0" ttl="0">
  <omi:read msgformat="odf">
    <omi:msg>
      <Objects xmlns="odf.xsd"/>
    </omi:msg>
  </omi:read>
</omi:omiEnvelope>
```

Figure A.1: O-MI Webclient User Interface



```
<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <omi:response>
    <omi:result msgformat="odf">
      <omi:return returnCode="200">
      </omi:return>
      <omi:msg>
        <Objects>
          <Object>
            <id>OMI-Service</id>
            <Object>
              <id>Settings</id>
              <InfoItem name="num-latest-values-stored">
                <value unixTime="1459777928" dateTime="2016-04-04T16:52:08.024+03:00"
type="xs:integer">50</value>
              </InfoItem>
            </Object>
          </Object>
          <Object>
            <id>JavaHouse</id>
```

Figure A.2: O-MI Webclient User Interface

Appendix B

Security module database

This shows the scheme of database that was designed for the security module in Chapter 5.

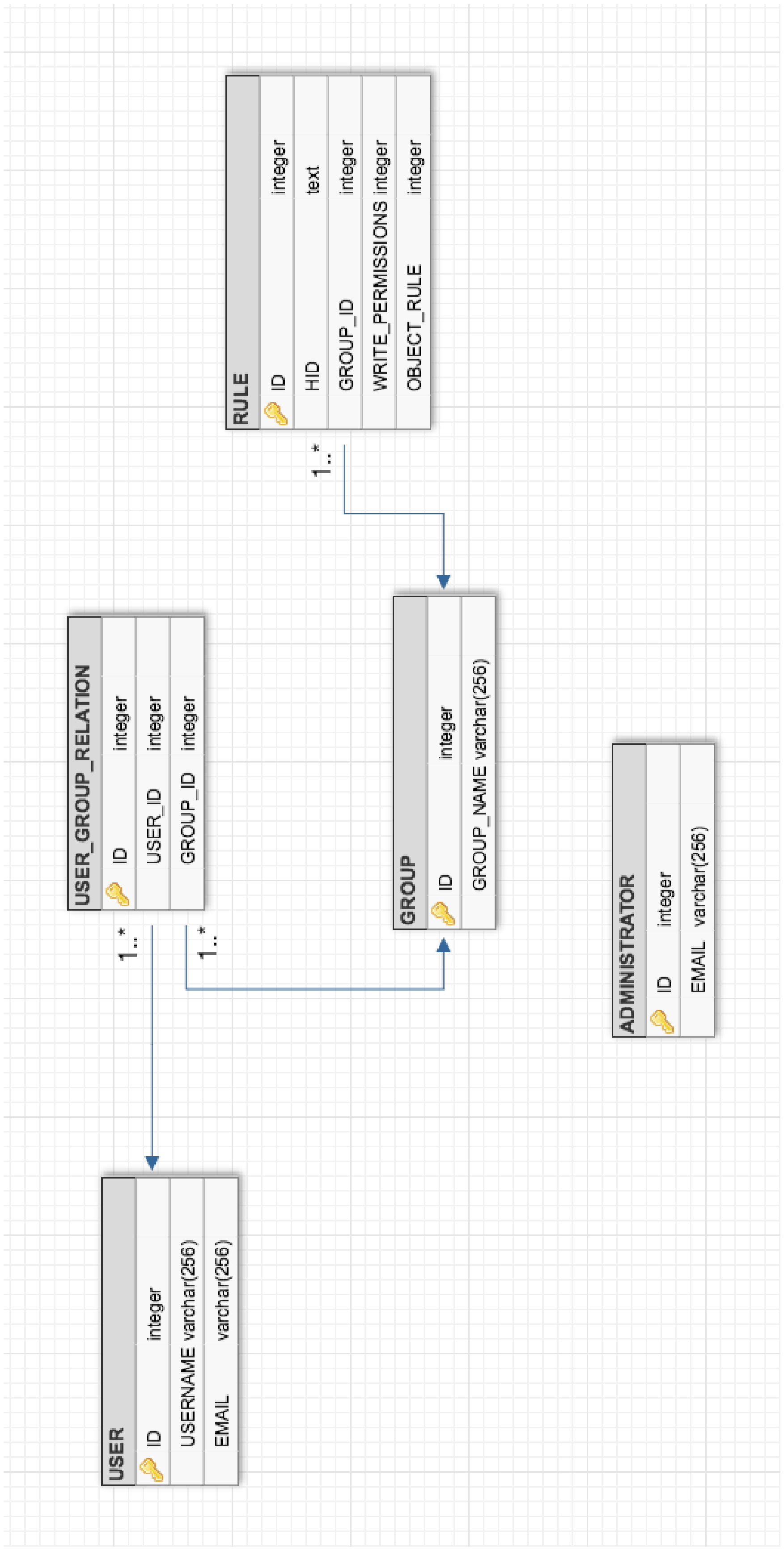
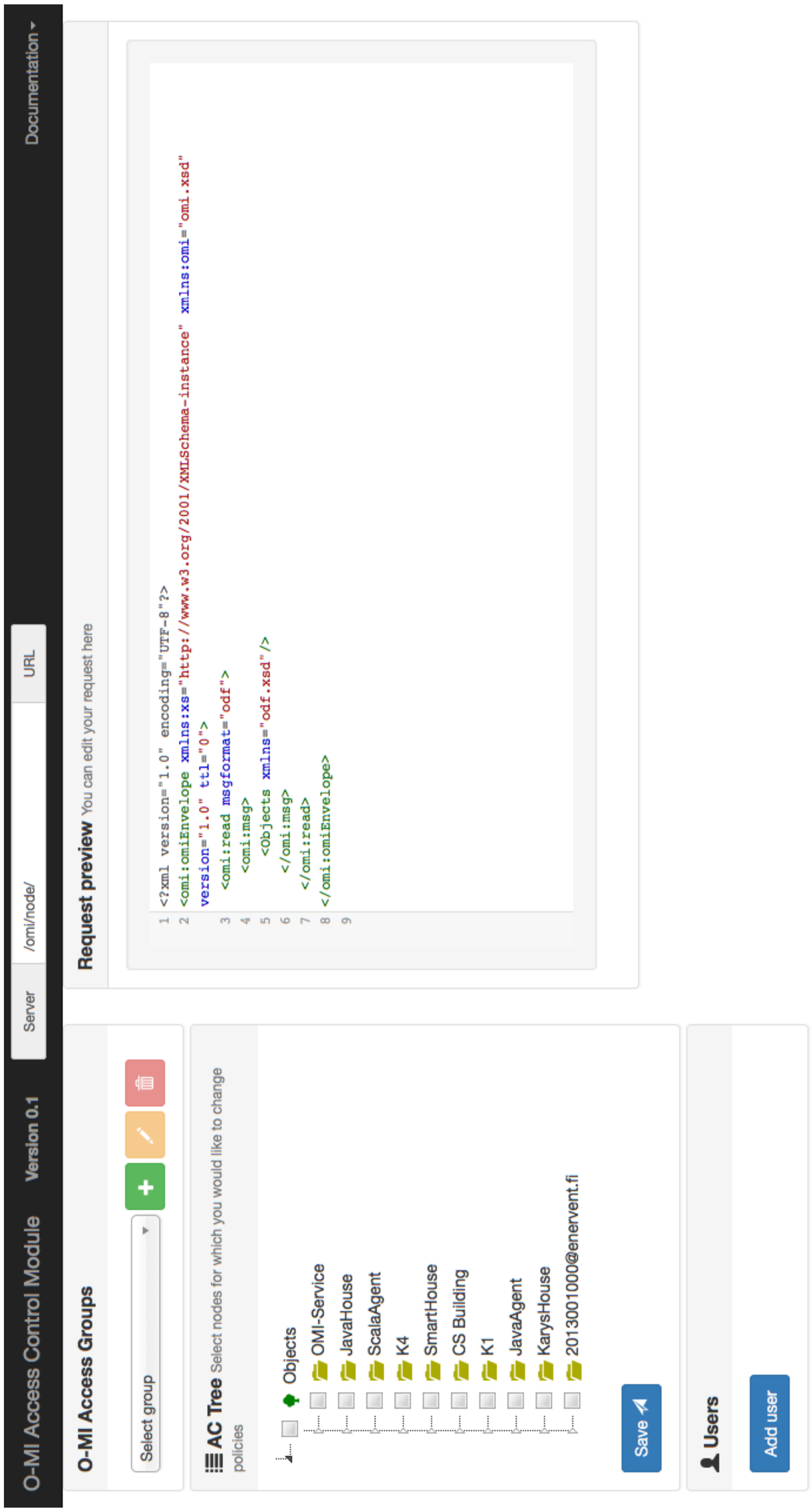


Figure B.1: Illustration of access control and authentication modules db

Appendix C

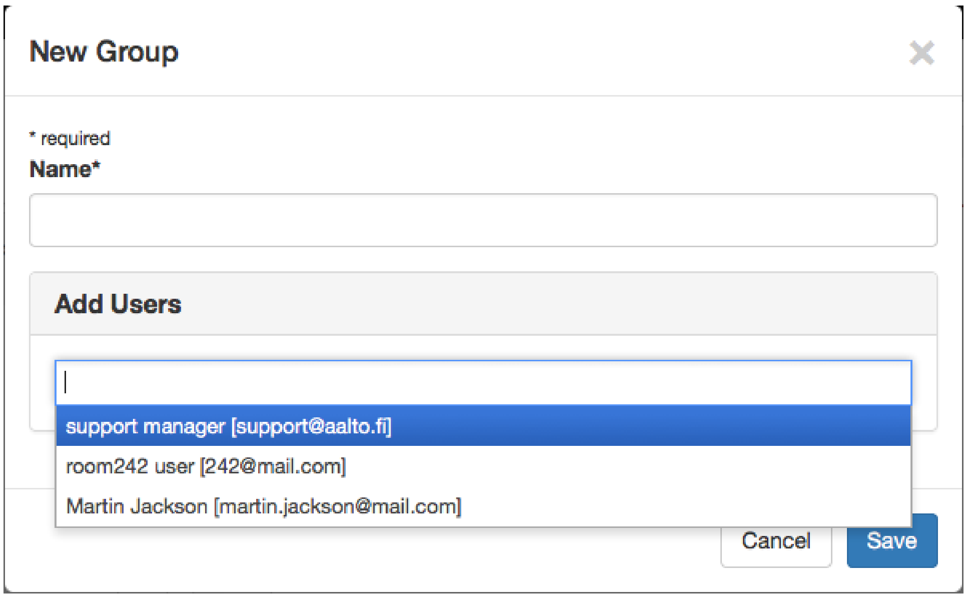
Security module admin panel

This appendix shows the interface for Administrators of the security module that was developed in this thesis. Main windows and parameters and the overall interface composition are presented.



```
<?xml version="1.0" encoding="UTF-8"?>
<omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd"
version="1.0" ttl="0">
  <omi:read msgformat="odf">
    <omi:msg>
      <Objects xmlns="odf.xsd"/>
    </omi:msg>
  </omi:read>
</omi:omiEnvelope>
```

Figure C.1: Security module admin panel User Interface


New Group

* required

Name*

Add Users

support manager [support@aalto.fi]

room242 user [242@mail.com]

Martin Jackson [martin.jackson@mail.com]

Cancel Save

Figure C.2: Create new user window

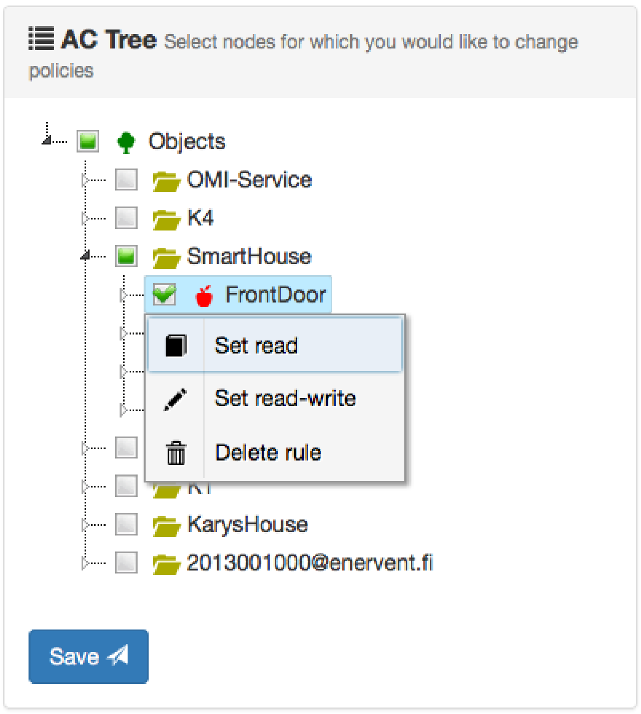

AC Tree Select nodes for which you would like to change policies

Figure C.3: Set permissions window

Bibliography

[1] P. Baran. Reliable digital communications using unreliable network repeater nodes. *RAND Corporation papers, document P-1995*, 1960.

[2] R. T. Braden. A server host system on the arpanet. *ACM New York, USA*, 1977.

[3] K. Hafner. Where wizards stay up late: The origins of the internet. *Journal of Engineering Education, Simon & Schuster*, 1998.

[4] R. Hauben. From the arpanet to the internet. *A Study of the ARPANET TCP/IP Digest and of the Role of Online Communication in the Transition from the ARPANET to the Internet*, 2001.

[5] W. Treese. The open market internet index for 11 november 1995. Treese.org. Retrieved 21 March 2016.

[6] Miniwatts Marketing Group. Internet world stats. November 2015.

[7] The Hive Reporters. The 8th continent needs a bill of rights. 12 March 2014.

[8] P. Emeagwali. Internet is the 8th continent. via YouTube, 31 October 2007.

[9] P. Crosman. Banks lose up to $100k/hour to shorter, more intense ddos attacks. American Banker, 23 April 2015.

[10] D. E. Sanger and N. Perlroth. Bank hackers steal millions via malware. The New York Times, 14 February 2015.

[11] A. Kanungo, A. Sharma, and C. Singla. Smart traffic lights switching and traffic density calculation using video processing. *Recent Advances in Engineering and Computational Sciences (RAECS), IEEE*, 2014.

[12] M. Chana, E. Campoa, D. Estèvea, and J. Y. Fourniolsa. Smart homes - current features and future perspectives. *Maturitas Volume 64, Issue 2, Pages 90–97, Elsevier*, 2009.

[13] F. K. Aldrich. Smart homes: Past, present and future. *Inside the Smart Home pp 17-39, Springer London*, 2003.

[14] O. Vermesan and P. Friess. Internet of things – from research and innovation to market deployment. *River Publishers*, 2014.

[15] H. Kopetz. Internet of things. design principles for distributed embedded applications. *Part of the "Real-Time Systems" Series pp 307-323, Springer US*, 2011.

[16] S. Jankowski, J. Covello, H. Bellini, J. Ritchie, and D. Costa. The internet of things: making sense of the next mega-trend. *Goldman Sachs Global Investment Research*, 2014.

[17] F. Xia, L. T. Yang, L. Wang, and A. Vinel. Internet of things. *International Journal Of Communication Systems*, 2012.

[18] Y. Wang and X. Zhang. Internet of things. *Proceedings from International Workshop IOT 2012, Changsha, China, August 17-19, Springer*, 2012.

[19] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks, vol. 54, no. 15, pp. 2787–2805, Elsevier*, 2010.

[20] F. Lin and W. Ye. Operating system battle in the ecosystem of smartphone industry. *International Symposium on Information Engineering and Electronic Commerce, pp. 617-621, IEEE*, 2009.

[21] T. Berners-Lee and M. Fischetti. Weaving the web: The original design and ultimate destiny of the world wide web by its inventor. *Harper Information*, 2000.

[22] S. Ziegler, C. Crettaz, and L. Ladid et al. Iot6 – moving to an ipv6-based future iot. *The Future Internet: vol. 7858 of the series "Lecture Notes in Computer Science" pp 161-172, Springer Berlin Heidelberg*, 2013.

[23] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta. The internet of things has a gateway problem. *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications, pp. 27-32, ACM New York, USA*, 2015.

[24] H. Zhang and L. Zhu. Internet of things: Key technology, architecture and challenging problems. *International Conference on Computer Science and Automation Engineering (CSAE), IEEE*, 2011.

[25] L. Tan and N. Wang. Future internet: The internet of things. *3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), IEEE*, 2010.

[26] D. Giusto, A. Iera, G. Morabito, and L. Atzori. The internet of things. *20th Tyrrhenian Workshop on Digital Communications, Springer*, 2010.

[27] R. H. Weber. Internet of things – new security and privacy challenges. *Computer law & security review 26, p. 23–30, Elsevier*, 2010.

[28] J. Hirsch. Hackers can now hitch a ride on car computers. Los Angeles Times, 13 September 2015.

[29] D. Goodin. 9 baby monitors wide open to hacks that expose users' most private moments. Ars Technica, 2 September 2015.

[30] S. Schneider. Understanding the protocols behind the internet of things. Electronic Design, 9 October 2013.

[31] The MQTT protocol official website. URL `http://www.mqtt.org`. Retrieved 21 March 2016.

[32] S. Lee, H. Kim, D. Hong, and H. Ju. Correlation analysis of mqtt loss and delay according to qos level. *International Conference on Information Networking (ICOIN), IEEE*, 2013.

[33] A. Stanford-Clark and H. L. Truong. Mqtt for sensor networks (mqtt-sn). Protocol specification version 1.2, IBM, 14 November 2013.

[34] T. Jaffey. Mqtt and coap, iot protocols. Eclipse newsletter, February 2014.

[35] C. Lesjak et al. Securing smart maintenance services: Hardware-security and tls for mqtt. *13th International Conference on Industrial Informatics (INDIN), IEEE*, 2015.

[36] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). RFC 7252, June 2014.

[37] W. Colitti, K. Steenhaut, N. De Caro, B. Buta, and V. Dobrota. Rest enabled wireless sensor networks for seamless integration with web applications. *8th International Conference on Mobile Adhoc and Sensor Systems (MASS), IEEE*, 2011.

[38] K. Hartke. Observing resources in the constrained application protocol (coap). RFC 7641, September 2015.

[39] L. Johansson. Xmpp as mom. *Greater NOrdic MIddleware Symposium (GNOMIS), Oslo, University of Stockholm*, 2005.

[40] P. Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. IETF RFC 6120, March 2011.

[41] J. O'Hara. Toward a commodity enterprise middleware. *Queue - API Design, Volume 5 Issue 4, pp. 48-55. ACM New York, USA*, 2007.

[42] OASIS AMQP Technical Committee. Oasis amqp version 1.0, sections 2.6.12-2.6.13, . Retrieved 21 March 2016.

[43] OASIS AMQP Technical Committee. Oasis amqp version 1.0, section 5.1, . Retrieved 21 March 2016.

[44] OASIS AMQP Technical Committee. Oasis amqp version 1.0, section 1.1, . Retrieved 21 March 2016.

[45] S. Tai and I. Rouvellou. Strategies for integrating messaging and distributed object transactions. *In IFIP/ACM International Conference on Distributed systems platforms, pages 308–330. Springer-Verlag New York*, 2000.

[46] The Open Group. An introduction to internet of things (iot) and lifecycle management. *White paper of The Open Group IoT Work Group*, 2016.

[47] The Open Group. Open data format (o-df), an open group internet of things (iot) standard, 2014.

[48] The Open Group. Open messaging interface (o-mi), an open group internet of things (iot) standard, 2014.

[49] D. Kiritsis, A. Bufardi, and P. Xirouchakis. Research issues on product lifecycle management and information tracking using smart embedded systems. *Advanced Engineering Informatics, Vol. 17, 189–202*, 2003.

[50] D. Kiritsis and A. Rolstadås. Promise-a closed-loop product lifecycle management approach. *In IFIP 5.7 Advances in Production Management Systems: Modelling and Implementing the Integrated Enterprise, NIST Publishing: USA*, 2005.

[51] H. B. Jun, J. H. Shin, D. Kiritsis, and P. Xirouchakis. System architecture for closed-loop plm. *International Journal of Computer Integrated Manufacturing, 20(7):684–698*, 2007.

[52] Akka Team. Akka framework documentation: Actors.

http://doc.akka.io/docs/akka/current/scala/actors.html. Retrieved 9 May 2016.

[53] K. Främling, D. Potter, K. Seidler, M. Neugebauer, and H. Do. Dr13.0: Promise end-to-end security white paper. *WP R13: PROMISE End-to-End Security*, 2007.

[54] H. Brody. How https secures connections: What every web dev should know, 2014.

[55] Kary Främling, Jan Holmström, Timo Ala-Risku, and Mikko Kärkkäinen. Product agents for handling information about physical objects. *Helsinki University of Technology*, 2003.