Dissertation

ARM: Adaptive Runtime Middleware

SAVVAS SAVVA

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2018

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ARM: Adaptive Runtime Middleware

Savvas Savva

Supervisor

Dr. George Papadopoulos

Co -Supervisor

Dr. Achilleas Achilleos

The Individual Diploma Thesis was submitted towards partially meeting the requirements for obtaining the degree of Computer Science of the Department of Computer Science of the University of Cyprus

May 2018

Acknowledgments

With the help of the Triune God and the Most Holy Theotokos, I managed to complete this work. I would like to express my thanks to my Chief Professor George Papadopoulo and my Co-Chief Professor Achillea Achilleos for the excellent cooperation we have had throughout this diploma thesis. The tips, guidelines and their advices, help me to complete this work. Special thanks to my Co-Chief Professor Achillea Achilleos for their help and support to be able to accomplish this work successfully. Also, thanks to Kyriaki Georgiou that create the middleware initially and deliver it to us.

I would like to dedicate my diplomatic work to my parents, grandparents, my two sisters and brother for the love they have shown me, and the support they have given me to encourage me to continue my work and achieve my goals.

Abstract

Nowadays Internet of Things (IoT) becomes very interesting topic. More and more people from various places in the World want to control devices from any place. Also people aims to interconnect their devices and bring heterogeneous devices together. That ecosystem will support software services as well. This logical devices can located anywhere in the world. Many physical or logical objects from everyone everyday life will be able to communicate with other devices or humans offering new services and applications.

To achieve this transition to the IoT World, a Middleware is a core element. A Middleware can help, because from its architecture can support information flow on connected systems in a network. Thus more complex heterogeneous systems can connected each other either locally or either on Cloud and help developers to deploy more easily solutions for users or business clients.

This thesis primary goal is to expand the Adaptive Runtime Middleware (ARM), which was building earlier in 2017 from Kyriaki Georgiou, to support Sensors. Especially, the expansion of the Middleware System to support Server Sent Events to manage Sensor Devices, so sensors can sent from it side all needed information when it is mandatory, without floating the network with unnecessary information.

Now the Middleware system it is self-configured and adopt automatically changes and expose functionality of the bundles as REST calls & SSE when is applicable to automatically generate Web-API. Also it expose the devices functionality by deploy again the Middleware. The Middleware support interoperability among heterogeneous devices and platforms.

Another goal is to Create Sensors form Scratch using Development Boards and then deploys its OSGi bundle to middleware. Through this process they have been created in total three Sensors (Motion, Light and Distance Sensors).

Our ultimate goal was to create a system that is auto-explanatory, easy-to-access, creative, easy-to-use, easy to deploy and clustered to be friendly for Client Application Developers (e.g. Android, iOS and Web Application Developers).

Table of Contents

Chapter 1

Introduction

1.1 Project Motivation

Internet of Things (IoT) was invented in previous decades; People want to have "connected" devices. IoT went in many people lives slowly in less noticeable applications in the previous years, but now over the years had become increasingly popular and can be found it on more and more applications.

Nowadays, a big amount of IoT devices and development boards appear every month. Anyone can have everything connected. Big and Small Companies start to produce their own IoT Devices. Many known and unknown Brands become popular for their IoT Devices. Many of those devices invented for entertainment purposes, for health purposes, for security purposes etc. Many of those IoT devices are heterogeneous to each other. Those were happened because many of those Devices have different architecture, have different communication protocols, have limited resources, had developed when newer technology not existed and for other commercial and non-commercial reasons. A Middleware come in to help to connect all those devices easily and handle communication between components to manage simplify all that complexity.

1.2 Project Concept

The main purpose of my thesis is to expand an Adaptive Runtime Middleware system for the Internet of Things, the ARM, to support Server Sent Events (SSE). Also, another important purpose is the deployment of sensor modules on a Development Board, like the Orange Pi.

ARM is dynamic middleware, which is developed in previous year as a master thesis in Computer Science. This middleware is self-configurable and self-adaptive to face the IoT demands. Originally, the Middleware it can expose the functionalities provided by the smart modules (OSGi bundles) through Representational State Transfer calls (REST calls). When an OSGi Bundle is deployed, the middleware discover its capabilities and subsequently expose them by re-generating and re-deploying the middleware Web Application Programming Interface (WEB API). Mainly was supported Actuator Plugins.

Focusing to improve and expand the Middleware in the best possible way to support SSE as well. So, using known technologies in the middleware, keep it simple and friendly to developers.

Now middleware is able to expose the functionalities provided by the OSGI bundles through REST calls & SSE as well, when is applicable. It can also support sensors that and actuators as well. Finally, it can bring even more devices together, that is heterogeneous.

Despite the changes that made in middleware, its core architecture is staying simple, as you can see in in Figure 1.1.

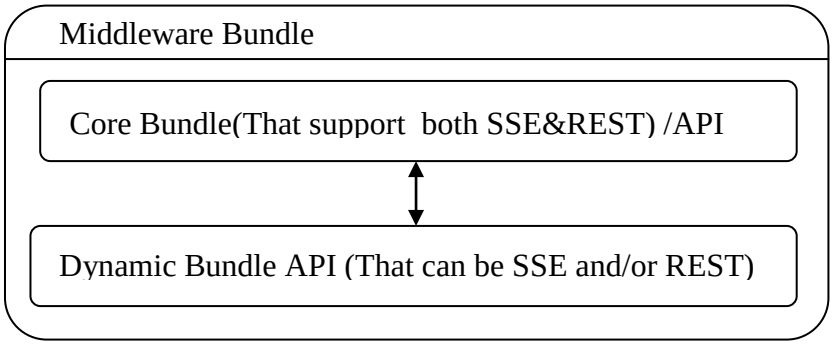


Figure 1.1: New Middleware Architecture.

For Sensors development, is used the Orange Pi Development Board, which hopefully can support a lightweight Debian based Linux distribution, the Armbian. As a result, the

creation of three motion Sensors (Distance, Motion & Light Sensor) is developed in the same Orange Pi Development Board.

The pyH3 python library is used to communicate with Orange Pi General-Purpose Input/Output (GPIO). The Flask (Python) Application deployment is done using a Web Server Gateway Interface server (WSGI server). Flask is a python micro framework. WSGI is a simple calling convention for web servers to forward requests to web applications or frameworks written in Python. The Flask application object is the actual WSGI application. The usage of Gunicorn server, which is a Python WSGI HTTP Server for UNIX, is important to serve successfully the Flask (Python) Application.

You can find more information about system implementation in next chapters.

Based on the above, the general scheme of the system is shown in Figure 1.2.

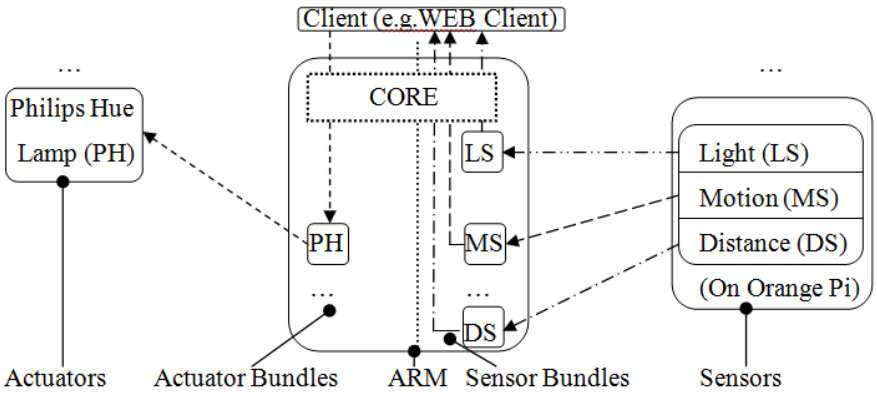


Figure 1.2: General scheme of the system.

1.3 Project Structure

This thesis is organized in the next 6 chapters as follows: Chapter 2 contains information about the technologies, for the core existing technology and the new technologies, which used to expand the middleware functionality. Chapter 3 describes the current Middleware specifications and the current Middleware architecture. Chapter 4 describes the Sensors and Server Sent Events implementations, with extra detail. Chapter 5 demonstrates a use case scenario that is based on sensors and actuators. In Chapter 6 there are the evaluation Methodology, which is used to evaluate the Middleware in this state, and a questionnaire, along with the evaluation results. Finally, in Chapter 7 there are some conclusions and future work which can be done in Middleware.

Chapter 2

Related Work

2.1 Internet of Things

The concept behind Internet of Things was the need to have connected Devices in the wide accessible Internet, like other humans was connect to Internet. Those Devices can have Hardware and and/or Software Services. These devices must be small enough and will not drain much power to operate. Also it must have high availability. That devises can be embedded devices which meet the above criteria.

Most enterprise projects are based on IoT solution cost-reduction to make IoT scalable, applicable and accessible to much more humans. The number of IoT enterprise employments is growing.

IoT can be applied in almost all Industries like Smart Homes, Smart Cities, Connected Buildings and Connected Cars. It can be also applied in healthcare and environment, which is one of the more reasonable applications for Internet of Things. [1]

For example, the distributed Sensors that is located all over the world and measures data about and counting different values to eventually manage to adapt a good ambiance and predict with more accuracy different weather phenomena that are being made and inform us accordingly.

Internet of things is very powerful, but if it will not be used as it should, it can become a threat to human being. Known technologies (Blockchain, RFID) can convert human Beings to pawns. Privacy in IoT must be utmost importance. [2]

2.2 Middleware for the Internet of Things

There is a lot of Middleware Systems out there that is proposed by many researchers. Many of that are remarkable, other is too complex and other is not scale well.

Many of that middleware have modular architecture (uses components) like OpenHab. OpenHAB is an open-source automation software it cover a lot home automation systems and technologies under the same umbrella, enabling the ability to the user to define the interaction of things and devices through automation rules and uniform user interfaces. This middleware is based on Java OSGi as well, like the ARM Middleware.

Also many of that middleware become powerful have the ability to store information in cloud like Tuya Smart and Eclipse Kura. Eclipse Kura [3] support communication with Eclipse Kapua platform, that is for IoT cloud Services. Such a Model can support device management and data management needed for tracking devices. This middleware based on Java OSGi too.

Also, another Middleware System, worth to mention, is the HomePort [4]. The HomePort Middleware provides REST interface to heterogeneous sensor networks. That Middleware use the Server Sent Events technology too.

ARM Middleware is a dynamic middleware system for the Internet of Things. When a smart module (sensor, actuator) is deployed, the system has the ability to discover it, detect their functionality, self-reconfigure, self-regenerate the system Web API and re-deploy the Middleware System. As a result new device capabilities/functionalities are dynamically injected and new Web API functions are created.

ARM goal is to enable interoperability among heterogeneous devices and platforms and automate device discovery and management. Another goal is to have a simple system for developers. ARM is not adding additional technologies and platforms that are unknown and increase the complexity for a solution for the IoT, but reduce the initial learning curve [5].

The proposed expansion in the ARM middleware is to add Server Sent Events functionality on the Middleware. Such an expansion will give the ability to install

Sensor Smart Modules (and not only support sensors by asking). ARM Middleware detect automatically the Server Sent Events that is defined in a Smart Bundle / OSGi Bundle (detect their functionalities) and self-reconfigure and self-regenerate the middleware's Web API and inject the new capabilities. SSE functionalities are exposed in the Rest Web API Level, as Rest, and they are descriptive to help even more developers. Server Sent Events can notify a Client application, a Reasoner plugin or a clever Rule engine, based on application needs. The System automation philosophy is kept to reduce learning curve and make a simpler system; which a developer can interact with easily.

2.3 REST and REST-full Web Services

REpresenational State Transfer (REST) is an architectural style. Rest is not defining a standard, but its implementations (RESTful implementations) use some well-known and accepted standards, like URI, HTTP (GET, POST, PUT, DELETE and PATCH), XML and JSON.

There is six formal architectural constrains, that define a RESTful system. A constrain can restrict the process and reply form server side to a client request. In this way the service will have some non-functional properties, that is simplicity, modifiability, performance, portability, scalability, reliability and visibility.

The First Architectural Constrain is Client-Server architecture. This Constrain is created to separate two concerns, the user interface concerns from the data storage concerns. Using this separation, results to improve the system scalability because it introduces simplified user components. The separation of the two concerns enables portability. This gives us the ability to have multiple platforms a uniform user interface. Also, satisfying this constrain enables the ability to have multiple organizational domains, which is a requirement for scalable internet.

The Second Architectural Constrain is Statelessness. This constrained is declared to define that must not save client side information in client server request interaction. The session state must stay in the client and not in the server. All information needed by client, to reply to server requests, must be kept in session.

The Third Architectural Constrain is Cacheability. This constrain is created to satisfy the caching need. There is the ability to cache the responses and get best application

performance is the same data that already retrieved before is used again. Also using this technology can benefit also the network traffic. As a result the traffic will be less using cache, and will not waste the network resources unnecessarily. The expiry flags must be declared carefully to manage the cached contents and so the outdated data will not be used. Also some type of data must be non-catchable, and this constrain satisfy this need.

The Fourth Architectural Constrain is to have Layered System. Satisfying this constrains enables a security level and give best application performance and response times, because client cannot detect if it is connected to the original root rooter or if it is connected to another intermediary (cache) server.

The Fifth Architectural Constrain is to have Uniform Interface. This is mandatory to destine REST services. To Satisfy this constrain, four smaller constrains must be satisfied to cover it. The resource identification in requests constrain. This means that other data format can be exists for communication outside of the system (like JSON, XML etc.) and internally processes the data in other format. So, that introduces the identification of a resource by its resource identifier. The resource manipulation through representations constraint, mean that when the resource representation and any metadata is known, then any Client Application Developer can use that to delete or modify that resource. The Self-descriptive messages constraint, introduce the idea that must there are enough information that describes the resource message and have complete knowledge about message and can easily process the resource message. An example is the MIME type. And finally, the hypermedia as the engine of application state (HATEOAS) constrains introduce an analogy to human web browsing, that visit a web page and from this point can travel like hypermedia system. The same concept applied as a REST constraint, but it means that a REST client must have the ability to discover all the available actions and resources by providing text that contains hyperlinks that link to all other available actions.

The Sixth Architectural Constrain is Code on Demand (that is an optional constrain). Satisfying this constrain, the client have the ability to download code form RESTful API’s and execute this code in client side (e.g. JavaScript, Flash SWF, Java Applets).

If a Service violates any of the above non optional constrains can’t be considered as a clear RESTful service.

On the other hand the architectural design constrains is little elastic in some applications but its API’s falsely tended to call clear RESTful. This APIs can be a

mixture of RESTful and another technology, that might violate a lot of or a little bit the RESTful system architectural constrains. So if an API is not clearly RESTful is good to mention it. ARM API at the moment is a mixture of REpresenational State Transfer (REST) and Server Sent Events (SSE). In next subchapters you can find more information about it.

2.4 Server Sent Events (SSE)

Server Sent Event (SSE) is a technology that a Client (e.g. Browser) receives updates from a Server using HTTP connection. SSE is an efficient real time streaming technology of text-based event data. This technology stands on two axes the EventSource API and MIME Type event-stream [6] [7] [8] [9]. In the SSE Client side, there is an EventSource interface that subscribes to Server Sent Events source. The EventSource interface resumes completely all the low level connection establishment and message parsing behind a plain Client API. Server sent push notifications to a registered Client as DOM events and MIME Type event-stream data, which is used to deliver every single updates.

The SSE technology can be an effective and necessary technology and can be done such a technology, by handling the events and sending updates when required and when an event exists. This technology sent real-time data in the Client effectively because it doesn't use any cache technology to store an event and Client receives it later. SSE use single long lived HTTP connection to guarantee low latency to conveyance the event-stream data. SSE has automatic functionality like tracking the last received message and automatically reconnection in case of disconnection. If the Client is a Browser, then the push notification data can be handled as DOM events.

Internally SSE implementation is cross Client XHR streaming and as a result the Client operates internally to do all the complex stuff like connection management and message parsing. So, SSE is stay simple, efficient technology for unidirectional real-time data flow using HTTP.

SSE also has some great features like event ID, automatic reconnection and can sent arbitrary events.

The Event Stream Format (MIME type: text/event-stream) have a basic form as follows: “data: notification_message\n\n”. The ability to add more fields to an event is very interesting but is not a necessary as well as sending the event data in JSON Format.

Here are some examples:

- Add Event name:

  “event: user_logon\n
  data: notification_message\n\n”

- Add Event ID and Sent multiline data:

  “id: 123\n
  data: notification_message_line_1\n
  data: notification_message_line_2\n\n”

- Retry to reconnect after 2 seconds:

  “retry: 2000\n
  data: notification_message\n\n”

- Sent JSON Formatted data:

  data: {\n
  data: "message": "a line",\n
  data: "Device_Id ": 12\n
  data: }\n\n

2.5 REST-full& SSE Architectural Similarities

There is no way to respect REST while use SSE on it. When a registration on an SSE takes place, then no one knows how many events will get as replies.

But, for comparison purposes, there is the ability to modify the SSE technology architectural constrains to bring it very close to REST architectural constrains. The First Architectural Constrain, Client-Server architecture can be satisfied, when the client triggers while the server does the processing.

The Second Architectural Constrain, Statelessness, can be satisfied, by storing client state on the client (HTTP is a stateless protocol).

The Third Architectural Constrain, Cacheability, can be satisfied, by not using cache header. This can be achievable because SSE doesn't use processing model and because in the article about REST [6] there is the above:

"Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable."

The Fourth Architectural Constrain to have Layered System can be satisfied, by adding another transform in the data stream layers (e.g. pipes and filters).

The Fifth Architectural Constrain to have Uniform Interface, which can be satisfied partially. Because can satisfy the identification of resources, by using URI's. Can satisfy, constrain manipulation of resources through representations, by using HTTP methods with the same URI. Can satisfy, the self-descriptive messages partially and by using content-type header and if add RDF to the data but, there is no standard which describes that the data is RDF coded and need to define new MIME type text/event-stream-rdf. Finally, can satisfy constrain hypermedia as the engine of application state, by sending links in the data.

The Sixth Architectural Constrain Code on Demand is satisfied anyway because that is an optional constrain.

If someone want to form SSE to be REST, then the SSE must comply with all the REST Constrains and this mean extra complexity, a lot of modifications, extra work for developers (on bundle creation), not a simple work for application developers and at the end only a subset of compatible SSE Devices, that can work with that solution. ARM is not such a middleware and has to include different technologies together because this is

the best solution and there is not such a rule, that you can't use two different technologies together in a WEB API [6], [10].

2.6 Web Server Gateway Interface (WSGI)

The Web Server Gateway Interface (WSGI) written in Python programming language. WSGI is a calling convention for Python web Servers to Forward Requests to frameworks and cross platform client applications (e,g. iOS, Android, Web Applications). WSGI tends to give simple implementation in application logic, especially when use it with other Python Microframeworks. WSGI have two sides the server (gateway) side and the application (framework) side. Between these two sides can be exists Middleware components. In server side take place either a light software defined webserver that can communicate with a full featured web server (e.g. Gunicorn) or take place a full featured web server standalone (e.g. NGINX, Apache). In application side can be found a Python Program (Script) that is callable (contains sometimes that is callable, like method, functions, class or instance with __call__ method). This Script can be used with a combination with another Python Programs (Script), frameworks and Libraries to enrich the application capabilities. Those Middleware components might be more than one and can collaborate and communicate with each other, with other WSGI middleware components or other separate WSGI Middlewares or other WSGI applications. Those middleware components in fact, it implements both sides of WSGI. This methodology fit best when the Orange Pi Development board is used to build the Sensors, but its usage is not limited on developments boards and in fact WSGI exists in many corporate applications [11].

Chapter 3

Middleware Specification and Architecture

3.1 Middleware Specifications

The ARM Middleware system stands on three main Specification axles. The first axle is Server Sent Events (SSE) architecture, the second axle is the REST architecture and the third axle is the OSGi bundles. These technologies can bring a dynamic middleware to life. Middleware is relying on Java OSGi to implement the Middleware "business logic".

3.1.1 OSGi Bundle Specifications

Any Developer including Third-Party company's developers can create its own smart OSGi bundle to attach it on ARM middleware and expose the functionality of a hardware device or a software service as a REST or SSE callable URL. Developers can include any amount of functions in that bundles for best user experience. Also Developers must not require to have knowledge to implement the REST Architecture in the bundles, but only need a very basic knowledge about Server Sent Events, to be able to build is application correctly, without need to know SSE low level internal

functionality and implementation. So, using this architecture will reduce the learning curve for new technologies in the lower possible amount.

To make this middleware possible to exist there is only a small addition over the traditional OSGi bundles, which is an extra XML file that is needed from middleware to be able to adapt accordingly. That XML file will be simple and will need only the fully qualified name of the implemented bundle class.

Once the bundle is added in the middleware will automatically detect it and retrieve all the available methods that developer declares in this bundle and create the internal corresponding java file and install the bungle to the middleware. Also, after that, the middleware will generate a new URL for each public method that have in that OSGi bundle. It will also create another URL that have the /def extension and will contain all the definition information that the developer insert while creating the bundle. Especially, it will have the name of the web method, the parameters and their type, the return type and all other additional information that the developer define in the Bundle. In addition to the previous process, Middleware can support Server Sent Events along with any additional information needed. Afterwards, when a URL is called and a web method is invoked the related method on the OSGi will be called and the OSGi bundle functionality will start.

Finally, in the middleware API exists a URL call that shows all the OSGi bundles that is available in the Middleware.

3.1.2 REST Specifications

The REST design is used to be able to expose the Smart Modules (OSGi Bundles) functionality to a bigger network, for example to the Internet. Middleware is an OSGi bundle, which is dependable for the automatic functionalities that, the middleware will provide, like self-configuration. In expansion, that bundle contains and runs a server, which can be utilized for distributing the URLs. Each URL will be created for mapping bundle functionality with a freely open URL, in conjunction with another URL which is giving all the information, related to the first exposed URL. The details that is shown in the second URL is the information that the bundle developer insert when create the bundle. Doing this, any client application developer can create its application faster and easily. Also some other URLs are used for navigation purposes.

The methodology between this communication is take place when a Client Application Developer wants to invoke a web method, from the middleware API, and if this method needs any parameters will be inserts it and in the callable API URL. When this URL is called, then it calls the web Method from the OSGi bundle, and then those executes the functionality from the developer Bundle and return to the invoker any result data. Then if there is any result data from the previous action, is returned to client API invoker [5].

3.1.3 SSE Specifications

The SSE architecture is used to enable the real-time updates to and from the Middleware System, with unidirectional HTTP connections. Using the same development methodology, that based on smart modules (Java OSGi bundles), the bundle creation can be deployed by consuming or producing real-time server sent events. Also by having bundles that can consume and produce at the same time real-time server sent events. The input Bundles implementation is based on the Smart Hardware Device or a Smart Software Service bundle Developer, which is very flexible to mix up technologies and ideas to create the Bundle in a way that can communicate correctly with its device or Service. Its work will be also simple because it has the flexibility to combine a lot of technologies that needed from the devices or it use it more and understand it well and use the already existed libraries that is now attached in the middleware and handle server sent event data in the best possible way. The OSGi Bundle functionality is exposed in the Middleware API level, like the other REST Calls, and it is managed by the middleware bundle that contains runs the server, which will be used for publishing the URLs. This approach is being used to have a Simple unify Middleware system for the Client Application Developer.

When invoking a web Server Sent Event method which is related to an OSGi Server Sent Event, the web method that was generated before will invoke the actual server sent event method from the installed bundle and register to that Server Sent Event and execute its functionality that might be have internally. That internal functionality can be connection to another SSE form a hardware device or a software service and create SSE from that, can be a live creation of server sent event using another technology, or can be a combination of technologies to create SSE. When a disconnection of SSE (Close) is

happened, then the appropriate Sensor can gets the appropriate disconnection message from middleware based on the Sensor bundle implementation and can closes or keeps alive the connection, based on the bundle implementation, which is declared by the OSGi Bundle Application Developer (in any hardware device or any software service system). SSE is flexible and functional, and can be used internally or externally in the middleware in the future.

3.2 Middleware Architecture

At the moment middleware have a very simple architecture that takes the OSGi bundles and expose its functionality by creating REST and SSE calls from it. This calls is well defined URLs that apart the Middleware API. The middleware can run from a server that is located locally that can be connected to the Internet, a WAN, a LAN and be able to handle all the connected devices in the LAN, which have their OSGi bundles. So, the middleware is very flexible, and can create virtual device groups even in the same LAN. The individual OSGi bundles can include all kind of functionality that the developer declares. These bundles are act as input for the middleware, and while is running have it installed is a part of it, because middleware automatically generate web methods and publish the related URL from those bundles.

In the Figure 3.1 is the graphic representation of ARM current architecture. In the middleware bundle exists the application server, which handles the Internet communication and the rest of the hardware device bundles in the LAN or the software services inside in the WAN. The middleware bundle, in the LAN range for hardware devices and in the WAN range for software services, communicates with the installed bundles and creates the web methods to be available outside the local network. From that web methods will be created the REST and SSE calls. These calls are URLs that apart the Middleware API. Another important aspect to mention is that the middleware support internal connections in Middleware, which gives the ability to create a powerful rule engine in a “Reasoner” Java OSGi bundle in the future, as shown in the figure 3.2.

This idea can be expanded furthermore and can be applied on both bigger and smaller networks. This architecture is easy to setup and scale very well. It can be applied in different locations for different applications and bundles without need to change anything in the middleware bundle. The only need is to provide the correct OSGi

bundles and so the Middleware be able to detect correctly the devices that must be attached to it and expose its functionalities each time. For example, installing the middleware in a house and installing the same middleware in a school, will not interfere with each other. And for example in the house bundle having a Philips hue light bundle, a Temperature Sensor bundle and a heater Boiler bundle. In the school, having three same type Philips hue light bundles, five Motion Sensor bundles and ten infrared which bundles to turn on projectors. Now let's guess that the house and School is located in different cities and flights for another country and forgets a projector open in school and in the house lefts a Philips hue bulb on. Using this middleware with a combination of a good client application a Client/User can manage all that devices with ease from the other country and turn them off. Another example let's say having the same home and same school and wanting to leave from school to go home and don't know if at home it's colt. Using this middleware with a combination of a good client application to manage all that devices, and set a rule that will enable the heater.

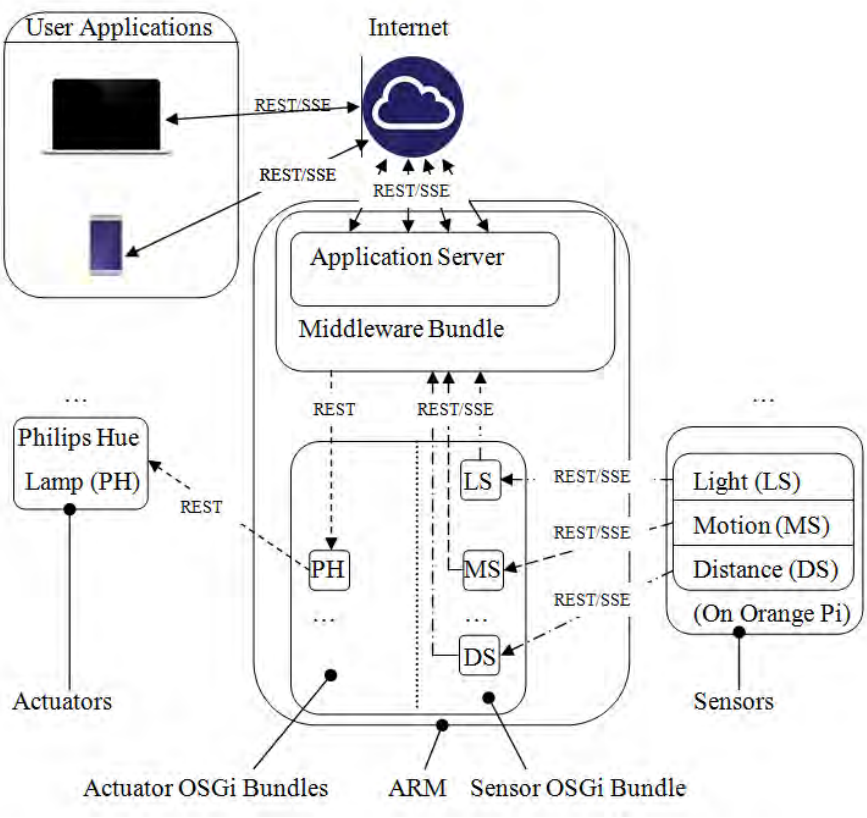


Figure 3.1: General scheme of the system.

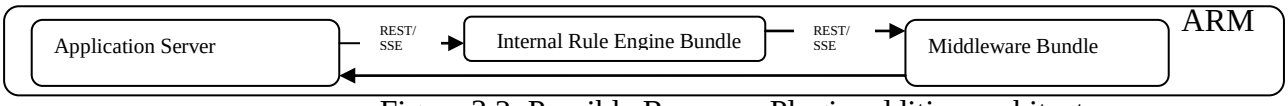


Figure 3.2: Possible Reasoner Plugin addition architecture.

Chapter 4

Sensors & Middleware Implementation

4.1 Development Board

There is a variety of development boards out there, to be able to build sensors. There are a lot of common name expensive and cheap boards to choose from. The choice to have something different from the traditional raspberry pi development board that is the most common out there can't be falsely. The best choice, for this scenario, is the Orange Pi One[1]. That is a simple all in one open-source development board in the cheapest category. These development boards have a great price tag, but the support from its company is not the best. Fortunately, Armbian distribution is compatible with the Orange Pi One [12]. Armbian is an Open Source Linux Operating System for ARM development boards. It is based on Debian (for its server releases) and Ubuntu (for both some server and Desktop releases). The Armbian Image is written on micro SD Card using the open source tool etcher.io[2]. Also Orange Pi development board has built in GPIO to circuit components and sensors from that without need to have extra

[1] http://www.orangepi.org/orangepione/
[2] https://etcher.io

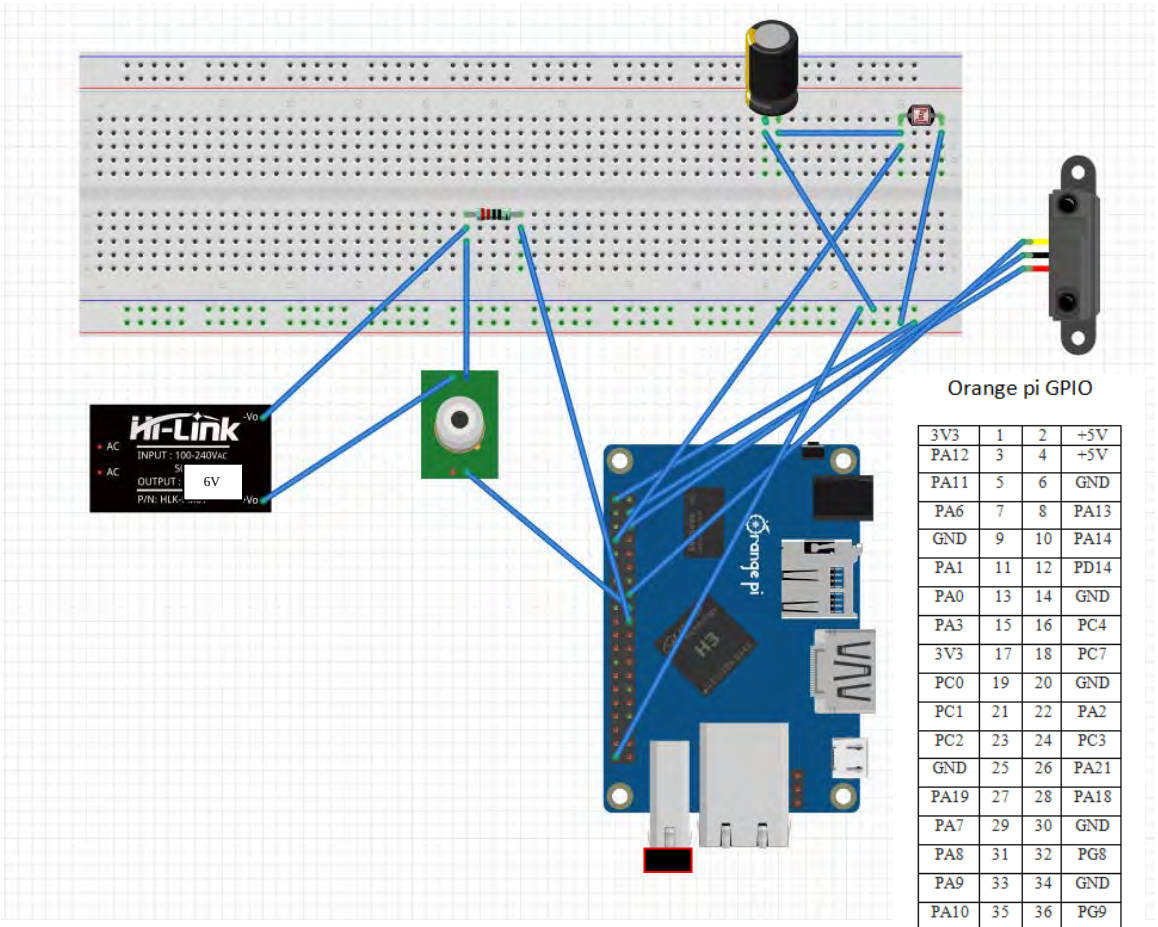
development boards. This Board has Quad Core ARM_Cortex CPU, 512MB DDR3 Ram and plenty of connectivity ports, and is sufficient to deploy three sensors on it. The lack of Wi-Fi in this board is not an issue, because there are a lot of other Wi-Fi USB dongles available to attach on it.

The three sensors are deployed on the same development board. One Motion Sensor, which is digital Sensor unit. One Distance Sensor that is also a digital unit. And also, one classic Light Sensor, that is analog.

The Digital Sensors can be connected directly to GPIO.

The Orange Pi and Raspberry pi GPIO of those devices is digital and not analog, but for that application this is not an issue, because using a breadboard, a capacitor and a modification in the Python script (discuss in later chapters about it), and enables the measurement of the appropriate values for an analog Light Sensors with success.

The figure 4.1 shows the Development Board and Sensors hardware connectivity.



3V3	1	2	+5V
PA12	3	4	+5V
PA11	5	6	GND
PA6	7	8	PA13
GND	9	10	PA14
PA1	11	12	PD14
PA0	13	14	GND
PA3	15	16	PC4
3V3	17	18	PC7
PC0	19	20	GND
PC1	21	22	PA2
PC2	23	24	PC3
GND	25	26	PA21
PA19	27	28	PA18
PA7	29	30	GND
PA8	31	32	PG8
PA9	33	34	GND
PA10	35	36	PG9
PA20	37	38	PG6
GND	39	40	PG7

Figure 4.1: Development Board and Sensors hardware connectivity.

4.1.1 Technologies

The software technologies, that anyone can use to deploy sensors on Orange Pi, vary. But, for this case it was preferable to go with Python Language, WSGI, Flask and the pyA20 Python Library package. Finally, Gunicorn used to serve the Flask application to the network, because the Flask build-in server is not scale very well. The WSGI was discussed earlier in Chapter 2. In summary, WSGI is a calling convention for Python web Servers to Forward Requests to Frameworks. WSGI simplicity can fit with Flask very well. Flask is a Microframework. In this sensor implementation you can find out that the WSGI application is the Flask application object. Also the pyA20 Library[3], that is used, is a package that is adapted for orangepi_PC_gpio_pyH3 and is fully compatible with Orange Pi One. This package is not only control the GPIO, but also control GPIO pins, I2C and SPI busses.

4.1.2 Flask Python Application

The Flask Applications tends to be easy and can be built without have to write a lot of code is. This Python Microframework is also lightweight. Is very flexible Solution and in Deployment can be hosted not only Self hosted locally, but also hosted on Heroku, OpenShift, Webfaction, Google App Engine, AWS Elastic Beanstalk, Local host Server with Localtunnel, Azure (IIS) and PythonAnywhere.

There is a procedure to prepare the Orange Pi sensor deployment environment to be able to, run successfully the Flask[4] Application. First requirement is the Python installation, then is the pyA20 installation then is the Flask installation (including Flask CORS and Flask SSE) and finally the Gunicorn Installation. In this procedure there are some dependencies that must be installed as well.

The Flask Application Object is the WSGI application. This application is basically a Python Script in its folder. This application is designed in a way to be able to serve it multithreaded, using Gunicorn. The Flask Application Anatomy is simple. First there are all the included libraries like the pyA20, sys, datetime, time, flask, flask_cors and flask_sse. Afterwards, there is the initializing section which the orange pi GPIO ports

[3] https://github.com/duxingkei33/orangepi_PC_gpio_pyH3/blob/master/README.txt
[4] http://flask.pocoo.org

initialized. Following there is some Flask and CORS definition commands and then there is the initialization of counters that is used in the code later. Following there is a lot of functions that is called from the “actual API” functions that defines the Sensor API.

The rest of this Functions is strait forward it check the sensor corresponding return value in GPIO and if there is a change with the previous state, it sent an SSE notification message to inform for that change. But in the Light Sensor, there is another extra function because that sensor is analog. There is no strait forward way in a digital GPIO to be able to get the value from the light sensor. Instead, charging up the capacitor and count the time it takes for it to send the input pin to high. Since the light sensor is a resistor, in darker there is higher resistance meaning the capacitor will take longer to charge and based on that measurement, the calculation of dark state or light state is done. The threshold of this value is software variable in the script as well.

The “actual API” functions defines each URL that can be an SSE call or be a REST call or return data using other MIME type or just return plain HTML. This is defined using the annotation @app.route(‘/name_of_route’). Using this technique the Sensor API is ready. For more clarification, the script is also located in Appendix A. The Sensor API hierarchical structure is located in the figure 4.2.

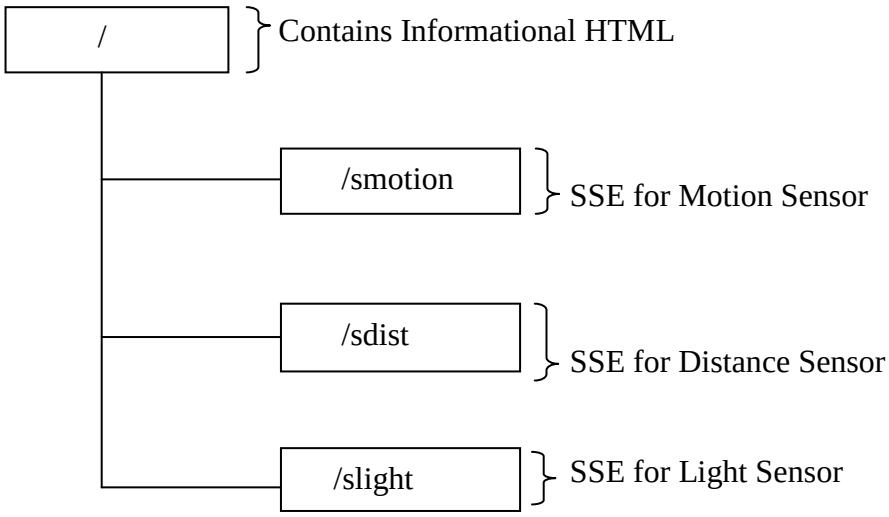


Figure 4.2: Sensor API hierarchical structure.

Finally in the Python script, there is some other Flask Attributes to set the multithreading behavior in the Flask Application.

At the end having Gunicorn to serve the Flask application, delivers optimal service experience. Gunicorn [13] means ‘Green Unicorn’ and is a Python WSGI HTTP Unix Server. Gunicorn server is compatible with a lot of web frameworks. It is also a fairly speedy and lightweight Server. It supports WSGI, Django, and Paster. It can offer easy and automatic worker process management. It’s very extensible server, because have various server hooks. It can handle multiple worker configurations [14], like gevent worker class that is async workers and can fit in the application needs. It gives us the ability to increase the gevent worker count for optimal performance. In figure 4.3 you can find the command that used to run the server.

```
sudo gunicorn --bind 0.0.0.0:80 Desktop:app --worker-class gevent --workers 3 --timeout 90
```

Figure 4.3: Command that used to run the server.

The General Application structure was designed this way, because this is a common kind of setup and to check if there is way to have SSE in both sides as well. There is also other ways to build those Sensors that can work as well.

4.2 Adaptive Runtime Middleware (ARM)

The Adaptive Runtime Middleware it was a REST Based Middleware before. A lot of modifications are done to it to be able to handle SSE. The ARM API now is a mixture of REST and SSE technology. These technologies can live together and there is no restriction that can exclude each other.

4.2.1 Technologies

In this Middleware are present many technologies. The Core OSGi Framework Specification is deployed with OSGi Equinox [15]. The RESTful web services are deployed based on JAX-RS community-driven standard and Jersey that is JAX-RS Reference Implementation. All the previous is connected together (at the service level)

with using the OSGi-JAX-RS Connector [16]. So, having an OSGi service and registering it as OSGi services, it will automatically be published as RESTful web service too. Also, this work form the other direction and a REST service can consumed as OSGi service as well.

The use of OSGi definition XML, remains the same, and must have the same name in all bundles ("osgi_plugin_def.xml"). This XML file is used for the middleware, because it's automatic detection and adaption and regeneration. An example of the structure of this file is located in figure 4.4.

```
<bundle-definition>
  <fullname>osgi_MotionSensor.MotionSensor</fullname>
  <filename>MotionSensor_1.0.0.jar</filename>
</bundle-definition>
```

Figure 4.4: Structure of the XML file osgi_plugin_def.xml.

Also, the same annotation for Bundles implementation is used as well. The annotation for Bundles is used for the OSGi bundles Documentation, to let the Client Application Developers know all the required information needed to create its application fast, efficient and accurate, without need to learn new things. All that information is listed in the API Level. The three different kinds of annotations: bundles annotations, methods annotations and for parameters annotations [5]. In the Figure 4.5 there is an example of it.

```
import osgi_annotations.interfaces.ClassDescription;
import osgi_annotations.interfaces.MethodDescription;

@ClassDescription(value = "MotionSensor is an osgi bundle that represent a motion sensor. "+"
    This plugin provide SSE that feeded from the sensor and then serve it to client.")
public class MotionSensor extends ResourceConfig {
    private static String sensorA = "smotion";
    private String LocalNetworkIP = "192.168.1.135"; // "192.168.31.161";

    @MethodDescription(value = "Return the Sensor type. 1 : Motion sensor, 2: Distance sensor 3: Light Sensor")
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public int getStype() {
        int stype = 1; /// motion sensor
        return stype;
    }

    @MethodDescription(value = "Return the Server Sent Event from the motion sensor. The event name is 'motion'. "+
        "When motion is present then a 'motion' data pass in else, it pass a 'none' data as message.")
    @GET
    @Consumes(SseFeature.SERVER_SENT_EVENTS)
    @Produces(SseFeature.SERVER_SENT_EVENTS)
    public EventOutput getServerSentEvents() {

        final EventOutput eventOutput = new EventOutput();
        Client client = ClientBuilder.newBuilder().register(SseFeature.class).build();
```

Figure 4.5: Annotations from Motion Sensor Bundle.

Furthermore, the core API REST Methods functionality remains the same and also the Path Constriction Logic remain the same as well. So the Middleware base URL gives a list with all the available bundles that its functionality URLs can be accessed through REST calls and SSE calls. In the Figure 4.5 there is an example of it, that shows three sensors (Motion, Distance, Light Sensors) and one actuator (Philips Hue Bundle).

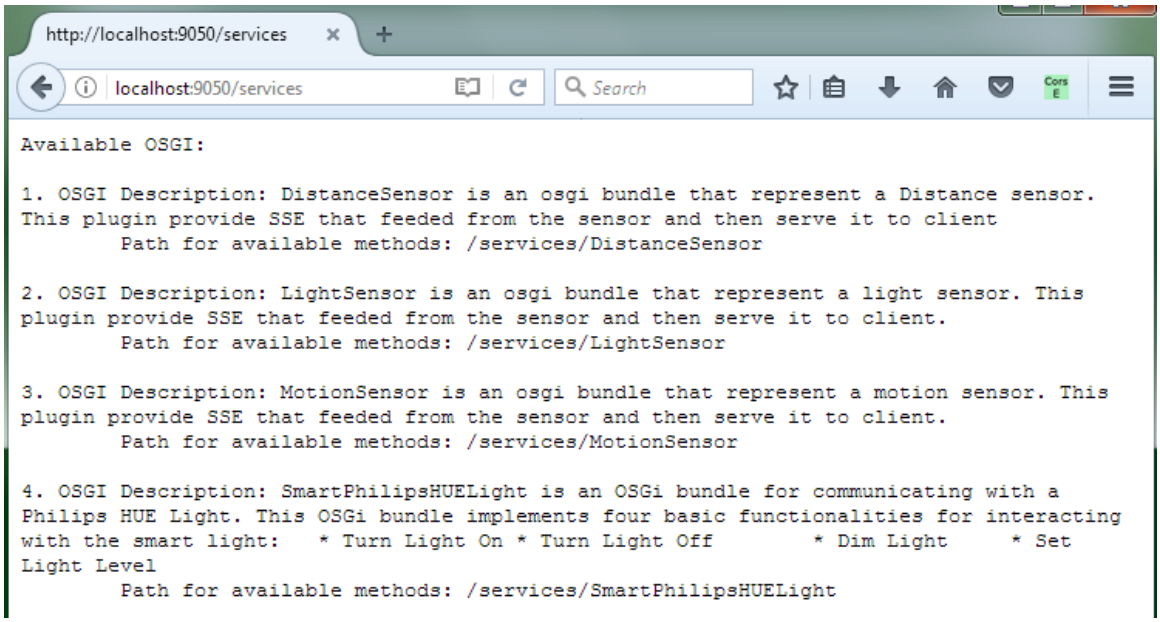


Figure 4.6: List with available OSGi bundles in middleware.

In addition, to be able to work with SSE [9], it requires to include the jersey-media-sse module and requires to have the javax.servlet API 3.x . SSE creates single unidirectional connection between Server and Client (see Chapter 2).

Finally, the middleware served using the Jetty Server that is embedded in eclipse, but can also run as standalone server [17].

4.2.2 Modifications to Support SSE

The old Middleware implementation wasn't support SSE. To be able to support SSE, some changes in the ARM OSGi Bundle are took place. Those changes are able to leave SSE to pass from the Middleware. Modifications were take place mainly in the ConnectionControl.java class file in the OSGi Bundle Middleware. The addition of

org.glassfish.jersey.media.sse.* in the created web (methods) class files and also the modifications in the entire Algorithmic logic of that class, is done to be able to include the SSE in the web (methods) class files that will be created, when the middleware run. For example, the createRestMethod function is changed and an extra if else is inserted, to connect SSE to correctly work as expected. Also from this point, the creation of another function that will expose the SSE to the API Level is done successfully. Furthermore modifications are done elsewhere need to make ARM include the SSE and work again.

Modifying the bundle wasn't sufficient. The eclipse IDE configuration needs also to be modified to support SSE. The final Target Platform Dependencies are in the figure 4.7.

Despite those modifications, the old REST behavior and functionality are remains the same. In addition, using this strategy and having bundles that can contains both functionality (methods) that must be exposed as REST, and functionality (methods) that must be exposed as SSE, all together in the same OSGi Bundle.

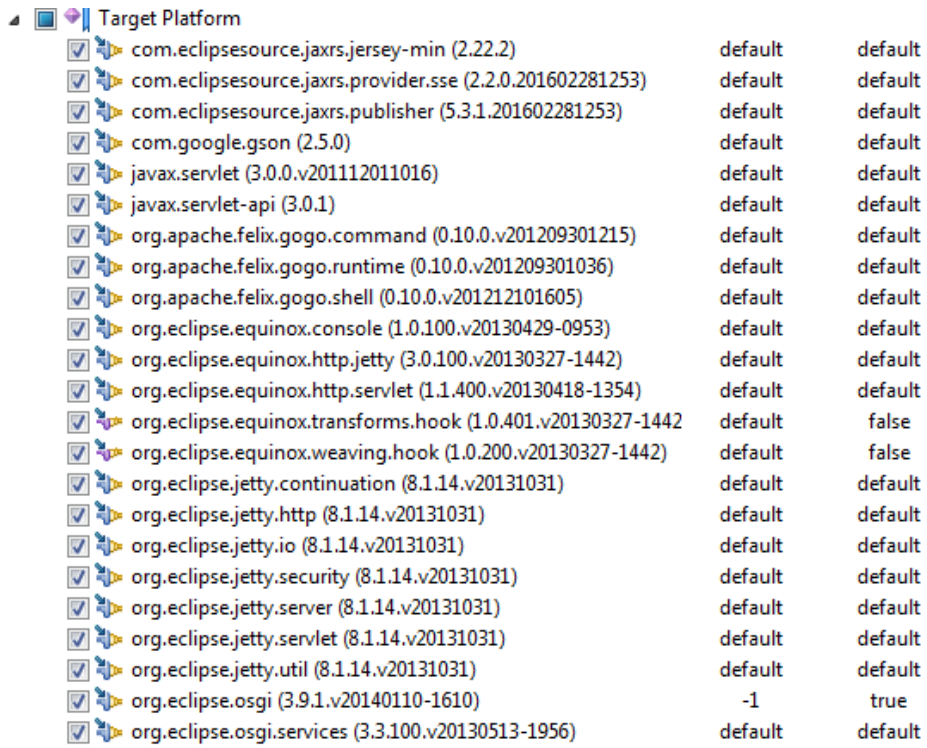


Figure 4.7: Target Platform Dependencies.

4.2.3 Sensors Bundles

The creation of Sensor Bundles that corresponds to the Sensor build that had previously was proportionate to the SSE behavior. The bundle creation is very flexible procedure. A bundle developer can combine technologies to fit in their Sensor Build. But, for this Sensor build, the activator class is standard and contains two standard functions - the start and stop function exact like the Philips Hue Bundle. But, in the "core" method in each Sensor bundle there are two functions. The First function is REST based function that returns the Sensor Type (e.g. 1: Motion sensor, 2: Distance sensor, 3: Light Sensor e.c.t.). The Second Function is the "SSE function" that is contains the SSE functionality and implementation in some cases, depending on where it will be applied. For, this sensor build, the usage of retransmission logic took place, to be able to create the bundle method. So, in the same method there is SSE consume and SSE Produce at the same time. Also, this method is threaded internally, after a point, to be able to serve SSE to more than one Client.

For more explanation, the Appendix B contains the core class files that correspond to the Sensors OSGi bundles (for the Motion Sensor, Distance Sensor and Light Sensor).

As a result, the Sensor OSGi path general hierarchy is shown in the figure 4.8

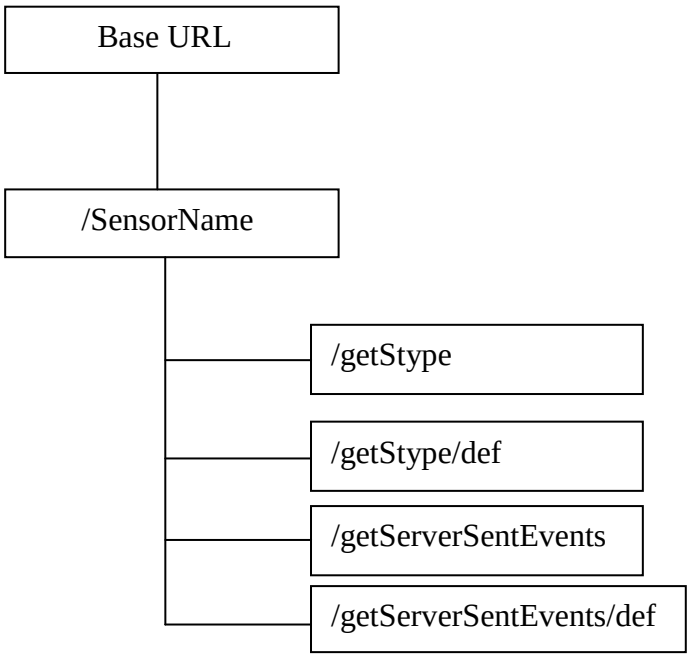


Figure 4.8: Sensor OSGi path general hierarchy.

Chapter 5

Use Case Demonstration

5.1 Use Case Scenario

There are a lot of use case scenarios that are applicable, and check the ARM functionality. For this thesis, any Local Area Network can be used, to let the devices to communicate and let the Middleware API to expose the Smart Modules functionality. Such a network can be connected to the Internet and expose the Smart Modules functionality though Internet as well. The scenario demonstrated in the next section uses a Philips Hue OSGi Bundle, a Motion Sensor OSGi Bundle, a Distance Sensor Motion OSGi Bundle and a Light Motion OSGi bundle. The connection with middleware is established using an HTML5 client, but any other Client Application can be used for the demonstration. The HTML 5 client enables the functionality of Client JS, which contains three static rules, for demonstration purposes.

5.2 Use Case Preparation Procedure

In the demonstration presented in this chapter, the whole process will be followed, from the implementation of a bundle providing functionality, up to the usage from the HTML5_Client3 in this scenario. Two walkthrough videos [5] [6] are also great clarification tools for all the connecting parts of the successful usage and applicability of the ARM.

5.2.1 OSGi Bundle

The system testing is done using some bundles. These bundles can be implemented in the same or other eclipse environment and workspace that the ARM is located. That bundle can be any bundle packed in .jar file that is satisfy the ARM requirements. In that OSGi Bundle exist the implementation of the functionality that must be exposed form ARM. In the figure 5.1 exist the Code snippet from Motion Sensor. Also, all the Bundles core Sensor class files that is used in the use case scenario can is located in the Appendix B.

```
import osgi_annotations.interfaces.ClassDescription;
import osgi_annotations.interfaces.MethodDescription;

@ClassDescription(value = "MotionSensor is an osgi bundle that represent a motion sensor. "+"
    This plugin provide SSE that feeded from the sensor and then serve it to client.")
public class MotionSensor extends ResourceConfig {
    private static String sensorA = "smotion";
    private String LocalNetworkIP = "192.168.1.135"; // "192.168.31.161";

    @MethodDescription(value = "Return the Sensor type. 1 : Motion sensor, 2: Distance sensor 3: Light Sensor")
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public int getStype() {
        int stype = 1; /// motion sensor
        return stype;
    }

    @MethodDescription(value = "Return the Server Sent Event from the motion sensor. The event name is 'motion'. "+
        "When motion is present then a 'motion' data pass in else, it pass a 'none' data as message.")
    @GET
    @Consumes(SseFeature.SERVER_SENT_EVENTS)
    @Produces(SseFeature.SERVER_SENT_EVENTS)
    public EventOutput getServerSentEvents() {

        final EventOutput eventOutput = new EventOutput();
```

Figure 5.1: Code snippet from Motion Sensor.

[5] https://www.youtube.com/watch?v=cEEjh0TPtVI
[6] https://www.youtube.com/watch?v=Jb6cBs8qi60

In this example, annotations exist in the bundle for documenting the bundle and for the method implementing the functionality. Annotations exist on both SSE and REST functionalities as well. Similarly, the other two Sensor main class file is annotated in the same way as well.

Finally, the bundle must be exported as “Deployable plug-ins and Fragments” and saved as JAR file, for distribution.

5.2.2 Addition of XML File

The XML file must be build form the Exported Package name and the Bundle Symbolic name, as specified in the Bundle manifest file.

In Figure 5.2, there are the XML files for each one Sensor bundles. Here, using the names that have use while developing the bundle and those names that also exist in the manifest file, that is created. Notice that the <fullname> encloses text, which has the pattern “<Exported-Package>.<Bundle-Symbolic-Name>”.

The XML file has to be included in the .jar file of the bundle that will be distributed. Bundle Developer must include one XML for each Bundle.

```
<bundle-definition>
    <fullname>
        osgi_DistanceSensor.DistanceSensor
    </fullname>
    <filename>
        DistanceSensor_1.0.0.jar
    </filename>
</bundle-definition>
```

```
<bundle-definition>
    <fullname>
        osgi_LightSensor.LightSensor
    </fullname>
    <filename>
        LightSensor_1.0.0.jar
    </filename>
</bundle-definition>
```

```
<bundle-definition>
    <fullname>
        osgi_MotionSensor.MotionSensor
    </fullname>
    <filename>
        MotionSensor_1.0.0.jar
    </filename>
</bundle-definition>
```

Figure 5.2: Sensors (Distance Light Motion Sensors) XML Files.

5.2.3 Input of OSGi Bundle to Middleware

When a valid bundle file is deployed in the middleware, ARM detects its functionalities of that bundle and automatically re-configure by creating the required class files and redeploy the middleware Web API to include new REST and SSE calls accordingly.

Also, all the definition REST calls is created for each new web method, REST call and SSE call, to let the Client Developer know all required information for the device. ARM will inform fully the Client Developers and they don't require communicating with the IoT device itself. The Definition for each SSE and REST call is located under: "<baseURL>/<BundleName>/<MethodName>/def".The definitions of the web methods is located under: "<baseURL>/<BundleName>".

Middleware empty list with available OSGi bundles is shown in the figure 5.3.

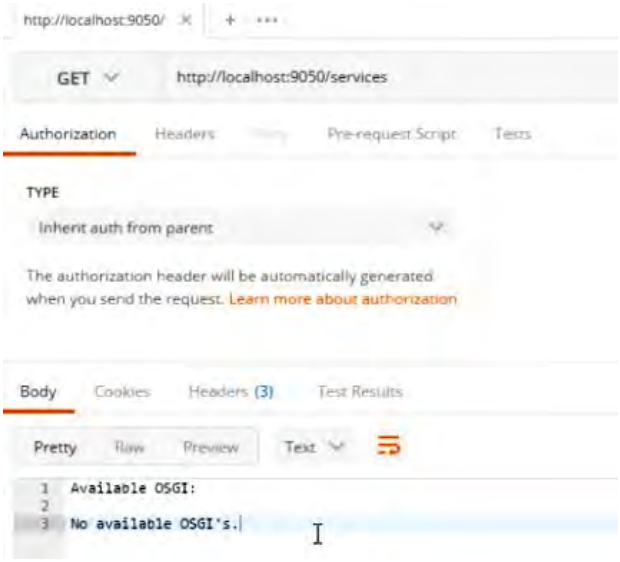


Figure 5.3: Middleware empty list with available OSGi bundles.

Afterwards, the Installation of Philips Hue Light take place and once the Philips Hue Light bundle is deployed in the middleware, the ARM finds and parses the XML file containing the bundle's full name. If the Philips Hue Light bundle is not already installed as happen in this use case scenario, the middleware will automatically install

and start it. ARM detects all the available functionalities from the annotations defined and re-configure the middleware by injecting the new Philips Hue Light functionalities as REST web methods and publishing their paths. In the figure 5.4 is the middleware updated list that include the Smart Philips Hue Light.

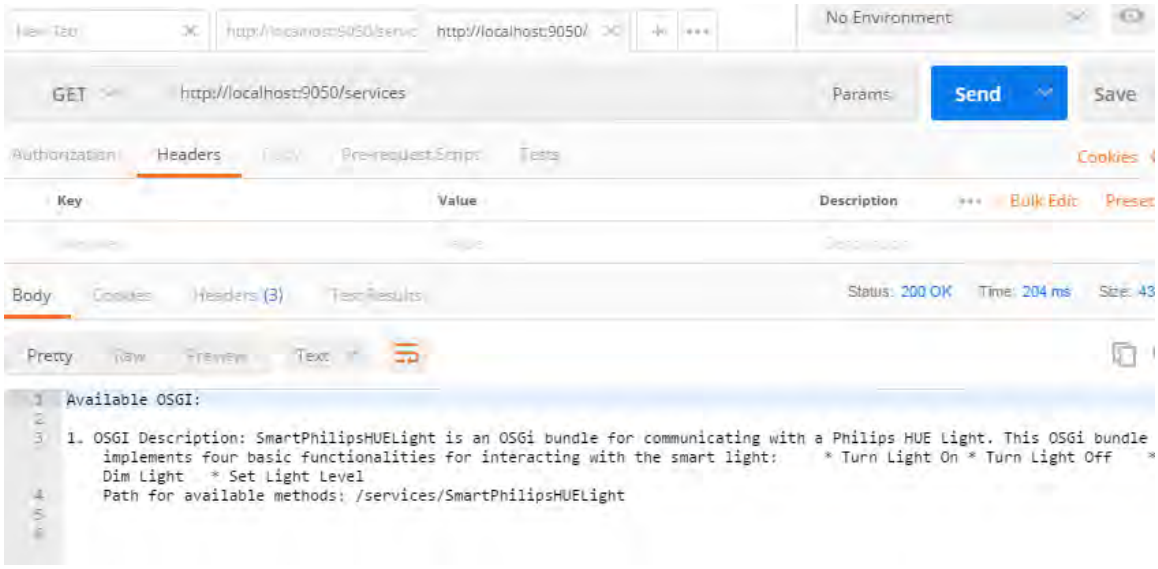


Figure 5.4: Middleware list with available OSGi bundles after Smart Philips Hue Light Installation.

Next, the Installation of Motion Sensor take place and once the Motion Sensor bundle is deployed in the middleware, the ARM finds and parses the XML file containing the bundle's full name. If the Motion Sensor bundle is not already installed as happen in this use case scenario, the middleware will automatically install and start it. ARM detects all the available functionalities from the annotations defined and re-configure the middleware by injecting the new Motion Sensor functionalities as REST & SSE web methods and publishing their paths. In the figure 5.5 is the middleware updated list that include the Motion Sensor. The Smart Philips Hue Bundle is not installed again.

Figure 5.5: Middleware list with available OSGi bundles after Motion Sensor install.

Finaly, the Installation of Light and Distance Sensor take place and once the Light and Distance Sensor bundles are deployed in the middleware, the ARM finds and parses the XML file for each one bundle and containing the bundles's full names. If the Light and Distance Sensor bundle are not already installed as happen in this use case scenario, the middleware will automatically install and start them. ARM detects all the available functionalities from the annotations defined and re-configure the middleware by injecting the new Light Sensor, Distance Sensor functionalities as REST & SSE web methods and publishing their paths. In the figure 5.6 is the middleware updated list that include the Light and Distance Sensor. The Smart Philips Hue and Motion Sensor Bundle is not installed again.

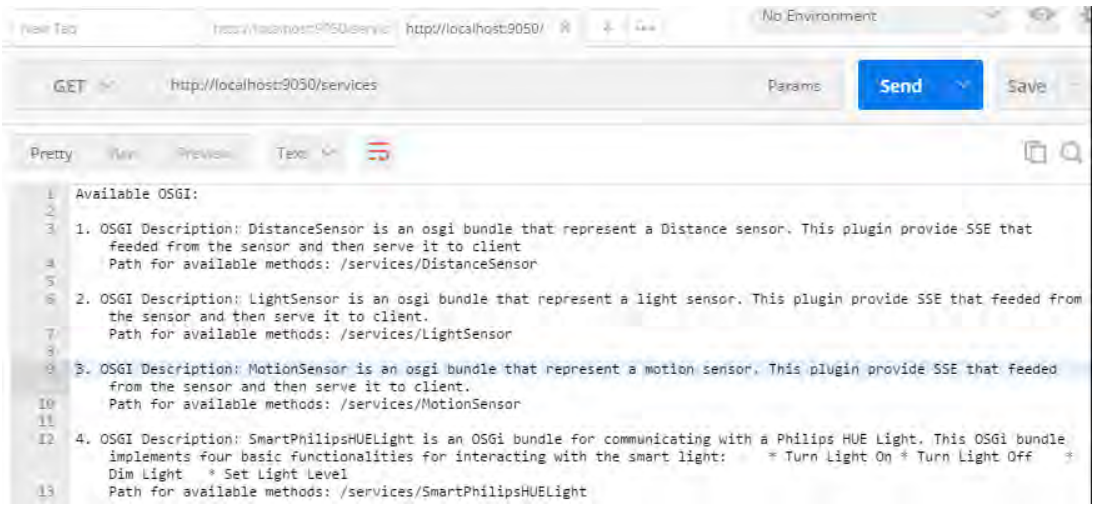


Figure 5.6: Middleware list with available OSGi bundles after Light and Distance Sensor Installation.

Lastly, navigating in the Middleware list with available REST & SSE calls for all the installed bundles, can be achived easily using Postman. For example, in figure 5.7 there are all the available REST & SSE calls for Distance Sensor Bundle.

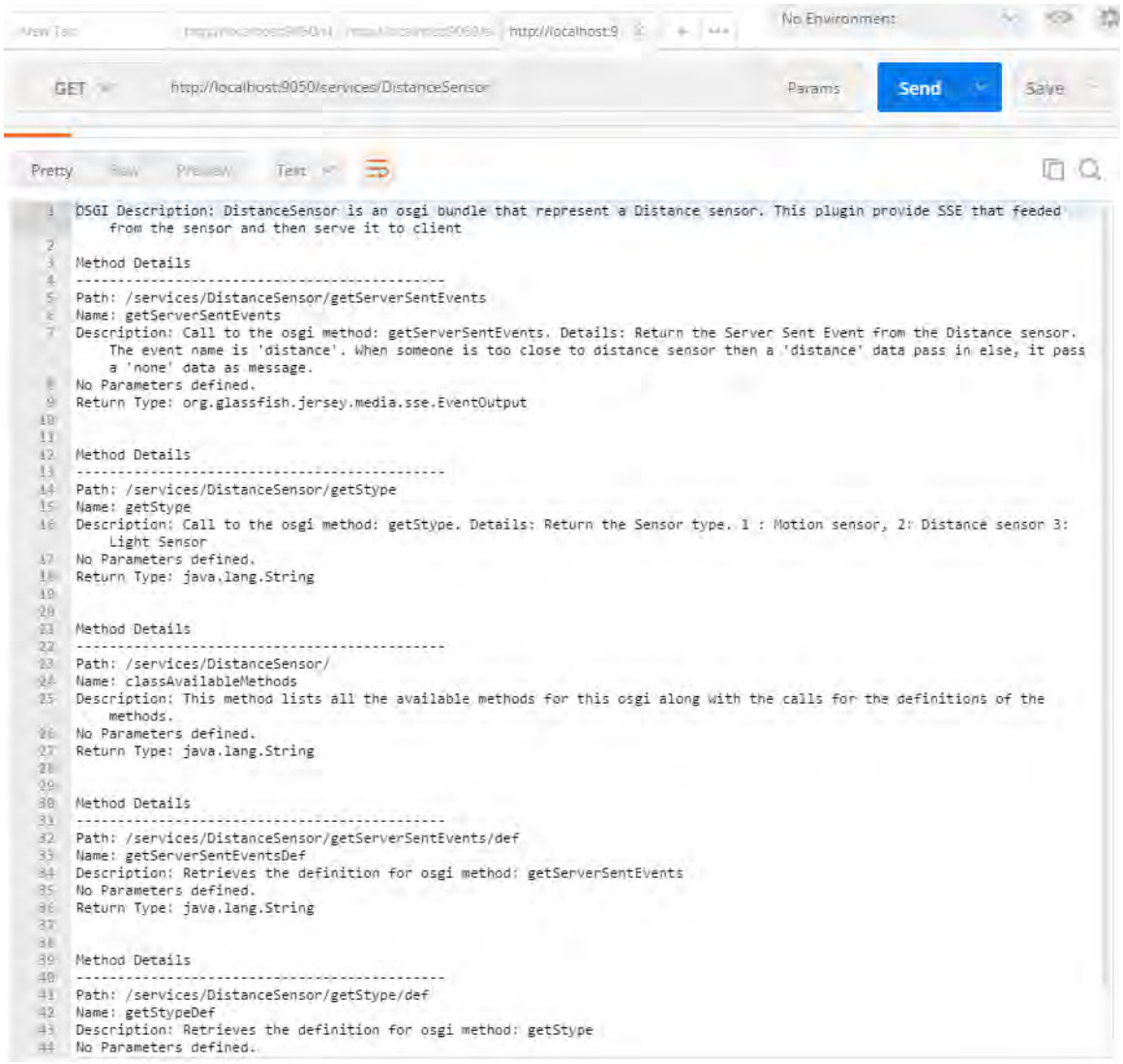


Figure 5.7: Middleware list with available REST & SSE calls for Distance Sensor Bundle.

5.2.4 Automatically Configured and Generated URLs

The Middleware behavior is automatic. So, a list with REST & SSE accessible web methods will be configured and deployed, each time an OSGi bundle is given to the middleware.

As a result, the Sensor OSGi path general hierarchy is shown in the figure 5.8, which list all the available REST & SSE web methods for Philips Hue Smart Light, Motion Sensor, Light Sensor and Distance Sensor. All that Generated URLs together among with the Base URL creates the final ARM API, for this use case scenario.

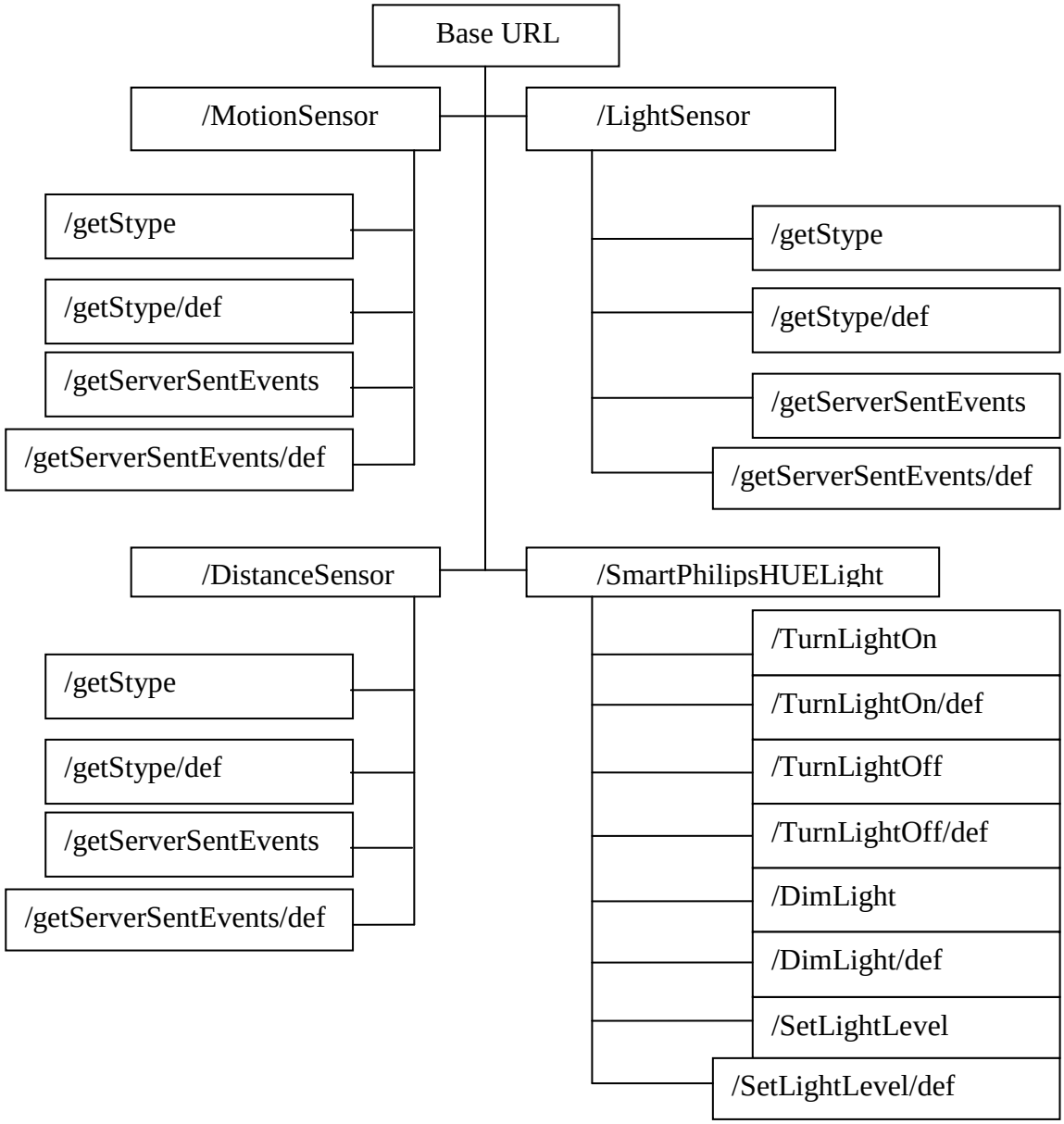


Figure 5.8: List with all available web methods for this demonstration.

5.2.5 Communication with Middleware from Client

To be able to complete a review for the ARM state a Client, that can consumes SSE, is created. In this Use Case Senario, the Client is an HTML 5 Client that is working alongside with Javascript (Client_JS.js), to create an event driven HTML 5 application that can test the current Middleware state. Having the HTML 5 Client Version 3, can check the SSE Functionality for all Sensors Modules together and seperatly. HTML 5 Client Version 3 source code and the Client.js source code is located in the Appendix C. A Screnshot of the HTML 5 Client Version 3 is in the figure 5.9 .

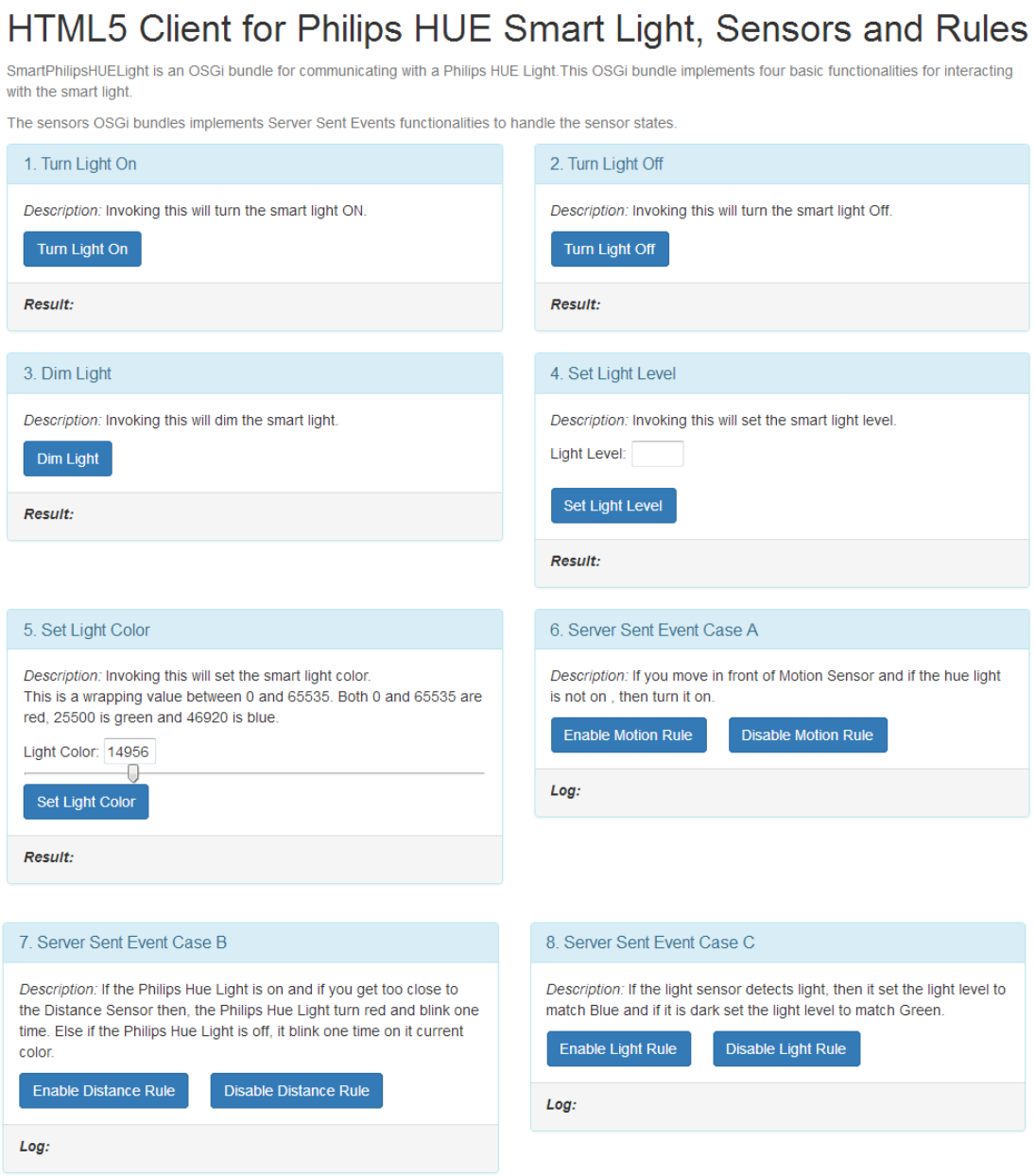


Figure 5.9: HTML 5 Client Version 3.

5.3 Demonstration of Use Case

When the use case preparation is done, then the scenario can run. The Demonstration procedure is recorded in two videos. The preparation of one short demonstration video[7] and one extended video[8] is done, through this this, to present with the best possible way the system in action. For demonstration is declared three static rules, but this rules can created automatic, maintained, stored and be dynamic, when the rule engine will take place in the future. The API exploration is done using postman. Initially, is presented the project Setup. The setup including the Orange Pi and Philips hue and the setup is explained little bit. Then, the HTML 5 application server is started up and arranged to present the system in four stages. Initially, in the in the use case scenario is presented the Middleware "colt start" when no one OSGi Bundle (Smart Module) is installed yet. Then, is presented the installation of Philips Hue Bundle and explore the Middleware API and show the regenerated API with the Philips Hue exposed functionalities. In the HTML 5 Client application is presented the on/off functionality for this Bundle (Smart Module). Next, is presented the installation of Motion Sensor. Here is presented the Middleware automatically regeneration of its API and presented the exploration of the new functionalities that is exposed in its API. In the HTML 5 application is presented the usage of the first static rule, which is when a motion is detected for the first time and if the Philips Hue Light is not on, then turn on the Philips Hue Light and then disconnect with the respective SSE and don't make any other changes. Finally, is presented the Light Sensor and Distance Sensor Bundles installation. Here is presented the Middleware automatically regeneration of its API and presented the exploration of the new functionalities that is exposed for that two new Bundles in ARM API. In the HTML 5 application is presented the usage of the second and third static rules. First is presented the third rule and if the light sensor detects light, then it set the light level to match Blue and when is selected if it is dark set the light level to match Green. Finally, is presented the second rule and if the Philips Hue Light is on and when is selected if you get too close to the Distance Sensor then, the Philips Hue Light turn red and blink one time. Else, if the Philips Hue Light is off and you come too close to distance sensor, Philips Hue Light blink one time on it current color.

[7] https://www.youtube.com/watch?v=cEEjh0TPtVI
[8] https://www.youtube.com/watch?v=Jb6cBs8qi60

Chapter 6

Evaluation

6.1 Evaluation Method

The ideal evaluation method, for this work is the Opinion based evaluation method, because a questionnaire is used in it. This technique, give the ability to collect feedback for the system from users and current system state. Also this technique gives many information and suggestions on possible ways to improve the system. The user's opinion matter, because gives measurements for system efficiency, adequate and complete. Also measuring user's satisfaction, user's reaction and user's acceptance, creates a better opinion for the system. [18] [19]

6.2 Evaluation Questionnaire

In the Questionnaire[9] there is an introduction paragraph and a small video in the beginning to get the responder in the context quickly. Afterwards, there are 8 Questions and finally a place for comments and suggestions. The questions are about the user acceptance (e.g. check difficulty, check reaction and acceptance) and user opinion (e.g. what reviewee believes) about the system.

The questionnaire is listed in the next page.

[9] The questionnaire is located under the following link: https://docs.google.com/forms/d/e/1FAIpQLScwmajWGSisxJHFTOm6nK8Nwl2gJukM__9TSeqd5k3IdWWsEA/viewform

Questionnaire:

- How difficult do you find creating cross platform client applications (e.g. Android, iOS, Web) using RESTful APIs?

Difficult 1 2 3 4 5 Easy

- How difficult do you find creating cross platform client applications (e.g. Android, iOS, Web) using language Specific Libraries (.jar, .dll)?

Difficult 1 2 3 4 5 Easy

- Do you find it useful that the ARM can automatically generate the REST APIs & Server Sent Events?

Yes No

- How difficult do you find to Communicate with IoT Devices using REST APIs & Server Sent Events?

Difficult 1 2 3 4 5 Easy

- Do you believe is necessary to communicate both with Software Services and IoT Devices using REST APIs & Server Sent Events?

Not Necessary 1 2 3 4 5 Necessary

- Do you believe that the ARM enables developers to communicate and control IoT Devices Easy?

Not At All 1 2 3 4 5 Very Much

- Do you believe that the ARM enables developers to define & implement dependencies (interaction) between heterogeneous IoT Devices easily?

Not At All 1 2 3 4 5 Very Much

- Do you believe that the ARM functionality is useful?

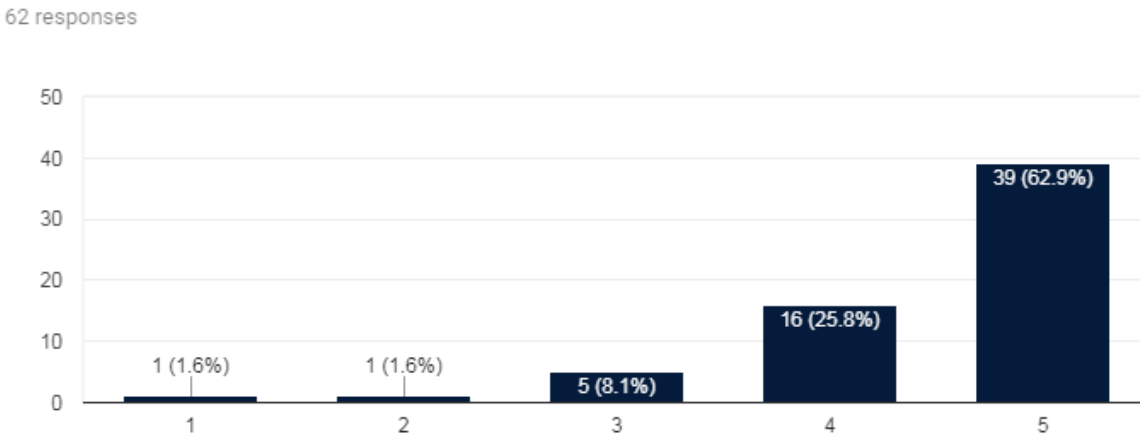
Yes No

- Comments and/or Suggestions

6.3 Evaluation Results

In this evaluation that is shown in the figures (Figure 6.1 – Figure 6.9), are collected the results below:

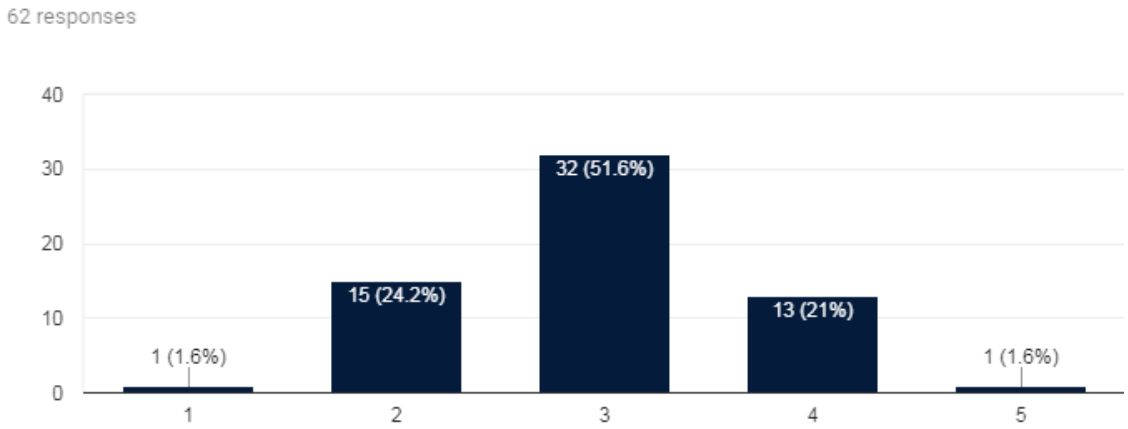
- *How difficult do you find creating cross platform client applications (e.g. Android, iOS, Web) using RESTful APIs?*



(Difficult 1 2 3 4 5 Easy)

Figure 6.1: First Question Results.

- *How difficult do you find creating cross platform client applications (e.g. Android, iOS, Web) using language Specific Libraries (.jar, .dll)?*



(Difficult 1 2 3 4 5 Easy)

Figure 6.2: Second Question Results.

- Do you find it useful that the ARM can automatically generate the REST APIs & Server Sent Events?

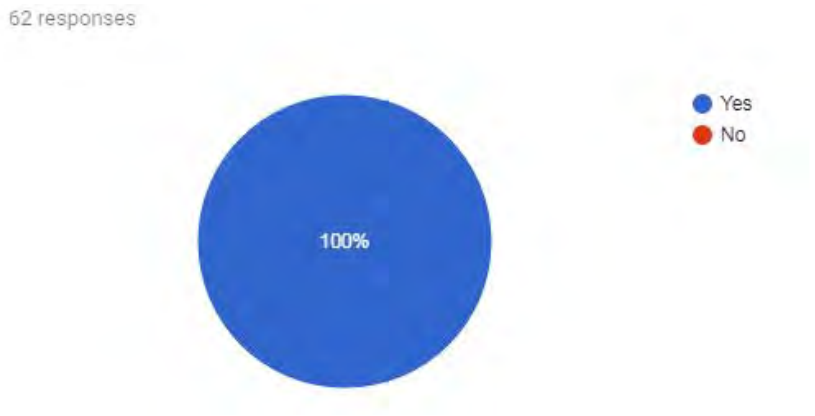
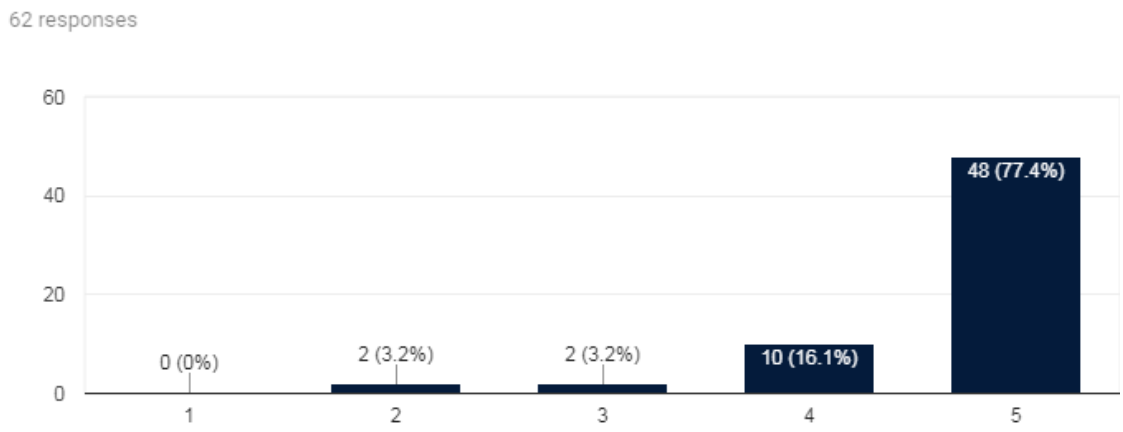


Figure 6.3: Third Question Results.

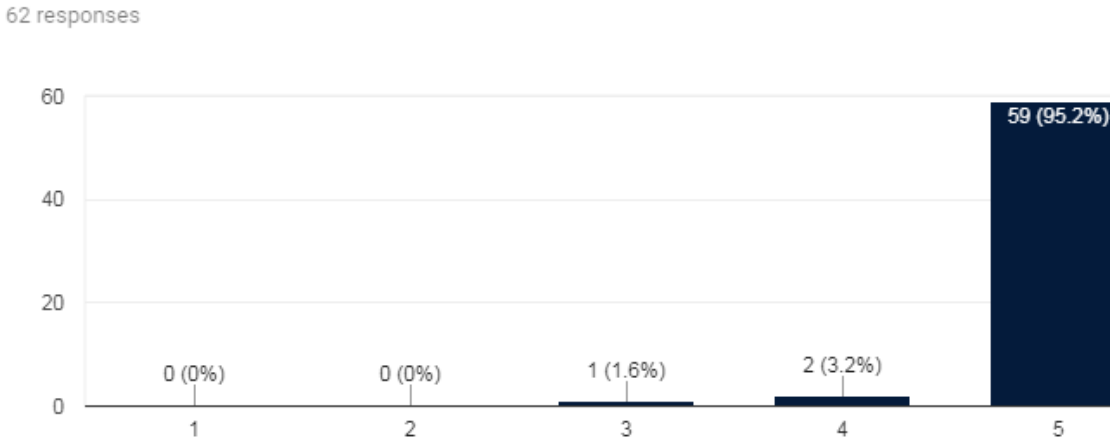
- How difficult do you find to Communicate with IoT Devices using REST APIs & Server Sent Events?



(Difficult 1 2 3 4 5 Easy)

Figure 6.4: Fourth Question Results.

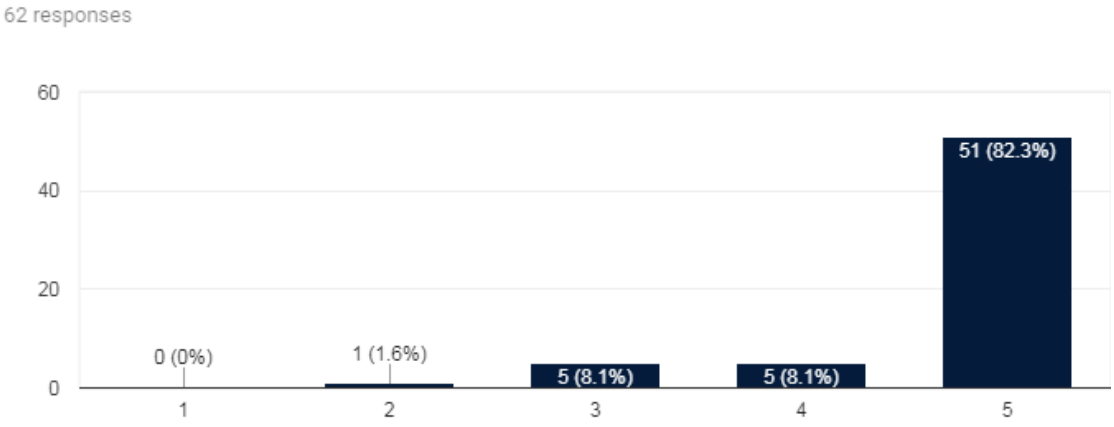
- Do you believe is necessary to communicate both with Software Services and IoT Devices using REST APIs & Server Sent Events?



(Not Necessary 1 2 3 4 5 Necessary)

Figure 6.5: Fifth Question Results.

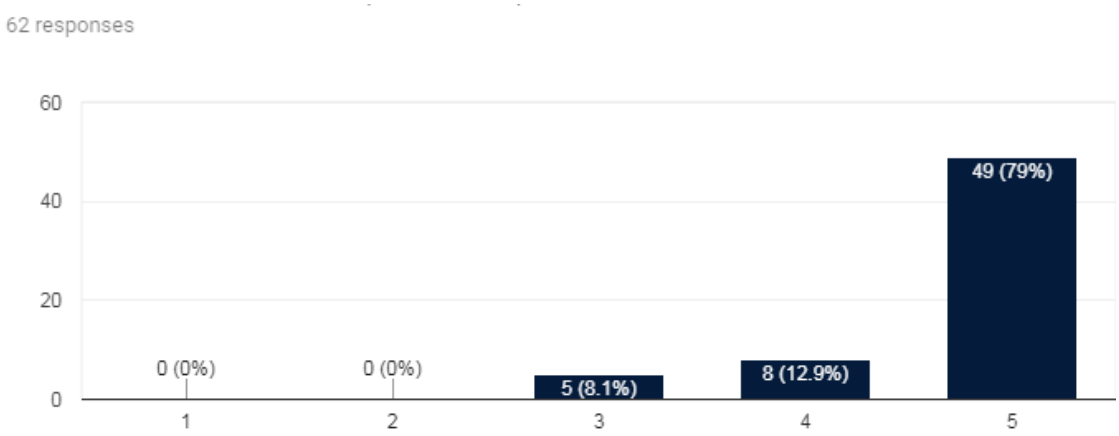
- Do you believe that the ARM enables developers to communicate and control IoT Devices Easy?



(Not At All 1 2 3 4 5 Very Much)

Figure 6.6: Sixth Question Results.

- Do you believe that the ARM enables developers to define & implement dependencies (interaction) between heterogeneous IoT Devices easily?



(Not At All 1 2 3 4 5 Very Much)

Figure 6.7: Seventh Question Results.

- Do you believe that the ARM functionality is useful?

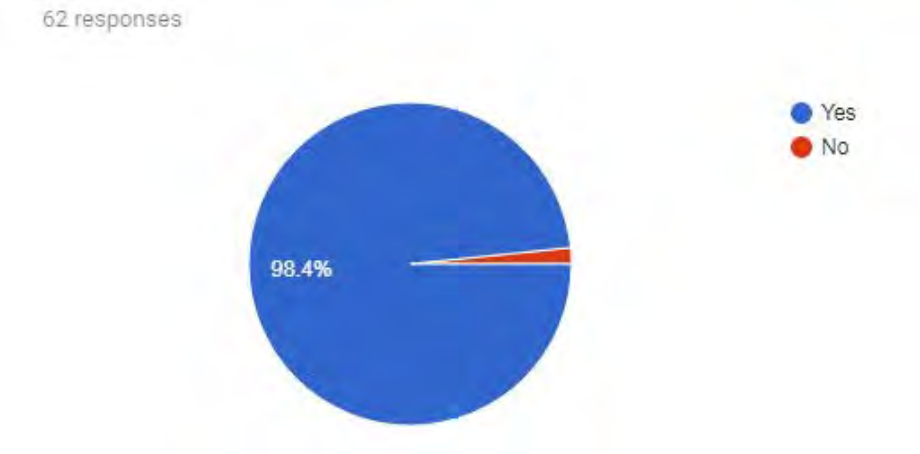


Figure 6.8: Eighth Question Results.

- *Comments and/or Suggestions*

5 responses

ARM functionality is useful only for very specific cases where very complex scenario requires it.
The beauty of IoT is to me, to simply interact with hardware rather than software.
Would be nice to have a web interface that can be used, maybe this has been done.
Points to consider: (1) security layer (2) swagger integration (popular framework in order to define rest API's) (3) auto discovery within a network? (Similar to apple's bonjour).
Very Good, If you have a lot of Devices to communicate with.

Figure 6.9: Comments and/or Suggestions.

From the above results, the ARM system tends to be a useful tool for Client Application Developers. Even though there are different values in the results, the average results, for ARM, are positive. Especially, a big amount of reviewee supports that is easier to create Client Application (e.g. Android, iOS, Web) using restful API's and SSE instead of using language specific libraries. As for the API automatic regeneration, all of the reviewees responded that is a useful. Also, they claim that there isn't find difficulty to communicate with IoT devices using the REST API's and SSE and that is a necessary feature is such a middleware. The support that ARM is an easy to use middleware, that let the developers communicate heterogeneous and non-heterogeneous with IoT Devices easily. A big amount of reviewees find that the ARM functionality is useful.

Despite the positive responses, form the questionnaire results feedback about future improvement that can be done in the system. For example, the swagger integration, the addition of security features (this is able with com.eclipsesource.jaxrs.provider.security [20]). They proposed also, the addition of Web Interface to create the connections on the system (that is planned to do when the Rules engine will be created in the future). Finally, it proposed the addition of an automatic discovery system of IoT devices in the network, to make the Middleware super easy to use.

Also, a small amount of the reviewees support that is best to communicate directly with the IoT devices, and based on the responses expose rest APIs and SSE from that.

Finally, an amount of the reviewees support that such a middleware can present only when a lot of IoT Devices exists interconnected. But this is planned to happen in the near future, because more and more companies will create more IoT devices that will have the ability to be Smart and connected to the Internet.

As every research on new technology has variations, ours also expected that would have deviations, because this is based on the developer's acceptance of this new technology. The fact that the programmer who took time to learn a technique as well uses a philosophy is giving difficulty to have to re-start something new.

What may have been enough is the fact that a Middleware can be considered a single point of failure but because of the way OSGi works (e.g. an OSGi Smart Module or device, but the rest of the Middleware will still work) and the fact that a backup system can be used to redirect the Client Application requests, this can blunt the problem.

On the other hand, all the answers are important, because is helpful for the system improvement and so that will be even more friendly and acceptable System for developers.

All the above delivers the feeling that ARM is a useful Middleware. Especially now when the IoT tends to grow and have more devices connected and a lot of new IoT devices are appear every single day.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Internet of Things tends to be more complex and demand. To be able to face this demand a Middleware is essential. This thesis shows that ARM can bring heterogeneous devices together. The Middleware stays a simplify solution and help Client application developers to write easily its applications and reduce the learning curve. The Middleware have the ability to handle a lot of OSGi components including those that have SSE functionality inside. When the OSGi Bundles (smart modules) is deployed the Middleware find out its available capabilities and reconfigure itself by re-generating and re-deploying the middleware WEB API. Because It's an auto adjustable system automatically adapt this changes and expose the functionalities of the smart modules through REST calls and/or SSE, when is applicable.

Using such a system, there are a lot of advantages. There are a lot of automated tasks in the system. The middleware now support various sensor types and expose them functionalities using Server Sent Events. Also can support heterogeneous Sensor types, Sensor that not support SSE de facto and even more software Services. ARM Developers creating a correct bundle will lead the system to create SSE on the fly, at Middleware runtime from the source of them choice.

The Middleware functionality is beneficial for both Client application developers (e.g. iOS developers, Android developers, WEB developers etc.) and ARM Developers as well.

7.2 Future Work

In this thesis, ARM is updated to make the functionalities of the bundles available through SSE and/or REST calls, when is applicable. This Middleware can be upgraded in many ways to be a powerful Middleware for the IoT.

Initially, a long term storage solution for the rules must be implementing. One approach is the Reasoner Plugins (see figure 3.2), which will store some rules in the Middleware, so the functionality that each user want to be saved, can be saved and be widely accessible. Another approach is to build that logic on node.js or PHP and store the rules here. Also, building hybrid web applications compatible with mobile devices can be done easily and lets the user be able to set long term rules and short term rules, based on their needs. So user can set long term and short term associations between the bundles. Also developer can set automatic associations and bundle triggers internally inside the application, to satisfy the user needs. In both approaches including a dynamic rule engine, that can do automatic a lot of functionality to simplify the Developer work, will be a great addition to ARM.

Also, another important aspect is the middleware's expansion to support all REST API Methods and be compatible with more complicated hardware and software making the functionalities of those bundles available through REST natively.

Finally, the security in the middleware must be considered. Some important security features that can be added on the middleware are for example the possibility to intergrade the Jersey/JAX-RS security features that is located in the bundle com.eclipsesource.jaxrs.provider.security [20]. Also, adding CORS Filters in the Middleware will let ARM to be friendlier to web browsers.

Βιβλιογραφία

[1] Padraig Scully, “IoT Analytics”, [Online]. Available: https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/

[2] Jan Henrik Ziegeldorf1, Oscar Garcia Morchon and Klaus Wehrle, “Privacy in the Internet of Things: threats and challenges”. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.795

[3] Eclipse, [Online]. Available: http://eclipse.github.io/kura/

[4] Thibaut Le Guilly, Petur Olsen, Anders P. Ravn, Jesper Brix Rosenkilde, Arne Skou, “HomePort: Middleware for Heterogeneous Home Automation Networks”. Available: http://people.cs.aau.dk/~thibaut/papers/HomePort.pdf

[5] Achilleas Achilleos, Kyriaki Georgiou, Christos Markides, Andreas Konstantinidis, George A. Papadopoulos, “Adaptive Runtime Middleware: Everything as a Service”. Available: https://www.researchgate.net/publication/319572045_Adaptive_Runtime_Middleware_Everything_as_a_Service

[6] “cs.uci.edu”, [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[4] Roy Fielding, “Economies of scale”, [Online]. Available: http://roy.gbiv.com/untangled/2008/economies-of-scale

[5] Things every developer should know, “Twitter Inc.”, [Online]. Available: https://developer.twitter.com/en/docs/basics/things-every-developer-should-know

[6] Ian Hickson, “Server-Sent Events”, [Online W3C Working Draft]. Available: https://www.w3.org/TR/2009/WD-eventsource-20090421/

[7] “HTML Living Standard Event Source”, “WHATWG community”, [Online Community Spec]. Available: https://html.spec.whatwg.org/multipage/server-sent-events.html

[8] Mozilla, “MDN web docs”, “Server-sent events” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events

[9] Sunil Gulabani, “Developing RESTful Web Services with Jersey 2.0”, “Packt Publishing Limited”. Available: https://www.packtpub.com/application-development/developing-restful-web-services-jersey-20

[10] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine, 2000.

[11] P.J. Eby, “Python Software Foundation”, [Online PEP]. Available: https://www.python.org/dev/peps/pep-3333/

[12] Armbian, [Online]. Available: https://docs.armbian.com/

[13] Benoit Chesneau, “Gunicorn Org”, [Online]. Available: http://docs.gunicorn.org/en/latest/design.html

[14] Spirula, “Gunicorn Worker Types”, “Spirulasystems” [Online]. Available: https://www.spirulasystems.com/blog/2015/01/20/gunicorn-worker-types/

[15] Eclipse, [Online]. Available: https://projects.eclipse.org/projects/rt.equinox/releases/3.9.0

[16] H. Staudacher, “GitHub, Inc,” [Online]. Available: https://github.com/hstaudacher/osgi-jax-rs-connector

[17] Lars Vogel, Simon Scholz, “Vogella” [Online]. Available: http://www.vogella.com/tutorials/OSGi/article.html#running-a-stand-alone-osgi-server

[18] Günther Gediga, Kai-Christoph Hamborg, Evaluation of Software Systems, University of Ulster, Newtownabbey, BT 37 0QB, N.Ireland

[19] Barbara Ann Kitchenham, Evaluating Software Engineering Methods and Tool, Part 1: The Evaluation Context and Evaluation Methods, “NCC Services Ltd”, National Computing Center Oxford Road, Manchester, M1 7ED, England

[20] H. Staudacher, “GitHub, Inc,” [Online]. Available: https://github.com/hstaudacher/osgi-jax-rs-connector/wiki/security

Appendix A

Desktop.py :

```
#imports
import sys, datetime, time, flask
from pyA20.gpio import gpio
from pyA20.gpio import port
from flask_cors import CORS, cross_origin
from flask_sse import sse

#initializing
gpio.init()
gpio.setcfg(port.PA7,gpio.OUTPUT)
gpio.setcfg(port.PC4,gpio.INPUT)
gpio.setcfg(port.PC7,gpio.INPUT)

#for flusk
app = flask.Flask(__name__)

#for cors
cors = CORS(app)
app.config['CORS_HEADERS'] = 'Content-Type'

#counters
flagMotion = 2
flagDistance = 2
flagLight = 2
connaliveM = 0
connaliveD = 0
connaliveL = 0

#function rc_ctime
def rc_time():
    count = 0
    gpio.setcfg(port.PA6,gpio.OUTPUT)
    gpio.output(port.PA6,gpio.LOW)
    time.sleep(0.1)
    gpio.setcfg(port.PA6,gpio.INPUT)
    while(gpio.input(port.PA6) == gpio.LOW):
        count += 1
    return count

#intitializing
time.sleep(3)
```

```
42.
43.
44. #Motion Check
45. def event_stream1():
46.     try:
47.
48.         global connaliveM
49.         global flagMotion
50.         #yielding "event: motion\n"
51.         while True:
52.             check = gpio.input(port.PC7)
53.             if (check and not(flagMotion == 1)):
54.                 print("motion")
55.                 yield "event: motion\ndata: motion\n\n"
56.                 connaliveM = 0
57.                 time.sleep(1)
58.                 flagMotion = 1
59.             elif (check and (flagMotion == 1)):
60.                 print ("--Double motion---")
61.                 connaliveM += 1
62.                 time.sleep(1)
63.             elif (not(check) and not(flagMotion == 0)):
64.                 yield "event: motion\ndata: none\n\n"
65.                 connaliveM = 0
66.                 time.sleep(1)
67.                 flagMotion = 0
68.                 print("none motion")
69.             elif (not(check) and (flagMotion == 0)):
70.                 connaliveM += 1
71.                 time.sleep(1)
72.                 print("--Dub none motion--")
73.             else:
74.                 connaliveM += 1
75.                 time.sleep(1)
76.
77.             if (connaliveM >= 15):
78.                 connaliveM = 0
79.                 print("/////Sync alive////")
80.                 yield "event: motion\ndata: s\n\n"
81.                 #time.sleep(1)
82.
83.     except:
84.         #gpio.close()
85.         print("Motion Sensor Stoped!")
86.
87. #Distance Check
88. def event_stream2():
89.     try:
90.
```

```
91.     global connaliveD
92.     global flagDistance
93.     #yielding "event: distance\n"
94.     while True:
95.       checkb = not(gpio.input(port.PC4))
96.       if (checkb and not(flagDistance == 1)):
97.         print("close")
98.         yield "event: distance\ndata: close\n\n"
99.         connaliveD = 0
100.                time.sleep(1)
101.                flagDistance = 1
102.            elif (checkb and (flagDistance == 1)):
103.               print ("---Double close---")
104.               connaliveD += 1
105.               time.sleep(1)
106.           elif (not(checkb) and not(flagDistance == 0)):
107.              yield "event: distance\ndata: none\n\n"
108.              connaliveD = 0
109.              time.sleep(1)
110.              flagDistance = 0
111.              print("none close")
112.           elif (not(checkb) and (flagDistance == 0)):
113.              connaliveD += 1
114.              time.sleep(1)
115.              print("--Dub none close--")
116.           else:
117.              connaliveD += 1
118.              time.sleep(1)
119.
120.           if (connaliveD >= 15):
121.              connaliveD = 0
122.              print("/////Sync alive////")
123.              yield "event: distance\ndata: s\n\n"
124.              #time.sleep(1)
125.
126.       except:
127.         #gpio.close()
128.         print("Distance Sensor Stoped!")
129.
130.     #Light Check
131.     def event_stream3():
132.       try:
133.
134.         global connaliveL
135.         global flagLight
136.         #yielding "event: light\n"
137.         while True:
138.           #print rc_time() #was 100000
139.           checkc = True
```

```
140.                 if (rc_time() >= 10000):
141.                     checkc = True
142.                 else:
143.                     checkc = False
144.
145.                 #check
146.                 if (checkc and not(flagLight == 1)):
147.                     print("dark")
148.                     yield "event: light\ndata: dark\n\n"
149.                     connaliveL = 0
150.                     time.sleep(1)
151.                     flagLight = 1
152.                 elif (checkc and (flagLight == 1)):
153.                     print ("---Double dark---")
154.                     connaliveL +=1
155.                     time.sleep(1)
156.                 elif (not(checkc) and not(flagLight == 0)):
157.                     yield "event: light\ndata: none\n\n"
158.                     connaliveL = 0
159.                     time.sleep(1)
160.                     flagLight = 0
161.                     print("none dark")
162.                 elif (not(checkc) and (flagLight == 0)):
163.                     connaliveL +=1
164.                     time.sleep(1)
165.                     print("--dub none dark--")
166.                 else:
167.                     connaliveL +=1
168.                     time.sleep(1)
169.
170.                 if (connaliveL >= 15):
171.                     connaliveL = 0
172.                     print("/////Sync alive////")
173.                     yield "event: light\ndata: s\n\n"
174.                     #time.sleep(1)
175.
176.             except:
177.                 #gpio.close()
178.                 print("Light Sensor Stoped!")
179.
180.         @app.route('/smotion')
181.         @cross_origin()
182.         def smotion():
183.             return flask.Response(event_stream1(), mimetype="text/event-
    stream")
184.
185.         @app.route('/sdist')
186.         @cross_origin()
187.         def sdist():
```

```
188.            return flask.Response(event_stream2(), mimetype="text/event-
    stream")
189.
190.        @app.route('/slight')
191.        @cross_origin()
192.        def slight():
193.            return flask.Response(event_stream3(), mimetype="text/event-
    stream")
194.
195.        @app.route("/")
196.        @cross_origin()
197.        def hello():
198.            return """
199.                <!doctype html>
200.                <html>
201.                  <head>
202.                    <title>OPi Sensor SSE</title>
203.                  </head>
204.                  <body>
205.                    <h1>OPi Sensor<h1/>
206.                    <h3> Current sse routes is /smotion, /sdist, /slight </h3>
207.                  </body>
208.                </html>
209.            """
210.
211.        #app multithreaded
212.        #app.run(threaded=True)
213.        if __name__ == "__main__":
214.            app.run(host='0.0.0.0')
215.        app.run(threaded=True)
```

Appendix B

LightSensor.java :

```
1.  package osgi_LightSensor;
2.
3.  import java.io.IOException;
4.
5.  import javax.ws.rs.Consumes;
6.  import javax.ws.rs.GET;
7.  import javax.ws.rs.Produces;
8.  import javax.ws.rs.client.Client;
9.  import javax.ws.rs.client.ClientBuilder;
10. import javax.ws.rs.client.WebTarget;
11. import javax.ws.rs.core.MediaType;
12.
13. import org.glassfish.jersey.media.sse.EventInput;
14. import org.glassfish.jersey.media.sse.EventOutput;
15. import org.glassfish.jersey.media.sse.InboundEvent;
16. import org.glassfish.jersey.media.sse.OutboundEvent;
17. import org.glassfish.jersey.media.sse.SseFeature;
18. import org.glassfish.jersey.server.ResourceConfig;
19.
20. import osgi_annotations.interfaces.ClassDescription;
21. import osgi_annotations.interfaces.MethodDescription;
22.
23. @ClassDescription(value = "LightSensor is an osgi bundle that represent a light
    sensor. This plugin provide SSE that feeded from the sensor and then serve it to
    client.")
24. public class LightSensor extends ResourceConfig {
25.     private static String sensorC = "slight";
26.     private String LocalNetworkIP = "192.168.1.135";
27.
28.     @MethodDescription(value = "Return the Sensor type. 1 : Motion sensor, 2:
    Distance sensor 3: Light Sensor")
29.     @GET
30.     @Produces(MediaType.APPLICATION_JSON)
31.     public int getStype() {
32.         int stype = 3; /// motion sensor
33.         return stype;
34.     }
35.
36.     @MethodDescription(value = "Return the Server Sent Event from the light se
    nsor. The event name is 'light'. When is dark then a 'dark' data pass in else, it pas
    s a 'none' data as message.")
37.     @GET
```

```
38.    @Consumes(SseFeature.SERVER_SENT_EVENTS)
39.    @Produces(SseFeature.SERVER_SENT_EVENTS)
40.    public EventOutput getServerSentEvents() {
41.
42.        final EventOutput eventOutput = new EventOutput();
43.        Client client = ClientBuilder.newBuilder().register(SseFeature.class).build(
    );
44.        WebTarget target = client.target("http://" + LocalNetworkIP + "/" + sensor
    C);
45.        final EventInput eventInput = target.request().get(EventInput.class);
46.
47.        new Thread(new Runnable() {
48.            @Override
49.            public synchronized void run() {
50.                try {
51.                    while (!eventInput.isClosed()) {
52.                        // If it is not sychronise well use:
53.                        // Thread.sleep(500);
54.                        InboundEvent inboundEvent = eventInput.read();
55.
56.                        if (inboundEvent == null) {
57.                            // connection has been closed
58.                            break;
59.                        }
60.                        try {
61.                            //Handle event and retransmit it. You can add event id if requir
    ed.
62.                            //For Debug
63.                            //System.out.println(inboundEvent.readData(String.class));
64.                            OutboundEvent.Builder eventBuilder = new OutboundEvent.B
    uilder();
65.                            eventBuilder.name(inboundEvent.getName());
66.                            eventBuilder.data(inboundEvent.readData(String.class));
67.                            OutboundEvent event = eventBuilder.build();
68.                            eventOutput.write(event);
69.                        } catch (IOException e) {
70.                            try {
71.                                eventOutput.close();
72.                                eventInput.close();
73.                            } catch (IOException ioClose) {
74.                                throw new RuntimeException("Error when closing the event
     output internal.", ioClose);
75.                            }
76.                            throw new RuntimeException("Error when writing or reading t
    he event.", e);
77.                        } catch (Exception e) {
78.                            e.printStackTrace();
79.                        }
80.                    }
```

```
81.
82.                 } catch (Exception e) {
83.                     e.printStackTrace();
84.                 } finally {
85.                     try {
86.                         if (!eventOutput.isClosed()) {
87.                             eventOutput.close();
88.                         }
89.                         if (!eventInput.isClosed()) {
90.                             eventInput.close();
91.                         }
92.                     } catch (IOException ioClose) {
93.                         throw new RuntimeException("Error when closing the event outp
    ut.", ioClose);
94.                     }
95.                 }
96.             }
97.         }).start();
98.         return eventOutput;
99.     }
100.
101.         }
```

MotionSensor.java :

```
1.  package osgi_MotionSensor;
2.
3.  import java.io.IOException;
4.
5.  import javax.ws.rs.Consumes;
6.  import javax.ws.rs.GET;
7.  import javax.ws.rs.Produces;
8.  import javax.ws.rs.client.Client;
9.  import javax.ws.rs.client.ClientBuilder;
10. import javax.ws.rs.client.WebTarget;
11. import javax.ws.rs.core.MediaType;
12.
13. import org.glassfish.jersey.media.sse.EventInput;
14. import org.glassfish.jersey.media.sse.EventOutput;
15. import org.glassfish.jersey.media.sse.InboundEvent;
16. import org.glassfish.jersey.media.sse.OutboundEvent;
17. import org.glassfish.jersey.media.sse.SseFeature;
18. import org.glassfish.jersey.server.ResourceConfig;
19.
20. import osgi_annotations.interfaces.ClassDescription;
21. import osgi_annotations.interfaces.MethodDescription;
22.
23. @ClassDescription(value = "MotionSensor is an osgi bundle that represent a mo
    tion sensor. This plugin provide SSE that feeded from the sensor and then serve i
    t to client.")
24. public class MotionSensor extends ResourceConfig {
25.     private static String sensorA = "smotion";
26.     private String LocalNetworkIP = "192.168.1.135"; // "192.168.31.161";
27.
28.     @MethodDescription(value = "Return the Sensor type. 1 : Motion sensor, 2:
    Distance sensor 3: Light Sensor")
29.     @GET
30.     @Produces(MediaType.APPLICATION_JSON)
31.     public int getStype() {
32.         int stype = 1; /// motion sensor
33.         return stype;
34.     }
35.
36.     @MethodDescription(value = "Return the Server Sent Event from the motion
    sensor. The event name is 'motion'. When motion is present then a 'motion' data
    pass in else, it pass a 'none' data as message.")
37.     @GET
38.     @Consumes(SseFeature.SERVER_SENT_EVENTS)
39.     @Produces(SseFeature.SERVER_SENT_EVENTS)
40.     public EventOutput getServerSentEvents() {
```

```
41.
42.         final EventOutput eventOutput = new EventOutput();
43.         Client client = ClientBuilder.newBuilder().register(SseFeature.class).build(
    );
44.         WebTarget target = client.target("http://" + LocalNetworkIP + "/" + sensor
    A);
45.         final EventInput eventInput = target.request().get(EventInput.class);
46.
47.         new Thread(new Runnable() {
48.             @Override
49.             public synchronized void run() {
50.                 try {
51.                     while (!eventInput.isClosed()) {
52.                         // If it is not sychronise well use:
53.                         // Thread.sleep(500);
54.                         InboundEvent inboundEvent = eventInput.read();
55.
56.                         if (inboundEvent == null) {
57.                             // connection has been closed
58.                             break;
59.                         }
60.                         try {
61.                             // Handle event and retransmit it. You can add event
62.                             // id if required.
63.                             // For Debug
64.                             // System.out.println(inboundEvent.readData(String.class));
65.                             OutboundEvent.Builder eventBuilder = new OutboundEvent.B
    uilder();
66.                             eventBuilder.name(inboundEvent.getName());
67.                             eventBuilder.data(inboundEvent.readData(String.class));
68.                             OutboundEvent event = eventBuilder.build();
69.                             eventOutput.write(event);
70.                         } catch (IOException e) {
71.                             try {
72.                                 eventOutput.close();
73.                                 eventInput.close();
74.                             } catch (IOException ioClose) {
75.                                 throw new RuntimeException("Error when closing the event
     output internal.", ioClose);
76.                             }
77.                             throw new RuntimeException("Error when writing or reading t
    he event.", e);
78.                         } catch (Exception e) {
79.                             e.printStackTrace();
80.                         }
81.                     }
82.
83.                 } catch (Exception e) {
84.                     e.printStackTrace();
```

```
85.                 } finally {
86.                     try {
87.                         if (!eventOutput.isClosed()) {
88.                             eventOutput.close();
89.                         }
90.                         if (!eventInput.isClosed()) {
91.                             eventInput.close();
92.                         }
93.                     } catch (IOException ioClose) {
94.                         throw new RuntimeException("Error when closing the event outp
    ut.", ioClose);
95.                     }
96.                 }
97.             }
98.         }).start();
99.         return eventOutput;
100.             }
101.
102.         }
```

DistanceSensor.java

```
1.  package osgi_DistanceSensor;
2.
3.  import java.io.IOException;
4.
5.  import javax.ws.rs.Consumes;
6.  import javax.ws.rs.GET;
7.  import javax.ws.rs.Produces;
8.  import javax.ws.rs.client.Client;
9.  import javax.ws.rs.client.ClientBuilder;
10. import javax.ws.rs.client.WebTarget;
11. import javax.ws.rs.core.MediaType;
12.
13. import org.glassfish.jersey.media.sse.EventInput;
14. import org.glassfish.jersey.media.sse.EventOutput;
15. import org.glassfish.jersey.media.sse.InboundEvent;
16. import org.glassfish.jersey.media.sse.OutboundEvent;
17. import org.glassfish.jersey.media.sse.SseFeature;
18. import org.glassfish.jersey.server.ResourceConfig;
19.
20. import osgi_annotations.interfaces.ClassDescription;
21. import osgi_annotations.interfaces.MethodDescription;
22.
23. @ClassDescription(value = "DistanceSensor is an osgi bundle that represent a D
    istance sensor. This plugin provide SSE that feeded from the sensor and then ser
    ve it to client")
24. public class DistanceSensor extends ResourceConfig {
25.     private static String sensorB = "sdist";
26.     private String LocalNetworkIP = "192.168.1.135";
27.
28.     @MethodDescription(value = "Return the Sensor type. 1 : Motion sensor, 2:
    Distance sensor 3: Light Sensor")
29.     @GET
30.     @Produces(MediaType.APPLICATION_JSON)
31.     public int getStype() {
32.         int stype = 2; /// motion sensor
33.         return stype;
34.     }
35.
36.     @MethodDescription(value = "Return the Server Sent Event from the Distanc
    e sensor. The event name is 'distance'. When someone is too close to distance se
    nsor then a 'distance' data pass in else, it pass a 'none' data as message.")
37.     @GET
38.     @Consumes(SseFeature.SERVER_SENT_EVENTS)
39.     @Produces(SseFeature.SERVER_SENT_EVENTS)
40.     public EventOutput getServerSentEvents() {
41.
```

```
42.         final EventOutput eventOutput = new EventOutput();
43.         Client client = ClientBuilder.newBuilder().register(SseFeature.class).build(
    );
44.         WebTarget target = client.target("http://" + LocalNetworkIP + "/" + sensor
    B);
45.         final EventInput eventInput = target.request().get(EventInput.class);
46.
47.         new Thread(new Runnable() {
48.             @Override
49.             public synchronized void run() {
50.                 try {
51.                     while (!eventInput.isClosed()) {
52.                         // If it is not sychronise well use:
53.                         // Thread.sleep(500);
54.                         InboundEvent inboundEvent = eventInput.read();
55.
56.                         if (inboundEvent == null) {
57.                             // connection has been closed
58.                             break;
59.                         }
60.                         try {
61.                             // Handle event and retransmit it. You can add event
62.                             // id if required.
63.                             // For Debug
64.                             // System.out.println(inboundEvent.readData(String.class));
65.                             OutboundEvent.Builder eventBuilder = new OutboundEvent.B
    uilder();
66.                             eventBuilder.name(inboundEvent.getName());
67.                             eventBuilder.data(inboundEvent.readData(String.class));
68.                             OutboundEvent event = eventBuilder.build();
69.                             eventOutput.write(event);
70.                         } catch (IOException e) {
71.                             try {
72.                                 eventOutput.close();
73.                                 eventInput.close();
74.                             } catch (IOException ioClose) {
75.                                 throw new RuntimeException("Error when closing the event
     output internal.", ioClose);
76.                             }
77.                             throw new RuntimeException("Error when writing or reading t
    he event.", e);
78.                         } catch (Exception e) {
79.                             e.printStackTrace();
80.                         }
81.                     }
82.
83.                 } catch (Exception e) {
84.                     e.printStackTrace();
85.                 } finally {
```

```
86.                 try {
87.                     if (!eventOutput.isClosed()) {
88.                         eventOutput.close();
89.                     }
90.                     if (!eventInput.isClosed()) {
91.                         eventInput.close();
92.                     }
93.                 } catch (IOException ioClose) {
94.                     throw new RuntimeException("Error when closing the event outp
    ut.", ioClose);
95.                 }
96.             }
97.         }
98.     }).start();
99.     return eventOutput;
100.            }
101.
102.        }
```

Appendix C

HTML5_Client3.html :

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <meta charset="ISO-8859-1">
5.  <meta content="text/html" http-equiv="Content-Type">
6.  <meta content="utf-8" http-equiv="encoding">
7.  <title>HTML5 Client for Philips HUE Smart Light</title>
8.  <script type="text/javascript" src="Scripts/jquery-3.2.0.min.js"></script>
9.  <script type="text/javascript" src="Scripts/Client_JS.js"></script>
10. <link type="text/css" rel="stylesheet" href="Style/Client_CSS.css" />
11.
12. <meta name="viewport" content="width=device-width, initial-scale=1">
13. </head>
14. <body>
15.    <div class="container-fluid">
16.       <div class="row">
17.          <div class="col-xs-12 col-md-12">
18.             <h1>HTML5 Client for Philips HUE Smart Light, Sensors and Rules<
    /h1>
19.          </div>
20.       </div>
21.       <div class="row">
22.          <div class="col-xs-12 col-md-12">
23.             <p class="text-
    muted">SmartPhilipsHUELight is an OSGi bundle for
24.                communicating with a Philips HUE Light.This OSGi bundle imple
    ments
25.                four basic functionalities for interacting with the smart light.</p>
26.             <p class="text-
    muted">The sensors OSGi bundles implements Server
27.                Sent Events functionalities to handle the sensor states.</p>
28.          </div>
29.       </div>
30.       <div class="row">
31.          <div class="col-xs-12 col-md-6">
32.             <div class="panel panel-info">
33.                <div class="panel-heading">
34.                   <h2 class="panel-title">1. Turn Light On</h2>
35.
36.                </div>
37.                <div class="panel-body">
38.                   <p>
```

```
39.                     <i>Description:</i> Invoking this will turn the smart light ON.

40.                 </p>
41.                 <button type="button" id="TurnLightOnBtn" name="Turn Light
    On"
42.                     onclick="Javascript:TurnLightOn('TurnLightOnResult')"
43.                     class="btn btn-
    primary" role="button">Turn Light On</button>
44.             </div>
45.             <div class="panel-footer">
46.                 <label><i>Result: </i></label> <span id="TurnLightOnResult">
    </span>
47.             </div>
48.         </div>
49.     </div>
50.     <div class="col-xs-12 col-md-6">
51.         <div class="panel panel-info">
52.             <div class="panel-heading">
53.                 <h2 class="panel-title">2. Turn Light Off</h2>
54.             </div>
55.             <div class="panel-body">
56.                 <p>
57.                     <i>Description:</i> Invoking this will turn the smart light Off.

58.                 </p>
59.                 <button type="button" id="TurnLightOffBtn" name="Turn Light
    Off"
60.                     onclick="Javascript:TurnLightOff('TurnLightOffResult')"
61.                     class="btn btn-
    primary" role="button">Turn Light Off</button>
62.             </div>
63.             <div class="panel-footer">
64.                 <label><i>Result: </i></label> <span id="TurnLightOffResult">
    </span>
65.             </div>
66.         </div>
67.     </div>
68. </div>
69. <div class="row">
70.     <div class="col-xs-12 col-md-6">
71.         <div class="panel panel-info">
72.             <div class="panel-heading">
73.                 <h2 class="panel-title">3. Dim Light</h2>
74.             </div>
75.             <div class="panel-body">
76.                 <p>
77.                     <i>Description:</i> Invoking this will dim the smart light.
78.                 </p>
79.                 <button type="button" id="DimLightBtn" name="Dim Light"
```

```
                    onclick="Javascript:DimLight('DimLightResult')"
                    class="btn btn-primary" role="button">Dim Light</button>
                </div>
                <div class="panel-footer">
                  <label><i>Result: </i></label> <span id="DimLightResult"></s
pan>
                </div>
              </div>
            </div>
            <div class="col-xs-12 col-md-6">
              <div class="panel panel-info">
                <div class="panel-heading">
                  <h2 class="panel-title">4. Set Light Level</h2>
                </div>
                <div class="panel-body">
                  <p>
                    <i>Description:</i> Invoking this will set the smart light level.

                  </p>
                  <span>Light Level: </span> <input type="text" id="lightLevelIn
put"
                    value="" style="width: 50px; height: 25px" /> <br /> <br />
                  <button type="button" id="SetLightLevelBtn" name="Set Light
Level"
                          onclick="Javascript:SetLightLevel('SetLightLevelResul
t', 'lightLevelInput')"
                          class="btn btn-
primary" role="button">Set Light Level</button>
                  </div>
                  <div class="panel-footer">
                    <label><i>Result: </i></label> <span id="SetLightLevel
Result"></span>
                  </div>
                </div>
              </div>
            </div>
            <div class="row">
              <div class="col-xs-12 col-md-6">
                <div class="panel panel-info">
                  <div class="panel-heading">
                    <h2 class="panel-title">5. Set Light Color</h2>
                  </div>
                  <div class="panel-body">
                    <p>
                      <i>Description:</i> Invoking this will set the smart lig
ht color.
                      <br />This is a wrapping value between 0 and 65535. B
oth 0 and
                      65535 are red, 25500 is green and 46920 is blue.
```

```
120.                     </p>
121.                     <span>Light Color: </span> <input type="text" id="light
    ColorInput"
122.                         value="14956" style="width: 50px; height: 25px" />
123.                     <!-- disabled="disabled" -->
124.                     <br /> <input type="range" min="0" max="65535" value
    ="14956"
125.                         step="100" onchange="showValue(this.value, 'lightCol
    orInput')" />
126.                     <button type="button" id="SetLightColorBtn" name="Di
    m Light"
127.                         onclick="Javascript:SetLightColor('SetLightColorResul
    t', 'lightColorInput')"
128.                         class="btn btn-
    primary" role="button">Set Light Color</button>
129.                 </div>
130.                 <div class="panel-footer">
131.                     <label><i>Result: </i></label> <span id="SetLightColor
    Result"></span>
132.                 </div>
133.             </div>
134.         </div>
135.         <div class="col-xs-12 col-md-6">
136.             <div class="panel panel-info">
137.                 <div class="panel-heading">
138.                     <h2 class="panel-
    title">6. Server Sent Event Case A</h2>
139.                 </div>
140.                 <div class="panel-body">
141.                     <p>
142.                         <i>Description:</i> If you move in front of Motion Sen
    sor and if
143.                         the hue light is not on , then turn it on. <br />
144.                     </p>
145.                     <button type="button" class="btn btn-primary"
146.                         onclick="Javascript:sse1()">Enable Motion Rule</butt
    on>
147.                     <span style="display: inline-block; width: 3%;"></span>
148.                     <!--
     <span style="display:block; height: YOURHEIGHT;"></span> -->
149.                     <button type="button" class="btn btn-primary"
150.                         onclick="Javascript:source1.close(); i=0;">Disable
151.                         Motion Rule</button>
152.                     <span> </span>
153.                 </div>
154.                 <div class="panel-footer">
155.                     <label><i>Log: </i></label> <span id="out1"></span>
156.                 </div>
157.             </div>
```

```
158.                 </div>
159.             </div>
160.             <div class="row">
161.                 <div class="col-xs-12 col-md-6">
162.                     <div class="panel panel-info">
163.                         <div class="panel-heading">
164.                             <h2 class="panel-
    title">7. Server Sent Event Case B</h2>
165.                         </div>
166.                         <div class="panel-body">
167.                             <p>
168.                                 <i>Description:</i> If the Philips Hue Light is on and i
    f you get
169.                                 too close to the Distance Sensor then, the Philips Hue L
    ight turn
170.                                 red and blink one time. Else if the Philips Hue Light is
    off, it
171.                                 blink one time on it current color. <br />
172.                             </p>
173.                             <button type="button" class="btn btn-primary"
174.                                 onclick="Javascript:sse2()">Enable Distance Rule</but
    ton>
175.                             <span style="display: inline-block; width: 3%;"></span>
176.                             <!--
     <span style="display:block; height: YOURHEIGHT;"></span> -->
177.                             <button type="button" class="btn btn-primary"
178.                                 onclick="Javascript:source2.close()">Disable Distance

179.                                 Rule</button>
180.                             <span> </span>
181.                         </div>
182.                         <div class="panel-footer">
183.                             <label><i>Log: </i></label> <span id="out2"></span>
184.                         </div>
185.                     </div>
186.                 </div>
187.                 <div class="col-xs-12 col-md-6">
188.                     <div class="panel panel-info">
189.                         <div class="panel-heading">
190.                             <h2 class="panel-
    title">8. Server Sent Event Case C</h2>
191.                         </div>
192.                         <div class="panel-body">
193.                             <p>
194.                                 <i>Description:</i> If the light sensor detects light, the
    n it
195.                                 set the light level to match Blue and if it is dark set the l
    ight
196.                                 level to match Green. <br />
```

```
                    </p>
                    <button type="button" class="btn btn-primary"
                        onclick="Javascript:sse3()">Enable Light Rule</button
    >
                    <span style="display: inline-block; width: 3%;"></span>

                    <button type="button" class="btn btn-primary"
                        onclick="Javascript:source3.close()">Disable Light Rul
    e</button>
                    <span> </span>
                </div>
                <div class="panel-footer">
                    <label><i>Log: </i></label> <span id="out3"></span>
                </div>
            </div>
        </div>
    </div>

    <script type="text/javascript">
        function showValue(newValue, elementId) {
            document.getElementById(elemendId).value = newValue;
        }
    </script>
</body>
</html>
```

Client_JS.js :

```
1.  // Philips hue Middleware Address
2.  var uriHue = "http://localhost:9050/services/SmartPhilipsHUELight/";
3.  // Sensors Middleware Address
4.  var uriMotion = 'http://localhost:9050/services/MotionSensor/getServerSentEve
    nts';
5.  var uriDistance = 'http://localhost:9050/services/DistanceSensor/getServerSentE
    vents';
6.  var uriLight = 'http://localhost:9050/services/LightSensor/getServerSentEvents';

7.
8.  var source1;
9.  var source2;
10. var source3;
11.
12. // Initialize sensor values
13. var temp1 = "none";
14. var temp2 = "none";
15. var temp3 = "none";
16.
17. var i = 0;
18. var dark = 1; // is day light not dark
19. // Rules Each rule is named as sse1 , sse2 ect.
20. function sse1() {
21.     // optional
22.     // TurnLightOff("out1");
23.     source1 = new EventSource(uriMotion);
24.     source1
25.         .addEventListener(
26.             'motion',
27.             function(e) {
28.                 // console.log('System status is now: ' + e.data);
29.                 // Rule Actions
30.                 console.log('System status is now: ' + e.data);
31.                 if (e.data == "motion") {
32.                     document.getElementById("out1").innerHTML = e.data;
33.                     if (i == 0) {
34.                         TurnLightOn("out1");
35.                         i = 1;
36.                         document.getElementById("out1").innerHTML = e.data;
37.                         // hue is on, ok all done
38.                         source1.close();
39.                     }
40.                 } else if (e.data == "none") {
41.                     document.getElementById("out1").innerHTML = e.data;
42.
```

```
43.                     } else if (e.data == "s") {
44.
45.                     } else {
46.                         document.getElementById("out1").innerHTML = "Invalid eve
    nt inserted. Compromized";
47.                     }
48.                 });
49. }
50.
51. function sse2() {
52.     source2 = new EventSource(uriDistance);
53.     source2
54.             .addEventListener(
55.                     'distance',
56.                     function(e) {
57.                         console.log('System status is now: ' + e.data);
58.
59.                         // Rule Actions
60.                         if (e.data == "close") {
61.                             document.getElementById("out2").innerHTML = e.data;
62.                             // there is someone too close, turn light red and
63.                             // dim it.
64.                             DimLight("out2");
65.
66.
67.                             //lightLevelInput
68.                             document.getElementById("lightLevelInput").value = 245;
69.                             SetLightLevel("out2", "lightLevelInput");
70.
71.                             //lightColorInput
72.                             document.getElementById("lightColorInput").value = 65535;
73.                             SetLightColor("out2", "lightColorInput");
74.                             document.getElementById("out2").innerHTML = e.data;
75.                         } else if (e.data == "none") {
76.                             document.getElementById("out2").innerHTML = e.data;
77.                             if (dark == 1) {
78.                                 document.getElementById("lightLevelInput").value = 125 ;

79.                                 SetLightLevel("out2", "lightLevelInput");
80.                                 document.getElementById("lightColorInput").value = 2550
    0;
81.                                 SetLightColor("out2", "lightColorInput");
82.                                 document.getElementById("out2").innerHTML = e.data;
83.                             }
84.
85.                             else if (dark == 0) {
86.                                 document.getElementById("lightLevelInput").value = 245;
87.                                 SetLightLevel("out2", "lightLevelInput");
```

```
88.                         document.getElementById("lightColorInput").value = 4692
    0;
89.                         SetLightColor("out2", "lightColorInput");
90.                         document.getElementById("out2").innerHTML = e.data;
91.                     } else {
92.                         document.getElementById("out2").innerHTML = "Invalid e
    vent inserted. Compromized";
93.                     }
94.
95.                 } else if (e.data == "s") {
96.
97.                 } else {
98.                     document.getElementById("out2").innerHTML = "Invalid eve
    nt inserted. Compromized";
99.                 }
100.
101.                    });
102.        }
103.
104.        function sse3() {
105.            source3 = new EventSource(uriLight);
106.            source3
107.                .addEventListener(
108.                    'light',
109.                    function(e) {
110.                        console.log('System status is now: ' + e.data);
111.
112.                        // Rule Actions
113.                        if (e.data == "none") {
114.                            document.getElementById("out3").innerHTML = e.dat
    a;
115.                            // there is light, turn bulb strong blue
116.                            dark = 0;
117.                            document.getElementById("lightLevelInput").value =
    245;
118.                            SetLightLevel("out3", "lightLevelInput");
119.                            document.getElementById("lightColorInput").value =
    46920;
120.                            SetLightColor("out3", "lightColorInput");
121.                            document.getElementById("out3").innerHTML = e.dat
    a;
122.                        } else if (e.data == "dark") {
123.                            document.getElementById("out3").innerHTML = e.dat
    a;
124.                            // there is dark, turn bulb light green
125.                            dark = 1;
126.                            document.getElementById("lightLevelInput").value =
    125;
127.                            SetLightLevel("out3", "lightLevelInput");
```

```
128.                                 document.getElementById("lightColorInput").value =
    25500;
129.                                 SetLightColor("out3", "lightColorInput");
130.                                 document.getElementById("out3").innerHTML = e.dat
    a;
131.
132.                             } else if (e.data == "s") {
133.
134.                             } else {
135.                                 document.getElementById("out3").innerHTML = "Inva
    lid event inserted. Compromized";
136.                             }
137.                         });
138.         }
139.
140.         // Philips hue
141.
142.         function TurnLightOn(resultId) {
143.
144.             CallToMiddleware("TurnLightOn", "Turn Light On", resultId);
145.         }
146.
147.         function TurnLightOff(resultId) {
148.
149.             CallToMiddleware("TurnLightOff", "Turn Light Off", resultId);
150.         }
151.
152.         function DimLight(resultId) {
153.
154.             CallToMiddleware("DimLight", "Dim Light", resultId);
155.         }
156.
157.         function SetLightLevel(resultId, inputField) {
158.
159.             var inputValue = document.getElementById(inputField).value;
160.
161.             if (inputValue == null || inputValue == "") {
162.                 alert('Please enter the light level.');
163.                 return;
164.             } else if (isNaN(inputValue)) {
165.                 alert('Please enter a positive whole number for light level, between
    1 and 254.');
166.                 return;
167.             }
168.
169.             CallToMiddleware("SetLightLevel/" + inputValue, "Set Light Level",
    resultId);
170.         }
171.
```

```
172.        function SetLightColor(resultId, inputField) {
173.
174.            var inputValue = document.getElementById(inputField).value;
175.
176.            if (inputValue == null || inputValue == "") {
177.                alert('Please enter the light color.');
178.                return;
179.            } else if (isNaN(inputValue)) {
180.                alert('Please enter a positive whole number for light level, between
    0 and 65535.');
181.                return;
182.            }
183.
184.            CallToMiddleware("SetLightColor/" + inputValue, "Set Light Color",
    resultId);
185.        }
186.
187.        function CallToMiddleware(methodName, logicalActionName, resultId)
    {
188.            var url = uriHue + methodName;
189.
190.            var xhr = new XMLHttpRequest();
191.            xhr.open('GET', url, true);
192.            xhr.send(null);
193.            xhr.onreadystatechange = function() {
194.
195.                if (xhr.readyState == 4) {
196.                    var resultTxt = "";
197.                    if (xhr.status == 200) {
198.                        if (xhr.response == true || xhr.response == "true") {
199.                            resultTxt = 'Success ' + logicalActionName + '!';
200.                        } else {
201.                            resultTxt = 'Error ' + logicalActionName + ': '
202.                                    + xhr.status + ' - ' + xhr.statusText;
203.                        }
204.
205.                    } else {
206.                        var resultTxt = 'Error ' + logicalActionName + ': '
207.                                + xhr.status + ' - ' + xhr.statusText;
208.                    }
209.
210.                    document.getElementById(resultId).innerHTML = resultTxt;
211.                }
212.            };
213.        }
```