TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TERO LATVALA

DEPLOYMENT OF A SERVICE-ORIENTED AUTOMATION PLATFORM FOR INTEGRATING SMART CITY APPLICATIONS

Master of Science thesis

Examiners: Prof. José L. Martinez Lastra, Dr. Jani Jokinen
Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sciences
on 4th November 2015

ABSTRACT

TERO LATVALA: Deployment of a Service-Oriented Automation Platform for Integrating Smart City Applications
Tampere University of Technology
Master of Science Thesis, 58 pages
February 2016
Master's Degree Programme in Automation Technology
Major: Factory Automation
Examiners: Professor José L. Martinez Lastra, Senior Research Fellow Jani Jokinen

Keywords: Arrowhead, Smart City, Service-Oriented Architecture

SOA (Service-Oriented Architecture) is an architectural style for creating software systems. It encapsulates the functional behaviour behind services which are loosely coupled to each other. SOA brings flexibility, agility and dynamicity to software systems because it enables the collaboration of systems developed with different technologies. Moreover, SOA reduces the cost of the development since already existing services can be reused in new systems.

Arrowhead Project aims to develop a technical framework, which enables the collaborative automation by networked embedded devices. The framework aims to solve the energy and competitiveness challenges with increased efficiency, which is gained by means of collaborative automation. Collaborative automation is enabled with SOA. The project is targeting to five different areas including Production, Smart Buildings and Infrastructures, Electro-Mobility, Energy Production and End-User Services, and finally Virtual Market of Energy.

During the project, project partners are testing Arrowhead Framework by developing pilot applications for the project domains to demonstrate the use of the framework at various use-cases. This thesis presents pilot applications developed at FAST-lab at Tampere University of Technology during Pilot Generation 2. The first application is a managing and monitoring system for street lights and the second one for engine block heaters. They demonstrate how Arrowhead Framework could be used to develop Smart City applications.

Since Arrowhead Framework is based on SOA, it should be possible to reuse the existing services in new applications. Therefore, a third pilot application was developed by integrating the components from the first two pilot applications. The resulting application controls the dimming of street lights based on the luminance and the heating time of engine block heaters based on the temperature. The application shows that Arrowhead Framework gives suitable tools to develop systems where the requested functionality by reusing the existing services in collaborative manner.

TIIVISTELMÄ

TERO LATVALA: Palvelukeskeiseen arkkitehtuuriin perustuvan automaatioalustan hyödyntäminen älykaupungin sovelluksissa ja niiden integroinnissa.
Tampereen teknillinen yliopisto
Diplomityö, 58 sivua
Helmikuu 2016
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Factory Automation
Tarkastajat: professori José L. Martinez Lastra, Yliopistotutkija Jani Jokinen

Avainsanat: Arrowhead, älykaupunki, Palvelukeskeinen arkkitehtuuri

Palvelukeskeinen arkkitehtuuri on ohjelmistojen suunnittelutapa. Tässä arkkitehtuurimallissa toiminnallisuus kapseloidaan palveluihin, jotka ovat löyhästi kytketty toisiinsa. Palvelukeskeinen arkkitehtuuri tuo ohjelmistoihin joustavuutta, ketteryyttä ja dynamiikkaa, koska eri tekniikoilla toteutetut järjestelmät pystyvät toimimaan yhdessä. Lisäksi järjestelmien toteutus nopeutuu, koska jo olemassa olevia palveluita voidaan käyttää uudelleen uusissa järjestelmissä.

Arrowhead-projektin tarkoitus on kehittää tekninen viitekehys, joka mahdollistaa toisiinsa kytkettyjen sulautettujen automaatiolaitteiden yhteistoiminnan. Viitekehys pyrkii lisäämään yhteistoiminnan avulla tehokkuutta ja vastaamaan siten energiatehokkuushaasteisiin sekä kasvavaan kilpailuun. Yhteistoiminta perustuu palvelukeskeiseen arkkitehtuuriin. Projekti on suunnattu viidelle eri osa-alueelle, jotka ovat tuotanto, älykkäät rakennukset ja infrastruktuurit, sähkökulkuneuvot, energiantuotanto ja loppukäyttäjien palvelut sekä energian virtuaalimarkkinat.

Projektin partnerit testaavat Arrowhead-viitekehystä projektin aikana kehittämällä pilottisovelluksia projektin eri osa-alueille. Ne demonstroivat viitekehyksen käyttöä eri käyttötapauksissa. Tässä opinnäytetyössä esitellään pilottisovellukset, jotka on kehitetty toisen pilottivaiheen aikana FAST-labissa Tampereen teknillisellä yliopistolla. Ensimmäinen sovellus on hallinta- ja monitorointijärjestelmä katuvaloille, ja toinen sovellus on vastaava järjestelmä autojen lämmitystolpille. Sovellukset demonstroivat Arrowhead-viitekehyksen soveltuvuutta älykaupungin sovelluksiin.

Koska Arrowhead-viitekehys perustuu palvelukeskeiseen arkkitehtuuriin, olemassa olevia palveluita pitäisi voida hyödyntää uusissa sovelluksissa. Tämän vuoksi kehitettiin kolmas pilottisovellus, johon on integroitu kahden ensimmäisen sovelluksen komponentteja. Luotu järjestelmä ohjaa katuvaloja kirkkauden perusteella sekä muuttaa lämmitystolppien lämmitysaikaa lämpötilan mukaan. Tämä kehitetty sovellus osoittaa, että Arrowhead-viitekehys tarjoaa tarvittavat työkalut kehittää järjestelmiä, joissa haluttu toiminnallisuus saavutetaan sovellusten uudelleenkäytöllä ja yhteistoiminnalla.

PREFACE

This master’s thesis was written at FAST-lab (Factory Automation Systems and Technologies laboratory) at Tampere University of Technology. I would like to acknowledge Professor José L. Martinez Lastra about interesting position at Arrowhead project at FAST-lab. Moreover, I would like to send a special thanks to Senior Research Fellow Jani Jokinen who introduced the position to me and instructed me during the writing process. Finally, I want to thank my family and friends for never-ending support.

Tampere, 23th February 2016

Tero Latvala

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BPM	Business Process Management
CN	Common Name
CoAP	Constrained Application Protocol
CORS	Cross-Origin Resource Sharing
CP	Communication Profile
DOM	Document Object Model
ESB	Enterprise Service Bus
EXI	Efficient XML Interchange
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technology
IDD	Interface Design Description
IoT	Internet of Things
JSON	JavaScript Object Notation
LED	Light-Emitting Diode
MVaaS	Materialized View as a Service
REST	Representational State Transfer
Scallop4SC	SCALable LOgging Platform for Smart City
SD	Service Description
SOA	Service-Oriented Architecture
SoSD	System-of-Systems Description
SoSDD	System-of-Systems Design Description
SP	Semantic Profile
SSH	Secure Shell
SysD	System Description
SysDD	System Design Description
SysML	Systems Modeling Language
UML	Unified Modeling Language
URI	Unified Resource Identifier
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

.

1. INTRODUCTION

Service-Oriented Architecture is popular approach to build software systems where the focus is on the use of services encapsulating the application behaviour. At SOA-based systems, business processes are built by combining a set of services together. Existing services can be reused in new business processes. Moreover, services can be developed with different technologies at SOA-based systems. SOA improves thus the efficiency and changeability of software architecture, but also speeds up the development of software systems [31]. It brings also dynamicity, agility and flexibility to software systems with loosely coupled services [13]. During the past years, SOA is proposed to many kinds of systems including public emergency logistics system [51], test framework for mobile applications [29] and Bank online application system [26].

Colombo et al present the vision about collaborative automation which is enabled by SOA [20]. The study states that collaborative automation would improve the reconfigurability of manufacturing systems and thus reduce the time downtime and increase efficiency [20]. Similar vision is adopted at Arrowhead Project. Arrowhead project aims to develop technical framework, which enables collaborative automation by networked embedded devices by means of SOA. Collaborative automation is seen as a solution for increasing competitiveness and energy challenges [10]. Since the network is based on SOA, interoperability should remove the technical limitations meaning that it should be possible to connect devices developed with any technology to the network. Moreover, SOA enables the reuse of components thus enhancing the efficiency of engineering processes when new applications are developed. It is envisioned that the framework will achieve 75% reduction to the effort of design process [10]. Arrowhead Project is targeting the framework to five different areas including Production, Smart Buildings and Infrastructures, Electro-Mobility, Energy Production and End-User Services, and finally Virtual Market of Energy.

During the project, project partners are testing Arrowhead Framework by creating pilot applications to project domains which demonstrate the use of Arrowhead Framework in various use-cases. Applications are developed in three consecutive phases of development called Pilot Generations. Two pilot applications were developed at FAST-lab at Tampere University of Technology during Pilot Generation 2. They demonstrate how Arrowhead Framework could be used to build Smart City applications. The applications are managing and monitoring systems: the first targeted to street lights and the second one targeted to engine block heaters respectively. Next, a third pilot application was developed by integrating components from the first two applications. In this application, street lights and engine block heaters are controlled with a separate controller system. New functionality is achieved by reusing the components in collaborate manner.

1.1 Problem Definition

Smart City has gained lots of interest recently. It aims to improve the use of public resources and to increase the quality of services offered to the citizens. It also aims to reduce energy consumption and the operational costs of the public administrations. ICT (Information and Communication Technology) is seen as a key enabler for Smart City since smartness is increased by means of Smart City applications.

Several researches have proposed various technological solutions for Smart City. Nevertheless, there is not any inclusive technical solution available yet. Since Arrowhead is targeting also to Smart Buildings and Infrastructures domain, Arrowhead Framework should be suitable for building Smart City applications. Therefore, the framework should be tested in this concept. Furthermore, since Arrowhead Network is based on SOA, it should be possible to reuse the services. This feature should be tested by integrating Smart City applications to build new functionality.

1.2 Work Description

This thesis presents Arrowhead Project and the most important technical aspects of Arrowhead Framework. Furthermore, a theoretical background is given about Service-Oriented Architecture in order to gain better understanding about the framework. Furthermore, the thesis gives a review about the recent research topics related to Smart City.

The main objective of this thesis is to give a detailed description about the pilot applications developed at FAST-lab at Tampere University of Technology. The goal is to present how Arrowhead Framework can be used to develop Smart City applications.

1.3 Methodology

This thesis has two main parts. The first part of the thesis gives an overview of theoretical background by means of literature review. Following topics are included:

- Service-Oriented Architecture.
- Smart City concept.
- Technologies at Smart City.
- Arrowhead Project and its goals.
- Arrowhead Framework.

The next part of the thesis gives detailed description about pilot applications including the following topics:

- Technologies used in development.
- Description of hardware.
- Description of software architecture.
- Analysis.

1.4 Thesis outline

This thesis includes five chapters. First chapter contains introduction, problem definition, work description and methodologies. Second chapter describes SOA and Smart City concept. It also introduces Arrowhead Project and Arrowhead Framework. Third chapter describes the methodologies used to develop the pilot applications. First, it describes how Arrowhead Framework was used in development and what technologies were used to develop the services. The second subsection includes a set of technologies needed to develop the user interfaces for the pilot applications. Fourth chapter presents the actual pilot applications. First, two managing and monitoring pilot applications called Light Management Tool and Engine Block Heater Controller are described. Hardware, software architecture and user interface are described in separate subsections. Next, a third pilot application, which integrates components from the first two applications, is described. Finally, the chapter includes a discussion section. Fifth chapter contains the conclusions and recommendations for future work.

2. THEORETICAL BACKGROUND

This chapter gives a theoretical background by means of literature review. It starts with an overview about Service-Oriented Architecture, which is an architectural style to create software systems. Next, a literature review is given about Smart City concept and the latest research topics related to the concept. Furthermore, the technologies of Smart City applications are also described. Finally, this chapter introduces Arrowhead Project and the most important technical aspects of Arrowhead Framework.

2.1 Service-Oriented Architecture

Service-Oriented Architecture is an architectural style to create software systems where the main components are loosely coupled, interoperable and distributed [12]. The idea behind SOA is to encapsulate application behaviour to an entity called **Service**. Services can be combined dynamically together to build business processes. Service is stored by an entity called **Service Provider**. Furthermore, Services are offered to another entity called **Service Consumer**. Service Consumer is typically a piece of software or an application, which somehow benefits from the service. The interaction between a consumer and a service is based on the interface of the service, which describes the rules for communication and the context of the messages used in communication. Request-response pattern is typically used at the interaction meaning that the consumer requests meaningful data from the service, which then responds with a message containing the requested content. Nevertheless, communication is not limited only to request-response pattern. [13]

Bean presents at [13] that some supporting technologies are needed to construct SOA-based implementations. First, SOA needs a network or a platform where service consumers can connect to the services. The technology is known as ESB (Enterprise Service Bus). ESB provides a network, delivery of messages and communication protocols. The consumer and the service are identified at the network with an identifier called **Endpoint**. Communication between the consumer and the service is based on sending the messages to these endpoints. Consumer needs to know the endpoint of the service in order to start the communication. It can be difficult to maintain the list of available services at large SOA-based systems. Therefore, a technology called **Service Registry** is needed. Bean describes Service Registry as a catalogue of services hosted at the network [13]. Thus, it helps with finding the needed service from the network. Bean mentions also BPM (Business Process Management) technology, which composes a set of services into new business services. The action of composing services into business services is called **Orchestration**. [13]

The typical application of SOA is Web Services. The architecture of Web Services includes Service provider, Service requester and Service registry. Service provider provides the interface of the Web Service and publishes the Web Service to Service registry. Service requester, which means Service Consumer, can then find the Web Service from the Service Registry and start the interaction with it. [27] The setup is presented in Figure 1.

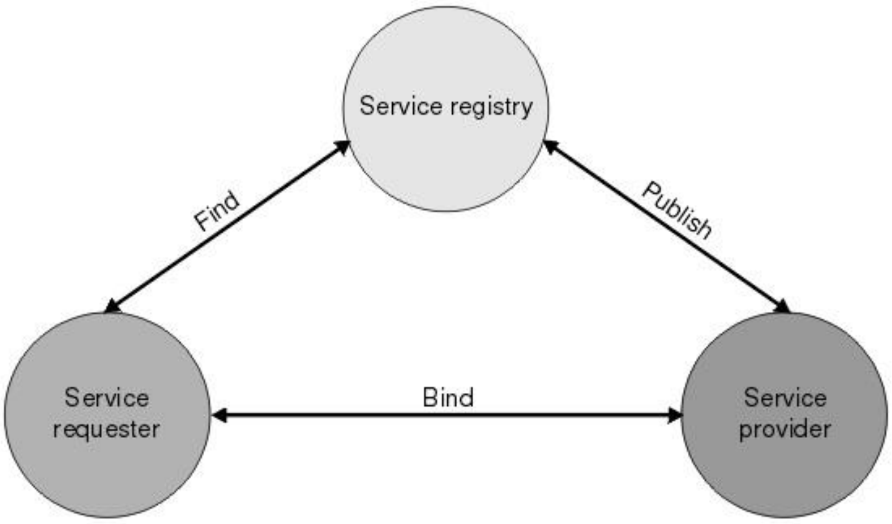


Figure 1. *Service-Oriented Architecture in Web Services [27].*

There is a set of principles which must be covered in SOA-based software systems. The principles are *Loose Coupling*, *Interoperability*, *Reusability*, *Discoverability* and *Composition*. [12] However, the list of principles has some variation in literature. For example, at [13] the list includes *Governance*, but *Composition* is not presented. Loose Coupling means that consumers are intentionally separated from services. There should not be any physical connection between a consumer and a service in order to avoid physical dependencies. Moreover, the communication should be based on messaging instead of direct communication. Interoperability means that consumers and services can be implemented with different technologies thus making SOA technology-independent. Moreover, Interoperability enables the collaboration and the message exchange of services even though they are developed with different technologies. Reusability aims to minimize costs by reusing the functionality already developed with existing services. Discoverability means the services must be discoverable in order to use them later. This is usually realised with Service Registry. Governance provides the rules for measuring the compliance of the principles and for correcting the possible noncompliance. Governance is sometimes considered a practice rather than a principle. [13] Finally, Composition enables the creation of new business processes with cooperation of services without changing their original content [12].

SOA has several benefits. It gives dynamicity, agility and flexibility to software systems. Moreover, SOA makes it possible to reuse earlier investments in technology thus reducing the costs. New business processes can be built by reusing the components which were designed for earlier implementations. SOA makes also the development process rapid. [13]

Some challenges are also identified at SOA. For example, Baskin et al argue that interoperability of SOA-based systems may bring lots of costs to the system maintenance, since maintenance personnel need to know multiple technologies and handle large documents [48]. Information security is also seen as a challenge. According to [38], the security of SOA is difficult to achieve, because it is required that every element, interaction and system is secure. Therefore, a single unsecure building block may compromise the whole SOA-based system.

2.2 Smart City

Smart City is quite new concept and thus it does not have a common definition yet. ICT technology is emphasized strongly at many definitions of Smart City. For example, Anthopoulos and Fitsilis define Smart City as an infrastructure and environment of services based on ICT that enhance city’s intelligence, the quality of life and other attributes such as entrepreneurship, education and transportation [9]. According to [43], Smart City is a next-generation city planning that aims to achieve sustainable society with ICT technologies. Monzon emphasizes at [30] the ICT-based solutions as a key element of Smart City, but he also states that simply deploying technologies to the city is misunderstanding the concept. At ASCIMER project, he defines Smart City as an integrated system where human and social capitals interact by using technology based solution. [30] Giffinger et al have slightly different approach since they define Smart City as a city well performing in a forward-looking way in six characteristics, which are Smart Economy, Smart People, Smart Governance, Smart Mobility, Smart Environment and Smart Living. The characteristics include ICT as well, but also things like social and human capital, political aspects, natural resources, transportation systems and quality of life. They used the definition of Smart City to compare the smartness of European medium-sized cities. [23]

Chourabi et al did a large research as presented at [19], where they studied Smart City literature to identify the trends around the concept and to understand it better. They defined a framework with eight important factors, each influencing each other, including technology, policy, organization, natural environment, governance, people communities, economy and built infrastructure. The framework can be used to determine the success factors of Smart City projects and to help in envisioning the first steps towards

Smart City. Chourabi et al stated that the technology can heavily influence other seven factors. [19] Again, ICT is strongly emphasized as an important building block at Smart City concept.

Smart City aims to improve the use of public resources, increase the quality of services offered to the citizens and reduce the operational costs of the public administrations [50]. According to [30], the main objective of Smart City projects is to solve urban problems in an efficient way to improve sustainability of the city and the quality of life of its inhabitants. According to [43], Smart City aims to sustainable society with ICT technologies by collecting data from various sensors and using it for value-added services. The services lead to an environmental-friendly city by saving energy, but they provide also safety, amenities and utilities to the city.

Most of the goals are reached with Smart City applications. Scientific papers envision many possible applications for Smart City concept. Su et al present applications at [41] such as a wireless network covering the whole city, which offers various services to people but also improves urban management systems. Moreover, they mention Smart Home applications providing an intelligent control of lights and other house equipment as well as alarm notifications. Furthermore, they state that Smart City would enable smart traffic management systems with better traffic control, smart public services and smart medical treatment. It would help to establish an urban model of green city and finally it would support tourism by offering services to tourists and by collecting tourism data. [41] Zanella et al give an overview at [50] to some of the Smart City applications envisioned at various scientific papers. Applications include a structural analysis system to provide information about the condition of historical buildings, optimized waste management system, monitoring system for air quality and noise at urban areas as well as for traffic congestion. Furthermore, Zanella et al mention also energy monitoring systems that measure the overall energy consumption of the city. Another energy-related application is Smart Lighting, which optimizes the street lights based on various parameters such as the presence of the people. Finally, Zanella et al mention Smart Parking, which helps to find free parking slots faster thus reducing the emissions and traffic congestions. [50]

Even though technology is seen as a key enabler when taking the first steps towards Smart City concept, other actions are needed as well. Van den Bergh and Viaene studied how to become a Smart City by analysing the city of Ghent in Belgium [46]. They identified six key challenges, focusing on city administration perspective, to overcome when realizing Smart City. The key challenges are related to city ecosystem, leadership, coordination mechanisms, business-IT alignment, organisational culture and experimental environment. Smart City needs clear IT strategy and adaption of new technologies, but also political willingness, long-term commitment and credibility towards parties. Finally, Van den Bergh and Viaene underline that cities should avoid getting stuck to experimental environment which tends to be the problem of the Smart City projects. [46]

2.3 Technologies of Smart City applications

As described before, ICT is seen as a key enabler in Smart City concept. Lots of research is done about technologies suitable for Smart City. In addition, several frameworks are developed to support the development of Smart City applications. This section gives a literature review about recent technology-related research topics within Smart City concept.

ICT-based Smart City systems can be built in many different ways. Anthopoulos and Fitsilis discovered several commonly used architectural styles by studying current Smart City solutions. According to the study, multi-tier is the most preferred architectural style at existing Smart City projects but SOA appeared to be famous architecture as well. Moreover, multi-tier is preferred in applications using IoT (Internet of Things) paradigm. Finally, event driven architecture was also used at some European research projects. [9]

IoT is a communication paradigm which interconnects everyday objects to a global network to exchange data between them. IoT is seen as a promising solution to meet the requirements of ICT systems within Smart City concept. [50] Zanella et al propose an architecture for IoT-based solution called Urban IoT. It uses EXI (Efficient XML Interchange) as a data format and CoAP (Constrained Application Protocol) as the application layer protocol. The network layer relies on 6LoWPAN technology. Service architecture uses Web Services and REST (Representational State Transfer) architectural style. The architecture of Urban IoT includes also servers for data, gateways and IoT nodes which are the actual devices of network collecting the data. [50]

Multiple projects are running at European Union, which aim to build IoT-based frameworks for Smart City concept, including SmartSantander [39] and RERUM [34]. SmartSantander project aims to build a platform which can be used to perform IoT experiments with Smart City services. Services are provided with thousands of sensors installed mainly in the city of Santander in Spain. The platform is based on multi-tier architecture including IoT device tier, gateway tier and server tier. [44] RERUM project focuses on building IoT-based framework which enhances reliability, security and privacy of Smart City applications. Smart City applications can have several security threats including the loss of measurement reliability, interference and denial of service, eavesdropping and data falsification. Technologies like secure self-configuration, reputation management framework and cryptographic mechanisms are going to be developed at RERUM project to overcome the security issues. [45]

Security of Smart City is also addressed at [47]. The research presents an approach called HiSPO for analysing threads and improving data security. It calculates thread factors for Smart City system describing its robustness against cyber-attacks. HiSPO approach starts by identifying threats from several areas including network, host, application, security policy, operational security and attack patterns. Data from public and commercial threat analysis systems is also used. Next, a threat model is created based on identified threads, and risk levels are defined. Finally, thread factors are calculated based on the data gathered from previous steps. Factors are presented in a threat report including vulnerability assessment, which gives a comprehensive state of the security of the system. However, the research states also that information security is not only a matter of technology but it needs good policies and effective business operations as well. [47]

Even though a lot of effort is put to IoT-based frameworks, there are also application-specific platforms for Smart City concept. For example, Takahashi et al have developed a platform called Scallop4SC (SCALable LOgging Platform for Smart City) for storing and processing data from households [43]. Scallop4SC manages two types of data including house log and house configuration. House log is the data collected from the devices and sensors of the house whereas house configuration is metadata including the list of the houses at Smart City and household compositions. Data is accessed through Web Services based API (Application Programming Interface). [43] Later Yamamoto et al introduced a materialized view for Scallop4SC which caches the application-specific data thus reducing the queries to raw data of the houses. However, a lot of experience was required from the developer in order to create Materialized views. [49] Therefore, Yamamoto et al encapsulated application-specific materialized views by means of cloud computing. They introduced an abstract cloud service called MVaaS (Materialized View as a Service). Instead of using application-specific materialized views, MVaaS creates application-specific views dynamically. Furthermore, the developer does not need to have the knowledge about the used technologies. [49]

Smart City services are based on huge amount of data which is mainly collected with sensors. Data can also be collected with participatory sensing meaning that people are collecting the data with for example their mobile devices [42]. Szabo et al. developed a framework based participatory sensing, which uses publish-subscribe service of XMPP (Extensible Messaging and Presence Protocol). The framework contains three different roles including Producer, Consumer and Service Provider. Producers are the information sources publishing raw data to data nodes. Service Provider is an entity, which turns the raw data into something more meaningful and publishes it to the data nodes. Finally, consumers can get access to the data by subscribing the nodes. The data is collected by means of participatory sensing with mobile devices thus making people Producers at this framework. [42]

2.4 Arrowhead Project

Arrowhead Project aims to develop a technical framework to enable a cooperative automation by networked devices. It is funded by ARTEMIS Industry Association. The project is a response to ARTEMIS-JU Call 2012 and it is running from March 2013 until February 2017. The project is really broad including almost 80 project partners. [11]

The main goal of Arrowhead Project is to enable a collaborative automation by networked embedded devices. The devices form a network called **Arrowhead Network**, which is based on SOA [10]. Services are offered by devices and systems at the network. Collaborative automation is achieved by combining the services. One of the key features of Service-Oriented approach is loose coupling, which enables the interaction of systems without any physical connections. Moreover, SOA should make it possible that anyone can design systems or devices with any technology and connect them to the Arrowhead Network. Furthermore, the project aims to enable the integration of Legacy Systems to the network with proper adapter technology. [10] At Arrowhead, Legacy System is a system that does not provide any services to Arrowhead Network [22]. Since collaborative automation is enabled with services of the systems, an adapter component is needed to offer the services.

Interoperability principle of SOA states that it should be possible to implement services and service consumers with any technology. However, it makes the documentation of services more difficult which can risk the reusability of services [16]. According to [16], unified approach leads to higher level of interoperability and helps to utilize the advantages of SOA. Therefore, there is a need for a technical framework to give common guidelines for the developers so that anyone can develop systems for Arrowhead Network.

The framework developed during the project is called **Arrowhead Framework**. It provides design patterns and guidelines for designing SOA-based Arrowhead compliant systems. In addition, a software framework consisting of a set of **Core Services** is available at the framework. Core Services are needed for supporting the interaction between the services at the network. They are offered by **Core Systems** thus providing the basic functionality, **Core Functionality**, to every device at the network. Furthermore, Arrowhead Framework provides documentation templates in order to give common approach to document the systems. Finally, the framework includes a set of principles to address technical property requirements, conformity requirements and a set of tools for conformity test and verification. Since Arrowhead Framework is rather complex, it includes also an Arrowhead Cookbook, which gives instructions how to use the framework. [10] [16]

Arrowhead Project targets Arrowhead Framework to five domains including Production, Smart Buildings and Infrastructures, Electro-Mobility, Energy Production and End-User Services, and finally Virtual Market of Energy. Project partners are testing the framework at the domains by developing pilot applications, which demonstrate the use of Arrowhead Framework in various use-cases. The applications are developed in three consecutive phases of development called Pilot Generations. Every domain is covered with various pilot applications. Production domain includes several applications aiming to the improvements of efficiency at machine operation and maintenance as well as energy savings. Moreover, machine monitoring is one of the key targets. Smart Buildings and Infrastructures domain aims to reduce energy consumption with solutions to urban environment. These applications are related to Smart City concept. Electro Mobility domain focuses on offering services for the recharge stations of electric vehicles. Energy Production and End-User Services domain is targeting to optimise the district heating systems. Finally, Virtual Market of Energy domain aims to create virtual energy markets for power plants. Markets are based on energy consumption data that is gathered with devices called Flex Offer. [10] In general, most of the applications are aiming to the reduction of energy consumption and improvements in efficiency.

The following subsections describe the technical aspects of Arrowhead Framework more in detail. Core Functionality is covered in section 2.4.1. Next, the security of Arrowhead Network is described in section 2.4.2. Finally, the document model of the framework is presented in section 2.4.3.

2.4.1 Core Functionality of Arrowhead Network

Arrowhead Framework includes three functional areas called Core Functionality. Core Functionality provides basic functionality to fulfil the SOA principles and information security at Arrowhead Network. It consists of three groups called Information Infrastructure, System Management and Information Assurance. Information Infrastructure fulfils discoverability and loose coupling principles of SOA. It thus helps to find services at Arrowhead Network and to provide the connection to them. It also provides information about the services. System Management manages late binging and the composition of the System-of-Systems which means a set of systems communicating to each other by using the Arrowhead Framework [22]. Finally, Information Assurance provides secure information exchange. [16] All three functional groups should be hosted at every Arrowhead Network [14].

Core Functionality is implemented with a set of software components called Core Systems. Currently, several Core Systems are identified for Arrowhead Framework. They are presented in Figure 2.

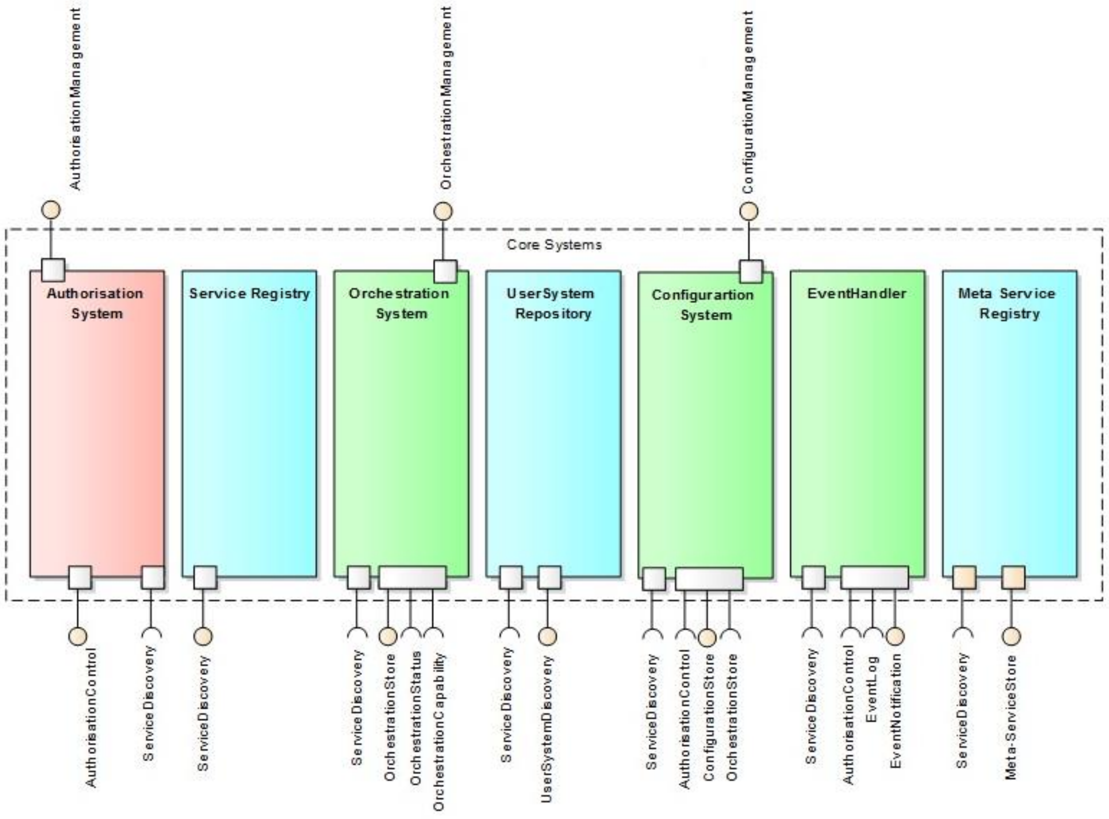


Figure 2. *Identified Core Systems [15].*

Information Infrastructure is implemented with Service Registry, User System Repository and Meta Service Registry. Service Registry manages and stores all services at the Arrowhead Network. A device can therefore find the requested service by using Service Registry. User System Registry is proposed to hold unique system identities. Finally, Meta Service Registry is proposed to be a store for metadata. However, in the current state of Arrowhead Framework, Meta Service Registry is only conceptual. Information Assurance is implemented with Authorisation System. It is a store for access rules to resources within the Arrowhead Network. Authorisation System ensures that only authorised systems can access the services. Finally, System Management is implemented with Orchestration System, Configuration System and Event Handler. Orchestration System manages the connection rules of the services and the composition of System-of-Systems. Configuration System is proposed to store configuration packages for the systems. The content of the configuration is system-dependent. Finally, Event Handler stores Event Log and notifies about the events. [15]

Since Arrowhead Framework is based on SOA, communication with Core Systems is based on services called Core Services. Currently, Arrowhead Framework contains several identified Core Services, which are presented in Table 1.

Table 1. *Identified Core Services [15].*

Information Infrastructure	Information Assurance	System Management
Service Discovery	Authorisation Control	Monitoring
Service Metadata	Authorisation Management	Deployment Access
Meta-Info Store	Authentication	Deployment Access Management
Meta-Info Management	Certificate distribution	Orchestration Management
User and Role	Security logging	Orchestration Store
Organisation and Role	Security Intrusion	Orchestration Capability
Software Distribution		Orchestration Status
Application Installation, Setup, Startup		

Even though Arrowhead Framework has several Core Systems and Core Services identified, only some of them are available for applications of Pilot Generation 2 because most of the Core Services are still at conceptual phase. A prototype is provided from Service Registry, Authorisation System and Orchestration System. Available Core Services are Authorisation Control provided by Authorisation Core System, Service Discovery provided by Service Registry and finally Orchestration Store provided by Orchestration System. Authorisation Control provides functions for checking if the service consumer is authorised to use the service. Authorisation is based on X.509 certificates. Service Discovery provides functions for publishing the services to Service Registry. In addition, identification data can be requested about the services available at Service Registry. Finally, Orchestration Store provides functions for reading the current orchestration configuration.

Pilot applications developed at Pilot Generation 2 are obligated to use the three Core System prototypes described above. Moreover, pilot applications should have both service provider offering services and service consumer. Service provider should use Authorisation System to authorise the service consumer before the service can be accessed. Moreover, service provider should be able to publish its services to Service Registry where service consumer can then discover them. Finally, Orchestration System should be used to manage orchestration rules. The resulting architecture is presented in Figure 3.

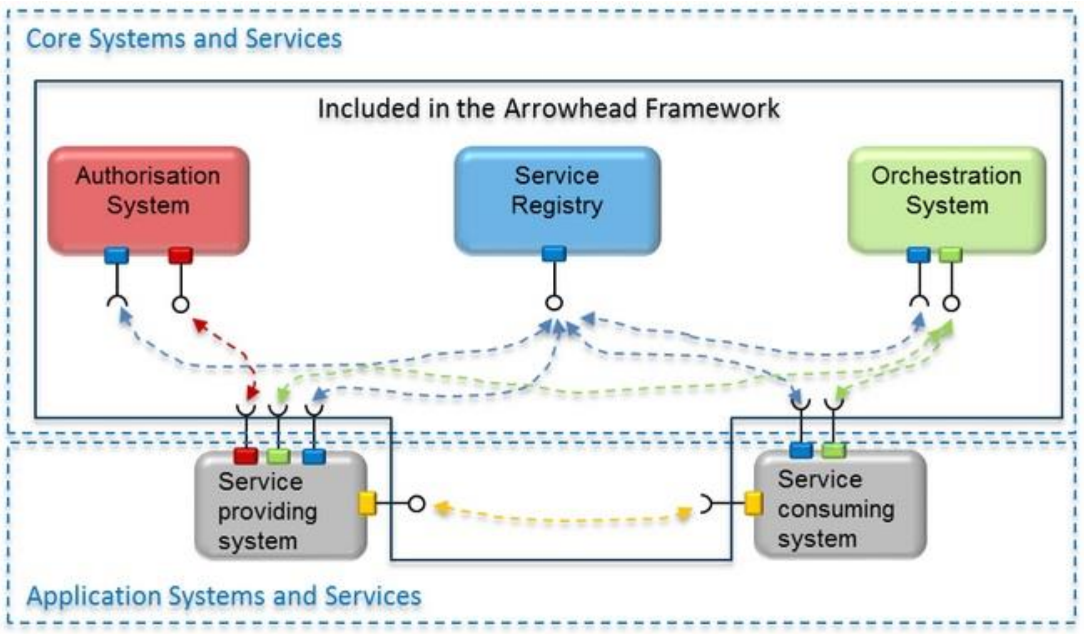


Figure 3. *Core Systems at Pilot Generation 2 [16].*

This presented architecture is the basis of Arrowhead Network. The network includes Core Systems and a set of application systems offering services to each other. During Pilot Generation 2, earlier described prototypes of Service Registry, Authorisation System and Orchestration System are hosted at a temporary server to provide the same Core Functionality to every pilot application. Application system can start using the Core Systems when connection to the server is established. Services published to Service Registry can be discovered by any application system within the network. Moreover, application systems can enable collaborative automation by combining services with Orchestration System. Finally, Authorisation rules can be configured individually to every service at the network with Authorisation System. Time spent in development should reduce because the developer can focus on developing only the application systems. There is no need to work with the SOA principles or information security because they are already covered with Core Systems. This proposed architecture of Arrowhead Network is presented in Figure 4.

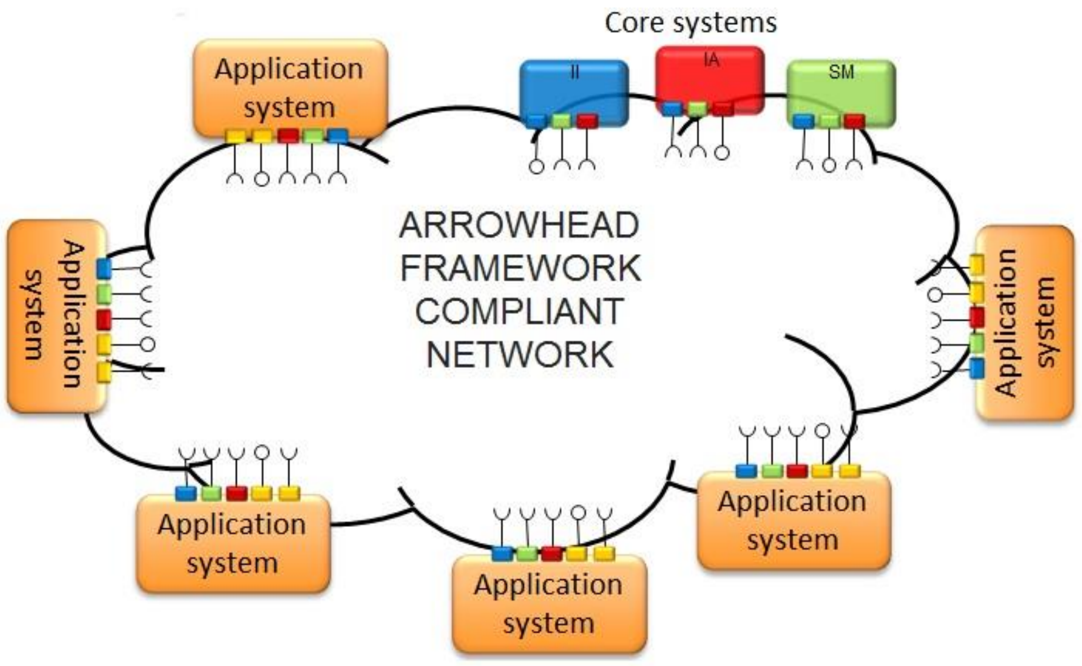


Figure 4. *Arrowhead Network.*

2.4.2 Security of Arrowhead Network

At Arrowhead Network, information security is handled with Information Assurance Core Functionality. Core Services like Authorisation, Authentication, Certificate Distribution, Security Logging and Security Intrusion are identified for Information Assurance. [16] Currently, information security is still under development and these above-mentioned services are not available yet. Certificates are already used, but they are distributed manually. TLS (Transfer Layer Protocol) is used to provide a secured connection to services.

TLS protocol provides communications security between two applications at the network. Secured connection is established at procedure called handshaking which is done before any data is transmitted. First, the client application connects the server application and requests a secure connection. Server proves its identity by sending a digital certificate to the client. The certificate contains several fields including the name of the server, public encryption key, which is another half of public-private key pair, and Certificate Authority. Certificate Authority is in general a trusted entity issuing the certificates. After the client has received the certificate, it can authenticate the server from the Certificate Authority that originally issued the certificate. In other words, Certificate Authority guarantees that the client can trust the server. Next, the client creates a unique session key, also known as a Master Secret, and encrypts it with the public encryption key of the certificate. Session key is then sent to the server. The message can be decrypted only with the private key matching to the public key. Thus, only the server that originally sent the certificate is able to decrypt the message, since it is the only entity having the private key. As a result, both client and server know the same session key, which they can use to encrypt and decrypt the data during the communication. Finally, the server sends “Finished” message encrypted with just created session key and sends it to the client. This step ends the handshaking. [36] This kind of security mechanism based on key-pairs is known in general as Public Key Infrastructure. There is a standard for the infrastructure called X.509 [37].

Hierarchical structures are common in Public Key Infrastructure. Certificate Authorities can certify another Certificate Authorities which then issue more certificates lower at the structure. The result is a chain of certificates one issuing another. When the owner of a certificate needs to be verified, certificate chain is followed back up during the certification verification until the certificate issued by root Certificate Authority, a trust anchor, is reached. This kind of structure is called Chain of Trust. Owner of any certificate at the chain can be trusted if the certificate chain can be followed back to the trust anchor. [35]

Arrowhead Network has internal certificate structure and root Certificate Authority which issues certificates to the project partners. Partners need to have their own certificates for their service providers. Moreover, the certificates must be chained with the root certificates. As a result, it can be guaranteed that the systems of the pilot applications are the part of the Arrowhead Network. When a service is requested from a service providing system, TLS protocol is used to encrypt the communication between a services provider and a service consumer. First, service provider sends its own certificate to the service consumer. Next, service consumer verifies that the certificate is chained to Arrowhead root certificates. This guarantees that the service producer can be trusted and service consumer can create a session key for the communication.

2.4.3 Arrowhead document model

As mentioned earlier, interoperability makes the documenting of SOA-based systems more difficult thus complicating the reuse of the services. Arrowhead Framework introduces a document model for giving a holistic way to document the developed applications. The goal of the document model is to give a common approach for documenting SOA-based systems at Arrowhead Network [16]. The model consists of both abstract description documents and detailed design description documents divided to three levels. Following document types are included at the framework: SoSD (System-of-Systems Description), SoSDD (System-of-Systems Design Description), SysD (System Description), SysDD (System Design Description), SD (Service Description), IDD (Interface Design Description), CP (Communication Profile) and SP (Semantic Profile). There is a template for every document type at the framework. In addition, the project partners are provided with Arrowhead Cookbook which gives additional help for creating the documents. [14][16] Since the overall documentation is distributed to multiple documents, referencing is needed between the documents in order to form a clear picture about overall System-of-Systems. The Arrowhead document model is presented in Figure 5.

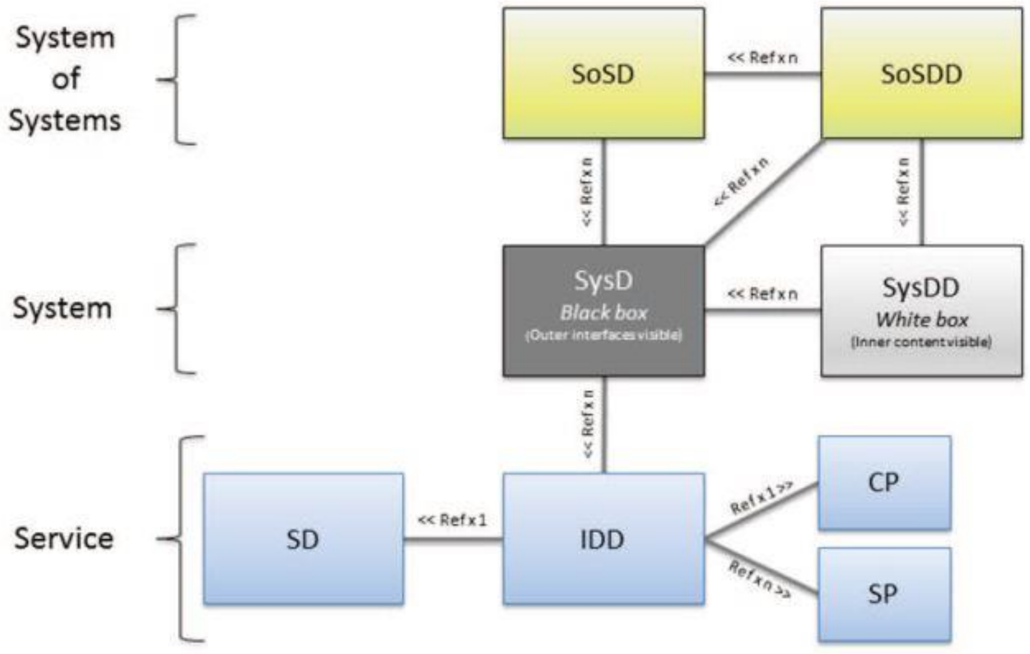


Figure 5. *The document model of Arrowhead Framework [16].*

At the top of the diagram, there is System-of-Systems level. At Arrowhead Framework, System-of-Systems is defined as a set of internal systems communicating to each other by using the Arrowhead Framework. System-of-Systems level is used to describe the actual pilot applications of Arrowhead Project. Therefore, every pilot application can be considered as a System-of-Systems. [22]

System-of-Systems level consists of SoSD and SoSDD documents. SoSD document describes the main functionalities and the architecture of System-of-Systems at abstract level. The template starts with a use-case section describing the overall behaviour of System-of-Systems. It is also proposed that behavioural diagrams such as UML (Unified Modeling Language) Activity Diagram should be used to give more information about the use-cases. SoSD does not present any technologies of the implementation. [5][16] Instead, SoSDD document is used to give a detailed technological description about the main functionalities and the architecture of System-of-Systems. It includes both software and hardware implementation. The technology description can contain for example network configuration, domain structure and start-up behaviour. Thus, SoSDD is considered as a deployment description of System-of-Systems. [6][16]

SoSDD document refers to SoSD document in order to connect the technological implementation to abstract description of System-of-Systems. Since System-of-Systems is the composition of several systems, references to System level documents are needed as well. Both SoSD and SoSDD documents have references to SysD documents of Systems which compose the System-of-Systems. In addition, SoSDD refers also to SysDD document of these systems. [6][16]

Next level at the document model is System level. At Arrowhead Framework, system is defined as something which provides or consumes services. System is an individual entity, which can be almost anything, for example a component or a device. System can include hardware but it can also be only a software component. [22]

System level consists of SysD and SysDD documents. SysD gives general description about the system at abstract level. The usage of the system is described with UML use-case diagrams. In addition, UML sequence diagram should be used to give more information about the interaction with the system. The main purpose of SysD document is to introduce all produced or consumed services of the system by referring to the IDD documents of these services. Similar to SoSD, technical details are not described at the SysD document. This is why SysD is also called a Black Box design. [7][16]

Whereas SysD document is an abstract description, SysDD document presents the actual technological implementation of the system which is why it is called a White Box design. SysDD is the only optional document of the document model. By making the SysDD document optional, partners are able to keep proprietary details of the system as a secret. SysD document describes the architecture of the system, interface implementation, access control mechanisms and also used programming languages. [8][16] SysDD contains a reference to SysD in order to connect the White Box and Black Box designs each other.

The bottom level of the document model is Service level. At Arrowhead Framework, service is something that changes information between producing and consuming system. Service can be realised by arbitrary number of producers and consumers. [22]

Service level consists of four documents: SD, IDD, SP and CP [16]. The documents are divided into technology independent and technology dependent documents. Service level documents are presented in Figure 6.

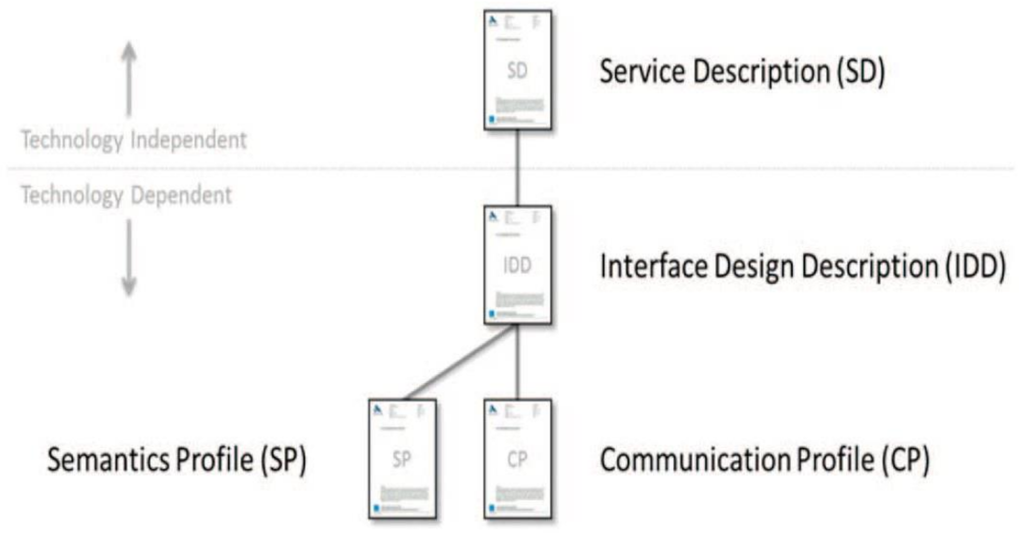


Figure 6. *Service level documents [16].*

SD document gives an abstract description about a service and its requirements. It should give engineers enough information about the service so that they can create a realization of the service producer or the service consumer by using the chosen technologies. [16]

SD document starts with the overview of the service. It is proposed that an abstract architecture description is given at this section with UML diagrams. Next, the interfaces of the service are presented at abstract level. The section does not have any specific form, but it is proposed that sequence diagrams are used with the text to give better understanding about the functionality. SD presents also the information model of the service at abstract level. UML Class Diagram and SysML (Systems Modeling Language) Parametric Diagram are proposed to give more information about the model. Finally, SD lists the non-functional requirements of the service. For example, response time and reliability can be typical non-functional requirements. [4][16]

IDD document describes how the service is realized with chosen technologies. The document starts with references to all SD documents it implements. This way the actual implementation is linked to the abstract service description. In addition, IDD includes references to CP and SP documents as well. Since the same service can be implemented with different technologies, there can be many IDD documents about the same service. [2][16]

Whereas SD document contains the abstract description about the interfaces, IDD document describes how they are actually implemented. Every interface and function should be described in separate subsections in detail. It is proposed to use UML Sequence Diagrams or Activity Diagrams to give better understanding how functions are used. In addition, a table form is available for describing the functions, their inputs, outputs and methods. Finally, IDD describes the information model of the service. [2][14]

CP document describes the technologies and standards used to build a communication profile for a service. CP document starts with an overview section giving an introduction to the technologies and specifications of the profile. Communication profile consists of transfer protocol, security mechanism and data format. The overview section presents also the unique name of the profile which is the combination of the used technologies. Therefore, the name of the profile can be for example CoAP-TLS-XML (Extensible Markup Language). The next section of the document describes how to implement different message exchange patterns by using the chosen transfer protocol. Typical message exchange patterns are Request-Response, One-to-Many and Publish-Subscribe patterns. Every supported pattern is described in own subsection. This is followed by the security section. It describes how the chosen security mechanism handles security issues. The fourth section of the CP document presents the endpoint of the service. The endpoint is presented in a table format although the content of the table might vary due the differences in protocols. Finally, the chosen data format is described at the next section. It is proposed that an example of the data format should be given. Specifications and standards used to build the profile are listed to the end of the document. [1][16]

The final document at Service level is SP document. It presents the encoding of data. The document starts with description of technologies and specifications that are used to build the Semantic Profile. Technologies of choice can be for example XML and JSON (JavaScript Object Notation). The next section presents the encoding of data format. It is proposed to provide examples about the data format. Finally, SP document contains references to standards and demarcations used to build the Semantic Profile. [3][16]

3. METHODOLOGY

This chapter describes the technologies used in development of pilot applications. The applications consist of several software components connected to Arrowhead Network as well as separate web applications providing graphical user interfaces. In addition, pilot applications include hardware devices. Nevertheless, this chapter does not cover hardware design because the hardware is designed by project partners.

First part of this chapter describes the role of Arrowhead Framework at the development. First, it is described how Core Services are used at the pilot applications. Next, Arrowhead Management Tool is presented. It is a monitoring and managing application for Core Systems. After this, the development of the services is described. Services are based on REST architectural style. Finally, an overview of using Arrowhead document model is given.

The next part of this chapter introduces the technologies used in web applications. The applications were developed with Bootstrap framework version 3.3.4 and jQuery version 1.11.1. Graphs were created with Morris JS library version 0.5.0. Finally, a technology called CORS (Cross-Origin Resource Sharing) is presented. It was needed to enable the AJAX (Asynchronous JavaScript and XML) requests to the services from the web applications because they were running locally on the computer outside of Arrowhead Network.

3.1 Using the Arrowhead Framework

This section describes how Arrowhead Framework was used as a part of pilot applications. The development was done according to the guidelines presented in Arrowhead Cookbook version 1.5, which is available for project partners. Cookbook introduces a set of minimum requirements how applications should function with the Core Systems. Furthermore, it gives instructions for using the document model. [14]

Section 3.1.1 describes the method for using Core Services as a part of the pilot applications. Section 3.1.2 introduces an Arrowhead Management Tool which is web application providing a graphical user interface for Core Systems. Section 3.1.3 discusses service development. Finally, section 3.1.4 discusses how Arrowhead document model was used.

3.1.1 Core Services

As mentioned earlier, Arrowhead Framework gives a set of minimum requirements how a system should function with Core Services at Arrowhead Network. According to [14], the system should at least use the Service Discovery provided by Service Registry, it should be able to authorise service consumers with Authorisation System and handle certificates. Finally, it should be able to support the Orchestration functionality by using the Service Orchestration. To cover these requirements, Core Services must be included in the pilot applications.

A plugin was distributed for the project partners. It provides an API for the Core System prototypes. The prototypes are hosted at a temporary server where all project partners have an access. With the plugin, partners can use the Core Services at the pilot applications through API. With this plugin, services can be published to Service Registry and they can be searched by consuming systems, authorisation can be checked from Authorisation System and finally orchestration rules can be requested from Orchestration System.

3.1.2 Arrowhead Management Tool

Arrowhead Management Tool is a web application giving a graphical user interface for the Core Systems. Management tool is needed mostly to manage authorisation and orchestration rules, but also to monitor the Core Systems. It consists of three functional components called Authorisation MMI, Service Repository MMI and Orchestration MMI. The Component model of Arrowhead Management Tool is presented in Figure 7.

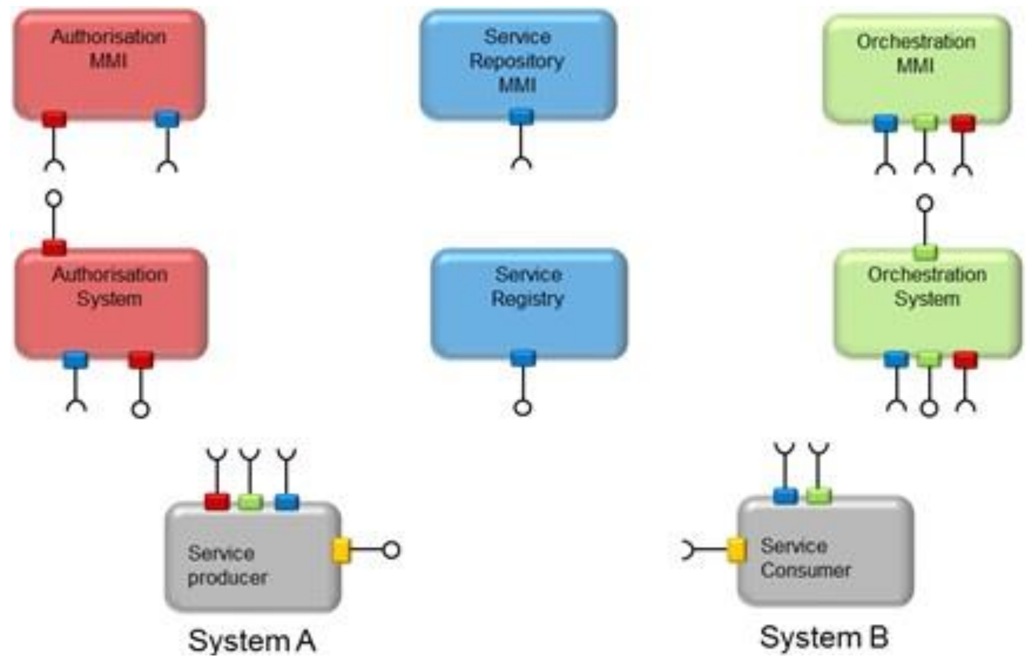
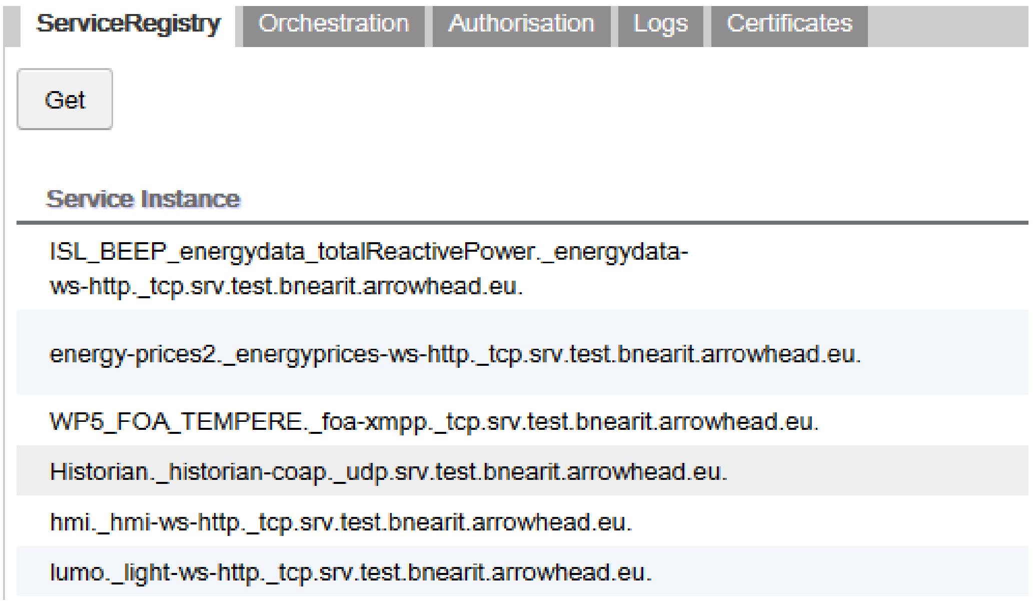


Figure 7. Arrowhead Management Tool components [14].

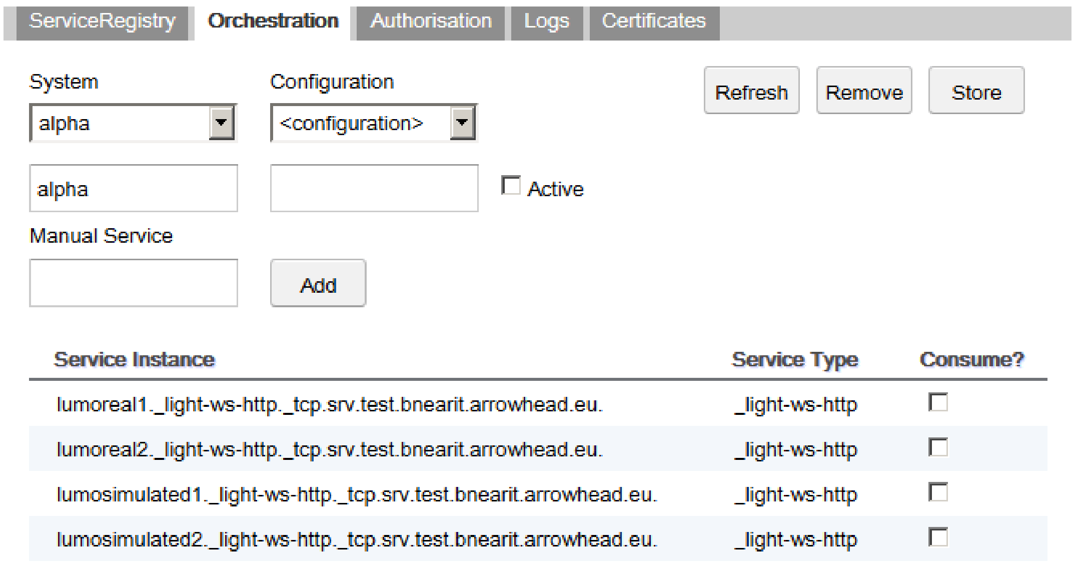


Service Instance
ISL_BEEP_energydata_totalReactivePower._energydata-ws-http._tcp.srv.test.bnearit.arrowhead.eu.
energy-prices2._energyprices-ws-http._tcp.srv.test.bnearit.arrowhead.eu.
WP5_FOA_TEMPERE._foa-xmpp._tcp.srv.test.bnearit.arrowhead.eu.
Historian._historian-coap._udp.srv.test.bnearit.arrowhead.eu.
hmi._hmi-ws-http._tcp.srv.test.bnearit.arrowhead.eu.
lumo._light-ws-http._tcp.srv.test.bnearit.arrowhead.eu.

Figure 8. *ServiceRegistry tab of Management Tool.*

Figure 8 presents the first section of Management Tool. It provides an access to Service Registry. It shows the list of all service instances published to the Service Registry. GET-button updates the list. By using this section, a developer can verify that the services instances of the pilot application are published correctly to the Service Registry by searching for them from the list.

The Orchestration section of Management Tool is used to manage the orchestration rules. The orchestration section is presented in Figure 9.



System

alpha

Configuration

<configuration>

alpha

Active

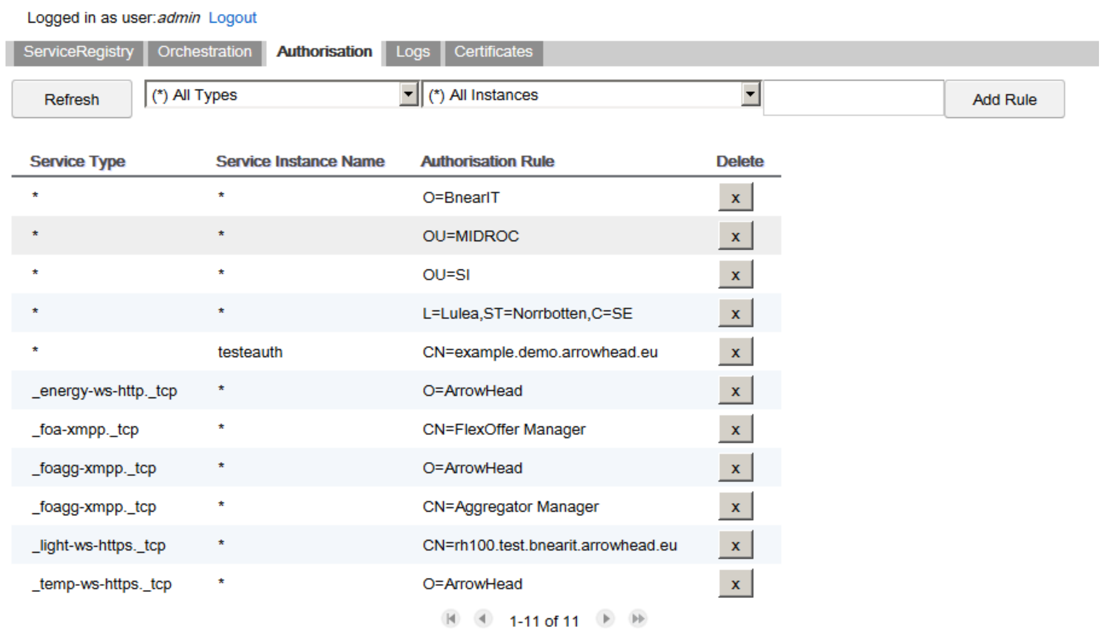
Manual Service

Service Instance	Service Type	Consume?
lumoreal1._light-ws-http._tcp.srv.test.bnearit.arrowhead.eu.	_light-ws-http	☐
lumoreal2._light-ws-http._tcp.srv.test.bnearit.arrowhead.eu.	_light-ws-http	☐
lumosimulated1._light-ws-http._tcp.srv.test.bnearit.arrowhead.eu.	_light-ws-http	☐
lumosimulated2._light-ws-http._tcp.srv.test.bnearit.arrowhead.eu.	_light-ws-http	☐

Figure 9. *Orchestration tab of Management Tool.*

Managing the orchestrations rules with Orchestration tab is straightforward. First, the name of the system is either selected from the dropdown list or typed manually to the field under the list. Similarly, the configuration is chosen from the list or a new configuration is defined by typing its name to the field under the dropdown list. Next, a user can define orchestration rules by choosing the services to consume by adding a tag to the box at the end of the service description. The configuration is then saved with Store-button. If the application has multiple configurations, one of them can be activated by tagging the Active-box. When the application with corresponding system name is started, it fetches the configuration from Orchestration System.

Authorisation tab provides a graphical interface for Authorisation System. It is used to define the authorisation rules for the services. Authorisation tab is presented in Figure 10.



Service Type	Service Instance Name	Authorisation Rule	Delete
*	*	O=BnearIT	x
*	*	OU=MIDROC	x
*	*	OU=SI	x
*	*	L=Lulea,ST=Norrbotten,C=SE	x
*	testeauth	CN=example.demo.arrowhead.eu	x
_energy-ws-http._tcp	*	O=ArrowHead	x
_foa-xmpp._tcp	*	CN=FlexOffer Manager	x
_foagg-xmpp._tcp	*	O=ArrowHead	x
_foagg-xmpp._tcp	*	CN=Aggregator Manager	x
_light-ws-https._tcp	*	CN=rh100.test.bnearit.arrowhead.eu	x
_temp-ws-https._tcp	*	O=ArrowHead	x

Figure 10. *Authorisation tab of Management Tool.*

Since Arrowhead uses X.509 certificates, authorisation rules are based on allowing systems with certain certificates to consume the services. CN (Common Name) field of the certificate is used as an authorisation rule. A system with authorised CN at its certificate is authorised to use the service. Authorisation rules can be defined for individual services or for every service with chosen type by using an asterisk. For example, it can be seen from the figure that all consumers having a certificate with rh100.test.bnearit.arrowhead.eu as a CN field are authorised to consume all services with type _light-ws-https_.tcp.

The next section of Management Tool is used to inspect log files. Log includes information about the events at Core Service. The final section of Management Tool is Certificate Management. The user can read the content of different certificates from this tab. In addition, it is possible to save new certificates and to import existing certificates. Neither of the tabs was used during the development of pilot applications.

3.1.3 Service development

Pilot applications described in this thesis use REST architectural style in services thus making them RESTful Web Services. In this architectural style, data and functionality are considered as resources and they are identified with URI (Uniform Resource Identifier). Resources can be accessed with HTTP (Hypertext Transfer Protocol) methods including GET, POST, PUT and DELETE. [32] Therefore, the message exchange with services is done by requesting different URIs with HTTP methods.

Services comply with request-response pattern. Interfaces were designed to support HTTP GET for requesting data and HTTP POST for modifying data or commanding devices. PUT and DELETE methods were not used. Service providers return the requested data in HTTP response entity bodies in JSON format. HTTP status codes were used to give more information to the consumer. In general, status code 200 (Ok) was used with successful request whereas code 500 (Internal Server Error) was used as a general code in most error situations. Additionally, code 403 (Forbidden) was used when service consumer was not authorised to use the service. Finally, 400 (Bad Request) was used to prevent the service consumer from sending unsupported input data.

3.1.4 Document model

Arrowhead Document Model was used to document the pilot applications. SD and IDD documents were created to describe every service of the pilot applications. SP documents were also created to describe the semantic profiles of the services. SysD documents were created to describe Engine Block Heater Controller, Light Controller, Light Managing System and Controller System. These systems are presented later in this thesis at Chapter 4. One SoSD document was created to describe Urban Management System which is later presented at subsection 4.3. SysDD document was not created since it is not a mandatory document. SoSDD and CP documents were left out because they included several sections which were not possible to fill yet. Documents are not presented in this thesis due their long length.

Some challenges were faced with the documents. First, it turned out that it is really time-consuming to create them. Moreover, changes to one document caused sometimes changes to other documents as well. Finally, even though Arrowhead Framework includes a template for every document type, it was still quite difficult to understand how the documents should be filled.

3.2 Web applications

This section describes the technologies used to develop the web applications. Web applications provide a graphical user interface for the pilot applications. Technologies used in the web applications are Bootstrap framework version 3.3.4, jQuery version 1.11.1 and Morris JS version 0.5.0. In addition, CORS technology was also needed since the web applications were running at different domain than Arrowhead Network. Chosen technologies are presented in following subsections.

3.2.1 Bootstrap framework

Bootstrap is HTML, CSS and JavaScript framework for developing responsive web applications. It was originally developed by a designer of Twitter. Bootstrap framework has been designed especially for mobile devices. Being responsive means that the components at web application can scale automatically based on the size of the display. Bootstrap is also open source giving the users a possibility to customize it freely. [17]

Bootstrap offers a set of visualisation components for web applications. Adding the components to the application is simple. In most cases the developer can just copy the ready-made source code from Bootstrap web page and add it to the HTML document. Moreover, some of the components have built-in functionality what reduces the amount of JavaScript code needed. Bootstrap is documented well and the user is supported with numerous examples. This makes the learning curve rather small. Bootstrap version 3.3.4 was used at the web applications.

3.2.2 jQuery

jQuery is a JavaScript library, which makes the writing of JavaScript programs easier. It is mainly used to manipulate the DOM (Document Object Model) tree of the HTML document and to perform AJAX requests. [28] Especially the support for AJAX requests makes jQuery an essential part of developed web applications because the resources of RESTful Web Service can be accessed with it. Another asset of jQuery is the way it handles JSON format. JSON data does not need to be parsed but the fields can be accessed instantly just by using the identifier names.

jQuery is not trouble-free. The biggest problem is that manipulating the DOM tree is time consuming. Especially large applications might cause a burden to the developer due the size of the DOM tree. Nevertheless, jQuery is quite fast to learn and applications developed at this project were rather simple what supports the decision that jQuery was chosen. Version 1.11.1 was used at the web applications.

3.2.3 Morris JS

Morris is a JavaScript library for drawing charts. It supports multiple chart types including line charts, bar charts, area charts and donut charts. In addition, charts can be configured to scale their size automatically. The library includes a user-friendly API making it easy to use. [40]

Since the web applications are a monitoring and management application, graphical data presentation is a descriptive way to present the data. Automatic scaling is a useful feature for presenting the data, which is why Morris JS library was chosen. Version 0.5.0 was used at the web applications.

3.2.4 Cross-Origin Resource Sharing

Web applications have a policy called Same-Origin which prevents the applications from using the resources from another domain. Moreover, it also limits HTTP requests to other domains. [21] Since web applications were running locally at the computer whereas the rest of the components were at Arrowhead Network, the domains were different. Therefore, it was not possible to perform AJAX requests to the services from the web applications without bypassing Same-Origin policy.

CORS is a mechanism that allows the access to resources at other domains. It is based on HTTP headers. CORS is enabled from the server side by adding a header *Access-Control-Allow-Origin* to the HTTP response. The value of the header defines the origin domains accepted by the server. The value can be either the name of a specific domain or it can be an asterisk, which makes the resource available for every domain. CORS includes a set of additional headers as well but it is not mandatory to use them. [21]

After CORS was enabled, web applications were able to use the services within Arrowhead Network. CORS was enabled by adding Access-Control-Allow-Origin header to the HTTP responses, which were sent by service providers.

4. IMPLEMENTATION

This chapter describes the pilot applications developed for the Arrowhead Project at FAST-lab Tampere University of Technology. They were targeted to Smart Buildings and Infrastructure domain. Two pilot applications were deployed. Deployment included the installation of hardware devices to the university and software development. Furthermore, third application was developed by reusing the components from the first two applications.

Pilot applications demonstrate how Arrowhead Framework can be used to develop Smart City applications. The first application is Light Management Tool used for monitoring and managing street lights. It was developed in cooperation with C2 SmartLight Oy. Lights are controlled with a device called SmartLumo designed especially for controlling the LED (Light-Emitting Diode) luminaires. The second application is called Engine Block Heater Controller, which is a monitoring and managing tool for engine block heaters. This application was developed with THT Control Oy. Finally, a third pilot application called Urban Management System was created by integrating the components from the first two applications. At this application, street lights and engine block heaters are controlled with separate Controller System. Therefore, the functionality is achieved by using multiple systems in collaborative manner.

Pilot applications are presented in following subsections by describing the hardware setup, software architecture and the user interfaces. Light Management Tool is described at Section 4.1 and Engine Block Heater Controller at Section 4.2. Next, Urban Management System is presented at Section 4.3. Finally, a discussion is given at Section 4.4.

4.1 Light Management Tool pilot application

Light Management Tool is a management and monitoring tool designed for maintenance personnel. It demonstrates how street lights could be controlled by using Arrowhead Framework. The application is developed in cooperation with C2 SmartLight Oy.

The application consists of a smart light controller called SmartLumo developed by C2 SmartLight Oy. It is used to control the LED luminaires. SmartLumo is not Arrowhead compliant and thus it cannot offer services to other devices at Arrowhead Network. Therefore, an adapter component called Light Controller was used to offers services to the network. Since the application is a demonstration rather than commercial application, the setup includes only one SmartLumo and a luminaire. Therefore, Light Controller was designed to provide also a simulated version of SmartLumo to simulate bigger amount of luminaires. Next, a composing software component called Light Management was developed. It consumes a set of Light Controllers and thus provides an API for accessing multiple SmartLumos. Since Light Management is a software component, it was possible to connect it to Arrowhead Network without adapters. Finally, a web application was developed to provide a graphical user interface for the application.

The implementation is presented at following subsections. Subsection 4.1.1 describes the hardware setup, Subsection 4.1.2 presents the software architecture and finally Subsection 4.1.3 presents the user interface.

4.1.1 Hardware setup

This section describes the hardware used at the implementation. Since the pilot application is a project demo rather than a commercial application, hardware needed to be suitable for demo purposes. Therefore, a small LED luminaire was chosen instead of actual street lights. Following hardware was used:

- C2 SmartLumo
- Motion sensor
- Hidealite 1202 Multi
- Hidealite Jolly Pro

C2 SmartLumo is an intelligent light controller unit used for controlling LED luminaires. It is designed specifically for the needs of industry, street lights, parking lots and parks. It has multiple controlling options including schedule-based control and sensor-based control with assistance of motion detectors. SmartLumo can be operated as a stand-alone controller or as a part of distributed control solution with multiple SmartLumos. Communication between the controllers is done wirelessly with ZigBee thus reduces the wiring in implementations. SmartLumo offers 0–10V control voltage for controlling the luminaires. The output is based on the configurations of the device: Output values can change based on time periods of the day or the sensor input. [18]

In this implementation, one C2 SmartLumo was used in two different modes, which are manual mode and scheduled mode. In manual mode, the output signal of the controller can be manually set to a certain level. In scheduled mode, the output signal is set based on a dimming profile and motion detection. The profile was defined so that in normal state the output is rather low, approximately 1V, but when motion is detected, SmartLumo sets the output to a certain level based on the time of the day. After approximately 10s the output level decreases to 50% and finally after 20s the output reverts back to 1V. To enable this behaviour, a motion sensor was required. This setup includes a motion sensor connected to secondary C2 SmartLumo. When motion is detected, this controller sends control commands to the primary SmartLumo, which then sets the output to the requested level.

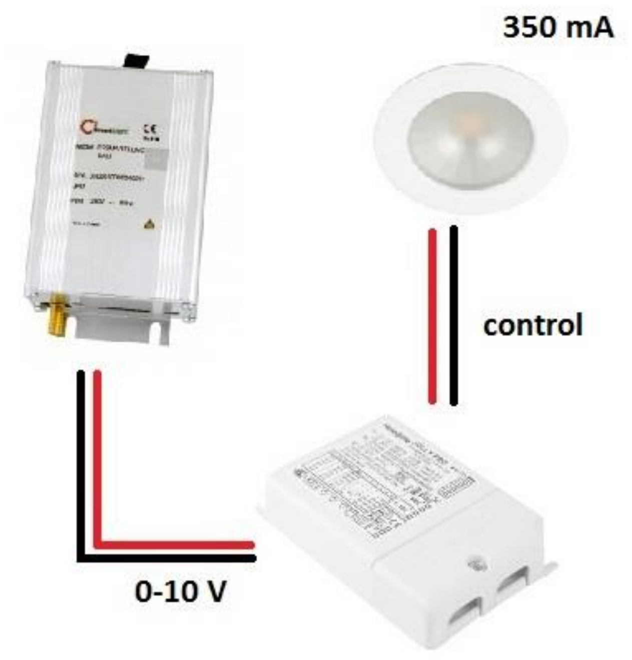


Figure 11. *Hardware setup of Light Management Tool.*

Figure 11 presents the setup used at Light Management Tool. Secondary SmartLumo is left out from the figure. The setup includes SmartLumo, 1202 Multi LED luminaire and Jolly Pro driver. Jolly Pro is a driver for LED luminaires with a built-in dimmer function. It can operate luminaires with both constant voltage and constant current. The device provides various ways for regulating the dimming including 1–10V voltage signal, a potentiometer and a push button. [25] In this setup, Jolly Pro was used to regulate Hidealite 1202 Multi, which is a 3.5W LED luminaire with a constant current of 350mA [24]. Jolly Pro was connected to the output of SmartLumo. Dimming was thus controlled with 1–10V voltage received from SmartLumo.

4.1.2 Software architecture

This section describes the software architecture of Light Management Tool. The software architecture consists of three systems including Light Controller, Light Management and User Interface. The setup includes also Arrowhead Core Systems, which are running at temporary server at Arrowhead Network. Software architecture is presented in Figure 12.

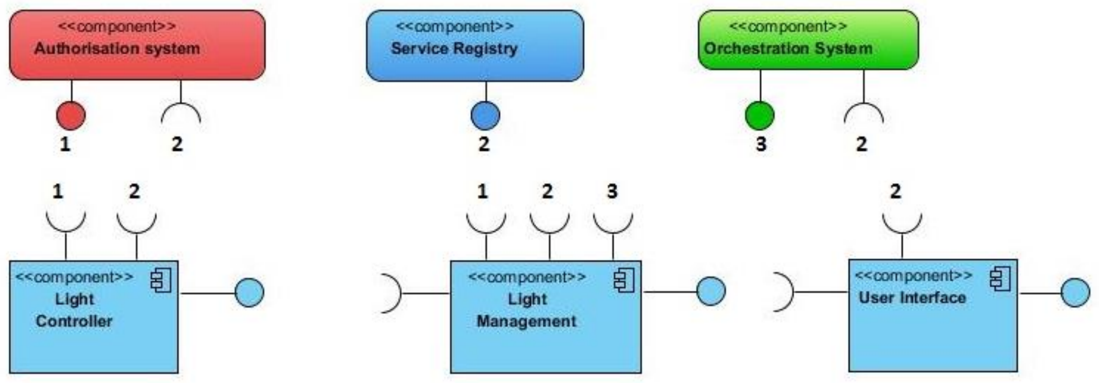


Figure 12. *Software architecture of Light Management Tool.*

Light Controller is an adapter component, which connects SmartLumo to Arrowhead Network. Adapter component is needed because SmartLumo is not Arrowhead compliant by itself. Instead, SmartLumo is operated in Arrowhead Network through the RESTful interface provided by Light Controller. The communication between Light Controller and SmartLumo is enabled with SSH (Secure Shell).

Light Controller publishes its endpoint to Service Registry. Therefore, it can be discovered and consumed by other devices at the network. Light Controller uses also Authorisation System to verify that the consumer is authorised to use its services. Authorisation rules can be set for the Light Controller by using the Authorisation System via Arrowhead Management Tool. Light Controller does not use Orchestration System since it does not need to consume anything. RESTful interface of Light Controller is presented in Table 2.

Table 2. *Interface of Light Controller.*

Function	**URL**	**Method**	**parameters**	**Output**

Get status.	/status	GET	-	{
				“name” : String,
				“output” : Integer,
				“timestamp” : String,
				“mode” : String
				}

Set output.	/output	POST	**query parameter:**
			level = [0..100]
				{
				“name” : String,
				“output” : Integer,
				“timestamp” : String,
				“mode” : String
				}

Set operating mode.	/settings	POST	**query parameter:**
			mode =
			{ Simulated | Manual }
				{
				“name” : String,
				“output” : Integer,
				“timestamp” : String,
				“mode” : String
				}

The interface of Light Controller offers functions for reading the current status of C2 SmartLumo with HTTP GET request. In addition, Light Controller offers a function for changing the operating mode between simulated and manual mode. Finally, the output can be set to certain percentage level if SmartLumo is in manual mode. Changes are made with HTTP POST request and new data is passed to the Light Controller with query parameters.

Errors are managed with HTTP status codes. If request is done with an incorrect value, Light Controller responds with an HTTP response containing a status code 400 (Bad Request). If Light Controller encounters problems with SSH connection, it responds with status code 500 (Internal Server Error). Finally, if the consumer is not authorised to use the services, Light Controller responds with status code 403 (Forbidden).

As can be seen from the table, same form is used in every JSON message that is received from the Light Controller. The message includes the name of the device, output value, the time of request and the current operating mode. An example message is presented below.

```
{
"name": "lumo10",
"value": 26,
"timestamp": "2015-08-11 13:25:53",
"mode": "Scheduled"
}
```

The example JSON message states that the output of lumo10 is 26% of the maximum output. In addition, SmartLumo is at scheduled mode. Timestamp shows the time of request.

The next component in the software architecture is Light Management. It is a composing software component providing a RESTful interface for operating a set of Light Controllers. Light Controllers must be consumed in order to use their interfaces. Light Management uses Service Registry to discover the Light Controllers. Furthermore, Orchestration System is used because Light Management might consume numerous Light Controllers at the same time and there is a need for multiple setups. Light Management can easily change the set of consumed Light Controllers by fetching new orchestration rules from Orchestration System. In commercial application, orchestration rules could be used to change for example the street section. Finally, Authorisation System is used to verify that the consumer is authorised to use the services of Light Management. The interface of Light Management is presented in Table 3.

Table 3. *Interface of Light Management.*

Function	**URL**	**Method**	**Parameters**	**Output**
Get the name of every consumed Light Controller.	/devices	GET	-	{ “devices” : [String] }
Get status of every consumed Light Controller.	/status	GET	-	{ "allReadings" : [{ “name” : String, “output” : Integer, “timestamp” : String, “mode” : String }] }
Get status of one specific Light Controller.	/status/<device>	GET	**Path parameter:** <device>	{ “name” : String, “output” : Integer, “timestamp” : String, “mode” : String }
Set the output to every consumed Light Controller, which are in manual mode.	/output	POST	**query parameter:** level = [0..100]	{ “modified” : [{ “name” : String, “output” : Integer, “timestamp” : String, “mode” : String }] }

Set the output to one specific Light Controller.	/output/<device>	POST	**Path parameter:** <device>
			
			query parameter: level = [0..100]
				{
				“name” : String,
				“output” : Integer,
				“timestamp” : String,
				“mode” : String
				}

Set the mode to every consumed Light Controller.	/settings	POST	**query parameter:** mode = { Simulated | Manual }	{
				“modified” :
				[{
				“name” : String,
				“output” : Integer,
				“timestamp” : String,
				“mode” : String
				}]
				}

Set the mode of one specific Light Controller.	/settings/<devic e>	POST	**Path parameter:** <device>
			
			query parameter: mode = { Simulated | Manual }
				{
				“name” : String,
				“output” : Integer,
				“timestamp” : String,
				“mode” : String
				}

The interface offers basically the same functionality as the interface of Light Controller but there are functions to operate multiple Light Controllers together as well. In order to use the interface, at least one Light Controller must be consumed. The interface offers a function to get the list of consumed Light Controllers in an array. An example response message is given below.

```
{
“devices” : [ “lumo10”, “lumo11” ]
}
```

In this case, two devices, lumo10 and lumo11, are consumed. Light Management can now be used to request their services. Light Management can operate both devices at the same time, or requests can be targeted to one device at the time. The target device is defined at URI. The data formats of Light Management are based on the data format of Light Controller. If only one Light Controller is accessed, the JSON message received from Light Controller is simply resent from Light Management. When multiple Light Controllers are accessed at the same time, Light Management combines every received JSON message to an array. The array is named with distinctive identifier. For example, if status is request from multiple devices, the identifier is “allReadings”. If control commands are sent to multiple devices, the identifier is “modified”. This way a user can instantly see the type of response. An example JSON message is given at the next page. The example presents a message received after the operating mode of both devices, lu-mo10 and lumo11, is changed to Manual.

```
{
“modified”:
[{
"name": "lumo10",
"value": 26,
"timestamp": "2015-08-11 13:25:53",
"mode": "Manual"
} , {
"name": "lumo11",
"value": 26,
"timestamp": "2015-08-11 13:25:53",
"mode": "Manual"
}]
```

Similar to Light Controller, HTTP GET methods are used to request the data and HTTP POST are used to do the modifications. New data is sent with query parameters. Errors are managed with HTTP status codes. If Light Management tries to access a Light Controller which is not consumed, the interface responds with status code 404 (Not Found). Incorrect parameters are not handled at Light Management because that error handling is already implemented to the Light Controller, which sends HTTP response with status code 400 (Bad Request). The response is then received by Light Management, which forwards the same error message to the user. In case of possible internal errors at Light Management, HTTP status code 500 (Internal Server Error) is used. Finally, Light Management responds with HTTP status code 403 (Forbidden), if the consumer is not authorised to use its services.

Final Component of the software architecture is User Interface. User Interface offers a RESTful interface with the same functionality as Light Management but it adds CORS headers to the HTTP responses. Since the web application is running at different domain, CORS is needed enable the AJAX requests to the services at Arrowhead Network.

Since User Interface is only needed to enable the AJAX requests from the web application, there is no need to publish it to Service Registry. Although. User Interface uses the Service Registry to find the endpoint of Light Management, which it needs to consume. When data is requested from the User Interface, it forwards the request to Light Management. When Light Management replies with HTTP response, User Interface adds CORS headers to the response and resends the response to web application.

4.1.3 User interface

This section describes the user interface of Light Management Tool. It is a web application providing a graphical user interface for managing and monitoring Light Controllers. The application requests data with AJAX requests from user interface software component as presented in 4.1.2. The user interface is based on polling meaning that new data is requested after constant intervals. In this implementation, the user interface was used on a PC with Mozilla Firefox version 38.3.0. The user interface is presented in Figure 13.

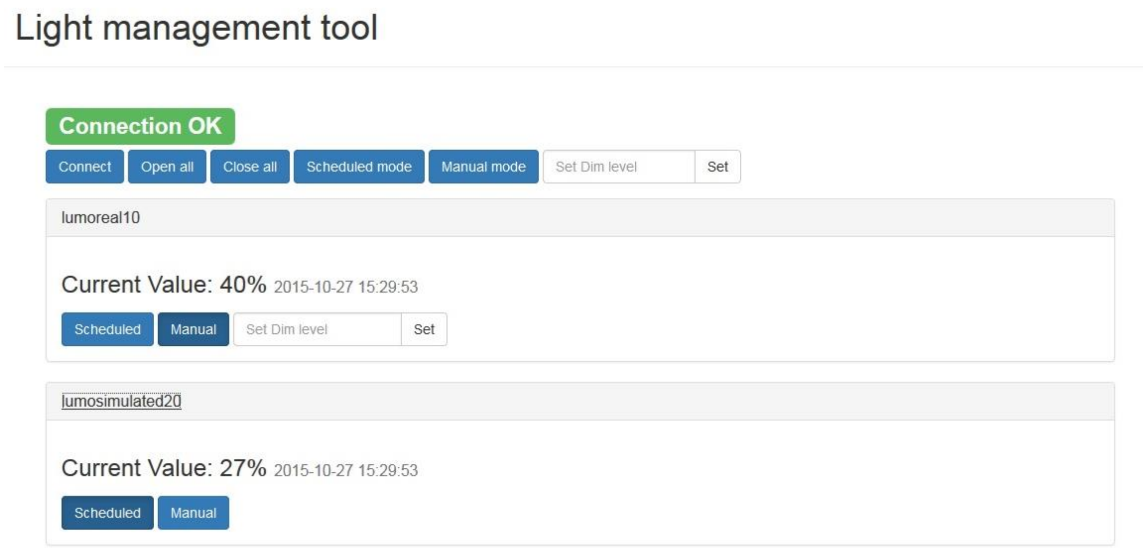


Figure 13. *User Interface of Light Management Tool.*

The user interface can be divided into two sections. At the top, there is a status field informing the user about the status of the connection. “Connection OK” means that the polling sequence is running normally. In case of error, the text changes into “Connection Down” and the polling stops. The top section has also a set of buttons. The buttons are described in a list below.

- Connect: Establish connection.
- Open All: Expands every Light Controller panel.
- Close All: Collapses every Light Controller panel.
- Scheduled Mode: Sets every Light Controller to scheduled mode.
- Manual Mode: Sets every Light Controller to manual mode.
- Set: Sets a new dimming value for every device, which is in manual mode.

The first push button is used to establish a connection. The next two buttons are used to manipulate the view. They expand and collapse the fields containing the information of individual Light Controllers. This feature helps to inspect quickly the status of every Light Controller. The next two buttons changes the operating mode of every consumed

Light Controller. This feature is useful if large numbers of devices are consumed. Finally, an input field is used for setting the dimming level for every Light Controller.

The second section displays consumed Light Controllers in separate panels. The panel consists of the header and the body. The header has a text field showing the name of the device whereas the body has device-specific controllers and displays. The body has a text field showing the current output level of Light Controller. Finally, there is a timestamp describing the time when the value was changed.

There are two buttons under the text field for changing the operating mode of the device. The button representing the current mode is highlighted. When the mode is set to manual, a new input field appears to the body. This field is used to manually set the output level of the device. When the device is set back to scheduled mode, the input field disappears.

Every consumed Light Controller has its own panel. The user interface is dynamical meaning that if the number of consumed Light Controllers changes during the runtime, the user interface adds and deletes panels automatically during the next polling interval. This can happen if for example orchestration rules are changed.

4.2 Engine Block Heater Controller pilot application

The second pilot application is called Engine Block Heater Controller. It demonstrates how Arrowhead Framework could be used to manage and monitor engine block heaters. The heater is developed by THT Control Oy who is also the main partner at this implementation. Since the application is a demonstration, the setup includes only one engine block heater. The heater is not originally Arrowhead compliant, so it is connected to Arrowhead Network with an adapter component. Finally, the implementation includes a web application providing a graphical user interface for the application.

Engine Block Heater Controller is described at following subsections. Hardware setup is described at Section 4.2.1, the software architecture is introduced at Section 4.2.2 and finally Section 4.2.3 describes the user interface.

4.2.1 Hardware setup

The hardware setup of the implementation consists of one engine block heater only. It was installed in parking area at Tampere University of Technology for this pilot application. Engine block heater is a car heating post developed by THT Control Oy. There are a few variations of the heater available providing a different number of power sockets. The heater includes also an LED luminaire to help the user to find the power sockets at dark. In addition, the LED has also a blinking function, which helps the user to identify the heater from a parking area. Engine Block Heater is presented in Figure 14.

Figure 14. *Engine block heater.*

The heater offers a set of intelligent features including a heating time, which changes based on the temperature. This feature makes sure that a car is not heated longer than necessary and energy is not wasted to unnecessary heating. Additionally, the heater provides many kinds of statistics that can be requested from the heater such as the number of the heat events and the total energy consumption of the day. Communication with the heater is based on HTTP requests to THT server.

A set of engine block heaters forms a zone. One of the heaters at zone, known as a Gateway, manages all data transmission between the zone and the THT server. Other heaters at the zone are only slaves and thus they cannot connect directly to the THT server. Instead, heaters within the zone can communicate with the gateway wirelessly which then sends the data to the THT server.

4.2.2 Software architecture

This section describes the software architecture of Engine Block Heater Controller pilot application. The architecture consists of two main components called Engine Block Heater Controller and User Interface. Furthermore, Arrowhead Core Systems are included to the architecture same way as in Light Management Tool. They are running at a temporary server and can be accessed through API. The architecture is presented in Figure 15.

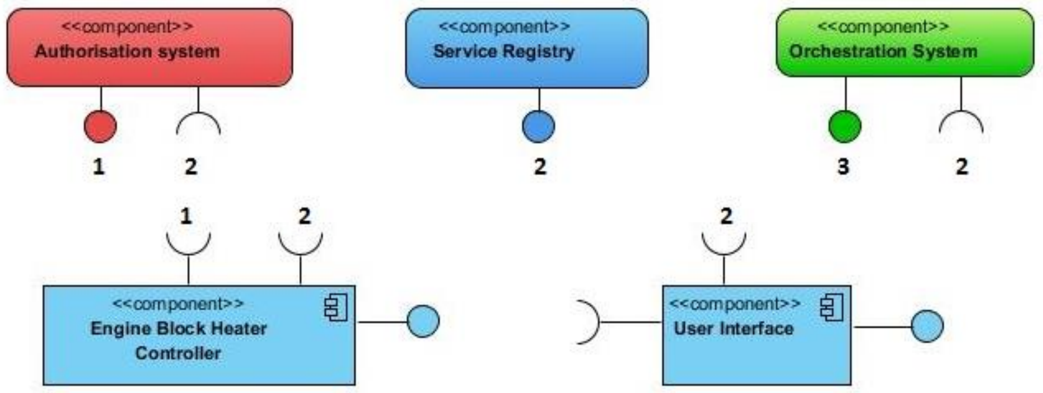


Figure 15. *Software architecture of Engine Block Heater Controller.*

Since the engine block heater is not Arrowhead compliant, it cannot be connected directly to Arrowhead Network. Therefore, an adapter component is needed. Engine Block Heater Controller is designed for this purpose. It provides a RESTful interface for the engine block heater. The heater is accessed from Engine Block Heater Controller with HTTP requests, which are sent to THT server. This way the heater can be operated from Arrowhead Network.

Engine Block Heater Controller uses Authorisation System and Service Registry Core Systems. Authorisation System is used to verify that the consumer is authorised to use the services of Engine Block Heater Controller. Furthermore, Service Registry is used to publish the endpoint of Engine Block Heater Controller thus making it discoverable for other systems at Arrowhead Network. Orchestration System is not needed at this implementation because the setup includes only one heater. Thus, there is no need for orchestration.

As described in previous subsection, a set of heaters forms a zone. Moreover, the zone has one heater called Gateway, which manages all data transmission between the zone and the THT Server. Even though the implementation includes only one heater, Engine Block Heater Controller was designed so that the whole zone can be accessed through the same interface. The RESTful interface of Engine Block Heater Controller is presented in Table 4.

Table 4. *Interface of Engine Block Heater Controller.*

Function	URL	Method	Parameters	Output

Request data from zone.	/zone	GET	**query parameter:**
			attr = { Temperature | Light | Current | OutputOn | TodayHeatEvent | TodayOnMinutes | TodayEnergy | YesterdayHeatEvents | Yester-dayOnMinutes | YesterdayEnergy }
				{
				“attr” : String,
				“value” : String,
				}

Send data to zone.	/zone	POST	**query parameters:**
			attr = { Temperature | Light }
			value = [-50…50] or [0 | 1]
				{
				“attr” : String,
				“value” : String,
				}

Request data from pole.	/zone/<pole>	GET	**Path parameter:**
			<pole>
			
			query parameter:
			attr = { Current | OutputOn | TodayHeatEvent | TodayOnMinutes | TodayEnergy | Yester-dayHeatEvents | Yester-dayOnMinutes | YesterdayEnergy }
				{
				“attr” : String,
				“value” : String,
				}

Send data to pole.	/zone/<pole>	POST	**Path parameter:** <pole> **query parameters:** attr = Blink value = [0 | 1]	-

Request data from socket.	/zone/<pole>/<socket>	GET	**Path parameter:**
			<pole>
			<socket>
			
			query parameter:
			attr = { Current | ManualMinutes | ManualOn | MinutesUntilReady | TodayEnergy | TodayHeatEvent | TodayOnMinutes | YesterdayHeatEvents | Yester-dayOnMinutes | YesterdayEnergy |
			OutputOn |
			HeatMinutes }
				With Current:
				{
				“current” : String,
				“timestamp” : String,
				}
				
				With others:
				{
				“attr” : String,
				“value” : String,
				}

Send data to socket.	/zone/<pole >/<socket>	POST	**Path parameter:**
			<pole>
			<socket>
			
			query parameters:
			attr = OutputOn
			value = [0 | 1]
				{
				“attr” : String,
				“value” : String,
				}

Since the zone can consist of several heaters with a different number of power sockets, the interface needs to be hierarchical enough to provide an access to every possible combination. The solution is to use path parameters. The interface is designed so that the user can define the target heater and target socket to URI. This way every possible socket can be accessed. Furthermore, the user can also request combined data from every heater at the zone or every socket at the heater. The interface is thus offering services at three levels including socket, heater and zone. The heater is called pole at the table.

Different attributes of the engine block heater are accessed by defining the target attribute with a query parameter. This way the interface is kept relatively simple even though there are many attributes available. The interface is designed so that data is requested with HTTP GET request and modifications are done with HTTP POST request. Query parameters define the requested attribute and the data to be sent.

Socket level offers the most of the functionality by providing many different attributes to be requested. Data is requested with HTTP GET. Additionally, every socket can be manually set on and off with HTTP POST request. The value is defined with the query parameter. Heater level has a function to blink the LED of the heater. Blinking is started with HTTP POST request. Some of the attributes at socket level can also be requested from the heater level. When the attribute is requested from the entire heater, the response will be the sum of the values combined from every socket at the heater. For example, the total energy consumption of the heater can be requested with a single HTTP GET request. The same logic is implemented to zone level as well. When the energy consumption is requested from the zone level, the sum of every pole’s energy consumptions at the zone is returned. This way the energy consumption of the whole zone can be requested with a single HTTP GET request. There is also a function for sending data to the zone level. New temperature can be sent to the zone with HTTP POST request. The temperature is sent with the query parameter and the value can be -50–50. Moreover, the same function can be used for setting the LED of every heater on or off. Again, the query parameter is used to send the value, which is in this case 0 or 1.

Engine Block Heater Controller uses JSON messages to send the data back to the user. Every function responds with a JSON containing the requested attribute and its value. An example JSON is given at the next page.

```
{
“attr” : “OutputOn”,
“value” : “1”,
}
```

The only difference in JSON messages is the response being received when current is requested from the socket. The response includes a timestamp, which is needed at web application in order to plot the current. An example is given below.

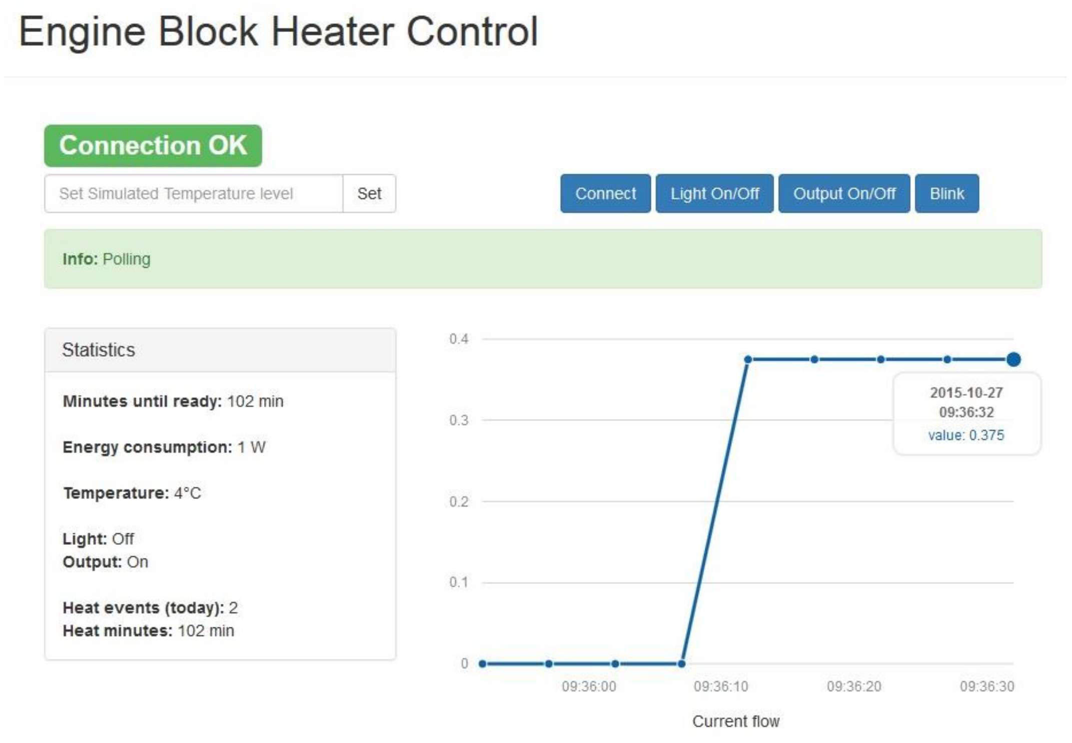
```
{
“current” : 0.5,
“timestamp” : "2015-08-11 13:25:53",
}
```

Engine Block Heater Controller uses HTTP status codes to manage errors. If HTTP POST request includes an unsupported value, Engine Block Heater Controller responds with HTTP response containing a status code 400 (Bad Request). If the connection cannot be established to THT server, the status code 500 (Internal Server Error) is used. Finally, if the consumer is not authorised to use the services, Engine Block Heater Controller will response with HTTP response containing a status code 403 (Forbidden).

The second component of the software architecture is User Interface. It is very similar to the User Interface of Light Management Tool. It offers the same interface as Engine Block Heater Controller but it adds CORS headers to the HTTP responses received from Engine Block Heater Controller in order to enable AJAX requests. Since the user interface is a web application running at different domain than Engine Block Heater Controller, AJAX requests cannot be performed without enabling CORS. User Interface is not published to Service Registry, but it uses Service Registry to discover the endpoint of Engine Block Heater Controller which it needs to consume.

4.2.3 User interface

This section presents the user interface of Engine Block Heater Controller pilot application. It is a web application providing a graphical user interface. It is used to operate and monitor a single socket of engine block heater which was installed in parking area at Tampere University of Technology. The application was used on a PC with Mozilla Firefox version 38.3.0. Similar to Light Management Tool, this user interface is based on polling. Engine Block Heater Controller needs to be consumed in order to use the user interface. The User Interface is presented in Figure 16.


Engine Block Heater Control

Connection OK

Set Simulated Temperature level | Set

Info: Polling

Statistics

Minutes until ready: 102 min

Energy consumption: 1 W

Temperature: 4°C

Light: Off
Output: On

Heat events (today): 2
Heat minutes: 102 min

Figure 16. *User Interface of Engine Block Heater Controller.*

User Interface has three different main sections. The top section is a management section. Similar to Light Management Tool, it contains a simple status field informing the user about the state of the connection. Moreover, there is a text field giving additional information about the behaviour of the user interface.

The management section contains also operating buttons. It is notable that since the user interface is based on polling, the effect of the button cannot be seen before the next polling interval. That is why the information box is available to inform the user that the push button worked. Following buttons are included:

- Connect: Establish connection
- Light On/Off: Sets the LED light of the heater on or off.
- Output On/Off: Sets the output of the power socket of the heater on or off.
- Blink: Starts to blink the LED of the heater. Blinking stops automatically.

The left-hand section of user interface is called statistics. It presents status information about the monitored power socket. The information is updated after every polling interval. The content of the section is described in following list:

- Minutes until ready: When heating is started, this section describes the remaining heating minutes.
- Energy consumption: Total energy consumption of the day in Watts.
- Temperature: Current temperature in Celsius.
- Light: The status of the light (On/Off)
- Output: The status of the power socket (On/Off).
- Heat events (today): The number of heating events during the day.
- Heat minutes: The duration of the heating.

The temperature affects the heat minutes so that lower temperature results longer heating time. Since this implementation does not include a temperature sensor, the temperature can be set with an input field which is located at the upper section of the user interface.

The right-hand section of user interface includes a graph. The graph plots the current flow of a power socket when the output is on. The Y-axis presents the current flow while X-axis presents the timestamps. This way a graphical presentation can be given about a current flow in a function of time. The graph is updated after every polling interval.

4.3 Integration of pilot applications

Since Arrowhead Network is based on SOA, it should be possible to reuse existing components in new applications. Moreover, one of the main goals of Arrowhead is to enable collaborative automation which is achieved by combining services together to create new functionalities. Therefore, a third pilot application was developed to test the reusability of the components and to demonstrate collaborative automation. The resulting application is called Urban Management System.

The goal was to create System-of-Systems, which can manage multiple urban systems with centralized control. The application was created by integrating both Light Management Tool and Engine Block Heater Controller into it. Next, a separate Controller System was developed. It was designed to send new dimming values to SmartLumos and to change the heating time of engine block heaters. The control is based on sensor input received from Temperature-Luminance sensor. Controller System can access the devices by using the APIs of Light Controller and Engine Block Heater Controller.

Controller System controls SmartLumos as the day gets brighter or darker. This way the luminaires are actively adjusted to produce only the necessary amount of light. Furthermore, Controller System sends also a new temperature to the engine block heater in constant intervals. This way the heating time of the heater is always adjusted to the temperature. In commercial application, this kind of setup could reduce the energy consumption. Since Urban Management System is only a demonstration, it uses the same hardware as described in previous sections. In commercial application, Controller System could control for example every street light and engine block heater of a city.

The sensor input is received from Lux 34 Temperature-Luminance sensor developed by Produal. It provides a 0–10V voltage signal for temperature and another 0–10V voltage signal for luminance. The sensor supports the temperature readings from the scale -50°C to 50°C and luminance reading from the scale 0lux to 10000lux. The sensor is designed for outdoor usage. [33] It is presented in Figure 17.

Figure 17. *Produal Lux 34 light level and temperature transmitter [33].*

Lux 34 is not Arrowhead compliant by itself, which means that a software adapter was needed to connect the sensor to Arrowhead Network. During the development, it was noticed that there should also be a simulated version of the sensor for demonstration purposes. Thus, the adapter component was designed to support both real and simulated version of the sensor. In order to use the simulated sensor, a new web application providing a graphical user interface was developed for this implementation. Web application was again running at different domain outside of Arrowhead Network what forced to enable CORS at the adapter component. The web application was named as Sensor Simulator.

The software architecture of Urban Management Tool is introduced at Section 4.3.1. Sensor Simulator is presented at Section 4.3.2.

4.3.1 Software architecture

This section presents the software architecture of Urban Management Tool. The implementation includes the components from both Light Management Tool and Engine Block Heater Controller. In addition, an adapter component for Temperature-Luminance sensor was included, because Lux 34 sensor is not Arrowhead compliant by itself. The adapter is called TLsensor. Controller System was also added to the architecture. Software architecture is presented in Figure 18. Numbers are used to clarify the connections between producers and consumers.

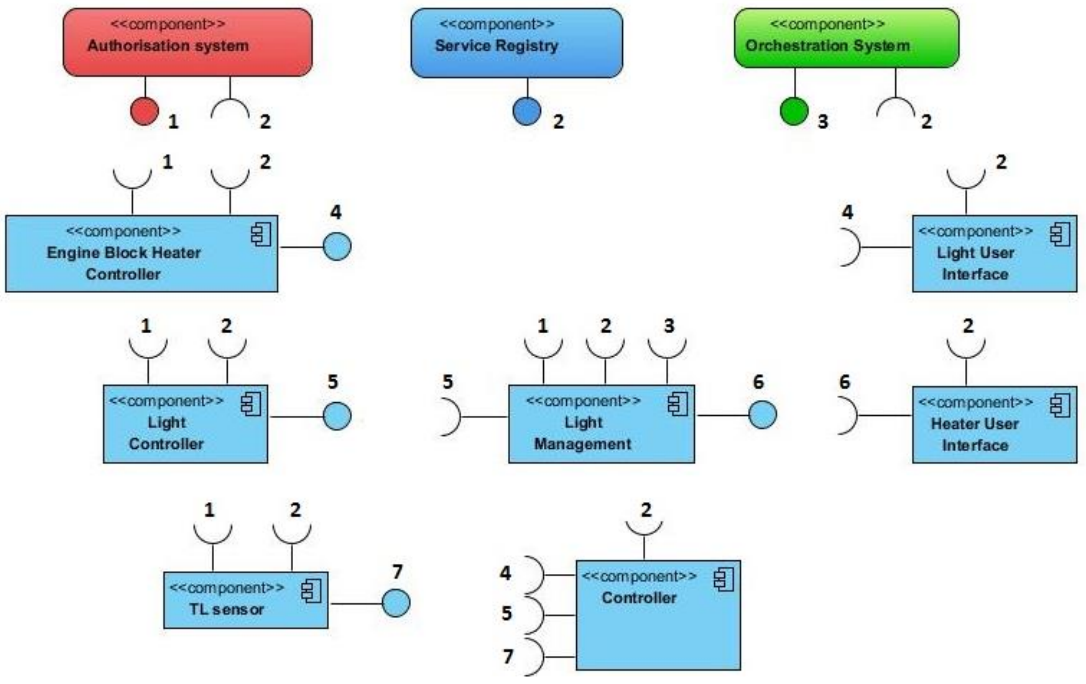


Figure 18. Software architecture of integrating implementation.

TLsensor is an adapter component offering a RESTful interface for Lux 34 sensor. It uses Authorisation System and Service Registry Core Systems. Authorisation System is used to verify that the consumer is authorised to use the services of TLsensor. Service Registry is used to publish the endpoint of TLsensor so that other systems at the Arrowhead Network can later discover it. TLsensor was designed to offer also a simulated version of Lux 34 sensor, which can be operated with web application called Sensor Simulator. Sensor Simulator is described later at Section 4.3.2. The RESTful interface of TLsensor is presented in Table 5.

Table 5. *Interface of TLsensor.*

Function	URL	Method	Parameters	Output
Get sensor reading.	/reading	GET	-	{ “name” : String, “temperature” : Integer, “luminance” : Integer }
Set temperature.	/reading/temperature	POST	**query parameter:** value = [-50..50]	{ “name” : String, “temperature” : Integer, “luminance” : Integer }
Set luminance.	/reading/luminance	POST	**query parameter:** value = [0..10000]	{ “name” : String, “temperature” : Integer, “luminance” : Integer }
Get device settings.	/settings	GET	-	{ “name” : String, “type” : String }
Set operating mode.	/settings	POST	**query parameter:** mode = { Simulated | Real }	{ “name” : String, “type” : String }

The interface of TLsensor offers a function for requesting the reading of the sensor with HTTP GET request. Since TLsensor offers both real and simulated version of Lux 34 sensor, the interface includes also a function for changing the operating mode. The mode is changed with HTTP POST request by sending a new mode with the query parameter. There is also a function for reading the current operating mode. When the sensor is operated in simulated mode, its readings can be set manually. Therefore, the interface includes functions for setting the temperature and luminance with HTTP POST requests. New value is sent with the query parameter. Limits for the temperature and the luminance are based on the real sensor.

TLsensor uses JSON messages to send the data back to the requester. Both temperature and luminance values are included in the same message. Name field is used to give a unique name or ID to the sensor. An example message is given at the next page.

```
{
“name” : “TL10”,
“temperature” : 20,
“luminance” : 6000
}
```

In this case, the sensor is named TL10, the temperature is 20°C and the luminance is 6000lux. TLsensor uses also another data format describing the current operating mode. An example is given below.

```
{
“name” : “TL10”,
“type” : “Manual”
}
```

HTTP status codes are used to manage error situations. Error handling is similar as in previous implementations. If the user is using invalid parameters, HTTP response is returned with a status code 400 (Bad Request). In case of any internal problems, the status code will be 500 (Internal Server Error). Finally, if the consumer is not authorised to use TLsensor, HTTP response will contain a status code 403 (Forbidden).

Controller System is another new component at the implementation. As can be seen from Figure 18, Controller does not provide any services to Arrowhead Network. Instead, it only consumes services. Controller System uses Service Registry to find the endpoints of TLsensor, Engine Block Heater Controller and Light Controller.

Controller polls the sensor reading from TLsensor and uses the sensor data to operate SmartLumo through Light Controller and engine block heater through Engine Block Heater Controller. When new temperature is detected, Controller System simply sends it to the engine block heater through Engine Block Heater Controller. This causes the heater to change the heating time. Similarly, when Controller System detects new luminance, it sends a new output level to SmartLumo through the Light Controller. Output level is based on an algorithm, which scales the luminance linearly to the output level so that 10000lux luminance results 0% output level whereas 0lux results 100% output level. Additionally, Controller sets the LED of the Engine Block Heater when the luminance drops under a certain value. This helps finding the heater from the dark parking area.

4.3.2 Sensor Simulator

Urban Management Tool includes a graphical user interface called Sensor Simulator for simulated version of Lux 34 sensor. Sensor Simulator a web application using AJAX requests to access the services of TLsensor. Similar to user interfaces described at previous implementations, Sensor Simulator was running locally at the computer outside of Arrowhead Network. For this reason, CORS was needed at TLsensor. Sensor Simulator was used on a PC with Mozilla Firefox version 38.3.0. Sensor simulator is presented in Figure 19.

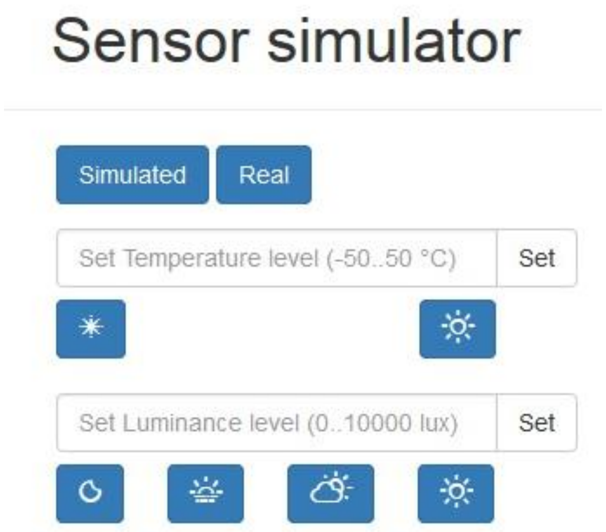

Sensor simulator
Simulated
Real
Set Temperature level (-50..50 °C)
Set
Set Luminance level (0..10000 lux)
Set

Figure 19. *User interface of Sensor Simulator.*

The top section includes two push buttons for changing the operating mode of the sensor. Active mode is highlighted when the connection to TLsensor is established. Under the mode buttons there are input fields for the temperature and the luminance. To be able to use the fields, TLsensor must be in simulated mode. Supported data ranges are displayed at the input fields. Under the input fields, there are several push buttons with predefined temperature and luminance levels. Temperature buttons set the temperature value to -20°C and 20°C whereas luminance buttons set the luminance value to 1000lux, 3000lux, 6000lux and 10000lux. Buttons are developed to simplify the demonstration sessions.

4.4 Discussion

Using the Arrowhead Framework during the development had a few challenges. In general, the Framework required lots of studying before it was possible to use it. The API used for accessing the Core Systems seemed quite large and had to be mastered before developing pilot applications. However, for a skilled developer this should not be too overwhelming. Arrowhead Cookbook provided important information about the framework, which made the learning faster. Therefore, Cookbook is essential document especially for a new developer.

Some problems were faced with a documentation model. Even though the framework includes templates for every document type, the documents are not unequivocal. This can cause too much variety to documents and thus risk the reusability of services. Next, maintaining and validating the documents needed much effort due the referencing system. If document were changed, references had to be changed to multiple documents. Moreover, it is still unclear how documents will be stored. Electrical documentation system would support Arrowhead better. The documents would follow more unified approach if the documenting were done with documentation software. Moreover, it would be easier to maintain references with the software.

Core Systems were included to every pilot application although Orchestration System was rarely used due the small size of the applications. Moreover, Orchestration System had some errors since it seemed that it was not possible to change the orchestration rules with Arrowhead Management Tool. Service Registry and Authorisation System worked as expected. RESTful seemed an appropriate technology for developing the services, since it was easy to document the interfaces and hierarchical structure was useful especially at Engine Block Heater Controller. The downside of RESTful was the lack of publish-subscribe communication protocol. This forced to use polling which caused continual requests to the services. This increased the data traffic and can cause performance problems in real commercial applications. It would be more effective to notify the consumer with events especially when the multiple consumers are accessing the same service.

The user interfaces of the pilot applications are rather exceptional components since they were running locally on the computer outside from Arrowhead Network. Normally, a device should be connected to the Arrowhead Network to consume the services within the network. Now this was bypassed with CORS technology making it possible to request the services from other domains. Furthermore, it was not possible to authorise user interfaces with Authorisation System. This approach cannot be recommended for real commercial applications due information security reasons. However, Arrowhead does not give any guidelines for user interfaces yet.

The pilot applications were successful and they showed that Arrowhead have potential in commercial Smart City applications as well. Light Management Tool could be used to monitor street lights from the different areas of the city. Orchestration System would make it easy to change the area to be monitored. Engine Block Heater Controller was developed with only one engine block heater but a commercial implementation would include dozens of heaters at multiple parking areas. The application could be used to monitor every engine block heater at a certain region. Moreover, if Orchestration System was included in the implementation, an operator could swap between the different areas of the city by using orchestration rules.

When Light Management Tool and Engine Block Heater Controller were integrated together, the devices were controlled with a separate Controller System. Commercial version of Urban Management Tool could include even more services with centralized control. The application provided an example about System-of-Systems where requested functionality was achieved with multiple systems using the services in collaborative manner. Arrowhead approach made the integration relatively easy. However, every software component was created by the same developer so there was no need to study the documentation of the systems beforehand. Moreover, same technologies were used in every component. It is unknown whether it would be more difficult to use the services made by other project partners with possibly different technologies.

5. CONCLUSION

This master’s thesis presented the recent research topics of Smart City concept. Smart City aims to improve the use of public resources, increase the quality of services offered to the citizens and reduce both the energy consumption and operational costs of the public administrations. ICT is considered as a key enabler of Smart City concept since most of the goals are reached with Smart City applications. There are several research papers proposing technologies and architectural structures to enable Smart City. There are also projects running at European Union to build technical frameworks for Smart City concept such as SmartSantander and RERUM.

This thesis introduced also Arrowhead Project, which aims to develop an Arrowhead Framework that enables collaborative automation by networked embedded devices. The project is running until February 2017. Networked devices form a network called Arrowhead Network. The architecture of the network is based on SOA. Thus, the interaction of devices within the network is based on services and application functionality can be built by using the services in collaborative manner. Arrowhead Framework includes a set of Core Systems providing the basic functionality such as information security to every device at the network. Moreover, they cover the principles of SOA such as discoverability and loose coupling. Currently, a prototype is available from three Core Systems including Authorisation System, Orchestration System and Service Registry. They are hosted at Arrowhead Network and their services can be used in applications through APIs.

Arrowhead Project is targeting several domains including Production, Smart Buildings and Infrastructures, Electro-Mobility, Energy Production and End-User Services, and finally Virtual Market of Energy. Project partners are testing Arrowhead Framework during the project by developing pilot applications which demonstrate the use of Arrowhead Framework in various use-cases. First, two pilot applications were developed during the Pilot Generation 2 of the project at FAST-lab at Tampere University of Technology. The applications are Light Management Tool used to manage and monitor the street lights, and Engine Block Heater Controller used to manage and monitor engine block heaters. Next, a third application called Urban Management System was developed to control urban systems with a centralized control system. It was created by integrating the first two pilot applications together. Street lights and engine block heaters were controlled with separate Controller System. The application changed the dimming of street lights based on the luminance level and the heating time of engine block heaters based on the temperature.

The pilot applications demonstrated how Arrowhead Framework could be used to develop Smart City applications. The applications worked well in demonstrations and they showed potential to commercial applications as well. Furthermore, Urban Management System showed that it was relatively easy to reuse components from existing systems to create an application where requested functionality is achieved with collaborative automation. On the other hand, the integration was easy, because every application was developed by the same developer and with the same technologies. Some challenges were faced with document model due its complexity. The framework required also lots of studying before it was possible to use it. Nevertheless, SOA-based Arrowhead approach seems suitable technology for building Smart City applications.

5.1 Future work

The work on Arrowhead Project continues at Pilot Generation 3, which is the last phase of the project. The project will run until February 2017. Arrowhead Framework seems a promising concept but it still requires lots of work to become a commercial platform. There are still unfinished things in Arrowhead Framework so it is expected that the framework will evolve during the rest of the project. For example, most of the Core Systems are still under development. New prototypes of Core Systems should be developed soon in order to test them in pilot applications. Work should be also focused on security mechanisms since Arrowhead does not have a suitable certificate distribution system yet. Finally, the document model needs more work. It should be considered whether the document model can be supported with software-based solution. Current document model is problematic because it allows too much variation to the documents and maintaining the documents requires lots of work.

Pilot applications introduced in this thesis can be developed more. Work should be focused especially to the hardware. Currently, Light Management Tool includes only one LED luminaire. The next step of the development should be to add more luminaires. The lack of hardware is a bigger problem at Engine Block Heater Controller. All the functions of the interface cannot be tested properly with only one engine block heater. For example, it is impossible to get the combined data from the zone level because there is only one heater at the zone. Therefore, the next step of the development should be to add more engine block heaters. Furthermore, the user interface needs to be upgraded as well since it is currently designed to monitor only one power socket. It should also be considered whether the pilot applications could be tested in more commercial environment. For example, it would be interesting to control an outdoor light with Light Management Tool instead of a small LED luminaire. Furthermore, Orchestration System should be included in both applications. Finally, Urban Management System could also be developed more. New services could be developed to expand the functionality of the application.

REFERENCES

[1] M. Albano, L. Ferreira, Communication Profile (CP) Template version 1.1, unpublished project material, 30th September 2015, 4 p.

[2] M. Albano, L. Ferreira, Interface Design Description (IDD) Template version 1.1, unpublished project material, 30th September 2015, 5 p.

[3] M. Albano, C. Chrysoulas, L. Ferreira, O. Jansson, Semantic Profile (SP) Template version 1.1, unpublished project material, 30th September 2015, 4 p.

[4] M. Albano, C. Chrysoulas, L. Ferreira, O. Jansson, Service Description (SD) Template version 1.1, unpublished project material, 30th September 2015, 5 p.

[5] M. Albano, C. Chrysoulas, L. Ferreira, O. Jansson, System-of-Systems Description (SoSD) Template version 1.1, unpublished project material, 30th September 2015, 7 p.

[6] M. Albano, C. Chrysoulas, L. Ferreira, O. Jansson, I. Soria, System-of-Systems Design Description (SoSDD) Template version 1.1, unpublished project material, 30th September 2015, 7 p.

[7] M. Albano, C. Chrysoulas, L. Ferreira, O. Jansson, System Description (SysD) Template – Black Box Design version 1.1, unpublished project material, 30th September 2015, 7 p.

[8] M. Albano, C. Chrysoulas, L. Ferreira, O. Jansson, I. Soria, System Design Description (SysDD) Template – White Box Design version 1.1, unpublished project material, 30th September 2015, 5 p.

[9] L. Anthopoulos, P. Fitsilis, Exploring Architectural and Organizational Features in Smart Cities, In 16th International Conference on Advanced Communication Technology (ICACT), IEEE, 2014. pp. 190–195.

[10] Arrowhead, web page. Available (accessed on 9th October 2015): http://www.arrowhead.eu/

[11] Artemis projects: Arrowhead. Available (accessed on 9th October 2015): https://artemis-ia.eu/project/49-arrowhead.html

[12] Y. Baghdadi, A Framework to Select an Approach for Web Services and SOA Development, In International Conference on Innovations in Information Technology (IIT), IEEE, 2012, pp. 277–282.

[13] J. Bean, SOA and Web Services Interface Design – Principles, Techniques, Standards, The MK/OMG Press. Boston, 2010, 372 p. Available (accessed on 13th January 2016): http://www.sciencedirect.com/science/article/pii/B9780123748911000010

[14] F. Blomstedt, C. Chrysoulas, G. Singler, P. Varga, Arrowhead Framework Cookbook version 1.5, unpublished project material, 2015, 32 p.

[15] F. Blomstedt, O.Carlsson, M. Johansson, P. Varga, Arrowhead generic SoSD – Generation 2 version 1.5, unpublished project material, 4th April 2015, 36 p.

[16] F. Blomstedt, L. Ferreira, M. Klisics, C. Chrysoulas, I. Martinez de Soria, B. Morin, A. Zabasta, J. Eliasson, M. Johansson, P. Varga, The Arrowhead Approach for SOA Application Development and Documentation, In Industrial Electronics Society (IECON), IEEE, 2014, pp. 2631–2637.

[17] Bootstrap framework, web page, Available (accessed on 3rd September 2015): http://getbootstrap.com/

[18] C2 SmartLumo Datasheet, Available (accessed on 1st October 2015): http://c2is.fi/c2-smartlumo/

[19] H. Chourabi, N. Taewoo, S. Walker, J.R. Gil-Garcia, S. Mellouli, K. Nahon, T.A. Pardo, H.J. Scholl, Understanding Smart Cities: An Integrative Framework, In 45th Hawaii International Conference on System Science (HICSS), IEEE, 2012, pp. 2289–2297.

[20] A.W. Colombo, F. Jammes, H. Smit, R. Harrison, J.L.M. Lastra, I.M. Delamer, Service-Oriented Architectures for Collaborative Automation, In 31st Annual Conference of Industrial Electronics Society (IECON), IEEE, 2005, pp. 2649–2654.

[21] Cross-Origin Resource Sharing, W3C Recommendation, 16th January 2014, Available (accessed on 29th October 2015): http://www.w3.org/TR/cors/

[22] L. Ferreira, Arrowhead-related Definitions version 2.3, unpublished project material, 10th April 2015, 11 p.

[23] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, N. Pichler-Milanovic, E. Meijers, Smart Cities: Ranking of European medium-sized cities, 2007, 28 p. Available (accessed on 11th January 2016): http://www.smart-cities.eu/download/smart_cities_final_report.pdf

[24] Hidealite 1202 Multi, Data sheet, 1 p. Available (accessed on 1st October 2015): http://www.hidealite.fi/Archive/FilesArchive/354730_555.pdf

[25] Hidealite Jolly Pro, Data sheet, February 2014, 6 p. Available (accessed on 1st October 2015): http://www.hidealite.se/Archive/FilesArchive/IM00031_Jolly_Pro_201402_sv_en_fi.pdf

[26] Z. Huanhuan, G. Zhen, Z. Xibo, Research on SOA-Based Integration Solution to Bank Online Application System, In International Conference on Computer Science and Network Technology (ICCSNT), IEEE, 2011, pp. 2235–2240.

[27] IBM, Web services: Key roles, web page, Available (accessed on 23rd October 2015): http://www-01.ibm.com/support/knowledgecenter/SSB23S_1.1.0.7/com.ibm.ztpf-ztpfdf.doc_put.07/gtps6/s6wsrol.html

[28] The jQuery Foundation, jQuery, web page, Available (accessed on 3rd September 2015): https://jquery.com/

[29] Z-F. Liu, B. Liu, X-P Gao, SOA Based Mobile Application Software Test Framework, In 8th International Conference on Reliability, Maintainability and Safety (ICRMS), IEEE, 2009 pp. 765–769.

[30] A. Monzon, Smart Cities Concept and Challenges: Bases for the Assessment of Smart City Projects, In International Conference on Smart Cities and Green ICT Systems (SMARTGREENS), IEEE, 2015 pp. 1–11.

[31] P. Offermann, M. Hoffmann, U. Bub, Benefits of SOA: Evaluation of an Implemented Scenario against Alternative Architectures, In 13th Enterprise Distributed Object Computing Conference Workshops (EDOCW), IEEE, 2009, pp. 352–359.

[32] Oracle, The Java EE 6 Tutorial, web page, Available (accessed on 2nd December 2015): https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html

[33] Produal Lux 34 light level and temperature transmitter, data sheet, 1 p. Available (accessed on 5th October 2015): http://www.admiclim.com/PDF/LUX34-100.pdf

[34] RERUM, web page, Available (accessed on 10th January 2016): https://ict-rerum.eu/

[35] RFC 4158, Internet X.509 Public Key Infrastructure: Certification Path Building, The Internet Engineering Task Force, September 2005, 81 p. Available (accessed on 2nd December 2015): https://tools.ietf.org/html/rfc4158

[36] RFC 5246, The Transport Layer Security (TLS) Protocol Version 1.2, The Internet Engineering Task Force, August 2008, 104 p. Available (accessed on 2nd December 2015): https://tools.ietf.org/html/rfc5246

[37] RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, The Internet Engineering Task Force, May 2008, 151 p. Available (accessed on 3rd December 2015): https://tools.ietf.org/html/rfc5280

[38] S. Simanta, E. Morris, S. Balasubramaniam, J. Davenport, D.B. Smith, Information Assurance Challenges and Strategies for Securing SOA Environments and Web Services, In 3rd Annual Systems Conference, IEEE, 2009, pp. 173–178.

[39] SmartSantander, web page, Available (accessed on 11th January 2016): http://www.smartsantander.eu/

[40] O. Smith. Morris.js, web page, Available (accessed on 3rd September 2015): https://morrisjs.github.io/morris.js/index.html

[41] K. Su, J. Li, H. Fu, Smart City and the Applications, In International Conference on Electronics, Communications and Control (ICECC), IEEE, 2011, pp. 1028–1031.

[42] R. Szabo, K. Farkas, M. Ispany, A.A. Benczur, N. Batfai, P. Jeszenszky, S. Laki, A. Vagner, L. Kollar, C. Sidlo, R. Besenczi, M. Smajda, G. Köver, T. Szincsak, T. Kadek, M. Kosa, A. Adamko, I. Lendak, B. Wiandt, T. Tomas, A.Z. Nagy, G. Feher, Framework for Smart City Applications Based on Participatory Sensing, In 4th International Conference on Cognitive Infocommunications (CogInfoCom), IEEE, 2013 pp. 295–300.

[43] K. Takahashi, S. Yamamoto, A. Okushi, S. Matsumoto, M. Nakamura, Design and Implementation of Service API for Large-Scale House Log in Smart City Cloud, In 4th International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2012, pp. 815–820.

[44] E. Theodoridis, G. Mylonas, I. Chatzigiannakis, Developing an IoT Smart City Framework, In 4th International Conference on Information, Intelligence, Systems and Applications (IISA), IEEE, 2013, pp. 1–6.

[45] E.Z. Tragos, V. Angelakis, A. Fragkiadakis, D. Gundlegard, C.-S. Nechifor, G. Oikonomou, H.C. Pohls, A. Gavras, Enabling Reliable and Secure IoT-Based Smart City Applications, In International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), IEEE, 2014, pp. 111–116.

[46] J. Van den Bergh, S. Viaene, Key Challenges for the Smart City: Turning Ambition into Reality, In 48th Hawaii International Conference on System Sciences (HICSS), IEEE, 2015, pp. 2385–2394.

[47] P. Wang, A. Ali, W. Kelly, Data Security and Threat Modeling for Smart City Infrastructure, In International Conference on Cyber Security of Smart Cities, Industrial Control System and Communications (SSIC), IEEE, 2015, pp. 1–6.

[48] L. White, N. Wilde, T. Reichherzer, E. El-Sheikh, G. Goehring, A. Baskin, B. Hartmann, M. Manea, Understanding Interoperable Systems: Challenges for the Maintenance of SOA Applications, In 45th Hawaii International Conference on System Science (HICSS), IEEE, 2012, pp. 2199–2206.

[49] S. Yamamoto, S. Matsumoto, S. Saiki, M. Nakamura, Materialized View as a Service for Large-Scale House Log in Smart City, In 5th International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2013, pp. 311–316.

[50] A. Zanella, N. Bui, A. Castellani, L. Vangelista, M. Zorzi, Internet of Things for Smart Cities, Internet of Things Journal Vol. 1, No. 1, IEEE, 2014, pp. 22–32.

[51] W. Ze-Lai, R. Guo-Zheng, F. Zhi-yong, W.Yao, Z. Chen, Public Emergency Oriented SOA-Based Logistics System, In International Conference on Computer Application and System Modeling (ICCASM), IEEE, 2010, pp. V1–588 – V1–591.