TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MUHAMMAD UMAIR HASSAN

MANAGEMENT OF IOT SYSTEMS FOR URBAN SERVICES

Master of Science Thesis

Examiner: Professor José L. Martinez Lastra
Examiner and topic approved on
07.09.2016

ABSTRACT

Muhammad Umair Hassan: Management of IoT Systems for Urban Services.

Tampere University of Technology
Master of Science Thesis, 55 pages
January 2018
Master's Degree Programme in Automation Technology
Major: Factory Automation and Industrial Informatics
Examiners: Professor José L. Martinez Lastra

Keywords: Arrowhead, Urban City, Internet of Things, Information and Communication Technology, Service-Oriented Architecture

Urban Cities are growing exponentially and with them the need for efficient management of city resources. SOA (Service Oriented Architecture) and IoT (Internet of Things) provides promising solutions for this issue. SOA brings interoperability, reusability and dynamic discovery. Whereas, IoT provide next evolution on Internet by allowing physical real-world objects to connect to the internet through unique identifiers.

Arrowhead Project has developed a framework to provide collaborative automation by enabling embedded devices to interact with each other. This project uses SOA to target five different domains. These domains include Smart Building and Infrastructure, Energy Production, Virtual Market of Energy, Production, and Electro-Mobility.

In the thesis, three applications were developed. Light Simulator was developed to demonstrate the working principle of the IoT enabled Light devices. Meter application was developed to monitor and manage IoT enabled systems. Finally, Energy Consumption service was developed as a pilot service for Arrowhead Project. This service was deployed on Arrowhead and it calculates the energy consumed by different real-time objects. Furthermore, since Arrowhead framework follows all principles of SOA, reusability of above services can provide composability to develop further applications.

PREFACE

This research was very interesting to me irrespective of the struggle that I had to face in the entire process. I learned a great deal from this master thesis. This thesis was done at Factory Automation Systems and Technologies laboratory at Tampere University of Technology. I want to thank Professor José L. Martinez Lastra for providing me with this wonderful opportunity and Senior Research Fellow Jani Jokinen for helping me in implementation phase. Furthermore, I want to thank Muhammad Ali, Arsalan and Ahsan for guiding me in successfully finalizing my research. Lastly but not least I want to acknowledge my family in motivating me and my wife, Staish, in supporting me at every step.

Tampere,13th January 2018

Muhammad Umair Hassan

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CC	Cloud Computing
CSA	Cloud Security Alliance
EDA	Event Driven Architecture
ESB	Enterprise Service Bus
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technology
IoT	Internet of Things
JSON	JavaScript Object Notation
LoRaWAN	Long Range Wide Area Networks
MIT	Massachusetts Institute of Technology
NFC	Near Field Communication
OECD	Organization for Economic Cooperation and Development
QoS	Quality of Service
REST	Representational State Transfer
RFID	Radio Frequency Identification
RPC	Remote Procedure Call
SBC	Single Board Computers
SC	Service Consumer
SD	Service Directory
SIoT	Social Internet of Things
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SP	Service Provider
URI	Unified Resource Identifier
WSDL	Web Services Description Language.
XML	Extensible Markup Language

1. INTRODUCTION

The evolution of automation and the industrial internet has brought the possibility of developing Smart Urban Cities. According to [10], the estimate is an increase in urban cities population over 2.9 billion in 2050. This rapid increase in urbanization demands efficient utilization of limited resources and better energy consumption. Smart cities are the key to satisfying the increasing demands of the urban cities.

Smart Cities is defined by Ferreira and Afonso [10], as a city which can implement clever solutions for modern cities by bringing both quantitative and qualitative improvements in productivity. There are three architecture styles that are mentioned to implement Smart Urban Cities by Anthopoulos and Fitsilis [12]. These architectures are named as Multi-tier architecture, Event Driven Architecture, and Service Oriented Architecture. Service-Oriented Architecture is most commonly seen approach when the aim is to have the services that form the infrastructure for the implementation of Urban services. SOA, makes use of composition and reusability, by combining different existing services. This allows the SOA-based systems to adapt new environments and provide solutions with agility. Furthermore, services that are developed using this architecture also contains autonomy, that enables services the freedom from technology restrictions and eliminates compatibility issues.

Arrowhead Project follows collaborative automation approach and works on the same principle as Service Oriented Architecture. Collaborative automation in an open-network environment connecting many embedded devices with the use of SOA [40]. This approach is an effective approach in providing solutions for increasing energy challenges [40]. Due to the SOA principles, the arrowhead framework provides interoperability and reusability. Interoperability allows the services to be developed on a different technological platform and still be compatible with each other. Similarly, reusability allows them to be reused hence, reducing development costs. Arrowhead framework is expected to have a significant reduction of 75% in designing networked devices [40]. There are five areas in which Arrowhead project is trying to achieve desired results. These fields include Smart Buildings and Infrastructures, Production, Electro-Mobility, Energy Production, and finally Virtual Market of Energy [40].

The Internet of Things (IoT) envisions the future aspect of devices where the physical objects of daily life can communicate with each other over the internet by using micro-controllers and transceivers. IoT systems will hence, make the internet more integrated into everyday life [39]. This development concept has many applications in smart cities

that include home automation, industrial automation, medical aids, mobile healthcare, elderly assistance, intelligent energy management and smart grids, automotive, traffic management, and many others [39].

1.1 Problem Statement

Urbanization is constantly and exponentially increasing even where overall population growth is minimum [10]. Increase in population and urbanization brings in the difficulties in managing city resources since cities are being forced to overstretch their infrastructure and resource capacities. Implementing smart cities and encapsulating IoT with the help of Information and Communication Technologies (ICT) is the key to successfully reduce energy consumption and monitor other city resources.

There are many proposed architectures that are suitable for building smart urban cities. However, there arise a problem to monitor IoT services in an urban city due to large-scale applications. Furthermore, it is very complicated to manage the services deployed in urban cities and process the data which is generated by such services. Therefore, it is very important to develop an IoT management system that is capable of monitoring and controlling services as well as storing and processing data.

This thesis aims at developing such system by utilizing Arrowhead framework. This framework can be used to provide an effective system that can demonstrate the monitoring process for different IoT based systems and energy consumption analysis in a city.

1.2 Work Description

This thesis describes the Arrowhead Framework and how different applications can be developed on this platform. Moreover, the theoretical background presents different urban cities architectures for developing an effective service management system. The state of the art focuses on Service Oriented Architecture and Arrowhead framework.

The main task of the thesis was to demonstrate how an IoT management system can be developed on Arrowhead Framework for monitoring and managing services. A prototype case was developed to demonstrate light system control and monitoring. Furthermore, Energy Consumption pilot application was developed at FAST-lab located in Tampere University of Technology for Arrowhead Project to demonstrate working example of an energy consumption evaluation system for multiple physical objects in the urban city.

1.3 Thesis Outline

There are six chapters in this thesis. The first chapter provides the introduction. The introduction also contains problem statement and the work description of the thesis. In the

second chapter importance of incorporating smart applications in urban cities is discussed. Furthermore, different architectures and frameworks are discussed. The chapter further explains the state of the art technologies for developing a system which relates to the solution of problem statement. It also explains IoT systems and presents some case examples. A detailed description of Arrowhead framework is also presented in this chapter. The third chapter explains the methods to approach this framework and lastly some tools which were used in this thesis implementation. The fourth chapter discusses the implementation which includes prototype implementation and deployment. A case example of energy consumption monitoring system was developed and is explained in this chapter. In the prototype light monitoring and controlling, simulation application is presented which was developed on the local platform. In the deployment section, the abovementioned application and two real-time monitoring systems were developed for coffee machine and car heating. The fifth chapter contains the results of the implementation. Finally, the last chapter provides conclusion and future work.

2. BACKGROUND AND TECHNOLOGICAL SELECTION

This chapter discusses the state of the art. In the start of the chapter the meaning of smart cities is explained followed by the discussion on the importance and motives of having smart cities and smart applications in an urban city for managing city resources. This chapter gives a detailed description of IoT with some case IoT systems already developed for urban services. Furthermore, different architecture styles are discussed as proposed by different authors to implement these cities. In section 2.3.1, background study is carried out on why Service Oriented Architecture is the most important architecture for future implementation. The next section explains Web Services and its role in SOA. Section 2.3.3 then focuses on Arrowhead framework, which is the main developing platform used in the thesis.

2.1 Smart Services for Urban Cities.

Urban cities are the cities that facilitates through infrastructure a very high-density population of human beings. According to [10], it is estimated that until 2050 there will be an increase in population within urban cities by 2.9 billion. Cities are being forced to overstretch their infrastructure and resource capacities due to increase in population growth and urbanization. Urbanization is constantly and exponentially increasing even where overall population growth is minimal. According to Bélissent [19] growth in population is expected to be 3% in developed countries whereas urban population growth is expected to increase 18% between the year 2010 and 2050. This rapid urbanization indicates the emerging problem of providing more services. One proposed solution is to undertake smart city solution for providing services. Increasing services demands can be fulfilled only by using energy efficiently. Information and Communication Technology (ICT), when encapsulated with smart cities, can undertake this task. The above-mentioned increase in population growth means that the competition for limited resources will also increase which will give rise to the energy consumption. As Bélissent [19], stated in her research that according to The Organization for Economic cooperation estimates, non-OECD countries will consume 84% more energy compared to 14% increase in 33 OECD countries. Smartness of the city is no longer a superfluity. It is becoming a necessity for urban areas to adopt efficient infrastructure with smart communication techniques.

Smartness of a city is a highly ambiguous term and different authors have defined it in distinct perspective in relation to a smart city. According to [17], smart city provides a new generation of city which has ICT technologies embedded in it to facilitate sustainability in city resources. Ferreira and Afonso [10] defines the smart city as a city which can implement clever solutions for modern cities. Hence, bringing both quantitative and

qualitative improvements in productivity. Giffinger et al [16] provides a different approach by defining six different fields in which smart city must perform efficiently. These fields are categorized as Smart Economy, Smart People, Smart Environment, Smart Mobility, Smart Living and Smart Governance. Furthermore, they also debated in [16] that a smart city shall be able to provide means in above six fields to solve problems in an urban city hence, providing better life qualities. Another interesting opinion is by Nam and Pardo [13] defining the term smart in the smart city as more user-friendly than intelligent, the smart city needs to adapt itself to user needs. In all the above definitions, the authors have agreed that smart cities can be designed by incorporating smartness in the applications which are achievable through solutions provided by ICT.

Initiatives are taken by innovative governments to incorporate smart applications into cities services so that the increasing demands of urban cities can be fulfilled. Following some motives for developing smart applications to manage resources in a city are mentioned.

- Pressure on global Resources as the population rises: There are limited resources in the world and the increase in the overall population is a challenge, leading to scarcity. This pressure can be seen in the rise of demand for water and electricity. According to Foster [19], 20 countries faced water shortage in 1990 whereas in 1955 there were only seven. In their report, they predicted that this shortage can rise up to 18% of total world population. As indicated above the rise in energy demand is also following a steep curve. These situations demand efficient means of distributing resources which can only be possible by implementing smartness into urban cities.

- Formation of megacities due to rapid urban concentration: Megacities are the urban cities with more than million inhabitants. There were only two megacities in 1950, New York and Tokyo. However, today there are more than 11 megacities in Asia, 2 in Europe, 2 in North America, 4 in Latin America and 2 in Africa [19]. More of these cities are emerging and will be more by 2050. It can be very challenging to take care of all necessary infrastructure such as public safety, traffic, education, housing and healthcare without the help of technology.

- Variation in the quality of services due to uneven distribution: With the emergence of megacities, variation in services are also becoming an issue. Furthermore, the percentage of the older population is also on the rise which will require more healthcare facilities in certain parts of the city. Cities with aging markets need means to attract young citizens in order to cope with the competition of growing commerce and labor market.

Smart IoT based services seems like a promising solution in proper management of city resources. This approach can optimize utilization of finite resources in the field of transportation, utilities, healthcare, education, public safety, building management and city management [19].

- Transportation: Transportation can be optimized by real-time monitoring of the flow of traffic. This will provide the needed data to analyze the need for new lanes and avoiding possible traffic congestion. One example of smart transportation technique is mobile payment for parking applied in San Juan Province of Northwest Argentina.
- Utilities: Use of smart energy grid is being initialized to reduce energy waste by providing only the amount of energy needed by the user. Furthermore, the user is also notified about the amount of energy they are consuming. Innovative designs are made to power data centers to establish a use-based energy consumption network through solar energy.
- Healthcare: Records of the patients can be recorded electronically which will facilitate in sharing information between different clinics and pharmacies. Due to such an implementation, patients can get faster excess to the medicines and required services. One such initiative is eHealth taken by Egypt and Saudi Arabia to meet the healthcare demands at the national level.
- Education: Introduction of information technology into education sector can improve the overall quality of the education in higher institutes. The access to the study content is increased via the online provision of courses. For example, China has launched Blue Sky eLearning that focuses on providing a remote excess of educational facilities to rural areas.
- Public Safety: Public safety is of high priority to government officials. It is politically beneficial for them to initiate public safety projects. Introduction of IT services in such projects will enhance public safety by optimizing emergency services response time. Furthermore, it helps in organizing secure mass events by providing secure administration and surveillance of public places.
- Building management: Urbanization brings in the need of developing a management system for buildings. Building sector that includes both commercial and residential buildings consumes about one-fifth of energy consumption in the world. [14]. This energy is needed for all the facilities provided; such as the power, water, elevators, lights, heating, and cooling. IT services help in reducing the energy consumption by providing optimized ventilation and room automation systems.
- City management: Integrating IT systems in city management will provide with the necessary data that can be analyzed and used to resolve budget issues between different sectors. It will also provide surveillance reports which can improve overall management of city facilities such as security and transportation.

2.2 Internet of Things (IoT)

The Internet has been used in past 40 years to provide facilities to people in fields such as shopping, information recovery, and easy banking etc. Furthermore, it has brought them together by providing better communications e.g. forums, social media, and emails. Earlier, the internet was considered as many individual computers connected to form a network. However, these days internet can be found in all smart devices. Next evolution of the internet is related to the concept of connecting all the devices together for bringing unlimited opportunities. This evolutionary phase is being referred as the Internet of Things (IoT).

The IoT has the potential to impact people way of living in ways greater than internet alone has done. The Internet has a significant impact on publishing, communication and entertainment sector until now, but industries in other sectors have not been much affected. The IoT, on the other hand, will embed information and communication technologies within machines to optimize automation process and bring new opportunities. [31]

IoT however, is a very vast term and its understanding varies on how it is being observed. Gartner, the technology analyst company, defines it as "*The network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment [32]"*. Similarly, Höller et al. define the IoT as follows*: "The IoT is a widely used term for a set of technologies, systems, and design principles associated with the emerging wave of Internet-connected things that are based on the physical environment [37]".* The key factor to understand from these definitions is that IoT is not a single technology but it can be understood as many technologies working together in a synchronous way to achieve greater objective [31].

Executive director and co-founder of Auto-ID center, Kevin Ashton, at Massachusetts Institute of Technology (MIT) first introduced the term IoT in 1999 [37]. However, it was used in relation to Radio Frequency Identification (RFID) at that time [37]. Whereas, now the term as defined in above definitions, illustrates that all the devices are connected to the internet.

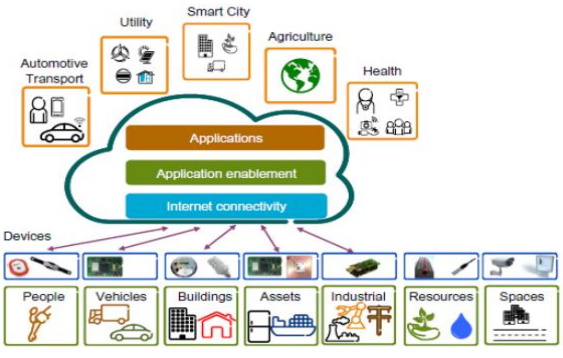

Figure 1. *Internet of Things [37].*

The broader concept of IoT is shown in Figure 1, which explains the concept of many different smaller services interconnecting to compose IoT solution such as smart cities. Furthermore, these services are connected to the cloud to enable resource and data sharing. IoT network consists of things that have unique identifiers. In general, the future of IoT universe will consist of IoT devices, cloud services, and software services that are owned, administered and operated by independent providers [33]. However, it is very important for the future IoT applications to manage the discovery, integration, and interoperability of things, cloud services, and third-party applications [33]. There are many proposed architectures to overcome this problem such as service-based IoT Architecture proposed by Dimitrios et.al. According to [33], every IoT component is a service which is integrated in a manner that provides dynamic discovery and composition. IoT Ecosystem is represented in conceptual form as shown in Figure 2.

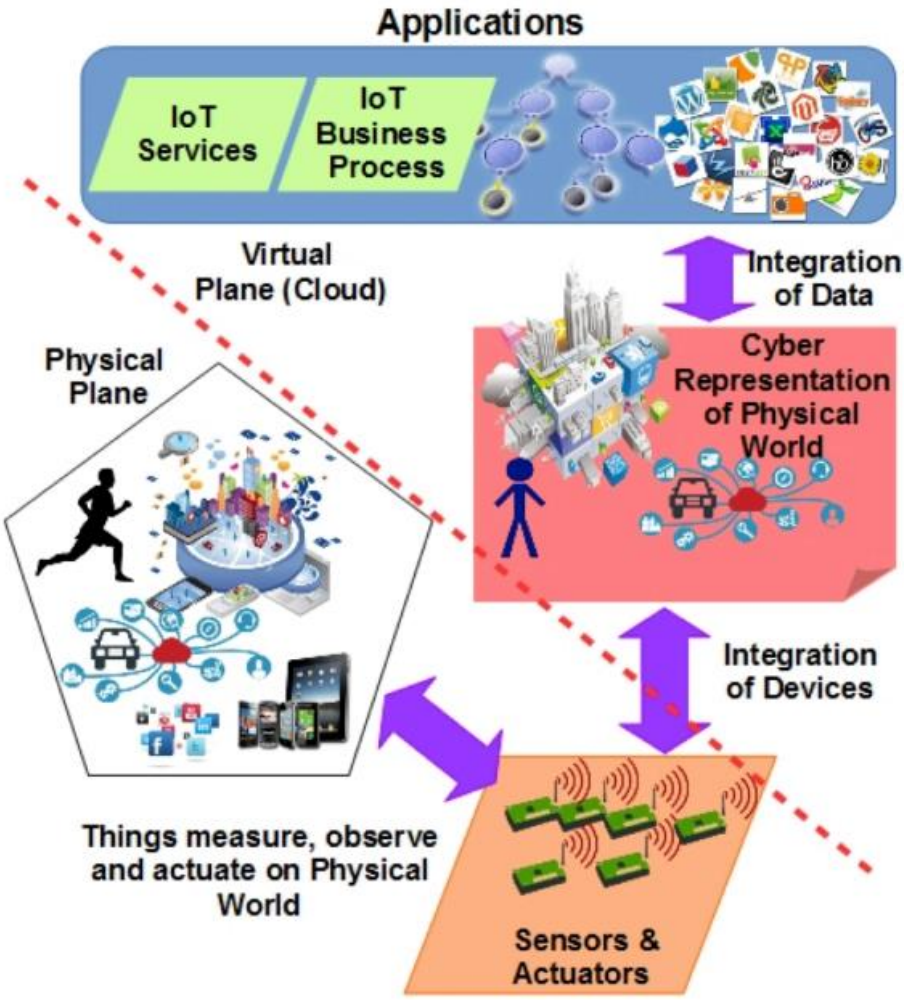

Figure 2. *IOT Ecosystem [33].*

The ecosystem can be divided into two planes, first is known as physical plane whereas the other as a virtual plane. The physical plane consists of all the objects connected to the IoT system such as humans, cars, devices, and infrastructure. Any change in the status of

the object is monitored and updated by the sensors connected to these objects. Physical world objects are represented in cyber form in the virtual plane. In most of the cases, this plane exists on the infrastructure hosted through cloud computing infrastructure. Furthermore, a virtual layer can be divided further into the application layer and cyber representation layer as shown in Figure 2. The virtual entities are modeled in cyber representation layer. The statuses of the cyber entities are reformed based on the sensor values in the actual world. Whereas, application layer consists of IoT services which are responsible for monitoring, controlling, and responding to the state changes.

Dimitrios et.al in [33], presented an example of monitoring and controlling indoor temperature. In order to support interactions between IoT devices in IoT ecosystem, there must be an association between the virtual layer and physical layer. This association must contain the data from the sensors which will then be used by the application layer to take necessary actions.

The new trend seen in the evolution of IoT systems is an integration of IoT into the industry. In particular, EU is leading a research, known as Industry 4.0, which aims at connecting machines and creating an intelligent value chain that can cooperate with each other to achieve the desired outcome autonomously. However, each industry component cannot easily be represented as a service and furthermore, implementing interoperability and integration among services in IoT at such level can be problematic. [33]

The above-mentioned integration problem is addressed by Dimitrios et.al in [33] through representing an abstract model for IoT.

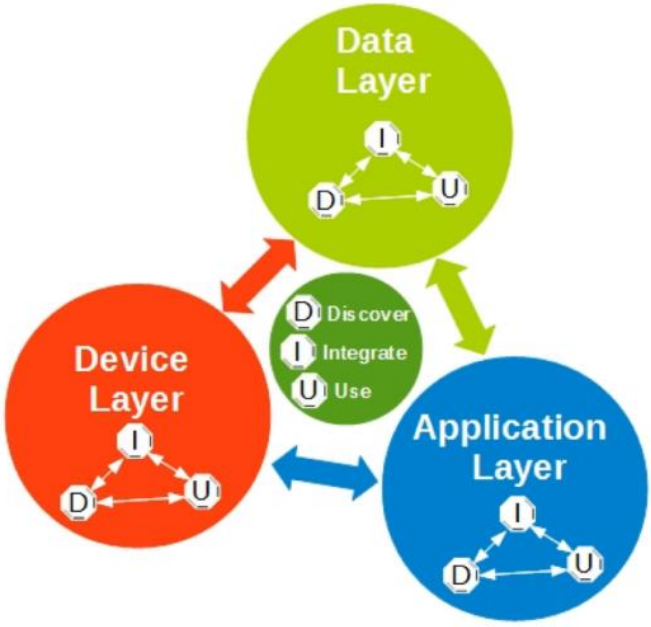

Figure 3. *IoT Abstract Model [33].*

Abstract model as shown in Figure 3 contain device layer, data layer, and application layer. These layers are explained in [33] and are discussed individually as follows.

- Device Layer: This layer is responsible for mapping physical world. It contains all the real hardware devices and focuses mainly on the discovery and integration of sensors attached to the devices.
- Data layer: Data layer is responsible for handling data that is received from the sensors. It further provides the data to the applications on their request. This layer focuses on storage and processing of data. Furthermore, this layer also checks the compatibility of received data and the requested data.
- Application layer: Incoming data sources are analyzed in this layer to produce information which is then used by the applications. Application layer focuses on the services which are being offered.
- Discover-Integrate-Use: Some functions are common to all the layers which provide the ability to discover, integrate, and use the services offered across the layers [33].

Another Architecture that is closely related to the work done in this thesis is Service-Oriented Architecture discussed in [34]. Section 2.3.1 provide with the general description of the tools and problems associated with this architecture. Following the steps involved in implementing SOA architecture in accordance with IoT is discussed. When implementing SOA in IoT, four layers are to be included as shown in Figure 4.

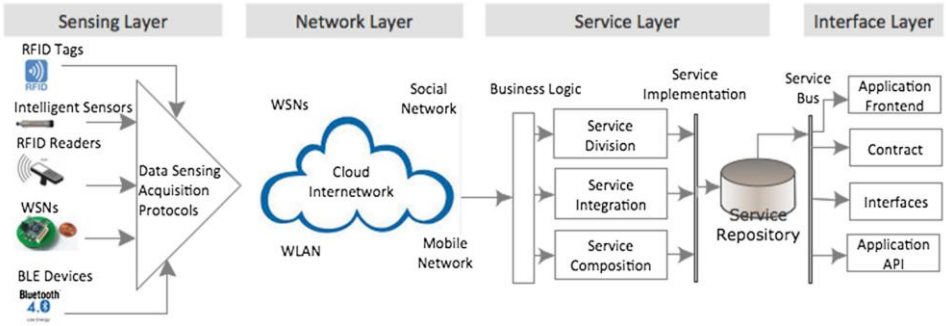

Figure 4. *SOA for IoT [34].*

1. Sensing layer: This layer is integrated by using all the available hardware objects. In other words, it consists of the physical objects which sense the changes in the physical world. When creating this layer, it is important to analyze the cost, size and communication method of the objects which are to be included in the layer. [34].
2. Network layer: Network layer supports the entire infrastructure over wired or wireless connections that connects things together. It allows the objects to be aware of their surroundings. The network layer is responsible for addressing the issues related to the security and privacy as well as the requirements of QoS and Network energy efficiency. [34]

3. Service layer: This layer relies mostly on middleware technologies to create and manage all the services required by the applications and users. Services running in this layer retrieves dynamically metadata about services. All the services-oriented activities are performed at this layer. These activities include managing data, storing data, exchanging data, search engines, communication between services, and defining ontologies. [34]
4. Interfaces layer: In an IoT network, there are many different services available which are provided by multiple and versatile vendors. These services are not always compatible with each other. Interface layer provides a mechanism that simplifies the interaction between the services. In other words, this layer provides the users the means to interact with other users and applications [34].

SOA is very useful for the programmers as it ensures the interoperability among the heterogeneous devices in multiple ways [34].

There are many common characteristics that are offered by IoT infrastructure irrespective of the architecture used. These characteristics are mentioned in [36] and include:

1. dealing with heterogeneity;
2. use of resource-constrained devices;
3. applications that require spontaneous interaction;
4. ultra-large-scale networks and a large number of events;
5. dynamic network behavior requirements;
6. context-aware and location-aware applications;
7. the need for distributed intelligence.

Achieving these characteristics are not impossible but can be challenging due to the distributed and diverse nature of the applications. Typically, IoT applications include many different technologies interconnected in the desired way to provide the desired solution. These technologies include Wireless Sensor Networks, RFID, NFC, barcodes, sensors and wireless communication modules [34]. Large amount of data is produced in IoT infrastructure which need to be analyzed properly and quickly to generate required information. Furthermore, energy consumption of the devices used in the infrastructure is also an important criterion in deciding the type of sensors and equipment to be used when creating a service. One possibility is Raspberry Pi for such applications due to its good hardware expansion capabilities and it is low-cost hardware with powerful functionalities [36]. Another alternative is Single-Board Computers (SBC) that provide similar functionalities to Raspberry Pi, but they are more expensive [36].

As explained earlier that energy consumption is an important factor in creating IoT infrastructure. Distributed nodes in IoT is battery operated and communication devices consume a lot of battery power. Most commonly used communication protocols are ZigBee or Bluetooth [36]. In case where it is required for the sensors to be distributed over long

distances, there exist propagation problems [36]. There are few emerging technologies such as LoRa/LoRaWAN (Long Range Wide Area Network), Sigfox or IQRF that aims at solving this issue [36]. They provide long distance solutions with energy saving possibilities.

2.2.1 Applications of IoT

In this section, few case examples of IoT applications are presented to highlight the implementation of state of the art. IoT has a great impact on society by providing applications in many fields. IoT has information retrieval applications for users which provide additional information on various products. Furthermore, increased number of products which have identification, are being manufactured to help manufacturers. IoT can also increase the effectiveness of communication techniques within traditional industries by providing better data and information exchange. [34]

The applications of IoT that are currently available can be categorized into industrial applications, social IoT, infrastructure and Healthcare applications. In industry business transactions are improved and smart networking possibilities are provided by using IoT. This improves the efficiency of processing information in real time. IoT can help the digital economy in reducing the gap between components due to improved networking. Moreover, Enterprises that use IoT based solutions can get more profit and can get real-time generated information that helps in making efficient decisions when making business models. Since IoT is connected globally it provides the business partners the opportunity to integrate enterprise resources in a better way. Table 1shows the industrial applications of IoT. [34]

Table 1. *IoT applications in Industry [34].*

Industrial Deployment	**Applications**
Logistics and SCM (Supply Chain) Management	Goods Position Monitoring; Theft prevention; Container monitoring in SC; SC events monitoring
Access control	NCF Access control system; E-home; Security infrastructure
Control of industrial processes	Intelligent Quality control system;

Social IoT known as (SIoT) as mentioned in [34], *"is proposed to describe a world where things around human being can be intelligently sensed and networked."* Scalability and discovery of the services can be improved through SIoT. Furthermore, implementing the security and privacy techniques of SIoT can improve overall IoT security. SIoT was encouraged by social networks that run on the internet such as Facebook and Twitter. Hence, it has a great impact on the lives of people. This architecture is represented in Figure 5.

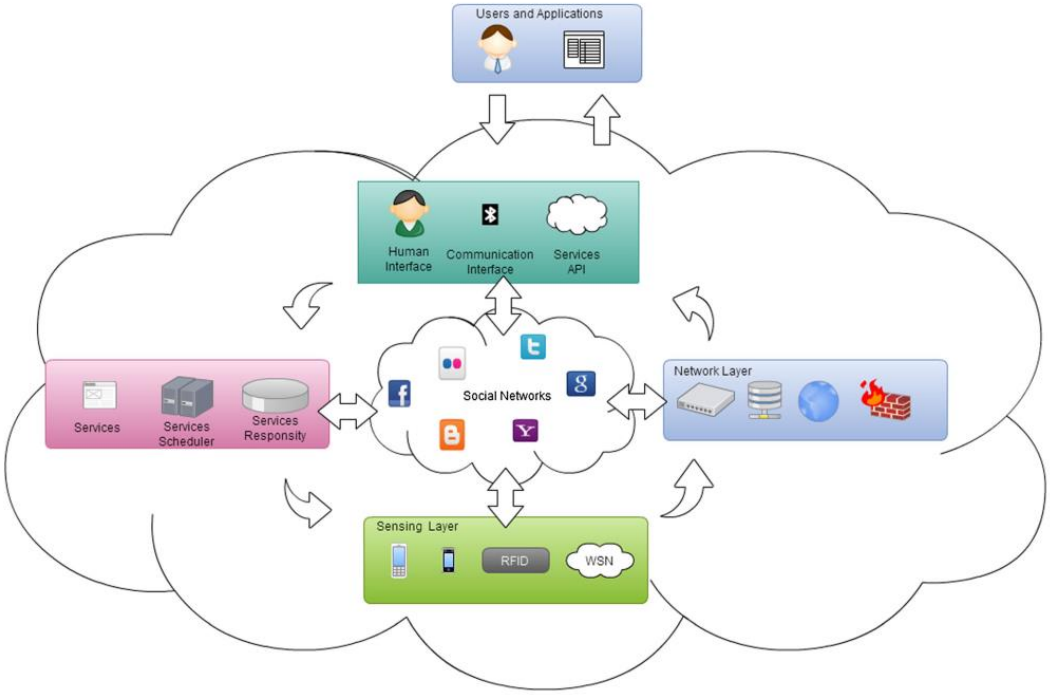

Figure 5. *Social IoT architecture [34].*

In healthcare IoT helps to improve quality of service and lower costs. There are many devices in multiple areas of biomedical that utilizes IoT. The driving force of adopting IoT into healthcare is the advances in sensor technologies. Moreover, IoT can provide solutions that provide living assistance to elderly people. As an example, wireless biosensors are in use for tracking, monitoring and taking care of elderly people. IoT will in future help in improving life quality and can prevent health problems. [34]

IoT applications are widely seen in smart cities, home automation and monitoring of environmental changes. IoT is helping in reducing waste in smart buildings. A case example is 'Sensing China', launched in June 2010 and uses identification tags to broadcast information to the internet [34]. This data can be used then to monitor for example the energy consumption.

2.2.2 Problems associated with IoTs

As seen with the other emerging technologies, IoTs also have few drawbacks and problems associated with them. The two challenges of most concern when it comes to IoT are security and privacy. There might be RFID tags attacks or data leakages. [34]

Furthermore, there is no fixed topology defined for automated management. Other challenges include providing diagnostic facilities to reduce malfunctioning of deployed services. The open nature of IoT itself raises questions about the security of the information and privacy since the information generated and circulated among services can contain very personal information about individuals. To avoid these problems the data must be encrypted, and a system shall be devised which can provide solutions for proper authentication of the services before the access to confidential data is provided. [38]

2.3 Architectures and Frameworks for implementing IoT management systems

According to Anthopoulos and Fitsilis [12] research of existing urban cities, many architectural styles have been developed for the implementation of smart cities. These architectures can be used to make an effective management system for IoT services. Three commonly observed architectures are multi-tier, Service Oriented architecture and Even-Driven architecture. Event Driven Architecture (EDA) [15], is based on the various events that occur in the smart cities. Events are triggers which are generated by real-world objects and upon these events a response can be produced. Even though European research is focusing on this approach, there is no clear existing implementation found in the research done by Anthopoulos and Fitsilis [12]. Service Oriented Architecture, explained in detail in Section 2.3.1, is emerging and gaining fame in modern Smart City applications. However, Anthopoulos and Fitsilis [12], concluded that multi-tier is most commonly used architecture. Multi-tier architecture is represented in Figure 6.

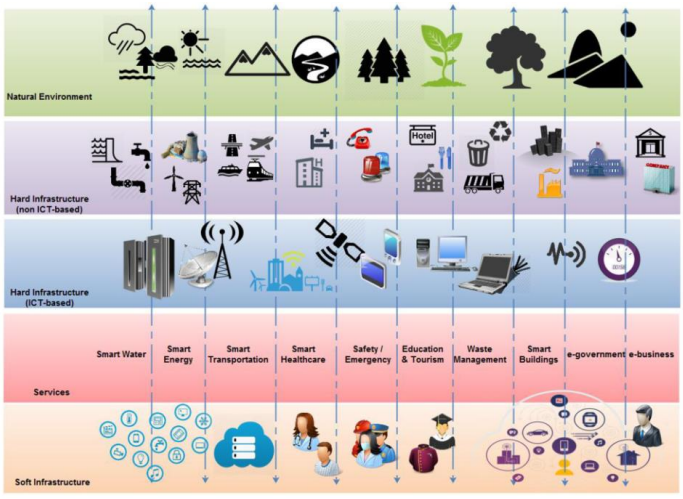

***Figure 6**. Generic illustration of Multi-tier ICT architecture for smart cities [12].*

The above-represented layers in Figure 6 must be present when implementing this architecture, with the utilization of both soft and hard infrastructure provided by the city. These layers are explained in [12] as:

- Layer 1: Layer 1 consists of Natural Environment which are concerned about all the features of environmental issues within the city locality.
- Layer 2: It is for hard infrastructure that does not have ICT induced, provided by human such as buildings, roads, bridges etc.
- Layer 3: This layer consists of all the hard infrastructures provided by humans and consists of smart hardware facilities. (i.e., data centers, servers etc.)
- Layer 4: All smart city services organized according to urban key- performance indicators.
- Layer 5: Individuals and people living in the city, categorizes as soft infrastructure. It also includes software, data, and smart applications.

2.3.1 Service Oriented Architecture

SOA, Service-oriented architecture utilizes services as the basic block. Different applications can be developed on different services. These services can complete overall business process by discovering and utilizing each other. A service according to [55] can be defined as any discrete function that encapsulates reusable business function and can also be utilized by external consumers. Services are well-defined functions and do not depend on other services. The service provider provides the services and publishes the location and endpoint of the service whereas, service consumers are the customers who use these services after finding the required published service endpoint from the service directory. Figure 7 shows the basic connection between SP and SC. In the figure, SC on the right sends a request to SP on the left whereas, SP sends a response message back to SC [29].

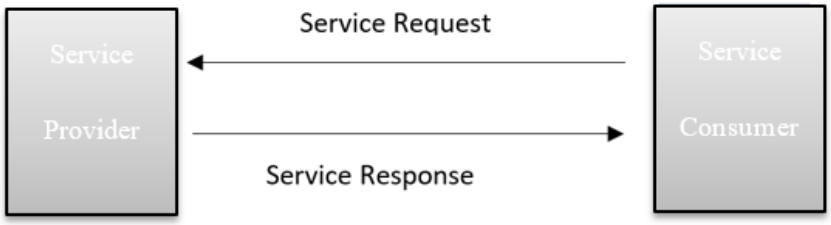

Figure 7. *Basic connection between Service Provider and Service Consumer [29].*

SOA can be divided into two components, basic and auxiliary services. Basic services consist of Service Provider (SP) and Service Consumer (SC) whereas Enterprise Service Directory (SD) or registry and Enterprise Service Bus (ESB) are auxiliary services [30]. It is very important to know the principles of SOA for successful implementation. These principles are discussed in [26] and are defined briefly as follows:

- **Abstraction:** Abstraction aims to conceal logic from outside world, which is used to design the service. The description of the service is only provided in the service contracts. Service meta information is categorized into four fields which are functional, technology, programmatic and service quality. Service contracts and service registry is then made by using this information. Open access is limited only to interested consumers and service owners.
- **Autonomy:** The aim of this principle is to make services which can control their logical operations. Autonomy provides the services with the freedom from the technology restrictions. It means that the services are platform independent and it is not necessary for the applications, built on these services, to follow the same development platform as the service itself. This also implies that the services can run on diverse platforms.
- **Loose coupling**: This principle states that all the services are independent. This highlights that in SOA services shall be able to engage with each other without knowing where that service is physically located. In other words, the services shall not be tightly coupled with each other and must be unaware of their surroundings.
- **Reusability:** As the name implies the services shall be able to use or be used as an existing asset. This can be only possible if the service is built generic in nature. Reusability provides cost avoidance opportunities and can be used by multiple consumers.
- **Discoverability:** All the metadata containing the information of a service, are stored in the service registry, as explained earlier in abstraction. Service consumer finds the required service from the service registry. This phenomenon is defined as discoverability, stating that the service shall have communicative metadata so that it can be easily found and used by the consumer. Discoverability is very important since it is impossible to construct an efficient infrastructure if the services cannot find each other.
- **Composition:** This principle states that services shall be able to participate in the composition and they can adapt to new business logic by making changes at the runtime. It promotes that services shall be able to create new solutions by using the existing services [26]. Composition reduces development costs by enabling dynamic reconfigurability of services without the need for hardcoding them.
- **Statelessness:** Statelessness is achieved by suspending state information management when necessary [26]. While doing so, the consumption of resources is re-

duced to the minimum. It is important that the developers should try to avoid unnecessary resource consumption so that it is possible for the service to satisfy more requests in order to achieve statelessness.

It is very important to have services discovered by the consumer, which is possible through web services. Web services can be defined as the set of protocols to publish, discover and use services in a standard form [20]. Web service is not the compulsory component in developing SOA, but they assist in its implementation. Table 2 shows a comparison of service principles between Web Services and SOA enabled services.

***Table 2**. Comparison of good service principles between SOA enabled Services and Web Services [28].*

Principles enabled by Web services	*Standardized*	Standards-based protocols.
	Technology neutral	Endpoint platform independence.
	Consumable	Enabling automated discovery and usage.
Principles enabled by SOA	*Reusability*	Use of Service, not reuse by copying of code/implementation.
	Abstracted	Service is abstracted from the implementation.
	Published	Precise, published specification functionality of service interface, not implementation.
	Formal	Formal contract between endpoints places obligations on provider and consumer.
	Relevant	Functionality presented at a granularity recognized as a meaningful service by the user.

SOA implementation brings many advantages and promises. It has provided software industry flexibility, lower cost, and autonomy but on the other hand, it also has some drawbacks such as security issues. These security risks become at large when the SoAs are deployed on the third-party cloud. Some of the principles of SOA such as abstraction help reduce security risks. However, security risks associated with SOA is still at large. Furthermore, Cloud Computing (CC) is currently the efficient mean of deploying SoAs.

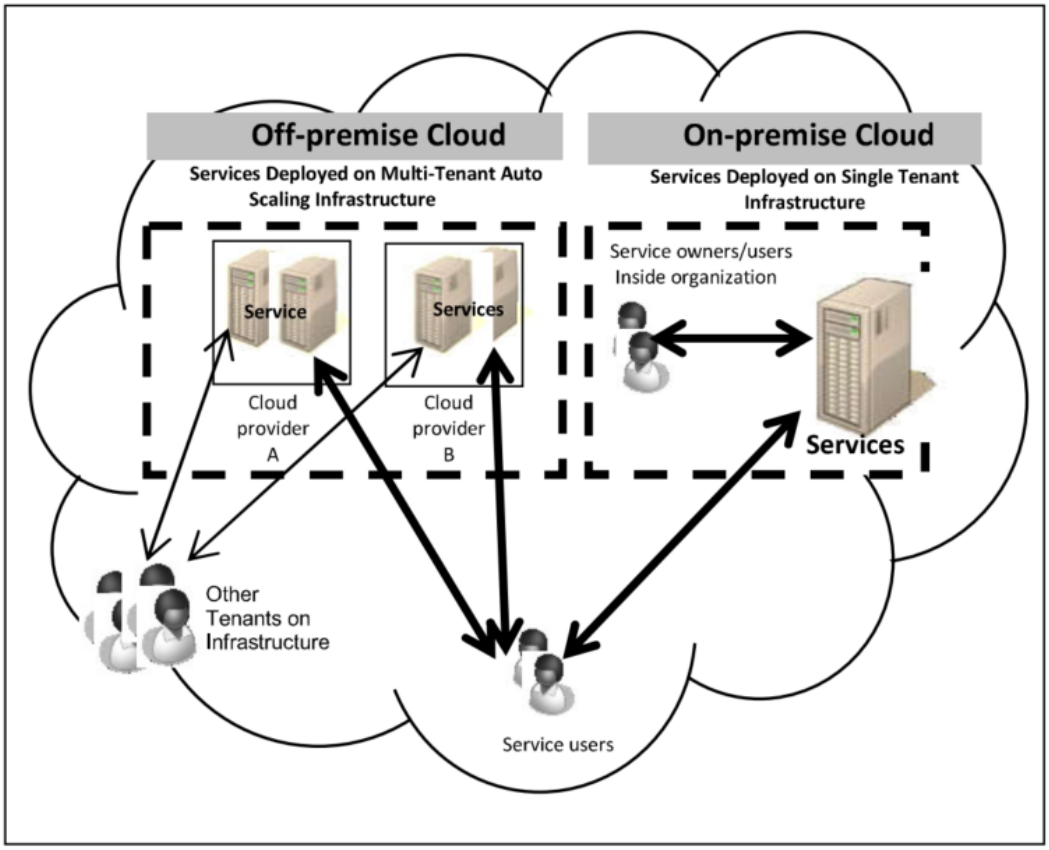

Figure 8. *SOA deployed on off-premises versus on-premises cloud computing [26].*

Figure 8 illustrates two methods of deployment on CC, on-premise and off-premise. In on-premises the services are running on the organization's own infrastructure and hence are only managed by the organization itself. In this scenario, the risks associated with SoAs due to CC are limited. Whereas, in the off-premise CC, widely in use nowadays, infrastructure is provided and managed by CC service providers [26]. Service providers offer auto-scalability and multi-tenancy which implies that the infrastructure is shared among many companies. Furthermore, services might demand the resources located on different virtual machines in distant locations. This method is essential when cost reduction and less utilization of limited resources is needed.

SOA is built on XML which is itself not very secure language. XML rewiring attack is a known XML exploit, despite the efforts of WS-Security and WS-Policy standards developed to secure XML based applications.

Security risks associated with SOA can be discussed separately as security risks related to each SOA principle discussed above. These risks are shown in Table 3. The first column shows the principle of SOA, applications in the second column whereas, the third and fourth column shows the technologies required to implement corresponding principle and the risks associated with off-premise CC respectively. Furthermore, these risks are mapped to CSA cloud attacked which were observed in 2013 [26].

Table 3. *SECURITY ISSUES RELATED TO THE UTILIZATION OF EACH SOA PRINCIPLE IN OFF-PREMISE CLOUD [26].*

SoA	**Applicable to**	**Vulnerable Technologies**	**Risks due to CC**	**CSA Notorious 9 Threats**
Abstraction	Service Contract	SOAP UDDI WSDL XML HTTP	1. Exposure 2. Redundancy and integrity issues 3. Access control 4. Trust	1.0 Data Breaches 4.0 Insecure Interfaces and APIs 3.0 Account or Service Traffic Hijacking
Discoverability	Service contract and Service Registry	SOAP UDDI WSDL XML HTTP	1. Exposure to cloud risks 2. Authentication and access control	1.0 Data Breaches 4.0 Insecure Interfaces and APIs 3.0 Account or Service Traffic Hijacking
Composability	Services	TCP communication XML	1. Lack of standards 2. QoS. 3. Availability 4.Trust 3. Compliance 4. Governance	1.0 Data Breaches 4.0 Insecure Interfaces and APIs 3.0 Account or Service Traffic Hijacking 5.0 Denial of Service 6.0 Malicious Insiders

Autonomy	Services	TCP communica-tion XML	1. Lack of standards and safe patterns 2. Exposure to cloud risks	1.0 Data Breaches 4.0 Insecure Interfaces and APIs 3.0 Account or Service Traffic Hijacking 9.0 Shared Technology Vulnerabilities
Formal contract	Service Contract	SOAP UDDI WSDL XML HTTP	1. Trust 2. QoS 3. Authentication and access control	6.0 Malicious In-siders 5.0 Denial of Ser-vice
Loose coupling	Services	TCP communica-tion XML	1.QoS 2.Exposure to cloud risks 3. Data Interception	5.0 Denial of Ser-vice 4.0 Insecure Interfaces and APIs 1.0 Top Threat: Data Breaches
Reusability	Services	TCP communica-tion XML	1. Compliance 2. QoS 3. Exposure to cloud risks 4. Authentication and access control	6.0 Malicious In-siders 5.0 Denial of Ser-vice
Stateless-ness	Services	TCP communica-tion XML	1.Exposure to cloud risks 2.Compliance 3. Trust	1.0 Data Breaches 4.0 Insecure Interfaces and APIs 3.0 Account Hi-jacking

The above-mentioned risks in the Table 3, associated with each principle are discussed in detail with some recommendations to minimize them, as follows.

- Abstraction: As explained earlier with less abstraction there will be more information exposed about the logic of the service to the outside world, making it an easier target to the attacker. Hence, the vulnerability is increased, and the security risks become a greater subject. However, more abstraction will decrease reusability. Another issue related to abstraction is access control. When the service is on-premise, the owner has access to all the sensitive information, such as source codes, design specifications, etc. Off-premises deployment of the SoAs increases the possibility of account hijacking since the SoAs are exposed to the outside world. Also, due to abstraction, composite services are not known to the developers and consumers. Therefore, it is recommended that when developing a service, the service developers shall balance the amount of abstraction. Likewise, during implementation services should be monitored appropriately and consistently for risks management.
- Discoverability: Due to this principle and the off-premise nature of deployment on the cloud, PAC files are exposed. Whereas in on-premise deployment, these files can only be attacked if the attacker is within the company premises. Furthermore, service broker adds an extra vulnerable layer of communication. Hence, it is recommended to use some sort of authentication between services and SB when communicating. It is also advisable to use automatic updating of services provided by many SoAs vendors to constantly update security software.
- Composability: Composability encourages reusing existing services to compose new services, but it lacks standard rules to define how to securely create new solutions. Moreover, creating new solutions will affect the parent services. The quality of service (QoS) created, is also affected since the CC infrastructure is unknown. CC is a multi-region infrastructure and therefore can cause issues related to compliance as well. Compliance issues relate to the problems arising due to the regulation of the country in which a sub solution is created, which may differ from the regulations of the country of parent services. Composability also increases transit time since the communication between composing services is augmented. Therefore, it is highly recommended to follow safe composing methods such as, encryption techniques and using digital signatures when transmitting data. Furthermore, the composed services must be monitored, and the compliance shall be analyzed with the specific country regulations before allowing the services to compose new solutions.
- Autonomy: When autonomy is implemented, the risk of exposing services on many different platforms also increases. Service autonomy requires a greater amount of trust between applications to avoid malicious modification and avoid service integrity issues [26]. Services need to communicate with each other in

order to maintain control over resources. More resources mean more communication which, in term implies that the possibility of security risks is of greater magnitude as more resources will be accessible for malicious attack. It is recommended to have a thorough assessment of the need for implementing autonomy. A strong validation shall be implemented to verify trusted inputs. Finally, WS-Security can provide trusted communication between applications and autonomous services.

- Loose Coupling: Implementation of this principle reduces throughput and increases latency to the response time since messages are transmitted over the internet. Furthermore, more messages are exchanged between the environment and the service. Encryption is one of the methods to avoid any data infringement. The bandwidth and the latency overhead can be reduced by using compression techniques.
- Reusability: Reusability principle is implemented to increase cost efficiency, but it can lead to compliance issues similar to composability. Another issue that can arise due to reusability is the difference in configurations in different CC infrastructures. This will lead to QoS variance and the change in the standards can lead to higher level of insecurity. This problem can be minimized by testing service on many different platforms and infrastructures before deploying them on a cloud. It is also important to make sure that the service data is not illegal throughout the service stages [26].
- Statelessness: When statelessness is implemented external components can be placed anywhere in the world. Therefore, it becomes a necessity to make sure that the latency limits are met in the cloud. Moreover, state of the service is being exposed while communicating between different clouds. Once more encryption is the basic method to secure the communication and like loose coupling compression techniques can help in reducing bandwidth and latency overhead [26].

2.3.2 Web-Based SOA

As explained earlier Web services are an essential component in SOA. It uses Web Services Description Language (WSDL) in order to describe services available. According to [22], *"WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information". WSDL* can provide the description of endpoints without being affected by the type of communication protocol used. Hence, it makes it an extensible language [22]. Extensible Markup Language (XML) is the default message format for WSDL.

XML is commonly used to make information formats and describe data since it is a very flexible. It can transfer structured data electronically over networks and internet. It is self-defining and self-describing. Element is defined by tags and is the basic building block in an XML document [27]. Furthermore, there is always a starting and ending tag in an

element [27]. This is illustrated in Figure 9 which shows that the city name is Burnsville. The starting tag is <city> whereas, the ending tag is </city> [29].

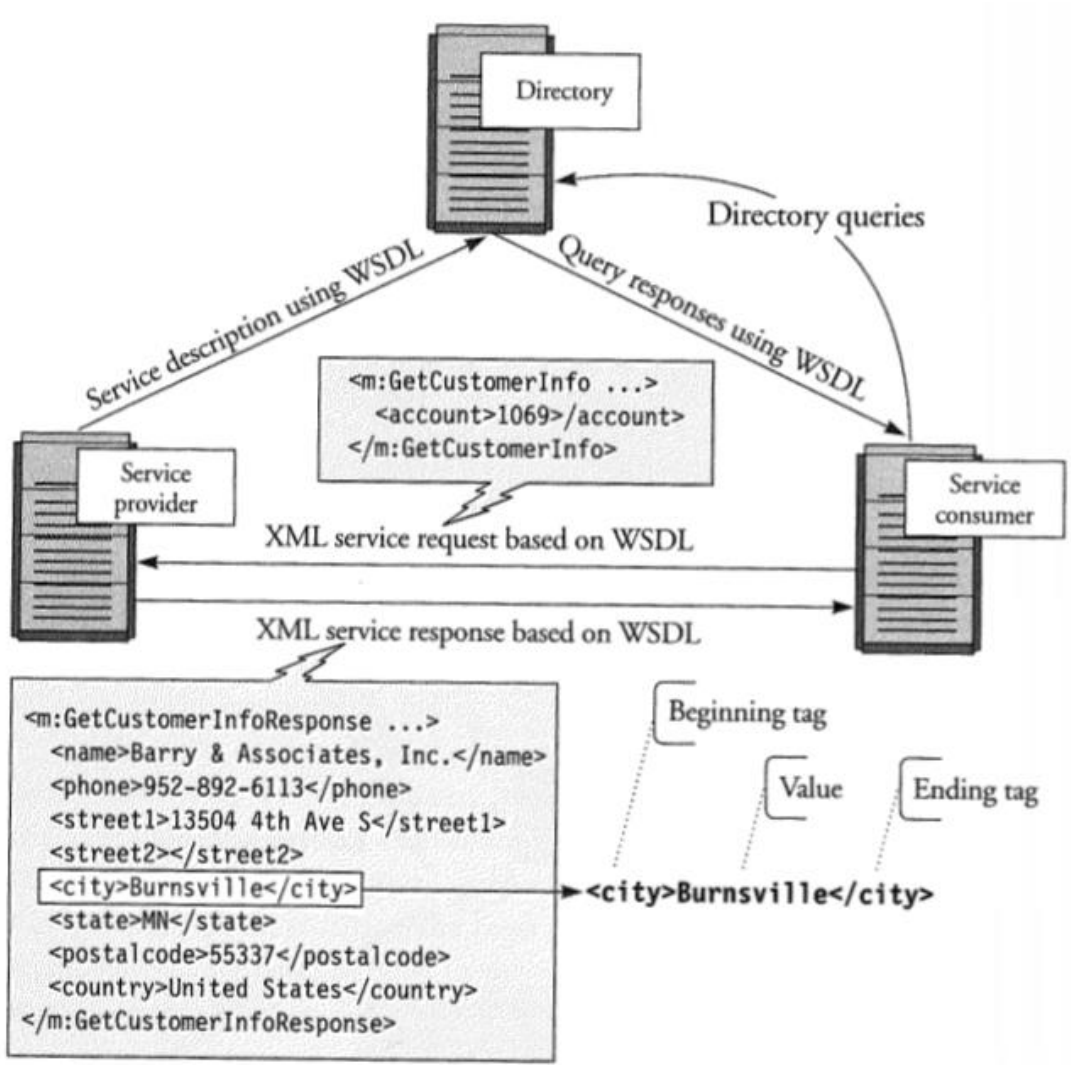

Figure 9. *XML message tags [29].*

There can be nested elements as well in XML. Furthermore, XML does not have any predefined tags. The name of the element describes the content whereas the structure relates the relationship. XML document shall be well organized and easily understood by XML parser. XML supports elements attributes and the characteristics of an element can be defined in the beginning tag. It has endless applications since it defines information in a standard way. These applications are able to perform normally with the addition of new data. Data is stored in different ways which is incompatible with each other in normal applications. This makes the exchange of data between different systems time-consuming since a large amount of data must be converted into compatible format before exchange and in this process, data is often lost [24]. XML is suitable for such applications since it provides data to all kinds of devices without extra processing.

Service Directory (SD) is the place where information for all published and available services described in WSDL, is held. One of the methods of creating such directory is to use Universal Description, Discovery, and Integration (UDDI). Finally, Simple Object Access Protocol (SOAP) is used to send WSDL among the services. SOAP provides envelop for sending web services messages over the internet [29]. There are two parts in this envelope. First one is an optional header which provides information on the authentication, encoding technique and the method of processing SOAP message by the recipient [29]. Whereas, the body of the message is described in the second part. There are two main encoding styles in SOAP, known as Remote Procedure Call (RPC) and Document-Literal. In this encoding style WSDL definition contains name and type of each argument. Whereas, operation arguments are defined by elements in the body of SOAP. The standard encoding format is used in order to serialize the data. SOAP specifications contain the information about this encoding format. RPC-style encoding interactions are represented in Figure 10.

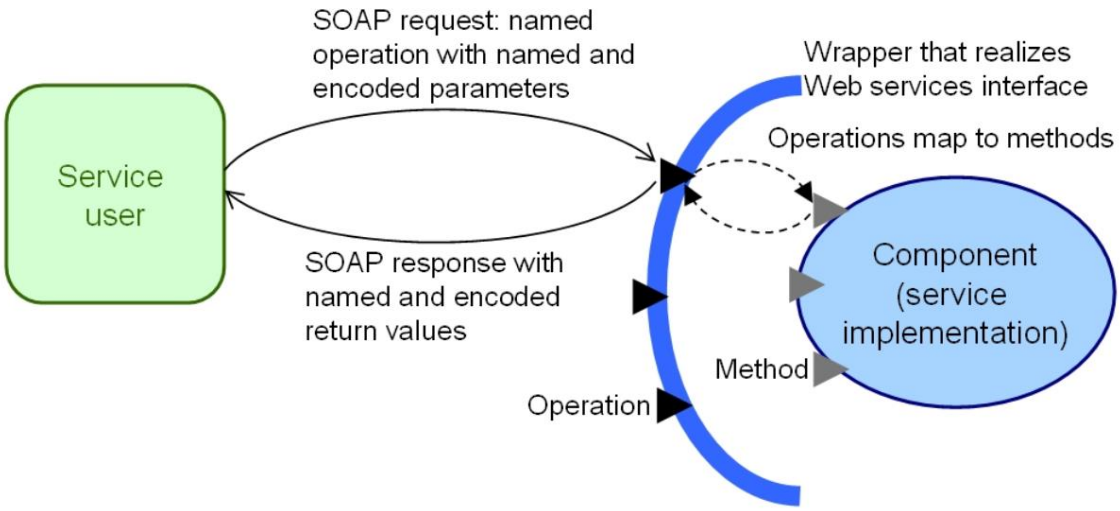

Figure 10. *RPC-style SOAP [21].*

This technique was popular as it has a simple programming model, but this technique can cause interoperability problems. [21].

Document-Literal SOAP can contain the business document in its request. In this approach, the body of the message follows no standard structure of XML content. There is no standard encoding standard used. So, in order to successfully encode SOAP body, rules are specified in XML schemas that are created by the developer of the service provider. [21]. Figure 11 shows this process.

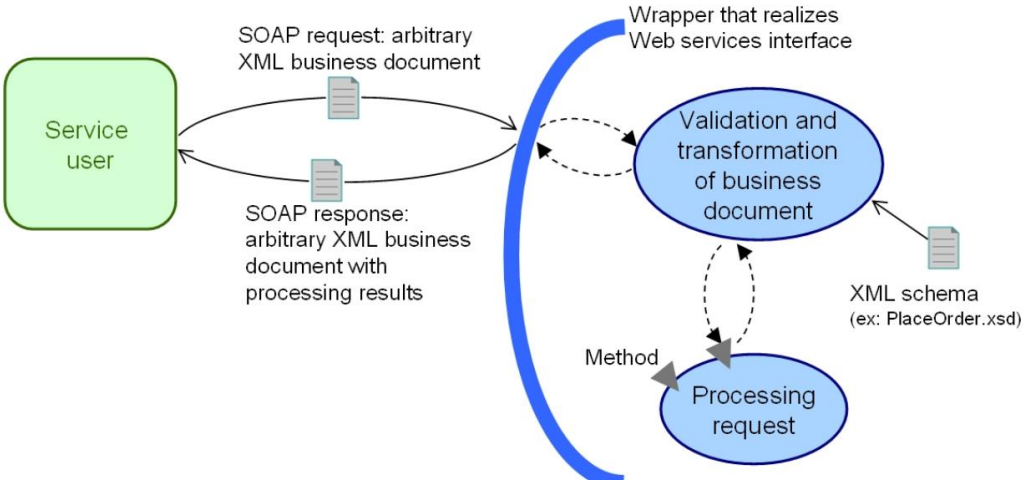

Figure 11. *Document-Literal SOAP [21].*

The commonly used protocol in SOAP is Hypertext Transfer Protocol. The Hypertext Transfer Protocol (HTTP) is a protocol that utilizes statelessness in all the requests and responses. This protocol operates at an application level and uses semantics which is extensible and message payloads that are self-descriptive in order to provide flexible interaction with information systems. Moreover, HTTP is designed in such a way that it can be used to translate communication between HTTP and non-HTTP based information systems. Hence, acting as an intermediate protocol for communication. Furthermore, different resources are identified in HTTP by using Unified Resource Identifiers (URIs). Intermediaries are used in HTTP to manage requests through connections. There exist three forms of intermediaries which are gateway, proxy, and tunnel. A single intermediary can act as all kind of intermediaries, switching between them, depending on the request nature. HTTP tunneling is a process by which the outside client can bundle all the information needed by the broker into normal HTTP request. A tunnel behaves like a blind relay since it does not change the messages between two connections. When both ends of the connection that are relayed closes, tunnel shuts down. A "proxy", is a message-forwarding agent and is selected by the client, using local configuration rules. It is used to receive requests for absolute URI of some type(s) and attempt to satisfy them by translating through the HTTP interface [25]. Whereas, a gateway acts like an origin server, aiming at translating and forwarding requests coming from outbound connections to other servers.

SOAP protocol is sometimes avoided in pure web services method by using *Representational State Transfer (REST),* which is a simpler method then SOAP. [21]. Roy Fielding [25] proposed Representational State Transfer (REST). The complexity and processing overhead is reduced due to use of HTTP only. Every information piece has a unique identifier which enables the information to be retrieved by using a unique endpoint. As an example, let take car fuel tank capacity as a service and car fuel efficiency as another service. Both these services can be utilized using REST as

www. example.com/tank

www. example.com/ fuel

REST relies on HTTP protocol to establish communication and carrying out four basic operations between the service provider and the service consumer. These four basic operations are POST, GET, PUT and DELETE. These four basic operations relate to create, retrieve, update and delete (CRUD) operations in a restful application. GET request asks the service provider to give data corresponding to the endpoint identifier, which in the above-mentioned example can be tank or fuel. POST requests inform the service provider to create the data. PUT indicates to replace the data with the new one and finally DELETE request, requests the service provider to delete the data. REST has many advantages over SOAP such as modifiability, simplicity, high interoperability and better performance as it can cache the responses when possible or desired. [21]

ESB provides support to the web services in routing messages. These messages can be from fixed to fixed applications or from one source to many applications. The messages can be either dynamic based on system availability or on load balancing. ESB is further equipped with authentication, encryption and authorization facilities. [21]. Figure 12 illustrates a basic overall process of Web Service discussed in detail above.

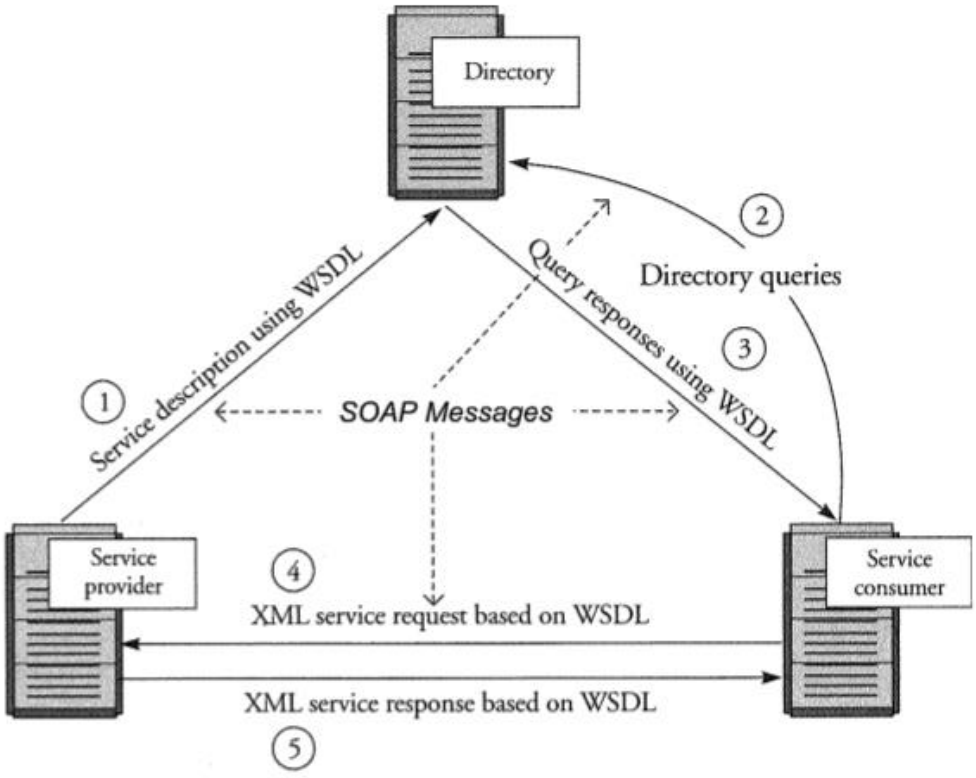

Figure 12. *Web Services Basics [29].*

Steps involved in publishing and consuming a web service as explained in [29] are as follows:

1. Service is published to a directory after being described using WSDL.
2. Service consumer queries the directory to get information about the location of the desired service and the means of communication.
3. The method of the communication required by the SP is defined by the part of WSDL which is passed to the service consumer by the provider.
4. WSDL is used by the SC to send a request to SP in the desired manner.
5. The response is provided by the SP to the request send by SC.

2.3.3 Arrowhead Framework

Arrowhead is based on SOA and follows all the principles mentioned above for SOA. Furthermore, Arrowhead provides a framework which can be used to develop a management system by following the certain set of rules defined by the Arrowhead project. The cookbook version 1.6, which is available for project partners, explain how core services should be utilized. Document model for these core services is also described below in this section. Arrowhead network also contains local cloud core system which can consist several local clouds. Each local cloud shall host at a minimum the mandatory services described in core services

Core services of Arrowhead are divided into mandatory core services and support core services. According to Arrowhead, mandatory core services are the Service Registry system, the Authorization system, and the Orchestration system.

- **Service Registry**

The service registry system is used to keep track of all the available services located in the local cloud. Arrowhead uses domain-based infrastructure, so it is mandatory for all the systems located in the local cloud to publish product services in their service registry. The management tool, explained later in the next chapter presents with the user interface of the service directory.

Arrowhead provides with different documents which allow the developer to follow in a unified way the principles of SOA. Service Discovery REST is one such document and explains the implementation model of service discovery using JSON over HTTP while utilizing Restful web services. Message sequence for Service Registry (REST) is shown in Figure 13.

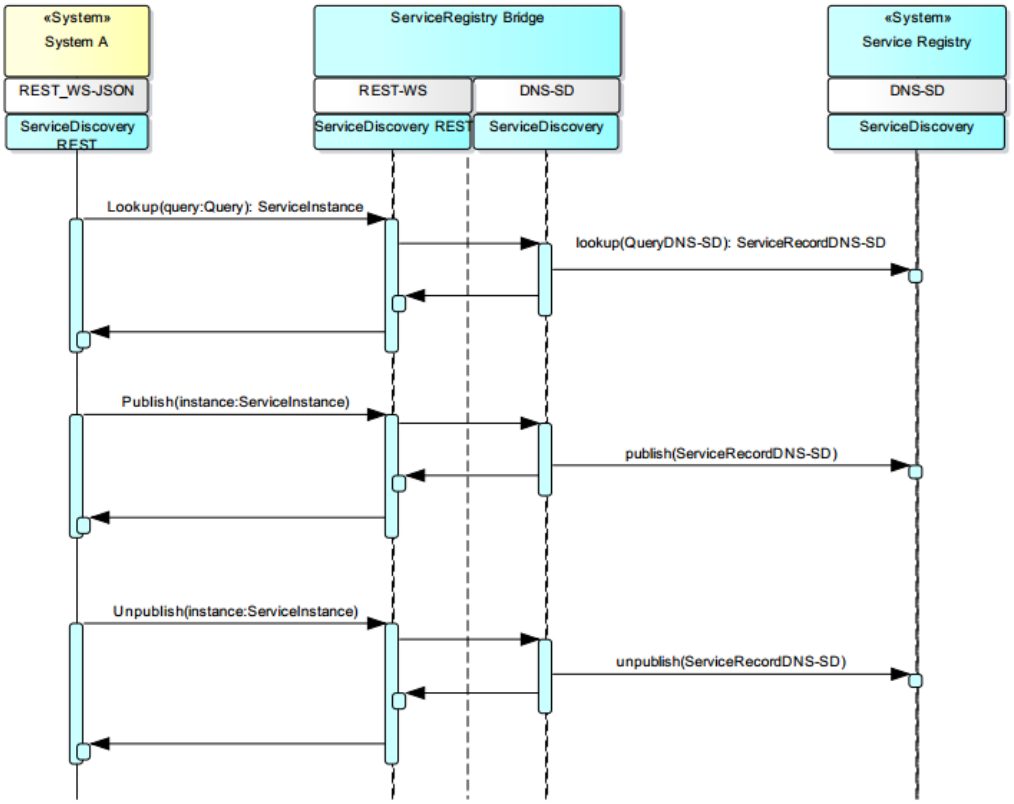

Figure 13. *Message Sequence in Arrowhead Registry Framework [1].*

The sequence describes how to lookup the services registered in the service registry. Furthermore, it also explains the dynamic associated with registering and unregistering the service. Service Registry Bridge is developed and used by Arrowhead framework to translate REST-WS to DNS-SD.

- **Orchestration System**

Service Orchestration provides with the functionality of re-usability and composability. The orchestrator follows the basic principles of loose coupling, autonomy, abstraction and statelessness. Furthermore, Arrowhead Service Orchestration also provides with dynamic service replacement if the service fails. The sequence shown in Figure 14 shows the dynamic behavior when retrieving configurations. System A registers a service to the registry which system B can discover and utilize later.

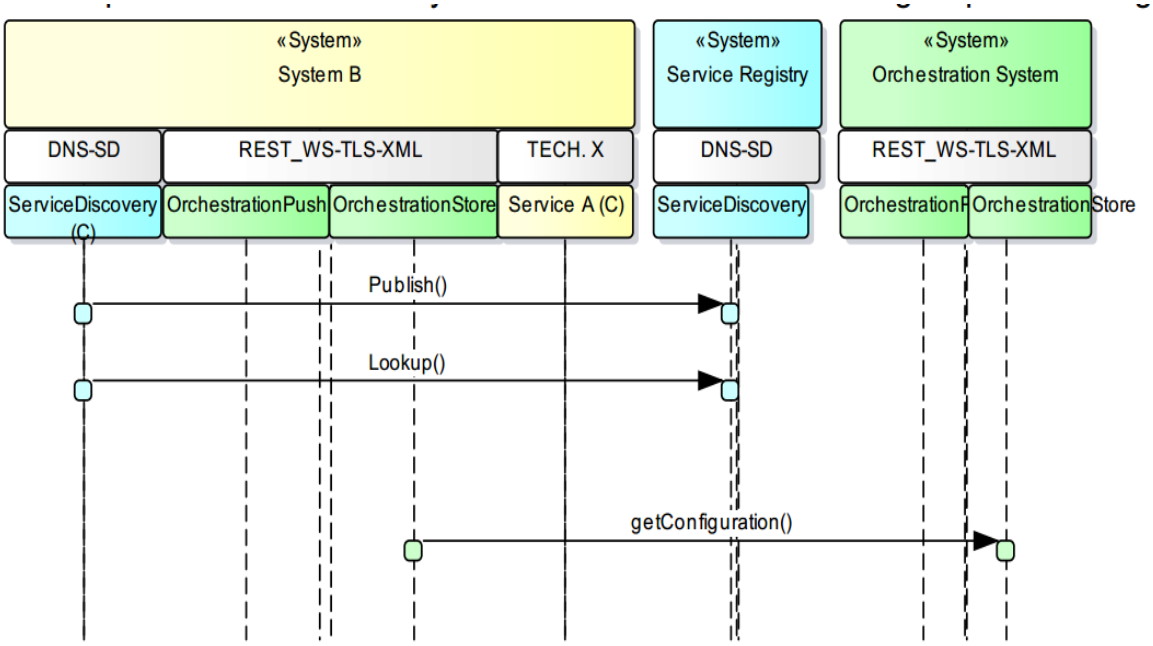

Figure 14. *Getting Configuration from Arrowhead Orchestration System [2].*

- **Authorization System**

Arrowhead framework provides two methods of authorization and authentication. The first method is based on certificates [X509] whereas the second method is based on Radius tickets. The tickets consist of a small group of bytes which contain the necessary information for each service to be authenticated. Authorisation Control core service is responsible for implementing authorization control. Authorize sequence is represented in Figure 15.

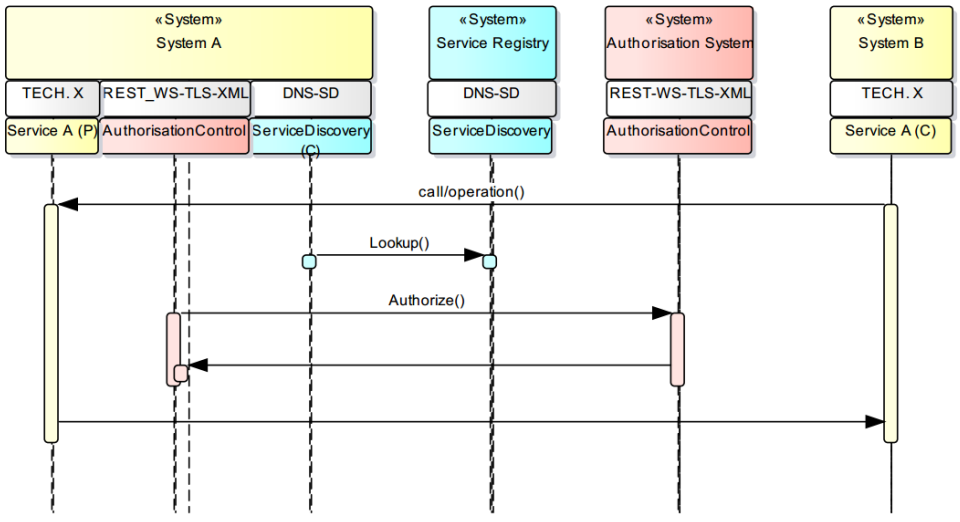

Figure 15. *Arrowhead Framework Authorize message sequence diagram [3].*

When system B sends a call operation to utilize the service A. Service A discovers the Authorisation Control in the Service Registry, then sends the authorize message. If the value received is true, the requested data is returned to the system B.

3. METHODS AND TOOLS

This chapter describes the tools and methods used in the development of an application for implementing IoT management system. The chapter is divided into two subsections describing the methods and tools respectively. The method used in this thesis for communication between services is Representational State Transfer (REST). This chapter also gives a brief introduction of the tools that can be used to develop this system. This section also explains how Arrowhead Management Tool can be used to publish and manage services on the cloud service provided by Arrowhead.

3.1 Methods

This section describes the method in this thesis for developing a management system for IoT services by using the Arrowhead framework. For the design of the application, the method used is the utilization of Representational State Transfer (REST) defined in Arrowhead framework. This section explains how REST method can be used to make HTTP calls.

Representational State Transfer (REST) uses HTTP to make calls and is a very lightweight framework. With the use of HTTP, REST can perform CRUD operations which are Create, Read, Update and Delete. In this method, each data and resource have a unique URI (Unified Resource Identifier) which can be used to operate CRUD operations. The methods used to perform these operations are GET, POST, PUT, and DELETE. GET is used to retrieve a resource from the given URI whereas DELETE erases the resources at that URI. POST is used when a subordinate is to be created for the resource at the addressed URI. PUT method is used when the data is to be updated.

REST is independent of language and platform. Therefore, it is used to develop RESTful applications. A RESTful application has the capability to run on several platforms such as mobile, social applications, web applications etc. Furthermore, REST can perform all the functions which are performed by Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL). REST can take advantage of HTTPS security features by establishing a secure connection over HTTPS. These security features are such as encryption or username and password tokens. The main characteristics of RESTful applications are that they are stateless, supports caching of resources, uses a client-server system, support proxy server, and uses logical URLs to identify resources.

HTTP status codes are used to further inform the user about the status of the request. Most commonly used status codes are status code 200 (OK) which is received when everything went as expected and the request was successfully executed, code 500(Internal Server

Error) is used to show the errors related to the server, code 404(Not Found) which happens when there is nothing to be found at the resource URI. Finally, 403(Forbidden) is used to indicate that the server is refusing to fulfill the request, mostly due to the reason that the requestor is unauthorized to the resource.

3.2 Tools

Several tools are available that can be used to support in the implementation of the technique. This section briefly discusses some of these tools such as Arrowhead Management Tool, NodeJS, jQuery, Mozilla HTTP Requester, CanvasJs, and Mongo DB

3.2.1 Arrowhead Management Tool

Arrowhead Management Tool provides a user interface which is built as a web application for monitoring authorization rules and orchestration.

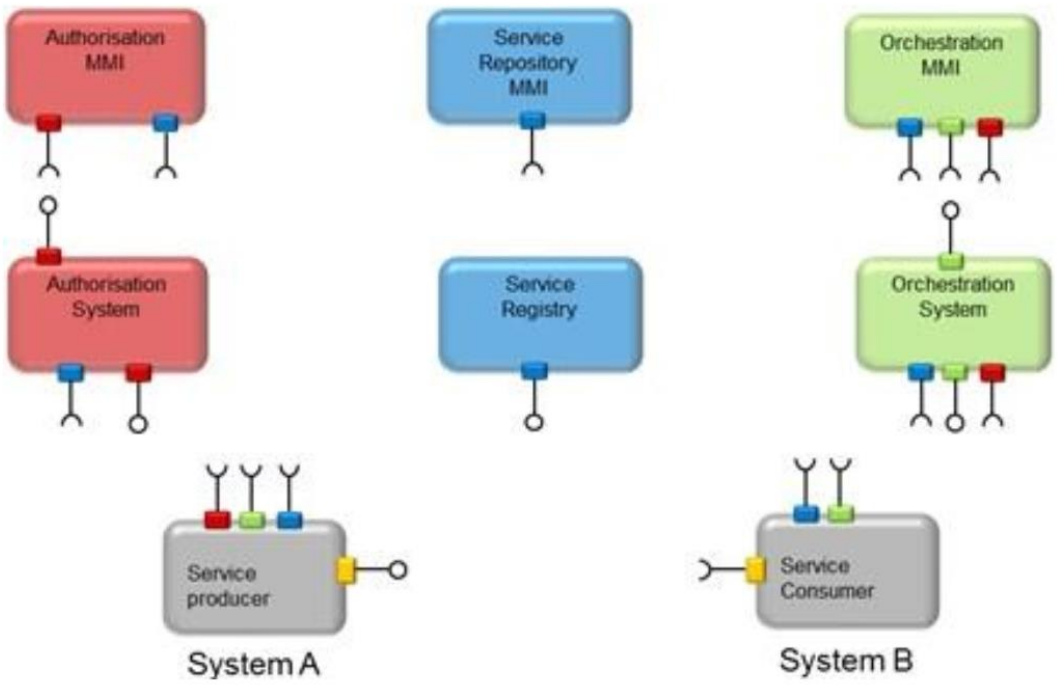

Figure 16. *Components of Arrowhead Management Tool [4].*

As shown in Figure 16, it consists of Authorization MMI, Orchestration MMI, and Service Repository MMI. Furthermore, it allows the user to visually see the available services in service registry.

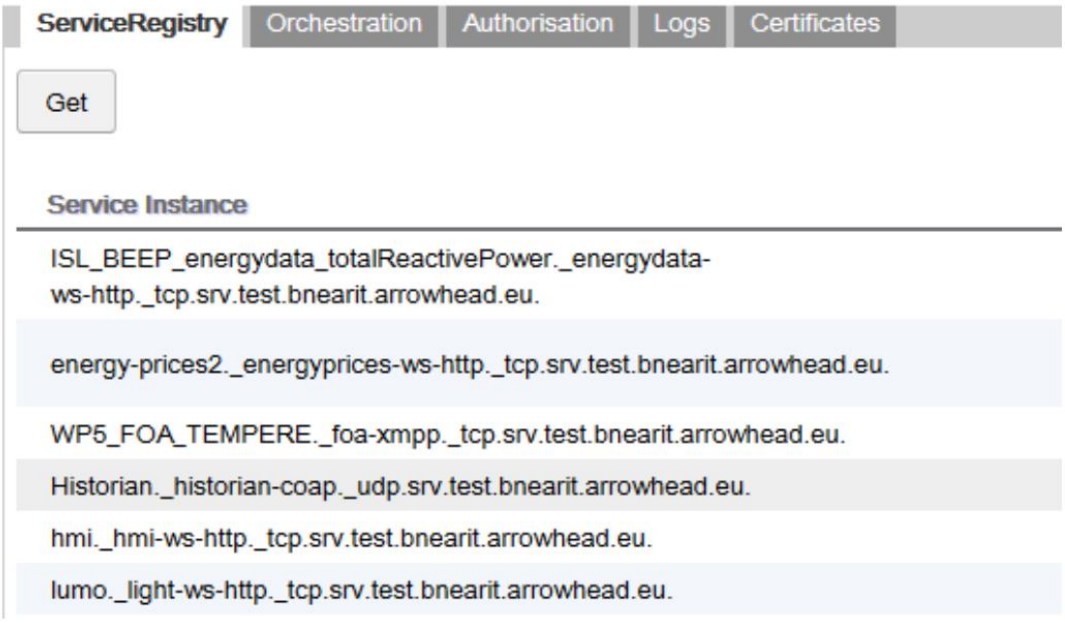

Figure 17. *Service Registry Component.*

Service Registry component as shown in Figure 17, enables the user to see all the services available at that moment. The list can be refreshed by clicking the 'Get' button. Service Instance shows the published services. Hence, after publishing a service the provider can verify that the process has been successfully completed by locating that service name in the service registry.

Next component of Arrowhead Management Tool is 'Orchestration' which describes the orchestration rules. This component tab is represented in Figure 18.

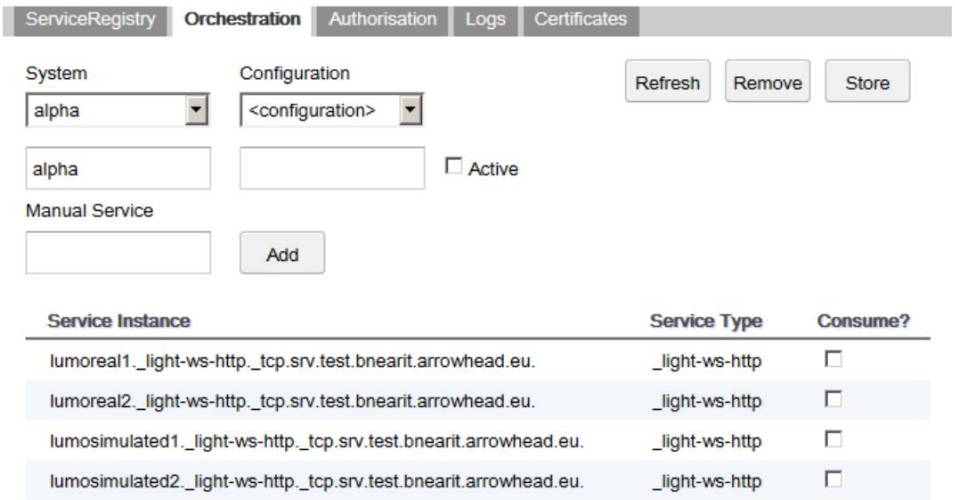

Figure 18. *Orchestration Component.*

To use this component; the user has to select the system from the drop-down menu. In case if the system is not available on the list, it can be entered in the manual tab. A system

can have one or several different configurations. Configurations drop-down menu enables the user to select the desired configuration and then activate that specific configuration by selecting the Active checkbox. To add service instances the consume checkbox is selected and then store button is pressed. There was a problem of not knowing the currently selected services if the page is refreshed while utilizing this component. To overcome this problem a small application was developed, which allowed reconfirming that the services are subscribed to a specific configuration of a system.

Authorisation component provides with the user interface for authorization rules. Arrowhead provided certificates [X509], which are used for allowing a system with the authorization to utilize services. Services can be authorized individually, or another way is to authorize a system to consume all the services by using the asterisk mark. New rules can be added in the 'Add Rule' box whereas 'Delete' enables the user to delete an existing rule. Figure 19 represents this component.

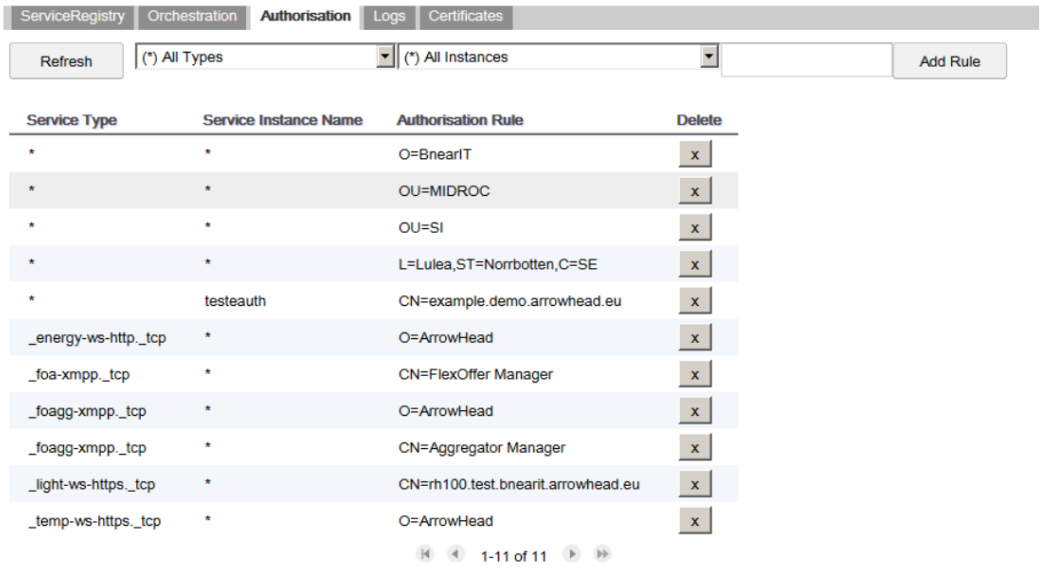

Figure 19. *Authorisation Component.*

3.2.2 Node JS

Node JS is a programming language which is built on chrome's V8 JavaScript engine. It is lightweight and is very efficient since it uses non-blocking I/O model which is also event-driven. Node is designed to handle many connections simultaneously and is used to build extensible networking applications. The calls between the connections are asynchronous. For each connection, a callback is fired which returns the desired output of each connection request. Some programming languages implement concurrency with different threads, which is relatively difficult and inefficient to use. Thread-based networking uses different cores of the system to do simultaneous tasks. Node can also make use of multicore environment with child processes. Node gives the opportunity to build a non-blocking system hence it is best for developing network applications. [5]

API reference documentation which can be found at [6], provides with different functions and objects present in NodeJS. It also explains arguments, return values and errors associated with a method.

3.2.3 MongoDB

MongoDB stores data in the form of JSON objects which can have variable structure. MongoDB stores relevant data together to enable faster query of the information. In MongoDB, new records can be created without initially defining the structure. This is possible because MongoDB uses dynamic schemas. It is very easy to simply add new fields or delete existing ones in this data model. Furthermore, it allows you to store arrays, complex structure and represent hierarchical relationships more conveniently. [7]

MongoDB provides some high functionalities which are not possible to be offered in simple key-value stores other than the rich set of features offered by MySQL. Table 4 illustrates these functionalities in comparison to MySQL.

Table 4: *Comparison between MySQL and MongoDB [7].*

Functionalities	**MySQL**	**MongoDB**
Rich Data Model	No	Yes
Dynamic Schema	No	Yes
Typed Data	Yes	Yes
Data Locality	No	Yes
Field Updates	Yes	Yes
Easy for Programmers	No	Yes
Complex Transactions	Yes	No
Auditing	Yes	Yes
Auto-Sharding	No	Yes

3.2.4 jQuery

jQuery is a JavaScript library which is built to enable fast handling and manipulation of HTML documents, animations and event handling. It is compatible with numerous browsers [8]. jQuery provides support for AJAX calls and it is easy to use due to the reason that it can handle JSON data with the field identifier without parsing them. Furthermore, jQuery is also extensible.

3.2.5 Mozilla HTTP Requester

Mozilla Http Requester is a development tool provided as an add-on for Mozilla Firefox. It is useful in developing RESTful applications. It can be used to verify the responses of different CRUD operations. Mozilla Http Requester supports all four (GET, POST, PUT and DELETE) functionalities. Another available option is Postman from Chrome. Figure 20 shows the User interface for this tool.

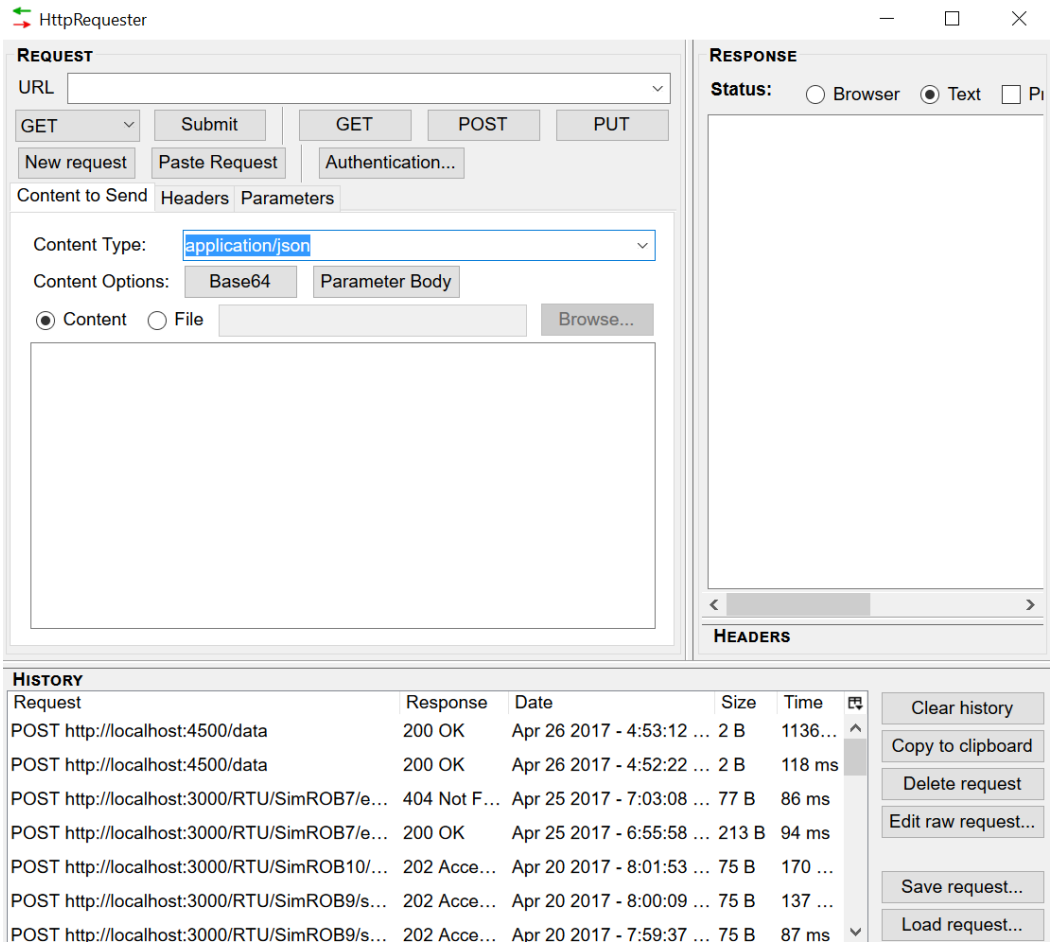

Figure 20. *Mozilla Http Requester.*

The resource URI is entered in the URL section. From the list, the type of operation (GET, POST etc.) is selected which is to be performed. Content is entered in the content field. In the case of GET request, content is left blank. Furthermore, type of content can be chosen from content type list. Response with the status code is represented in the left most column. This tool also stores the history of the requests for easy operation.

3.2.6 CanvasJs

CanvasJs is a library built on HTML5 and JavaScript to help in creating easy and attractive charts. It can run in many different environments such as iPhone, iPad, Android, Windows Phone, Microsoft Surface, Desktops, etc. [9]. Since it has a capability of running on many different environments, it makes it the best library to be used in our project without compromising any functionality. Other benefits of using it include high performance, elegant themes, support from developers and no dependency on any other library. Furthermore, it provides many different types of charts such as area chart, line chart, column chart, etc.

4. IMPLEMENTATION

This chapter describes the implementation of Arrowhead framework through the development of Energy Consumption service deployed on Arrowhead cloud server. The implementation is divided into two sections; prototype and deployment. In the prototype 'NodeArrowhead' is used to represent Arrowhead framework environment. 'NodeArrowhead' is available on GitHub and is an open source. It represents the Arrowhead implementation of core services. In the prototype three applications were developed; 'Light simulator', 'Meter' and 'Energy Consumption'. The functionality of each service is explained in section 4.1. Furthermore, Car heating and Coffee machine energy consumption data were also analysed to provide a real-time example of smart cities physical objects. In the deployment, the above-mentioned services were implemented on actual Arrowhead cloud service to facilitate the user to use energy consumption service analyses in order to monitor urban cities devices energy usage. Section 4.2 explains the deployment phase.

4.1 Prototype

Service Oriented Architecture is the foundation of Prototype implementation with the 'NodeArrowhead' used as the tool. Two services of 'NodeArrowhead' used are 'NodeServiceRegistry' and 'NodeOrchestrator'. 'NodeServiceRegistry' resembles the 'ServiceRegistry' core functionality of Arrowhead Framework. Whereas, 'NodeOrchestrator' provides full orchestration services on the local environment. A Prototype implementation is represented in the component diagram as shown in Figure 21.

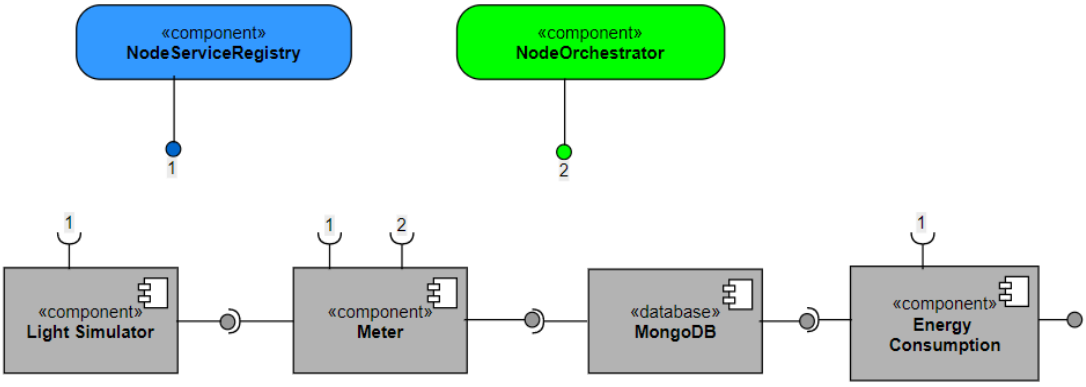

Figure 21. *UML Component Diagram representation of Prototype implementation.*

Light Simulator is an application, developed on ‘NodeJS’, and provides with a virtual representation of light sensor in the real world. It publishes its endpoints on the ‘NodeServiceRegistry’ so that it can be consumed by other services. Therefore, only required connection for this application is with the ‘NodeServiceRegistry’ as represented in Figure 21. Light Simulator does not consume any other service hence, it is unnecessary for this application to have orchestration rules. It runs on two modes; automatic and manual. In the automatic mode, the current luminosity data is returned to the user. This luminosity data is generated randomly in between 0%-100%. Whereas, in the manual mode the user can set the luminosity value through a POST command. Table 5 represents the RESTful interface for this service.

Table 5: *Light Simulator RESTful Interface.*

Mode	**Request URL**	**Method**	**Body**	**Response**

Automatic	/automatic/status	GET	-	{
				"name”: String,
				"type": String,
				"host": String,
				"port": Integer,
				"measurement”: Integer,
				"properties": JSON Array
				}

Manual	/manual/status	POST	value=
			[0...100]
				{
				"name”: String,
				"type": String,
				"host": String,
				"port": Integer,
				"measurement”: Integer,
				"properties": JSON Array
				}

Meter acts as a middleware application to fetch the readings from Light Simulator and store them in some database. We have used ‘MongoDB' in this implementation. Meter gets the orchestration rules form ‘NodeOrchestrator'. In the orchestration rules, a list of all the devices which meter must monitor is passed on. Meter further requests the ‘NodeServiceRegistry' to find the endpoints of the devices in order to connect with them. An individual request is sent to each device, in this case, ‘Light Simulator' to request the measurement every 1 minute and is stored in the MongoDB for further evaluation.

There are some functionalities provided in the meter to enable dynamic reconfigurability and autonomy. It reads orchestration rules every 10th cycle to confirm that the rules are

not changed. Moreover, if no orchestration rules are found the application waits until some orchestration rules are provided. In the case, if any of the device provided in the current orchestration rule unpublishes itself from ‘NodeServiceRegistry', then new orchestration rule is requested. This error handling is made possible by monitoring the Http Response status code. Similarly, if the device exists in the new orchestration but no response is received from the device then there exist the chances that the device endpoint has been changed. So, in this scenario, ‘NodeServiceRegistry' is checked again to update the endpoints.

All the communication is based on REST and the values are stored in JSON form in the database. The JSON data, stored in the database, is represented as below in Table 6.

Table 6: *Data stored to database.*

Type	**Data**
JSON	{"Name": String, "Date": String, "value": Integer, "timestamp": Integer }

.

The above-mentioned sequence for a single device is shown in Figure 22. Meter when starts, sends a request to NodeOrchestrator to get the device list. After it receives the list of devices from the NodeOrchestrator it requests NodeServiceRegistry to get the endpoints of the devices. It then further requests the devices to receive data and simultaneously sends a connection request to MongoDB. In the final step the modified data is stored in the database.

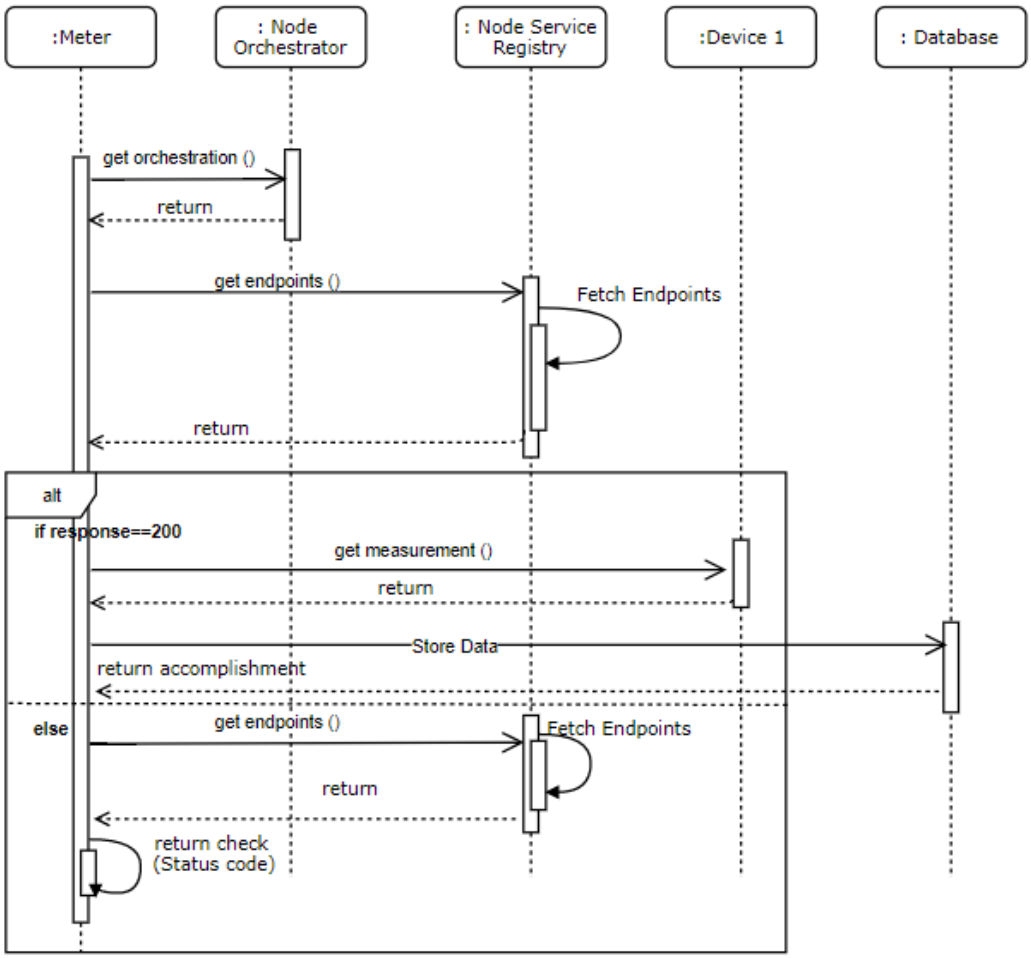

Figure 22. *Meter UML Sequence Diagram for prototype implementation.*

Energy Consumption service works in the same way when working in the prototype or in the Arrowhead environment. The basic difference is in the endpoints and in the publishing of this service endpoint to the NodeServiceRegistry or to the Arrowhead Registry respectively. Hence, this service is explained in the section 4.3 as a use case example.

4.2 Deployment

This section explains the deployment of energy consumption service on the Arrowhead framework. The data for the energy consumption evaluation is gathered from light simulator application and two real-time devices which are represented by car heating and coffee machine services. These services are also deployed on Arrowhead framework to facilitate meter in getting the desired endpoints from Arrowhead Registry. Deployment is carried out in the similar fashion as the prototype. In the first step the ‘Light Simulator’, ‘Coffee Machine’ and ‘Car Heating’ services are registered on the ‘Arrowhead Registry’ located in ‘BnearIT’ server. Following the meter is started which then requests the ‘Arrowhead Orchestrator’ for the orchestration rule. Finally, the data is stored on to the MongoDB or V3M database. V3M database was used by the project partners and is only accessible to them. The procedure for registering a service on the Arrowhead Registry, and

defining orchestration rules is explained in the tools subsection 3.2.1. This implementation is represented in a component diagram in Figure 23.

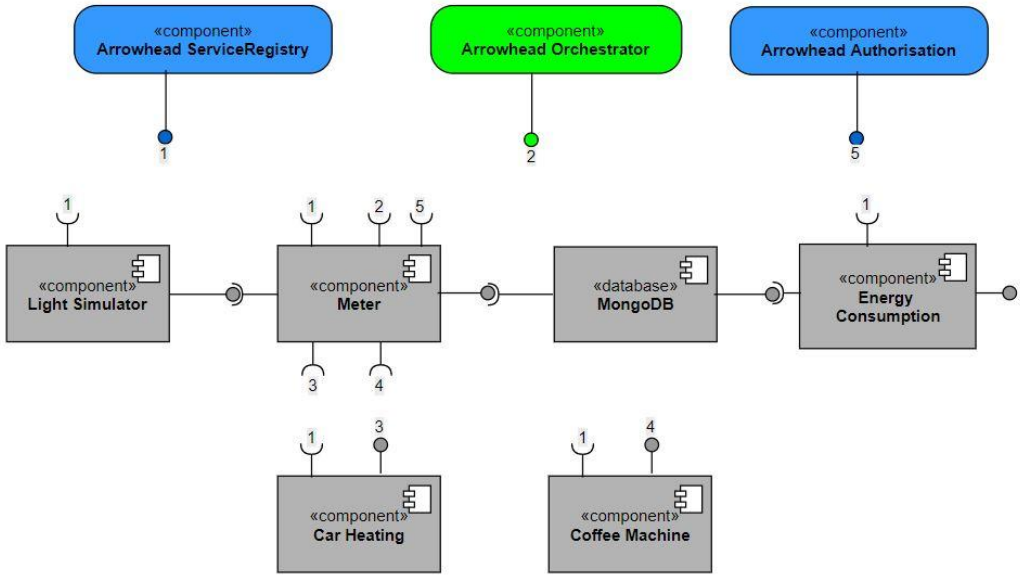

***Figure 23**. UML Component Diagram for Deployment of services on Arrowhead Framework.*

'Light Simulator' works in the similar fashion on Arrowhead Framework as mentioned in the Prototype section 4.1. The only difference is that instead of running on the local environment it is connected to the cloud through 'BnearIT' server and can be accessed anywhere around the world.

'Car Heating' and 'Coffee Machine' are two real-time devices located in the Tampere University of Technology. However, they have the same associations as the 'Light Simulator'. The requests sent to these devices are secured as they are only accessible to authorized users. Furthermore, data is requested once every day for the whole previous day from the Tampere University of Technology servers.

Meter has some modifications from the prototype implementation, though the working principle is same. This modified working principle is shown in Figure 24. It needs to connect to 'Arrowhead Orchestrator', which provides it with the set of rules. Device lists are provided in the orchestration rule. This orchestration rule is provided in XML format which must be converted into JSON and is restricted to only one station. These restrictions can be edited in the authorization tab of management tool, explained in earlier in Arrowhead Framework in section 3.2.1.

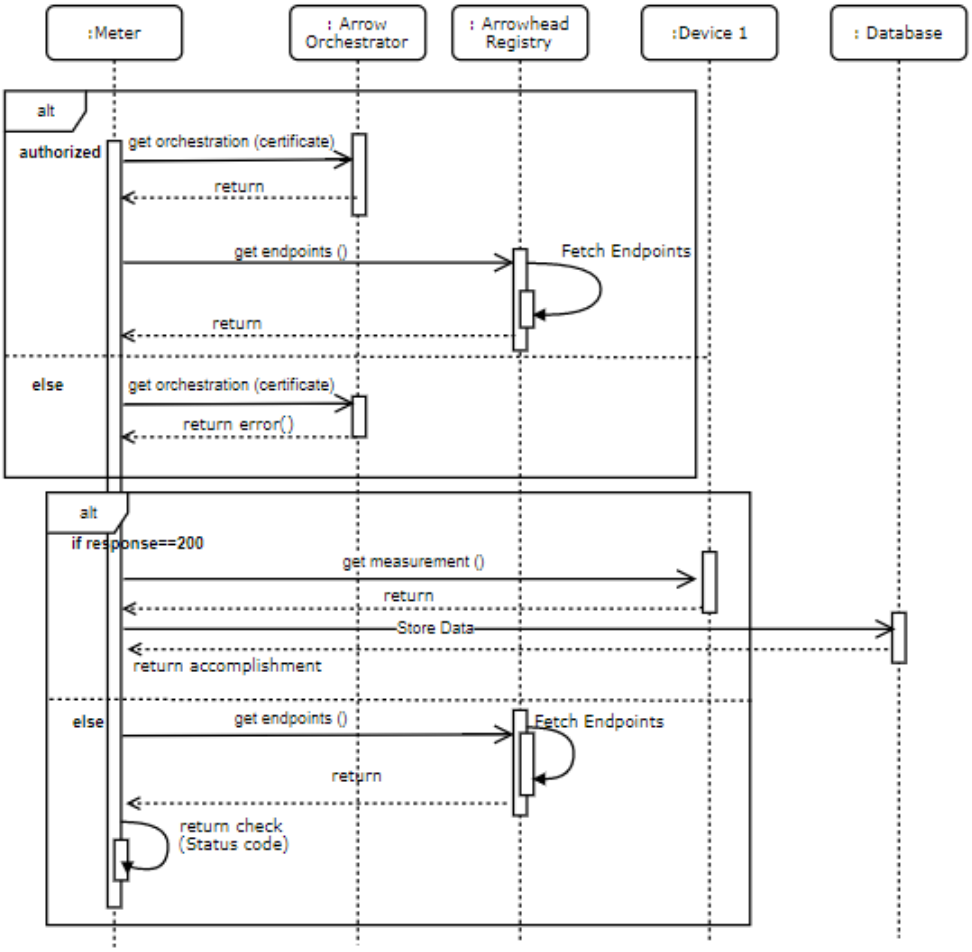

Figure 24. *UML Meter Sequence Diagram for Deployment.*

Every Arrowhead station has a designated certificate which was provided by the Arrowhead. These certificates are needed to be made compatible with JavaScript and must be included in the options field of the request when retrieving orchestration rules from Arrowhead Orchestrator. The certificate identifies the authorization of the request. If the requestor is authorized, then the orchestration rules are returned. otherwise an error message is returned, and the request is denied access to the orchestrator. Table 7 shows a successful reply from the Arrowhead Orchestrator.

Table 7: *Arrowhead Orchestrator Response.*

Function	**Method**	**Response**

Retrieve Orchestration Rule	GET	STATUS: 200
		{
		"rule":
		[Device 1_Type,
		Device 2_Type]
		}

4.3 Case Example of managing IoT service for Energy Consumption

Energy Consumption Service is built on the set of rules defined by the Arrowhead in order to realize this service. These rules were defined in Arrowhead design description document. The important set of rules are shown in Table 8.

Table 8: *Energy Consumption Service operation rules.*

Method:	/energyconsumption/{id}/{start}/{stop}
The method energy consumption returns the energy consumption value. The input parameters are used to specify different subsets. Parameter in curly brackets "{xx} "means that it can be left out but then all parameters after that must also be left out.	
Request:	The method that request to the web service REST has the following parameters: - **id**: the object id that denotes a specific energy consumption measurement object. This parameter should be one of the listed identities in the /object/ response with the exception of the id "ALL" that should return the combined measurement during a time interval. - **Start**: the starting time of the energy consumption measurement in UTC. - **Stop**: the ending time of the energy consumption measurement in UTC. http://[IP]: [PORT]/[PROJECT_NAME_WS]/energyconsumption/{id}/{start}/{stop} If {stop} is left out the requested energy consumption value is from start time to current. If {start} (and {stop}) is left out the returned the requested energy consumption value is the current value (cumulative energy meter value). If {id} (and {start} and {stop}) is left out a list of all available current energy consumption values (cumulative energy meter values) is requested. One measurement for each object(id). Example: GET http://[IP]: [PORT]/[PROJECT_NAME_WS]/energyconsumption/123/1461357723655/1461357725443
Response:	The response of this service is a plain value representing the consumed energy during the time interval. A negative value indicates generation of energy. If a system is unable to match the requested start and stop times for an object, the response shall contain energy consumption value for a period with a length and mean time as close as possible to the requested period. The Start and Stop parameters shall always state the period that corresponds to the provided Measurement value. - **Id:** A unique id (within the scope of the service)

	- **Type:** Enumeration of measurement object types. - **Name:** User readable name of the object - **Measurement:** An energy consumption measurement value of the object in kWh. The type is a floating-point value. - **Start**: the actual starting time of the provided energy consumption measurement in UTC. - **Stop**: the actual ending time of the provided energy consumption measurement in UTC. Example of the structure in JSON with only one object: {“Object1”: {“Id”: “123”, “Type”: “BUILDING”, “Name”: “Sydney Opera House”, “Measurement”: 1.56, “Start”: 1461357723600,” Stop”: 1461357725600}} Example of the structure in JSON with a list of two objects: { “Object List” : [“Object1”: { “Id” : “123”, “Type”: “BUILDING”, “Name”: “Sydney Opera House”, “Measurement” : 1.56, “Start”: 1461357723600,”Stop”: 1461357725600}, “Object2”: {“Id” : “A2F”, “Type”: “SMALL ELECTRIC DEVICE”, “Name”: “My Espresso Machine” , “Measurement” : 0.03, “Start”: 1461357723655,”Stop”: 1461357725443}]}

Energy Consumption Service follows all the above-mentioned rules and functionalities. There are few extra functionalities embedded in this service, such as the ability to display cumulative and non-cumulative readings on a single web application user interface. The service reads all the required data from MongoDB. Furthermore, energy consumption service is not restricted to only one device. This states that it can read multiple numbers of devices which are defined as configurations in the configuration file. This file is read by the service on every successful startup. Table 9 shows how this service can be seen in Arrowhead Management Tool Service Registry.

Table 9: *Arrowhead Management Tool.*

Service Instance	**Service Type**	**Host**	**Port**	**Properties**
energyconsumption._energy-ws-http._tcp.srv.test.bnearit.arrowhead.eu.	energy-ws-http	Arrowhead Channel	Port number	{path=/energyconsumption, version=1.0}

User Interface was developed to make visualization of energy consumption more efficient. Figure 25 represents the activity diagram for this interface. As shown in the activity diagram, once the web interface is requested it waits until the device list is requested. Device list shows all the available devices that energy consumption can monitor. These devices appear dynamically on the interface. The user then selects the devices that are to be monitored through checkboxes and the start time and end time is provided. Time fields

can be skipped as according to the instructions of energy consumption, explained in Table 8. Finally, the user must mention that in which mode the interface should response the results. There are two modes of operation. The cumulative mode takes all the values from the start time and adds up the values until the mentioned end time to represent a cumulative graph. If the end time is not mentioned, then the last possible date value stored in the database is considered. Likewise, in the non-cumulative mode, each value entries are displayed as individual bars.

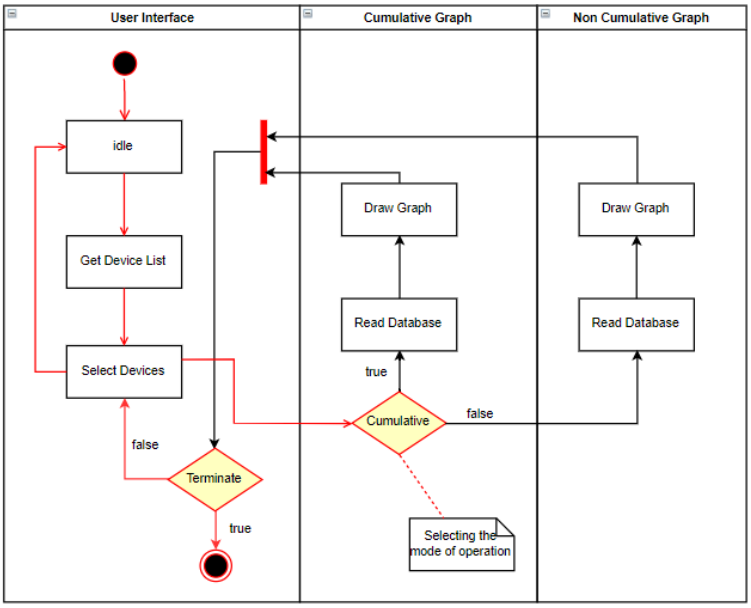

Figure 25. *User Interface UML Activity Diagram.*

User interface layout is shown in Figure 26. This layout represents the general idea of the view which is shown to the end user.

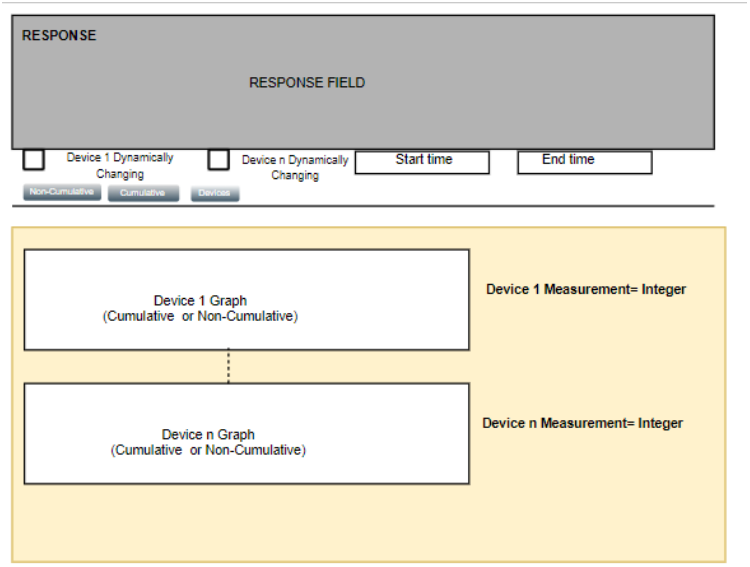

Figure 26. *General Overview of User Interface.*

5. RESULTS AND ANALYSIS

In this chapter, the results of the deployment are discussed. Energy Consumption service provides with the good graphical representation of the data. This data can be analysed to deduce the energy utilisation. The result of the implementation is the successful representation of managing IoT systems. Results of this implementation can be divided into three stages. The first one is the ability of the devices to register themselves on the Arrowhead Registry. The next successful stage is creating the managing system that fetches the orchestration rules, discovers the devices and stores the data into a database. Lastly, representing the consumption data on a user interface that is accessible irrespective of the location of the user. IoT system in this implementation consists of two real-time devices connecting to the Arrowhead framework. The list of devices that are registered and are in the orchestration rule can be displayed by using the devices tab. The results of this query are displayed in Figure 27.

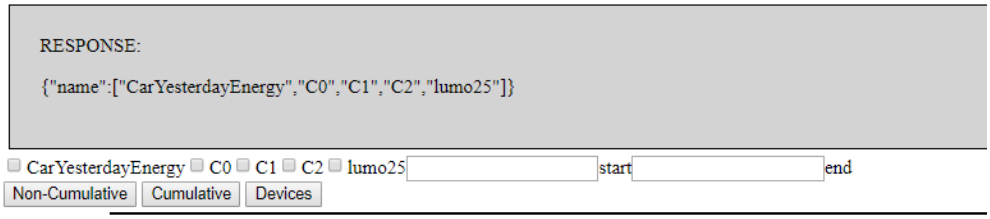

Figure 27. *Results of Device Query.*

The next results are the energy consumption analyses graphs. These consumption results can vary based on the method of the request in the implementation. Figure 28 shows the results of C0 (Coffee Machine) and C1 (Car Heating) when the desired outcome was the cumulative graph of the energy used.

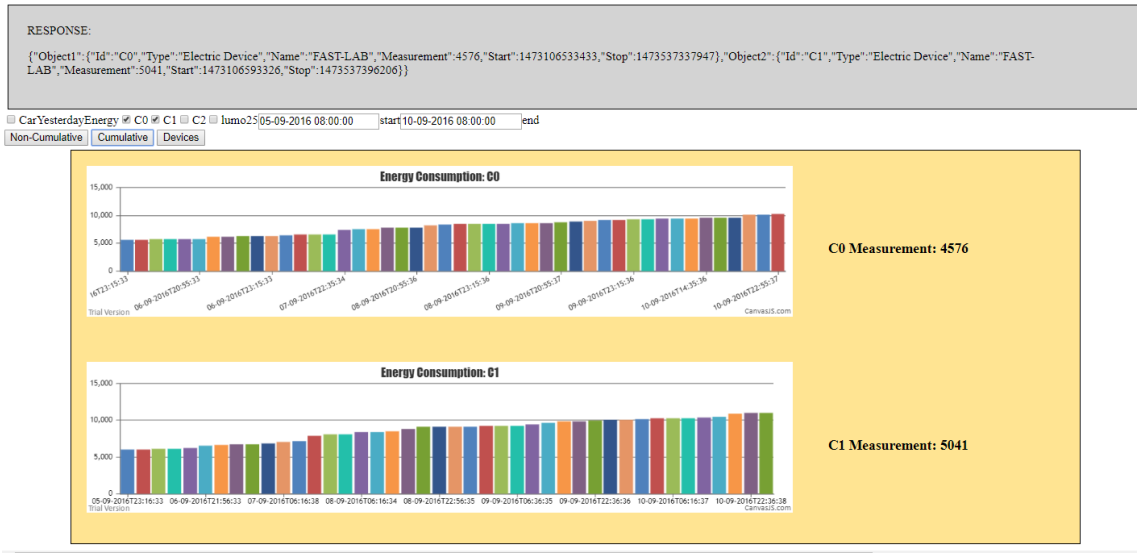

Figure 28. *Cumulative Graph of Energy Consumption.*

The figure above shows the consumed energy for 5 days between (05.09.2016 – 10.09.2016). The value as shown in the figure is 4576 for the coffee machine. The unit for the calculation is Watt. Whereas, it is 5.041 KW for Car heating. Furthermore, JSON string is returned in the response field. This response follows the same principles as described by the Arrowhead Energy Consumption design description document discussed in section 4.3.

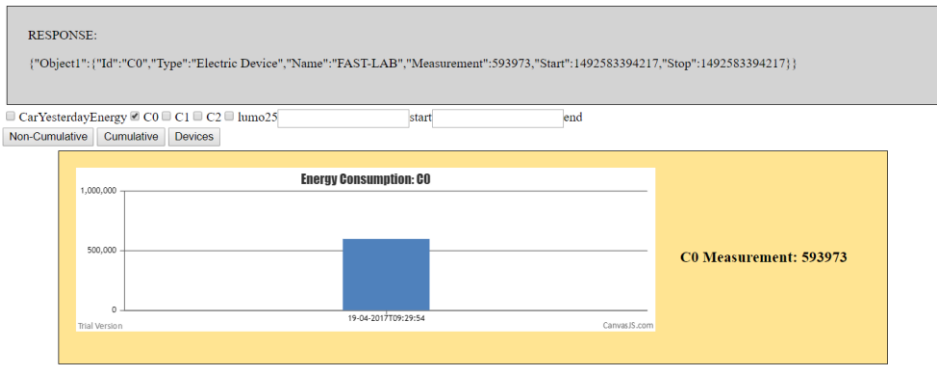

Figure 29. *Current Reading of Energy Consumption.*

If the start and end time are not provided, then the graph will represent the only current value that is the cumulative reading at the current time. This is represented for C0 in Figure 29.

If the desired results are non-cumulative data than the user can select non-cumulative mode. In that case, all the values will provide energy consumption data for each day. This is represented in Figure 30 for the same periods as cumulative graphs for better understanding.

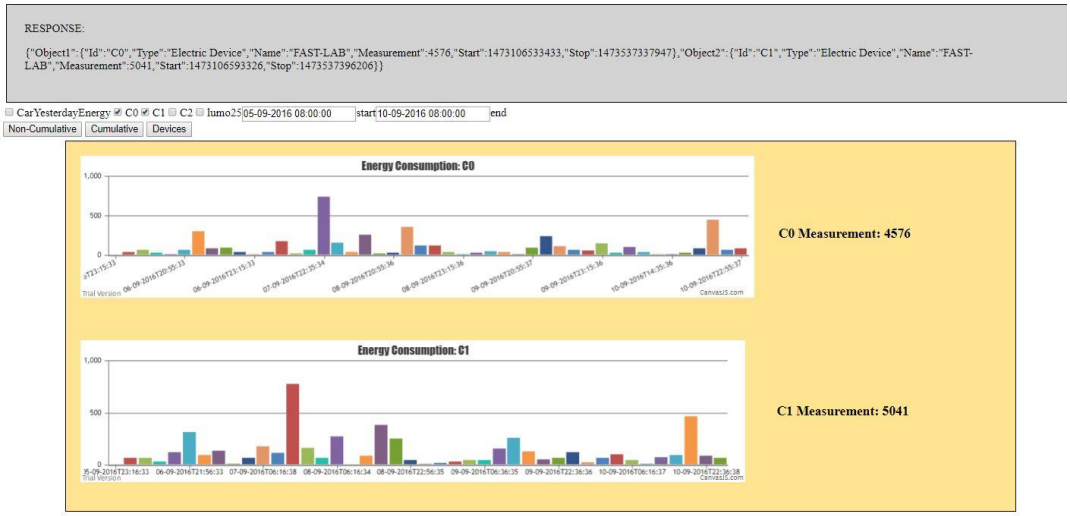

Figure 30. *Non-Cumulative Energy Consumption Graph.*

The measurement value is always calculated as the total energy consumed in the provided time span. Therefore, it is same as the cumulative value.

These graphs give a useful comparison of the energy consumption of each and every device that is connected to the IoT system. Furthermore, new devices can be added to the IoT system for monitoring energy consumption. Similarly, previous devices can easily be removed.

6. CONCLUSION AND FUTURE WORK

This master thesis focuses on the IoT systems in a Smart City. Smart City aims to provide efficient means of utilizing city resources and provides means of reducing energy consumption. Furthermore, it reduces administration costs and increases the quality of life for its inhabitants. These facilities are possible due to an evolution of information and communication technologies and the concept introduced by the Internet of Things. Many different types of research have been carried out to demonstrate feasible architecture styles for implementing smart cities. Most common among them are Event Driven architecture, Multi-Tier architecture and Service Oriented Architecture.

Arrowhead Framework is used in this thesis to develop applications in this thesis. This framework is provided by Arrowhead Project and is based on SOA. Hence, all the principles of SOA, such as interoperability, autonomy, loose coupling, discoverability, and composability is followed. There are three core systems that are offered by Arrowhead; Service Registry, Orchestration and Authorisation System. These systems enable developers to develop and then deploy the services that are compatible with Arrowhead Project. The scope of Arrowhead is divided into five different domains. These domains include Smart Building and Infrastructure, Energy Production, Virtual Market of Energy, Production, and Electro-Mobility.

Arrowhead Framework was used to develop prototype implementation and then it was deployed to Arrowhead. Light Simulator service demonstrates how different lights located at different locations can connect to the internet and then send information to the user. Furthermore, the intensity of the simulated lights can be controlled. This service was used in prototype implementation and was used as a demonstration. Another service 'Meter' was developed to act as middleware management service. This service was responsible for getting Light Simulator, Coffee Machine, and Car Heating. The service first reads the orchestration rules and then connect to Registry to get the endpoints of the devices. It then converts the data into preferred format and stores it into a database. Finally, a pilot service was developed for Arrowhead Project ‘Energy Consumption Service'. This service calculates the energy consumed by the above-mentioned devices and displays a graphical layout for the user.

These applications demonstrate how Arrowhead Project can be used to monitor, control and evaluate energy consumption in an Urban City using Arrowhead Framework and Web Services.

6.1 Future Work

The applications developed in this thesis can be upgraded by working on the hardware. All the readings obtained for Light Simulator can be converted to real-time data by integrating proper hardware with the light sensors. Light Simulator service is built generic in nature so that it is possible in future work to expand it to other devices such as surveillance cameras, automated doors, parking places etc. Since Meter is not restricted to the type or number of devices it can monitor, there is no major compatibility problems associated with different types of physical devices. Hence, it is possible to integrate more objects to this service. Raspberry Pi or some other powerful controller can be used for running meter service to provide a commercial solution. Energy Consumption service can also be improved by providing more functionalities such as runtime energy distribution control system to improve energy efficiency. Arrowhead Project can also be further developed especially, in the security and authorization by improving certificates distribution. There shall be better support available for the development of RESTful applications. Orchestration core service does not show the previous orchestration rules. This shall also be addressed further.

REFERENCES

[1]. F. Blomsted, Arrowhead_IDD_Service Discovery REST_WS-JSON-SPSDTR version 1.1, unpublished project material, 27th April 2016.

[2]. F. Blomsted, Arrowhead_IDD_Orchestraton Store REST_WS-TLS-XML-SPORCH version 1.3, unpublished project material, 10th May 2016.

[3] F. Blomsted, Arrowhead_IDD_ Authorisaton Control REST_WS-TLS-XML-SPAUTH version 1.3, unpublished project material, 10th May 2016.

[4] J.Delsing, Arrowhead Framework Cookbook version 1.6.1, unpublished project material, 2015, 19 p. [Online]. Available: https://forge.soa4d.org/plugins/scmgit/cgi-bin/gitweb.cgi?p=arrowhead-f/arrowhead-f.git;a=tree;f=4_How+to+implement+application+systems/1_Cook+book;h=41b05310a39bdc571b9c2d945c3c5899fd53229c;hb=HEAD

[5] N. Foundation, "About | Node.js", Nodejs.org, 2017. [Online]. Available: https://nodejs.org/en/about/. [Accessed: 30- May- 2017].

[6] N. Foundation, "Index | Node.js v7.10.0 Documentation", *Nodejs.org*, 2017. [Online]. Available: https://nodejs.org/api/. [Accessed: 30- May- 2017].

[7] MongoDB and MySQL Compared", *MongoDB*, 2017. [Online]. Available: https://www.mongodb.com/compare/mongodb-mysql . [Accessed: 30- May- 2017].

[8] j. jquery.org, "jQuery", *Jquery.com*, 2017. [Online]. Available: http://jquery.com/ . [Accessed: 30- May- 2017].

[9] "Introduction to CanvasJS JavaScript Charts | CanvasJS", *CanvasJS*, 2017. [Online]. Available: http://canvasjs.com/docs/charts/intro/ . [Accessed: 30- May- 2017].

[10] Jin, J., Gubbi, J., Marusic, S. and Palaniswami, M. (2014). An Information Framework for Creating a Smart City Through Internet of Things. *IEEE Internet of Things Journal*, 1(2), pp.112-121.

[11] Ferreira, J.C. & Afonso, J.L. (2011). Mobi-System: A personal travel assistance for electrical vehicles in smart cities, pp. 1653-1658.

[12] L. Anthopoulos, P. Fitsilis, Exploring Architectural and Organizational Features in Smart Cities, In 16th International Conference on Advanced Communication

[13] Nam, T. & Pardo, T.A. (2011). Conceptualizing smart city with dimensions of technology, people, and institutions, pp. 282-291

[14] Source: "International Energy Outlook 2010 - Highlights," U.S. Energy Information Administration press release, May 25, 2010 (http://www.eia.doe.gov/oiaf/ieo/highlights.html).

[15] L. Filipponi, A. Vitaletti, L. Landi, V. Memeo, G. Laura and P. Pucci,"Smart City: An Event Driven Architecture for Monitoring Public Spaces with Heterogeneous Sensors", in *Fourth International Conference on Sensor Technologies and Applications*, pp. 281-286, 2010

[16] 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.

[17] 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.

[18] Ji-chen and G. Ming, "Enterprise Service Bus and an Open Source Implementation", in *International Conference on Management Science and Engineering*, 2006.

[19] Forrester, "Getting Clever About Smart Cities: New Opportunities Require New Business Models", 2010.

[20] D. Sprott and L. Wilkes, "Understanding Service-Oriented Architecture", *Msdn.microsoft.com*, 2004. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa480021.aspx. [Accessed: 10- Aug- 2017].

[21] Bianco. Philip, Kotermanski. Rick, and Merson. Paulo, "Evaluating a Service-Oriented Architecture," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2007-TR-015, 2007. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8443

[22] E. Christensen, F. Cubera, G. Meredith and S. Weerawarana, "Web Service Definition Language (WSDL)", W3.org, 2001. [Online]. Available: http://www.w3.org/TR/wsdl. [Accessed: 10- Aug- 2017].

[23] M. Rouse, "SOAP (Simple Object Access Protocol)", Managing architectures, 2014.

[24]"XML Introduction", *W3schools.com*, 2017. [Online]. Available: https://www.w3schools.com/xml/xml_whatis.asp. [Accessed: 22- Aug- 2017].

[25] Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. R. Fielding, Ed., J. Reschke, Ed. June 2014. (Format: TXT=205947 bytes) (Obsoletes RFC2145, RFC2616) (Updates RFC2817, RFC2818) (Status: PROPOSED STANDARD) (DOI: 10.17487/RFC7230)

[26] A. Abuhussein, H. Bedi and S. Shiva, "Exploring Security and Privacy Risks of SoA Solutions Deployed on the Cloud", in *2012 International Conference for Internet Technology and Secured Transactions*, London, UK, 2012, pp. 1-5.

[27] M. Rouse, "XML (Extensible Markup Language)", Managing Architectures, 2014. [Online]. Available: http://searchmicroservices.techtarget.com/definition/XML-Extensible-Markup-Language. [Accessed: 22- Aug- 2017].

[28] "Understanding Service-Oriented Architecture", 2017. [Online]. Available: # https://msdn.microsoft.com/en-us/library/aa480021.aspx. [Accessed: 27- Aug- 2017].

[29] D. Barry and D. Dick, *Web services, service-oriented architectures, and cloud computing*, 1st ed. Waltham, MA: Morgan Kaufmann, 2013, pp. 23-26.

[30] Y. Baghdadi, "A framework to select an approach for Web services and SOA development", in *International Conference on Innovations in Information Technology (IIT)*, Abu Dhabi, United Arab Emirates, 2012, p. 279.

[31] M. De Saulles, The internet of things & business. Routledge, 2017, pp. 1-2.

[32] IT Glossary. [online]. Available at: www.gartner.com/it-glossary/internet-of-things/ .

[33] D. Georgakopoulos, P. Jayaraman, M. Zhang and R. Ranjan, "Discovery-Driven Service Oriented IoT Architecture", in IEEE Conference on Collaboration and Internet Computing (CIC), Hangzhou, China, 2015.

[34] S. Li, L. Xu and S. Zhao, "The internet of things: a survey", *Information Systems Frontiers*, vol. 17, no. 2, pp. 243-259, 2014.

[35] R. Fielding and R. Taylor, "Principled design of the modern Web architecture", *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115-150, 2002.

[36] Calvo, J. Gil-García, I. Recio, A. López and J. Quesada, "Building IoT Applications with Raspberry Pi and Low Power IQRF Communication Modules", *Electronics*, vol. 5, no. 3, p. 54, 2016.

[37] A. NEVAVUORI, “DEVELOPMENT OF AN INTELLIGENT DATA COLLECTION INSTRUMENT FOR MOBILE EQUIPMENT,” 2015.

[38] N. Javed, “IOT NODE EMULATION AND MANAGEMENT TESTBED,” 2013.

[39] A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, "Internet of Things for Smart Cities", *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22-32, 2014.

[40] "Arrowhead — Ahead of the future", *Arrowhead — Ahead of the future*, 2017. [Online]. Available: http://arrowhead.eu. [Accessed: 22- Nov- 2017].