CISTER

Research Centre in
Real-Time & Embedded
Computing Systems

BEng Thesis

Quality of Service for High Performance IoT Systems

Renato Ayres

Paulo Barbosa

CISTER-TR-161203

2016/10/31

Quality of Service for High Performance IoT Systems

Renato Ayres, Paulo Barbosa

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: 1120681@isep.ipp.pt, 1130648@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

The fourth industrial generation brought both solutions as challenges. It allowed greater efficiency and effectiveness in manufacturing, reducing both costs and wastes. However, it consists in the deployment of innumerable devices for data collection and control processes. This brings challenges such as interoperability between all these heterogeneous systems.

Thus, a group of partners, supported by the European Union, proposed a solution, the Arrowhead Framework. Its aim is to create a framework with a service-oriented architecture (SOA) enabling an abstract collaboration between all these different devices. While in development, the framework does not provide Quality of Service (QoS), which prevents its use in more demanding networks. This limitation was the central problem solved in this project.

This project focus on developing an architecture that provides QoS support in Arrowhead compliant systems. Here the main challenges addressed are the following: developing an architecture capable of working with different communication protocols and technologies; develop an architecture capable of working with an unlimited number of QoS requirements.

During the entire project, its development process consisted in two main iterations: the first was regarding the development of an architecture; the second consisted in the development of a pilot project based on the FTT-SE protocol that could test the architecture developed in the first iteration.

At last, the final product consists in two systems, one for QoS configuration and other for monitoring. These two systems are independent of each other. Regarding QoS requirements, only delay and bandwidth were implemented.

© CISTER Research Center
www.cister.isep.ipp.pt

Quality of Service for High Performance IoT Systems

LEI-DEI

2015/2016

1120681 - Renato Ayres de Sousa

1130648 - Paulo Miguel Santos Barbosa

ISEP Supervisors:

- Luis Lino Ferreira
- Paulo Baltarejo de Sousa

External Supervisor:

- Michele Albano

Quality of Service for high performance IoT systems

LEI-DEI

2015/2016

1120681 - Renato Ayres de Sousa

1130648 - Paulo Miguel Santos Barbosa

Degree in Informatics Engineering

October 2016

ISEP Supervisors:

- Luis Lino Ferreira
- Paulo Baltarejo de Sousa

External Supervisor:

- Michele Albano

«The only place **success** comes before work is in the dictionary. »

***Vince Lombardi**, American Football Coach.*

Acknowledgments

First, we want to thank the DEI department for the opportunity to attend LEI. These last few years have been very rewarding, full of challenges and learning. What we learned in ISEP, not only intellectually but as a personal level, made us more mature persons and more prepared for the labour market.

It should be noted that this internship would not have been possible without CISTER, especially professor Luis Lino Ferreira who proposed the realization of this project under PESTI. For him, Paulo Baltarejo De Sousa and Michele Albano a big thank you, they were always available and were a key support throughout the project.

Without forgetting all the other colleagues, we are very grateful to Bruno Silva for his cooperation and help in every stages of the internship, including the most difficult. Many thanks to Roberto Duarte, who despite also attending PESTI, helped us in the integration phase carried out in the final part of the project, and he was always available to help.

Paulo Barbosa/ Renato Ayres

I thank my family and friends that during these three strenuous years supported and helped me.

Paulo Barbosa

I would like to thank all my friends and colleagues for always helping when I asked. Thank you to my brother and two sisters for all their patience. A very special thank you to my mother for everything she does and sacrifices for the four of us without even asking for a simple thanks.

Renato Ayres

Abstract

The fourth industrial generation brought both solutions as challenges. It allowed greater efficiency and effectiveness in manufacturing, reducing both costs and wastes. However, it consists in the deployment of innumerable devices for data collection and control processes. This brings challenges such as interoperability between all these heterogeneous systems.

Thus, a group of partners, supported by the European Union, proposed a solution, the Arrowhead Framework. Its aim is to create a framework with a service-oriented architecture (SOA) enabling an abstract collaboration between all these different devices. While in development, the framework does not provide Quality of Service (QoS), which prevents its use in more demanding networks. This limitation was the central problem solved in this project.

This project focus on developing an architecture that provides QoS support in Arrowhead compliant systems. Here the main challenges addressed are the following: developing an architecture capable of working with different communication protocols and technologies; develop an architecture capable of working with an unlimited number of QoS requirements.

During the entire project, its development process consisted in two main iterations: the first was regarding the development of an architecture; the second consisted in the development of a pilot project based on the FTT-SE protocol that could test the architecture developed in the first iteration.

At last, the final product consists in two systems, one for QoS configuration and other for monitoring. These two systems are independent of each other. Regarding QoS requirements, only delay and bandwidth were implemented.

Keywords (Theme): Industry 4.0, Internet of Things, Quality of Service.

Keywords (Technologies): C, Flexible Time Triggered-Switched Ethernet, Java, MongoDB, MySQL, Representational State Transfer, Service-oriented Architecture.

Resumo

A quarta geração industrial trouxe tanto de soluções como desafios. Permitiu uma maior eficiência e eficácia na produção, reduzindo tanto custos como desperdícios. Contudo ela é construída na configuração de inúmeros dispositivos, para a recolha de dados e até para controlo de processos. Isto traz desafios, como a interoperabilidade entre todos estes sistemas heterogéneos.

O projeto Arrowhead é solução proposta por um conjunto de parceiros, financiados pela União Europeia. O seu objetivo é criar uma framework com uma arquitetura orientada a serviços (SOA) capacitando uma colaboração abstrata entre todos estes diferentes dispositivos. Já num estado avançado de desenvolvimento, a framework não providencia Qualidade de Serviço (QoS). Esta limitação foi o problema central solucionado neste projeto.

O presente projeto foca-se no desenvolvimento de uma arquitetura que suporte QoS em sistemas compatíveis com Arrowhead. Aqui os principais desafios abordados são: desenvolver uma arquitetura capaz de trabalhar com diferentes protocolos de comunicação e tecnologias; desenvolver uma arquitetura capaz de trabalhar com um número ilimitado de requisitos de QoS.

O processo de desenvolvimento durante o projeto dividiu-se em duas grandes iterações: a primeira consistiu no desenvolvimento da arquitetura; a segunda consistiu num desenvolvimento de um projeto piloto baseado no protocolo FTT-SE que pudesse testar a arquitetura desenvolvida na primeira iteração.

O produto final consiste em dois sistemas, um de configuração e outro de monitorização, independentes entre si. Os parâmetros de QoS implementados foram *delay* e largura de banda.

Palavras-chave (Tema): Indústria 4.0, Internet of Things, Qualidade de Serviço.

Palavras-chave (Tecnologias): Arquitetura Orientada a Serviços, C, Flexible Time Triggered-Switched Ethernet, Java, MongoDB, MySQL, REST.

Notation and Glossary

This Section shows all the concepts, symbols and acronyms on the body of this document.

Notation	Meaning
CISTER	Research Centre in Real-Time &Embedded Computing Systems
CPS	Cyber-Physical Systems
DCS	Distributed Control System
DEI	Departamento de Informática
ERP	Enterprise Resource Planning
FTT	Flexible Time-Triggered
FTT-SE	Flexible Time-Triggered on Switched Ethernet
IoT	Internet of Things
ISEP	Instituto Superior de Engenharia do Porto
JAX-RS	Java API for RESTful Web Services
LEI	Licenciatura em Engenharia Informática
MES	Manufacturing Execution System
QoS	Quality of Service
RDBMS	Relational Database Management System
REST	Representational State Transfer
RESTful	Characteristic of a device/system that conforms the constrains of REST.
RUP	Rational Unified Process
SCADA	Supervisory control and data acquisition
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SoS	System of Systems
TCP	Transmission Control Protocol
TCP/IP	Internet Protocol Suite
UML	Unified Modeling Language

Contents

Index of Figures

Index of Tables

Chapter 1. Introduction

1 Introduction

Introduction chapter begins by presenting the project context and the motivations that made the work possible. After, it is made a description of the organization responsible for the project, giving an insight of its work area and current research topics. The contributions of the work for other Arrowhead partners and for the students who developed it are also explained. The chapter ends with a report guide -summarizing each Chapter.

1.1 Framework

In the third and last year of the Informatics' Engineering Bachelor Degree, each student must attend an internship to apply the skills and the knowledge gained throughout the course in a real working environment. The internship is done in the context of the curricular unit Projeto/Estágio (PESTI) and has a minimum duration of one semester.

The internship was carried out in cooperation with the Research Centre in Real-Time & Embedded Computing Systems (CISTER), and focused on two research areas that the centre had already been working on: Cooperative automation; Internet of Things (IoT). The main purpose of the project was to design and implement a generic architecture that could guarantee Quality of Service (QoS) for IoT applications. The solution was implemented in an already developed framework, Arrowhead.

The Arrowhead Framework [1] is an European project constituted by more than 70 partners and has the goal to meet the following automation requirements: real time properties; security and safety; engineering of automation functionalities. Its vision is to enable, between network embedded devices, collaborative automation allowing interoperability of services provided by any device.

A pilot project was also developed to test and evaluate the proposed QoS architecture, and was deployed in a Flexible Time Triggered communication protocol.

1.2 Project Presentation

Devices for the IoT allow the development of applications that interact with embedded devices in a physical environment. It can potentially be everything that can interact with, for instance a power plug or medical gear, as long as they remain accessible through wired or wireless networks. After the initial phase of suitable hardware development, all these applications have an increasingly strong component of computer systems, namely their programming, configuration, monitoring and control. Some of these applications can only work satisfactorily for their users if certain QoS requirements are met. These requirements often include parameters like communications time delay, bandwidth requirements, reliability, etc.

The Arrowhead covers these problems at the global scale for five application areas, production, smart buildings, electro-mobility and virtual market of energy. Arrowhead architecture consists

in three core systems that support backbone operations. The first is the Orchestration System that coordinates the service requests done by consumers. The second is the Authorisation System that is responsible for controlling which service a consumer can consume and the third is the Service Registry System that manages all the services registration and discovering. The QoS functionality is integrated in the Orchestrator System, acting as supportive system.

The functionality of QoS in the Arrowhead Framework is of considerable importance to enable automation applications, and in fact many industrial scenarios require either a short end-to-end delay, or communication robustness. Therefore, the purpose of this project is to design and implement QoS mechanisms for Service Oriented Architecture (SOA), based on Representational State Transfer (REST). Such mechanisms are provided by systems that should perform three major processes:

- verify if a service request is feasible in the current state of the network;
- configure needed network actives or alter the operating parameters of the distributed applications.
- monitor, in real time, the status of the communications between applications to make sure that Service-level Agreements (SLA) are not broken.

1.3 Organization Overview/Presentation

CISTER (Research Centre in Real-Time and Embedded Computing Systems) is a Research Unit based at the School of Engineering (ISEP) of the Polytechnic Institute of Porto (IPP), Portugal created in 1997.

Since it´s creation, CISTER has grown to become one of the leading European research units. It has contributed and keeps contributing with seminal research works in a number of subjects:

- real-time communication networks and protocols;
- wireless sensor networks (WSN); cyber-physical systems (CPS);
- real-time programming paradigms and operating systems;
- distributed embedded systems;
- cooperative computing and QoS-aware applications;
- scheduling and schedulability analysis (including multiprocessor systems).

CISTER was, in 2004 and 2007 awarded the classification of Excellent in the FCT evaluations and is currently one of the most prominent research unit of ISEP. It has a strong and solid international reputation, built upon a robust record of accomplishment of publications and a continuous presence on program and organizing committees of international top conferences [2].

Regarding its research topics, CISTER has well-established roots in the real-time and embedded systems (RTES) scientific community. From the viewpoint of strategic vision, the unit is

consistently able to identify and contribute to emerging topics in the area, and continues to do so.

Following the strong tradition of developing foundational work in relevant topics, such as multiprocessor scheduling and wireless sensor networks, the unit fosters activities aligned with the international agenda. Along with their ubiquitous deployment, embedded platforms are becoming more complex as they grow more powerful, owing to their obligation to address ever-increasing demanding requirements. Their complexity and resource-awareness brings further challenges to the development of reliable and efficient systems, such as resource (e.g., CPU, memory, power) management, novel operating systems and virtual machines, timing analyses, etc.

In particular, with a strategic vision for the future, CISTER is working in emerging topics such as [3]:

1. programming paradigms for the next generation of computing systems;
2. modelling and analysing temporal behaviour;
3. handling the requirements of mixed-criticalities;
4. efficient management of energy resources;
5. networking communication protocols that have timeliness as a structuring concern while providing the required mobility, ubiquity, and pervasiveness;
6. systems theory that combines “physical concerns” (control systems, signal processing, etc.) and “computational concerns” (complexity, schedulability, computability, etc.);
7. increasing demands for quality of service and service level agreements at all layers of increasingly complex systems.

1.4 Project Charter

At the starting point of a project, some development teams write a project charter to clarify the major goals, tasks and each one´s roles. The charter is constituted by five elements:

- “statement”: is the problem that motivated the project realization;
- “scope”: is regarding the principal tasks of the project to accomplish the project goals;
- “goals”: are the functionalities that the project must accomplish;
- “business case”: is the principal reason of the project;
- “team members”: are the people that contributed to the project.

Since a team of developers participated in the project, each one with a different roles and tasks, it was considered beneficial to write a project charter to avoid misperceptions about all the project characteristics and consequently promote a fluid accomplishment of all future tasks.

Table 1 - Project Charter.

Statement	Arrowhead does not guarantee communication robustness.	
Scope	Develop a system to provide QoS and real time monitoring for Arrowhead, and in the end test it in a pilot project.	
Goals	Verify the feasibility of QoS objectives; Setup network actives and devices to ensure the QoS; Monitor, in real time, the performance of services; Detect if a QoS parameter is not being guaranteed anymore, or any other critical event.	
Business Case	Guarantee communication robustness.	
Team Members	**Project Owner**	European Union
	Project Champion	Luis Lino Ferreira – Research Associate of CISTER
	Project Leader	Michele Albano – Research Associate of CISTER
	Project Manager	José Bruno Silva –Research Associate of CISTER
	Development Team	Paulo Barbosa Renato Ayres
	Project Supporters	Csaba Hegedús – Research Associate of AITIA International Roberto Duarte- Undergrad Student of CISTER

As Table 1 depicts, the involved team members are grouped by six hierarchical roles. The top one is the Project Owner meaning the entity that funded the project. The Project Champion is the person who decides which persons are on board, he validates the solutions design and requirements, and is the first responsible of the project. Followed by the Project Leader that is responsible for leading and promoting the project, and has the vision of the process. Then, the Project Manager has a more direct contact with the development team, and contributes in technical problems and manages the team performance. The Development Team is responsible for implementing the solutions, and finally the Project Supporters, which are not directly involved in the project, may have simple tasks during the project, to support the development.

1.5 Contributions of this work

The principal contribution of this work is the addition of QoS processing in the Arrowhead Framework. Having QoS, the Arrowhead can be deployed on other networks that commonly have high traffic, expanding itself to many industrial scenarios that require as an example either a short end-to-end delay, or communication robustness, or both. The developed work was a commitment made by CISTER to another Arrowhead partner, and after its development it was integrated in the partner Arrowhead Framework implementation.

Finally, the work enabled the consolidation of the knowledge already acquired during the degree and the improvement of all skills. The presented problems here were challenging since they required the study of unknown technologies and several communications with foreign partners.

1.6 Document Organization

This report is divided into five main chapters, Context, Work Environment, Arrowhead Documentation/ Analysis and Implementation, Tests Description, and Conclusions.

Chapter 2, Context, starts by describing the problems in question. Further, the chapter portrays the business areas where the problems are present. It also gives an explanation of the studied fields that were critical for this project. More importantly, the chapter ends by explaining each of the documents present on Chapter 4. This last section is vital for a proper comprehension of the methodology used for the technical documentation.

Chapter 3, Working Environment, approaches the working methodologies and technologies used in this project. In addition, the planning of the project and the meetings occurred are explained in order to show the evolution of the work during all its lifetime.

Chapter 4, Arrowhead Documentation/ Analysis and Implementation, contains the mandatory technical documents, obeying the Arrowhead documentation methodology. These technical documents approach both solution analysis and implementation for each developed system, the QoSManager and the QoSMonitor.

Chapter 5, Tests Description, focus on software testing, by explaining essential concepts and the practiced patterns. Moreover, it also describes the performed tests for this project, dividing in two types, White-Box and Black-Box testing. This chapter is separated from Chapter 4 since most of the Arrowhead technical documentation does not contain software testing, with the exception of the Black-Box tests documented on only one document.

Chapter 6, Conclusion, describes the conclusions regarding all aspects of the project, both technical and management. First, it summarizes the work done, and follows by enumerating the fulfilled objectives. Afterwards, it ends by listing the strengths and setbacks of the developed work, suggesting different approaches and improvements, to implement as future work.

Chapter 2. Context

2 Context

This Chapter introduces the motivations and business areas related to the project. Additionally, the scientific areas and concepts that are considered necessary to understand the project are also explained. This chapter consists of five sections.

Section 2.1 presents the problem to be solved, detailing the reasons that caused the project development, focusing in QoS in IoT applications and the integration of the developed solutions with the Arrowhead Framework. Afterwards the section 2.2 describes the areas of IoT that have more to gain with this work, with the support of market statistics regarding its use in society.

Section 2.3 approaches the three principal scientific areas that required studying and analysis for the project development, IoT and Cooperative Automation, Network Monitoring and Flexible Time Triggered.

The Section 2.4 explains the concepts that are used to describe the architecture and that are vital to properly understand the framework.

Regarding the designed solution for this work problem, Section 2.5 describes it with the support of high-level diagrams for a better comprehension. First, it explains the SOA structure and the core systems of the Arrowhead Frameworks, such as the Orchestrator and ServiceRegistry. Afterwards, there is a description of how the QoS support was integrated in the Arrowhead, detailing its architecture and technologies.

Since the developed solution was integrated with the Arrowhead project, the technical documentation had necessarily to follow its methodologies, which are very specific for SOA automation applications. Therefore, the Section 2.6 helps to explain the concepts used. This methodology is then used on Chapter 4 in order to describe the work performed in this report.

2.1 The Problem

There are two problems approached in this project, the first is related to QoS requirements in systems. Nowadays, most automation applications are supported on systems with limited capabilities, the trend of applying IIoT and SOA architectures in these systems requires changes on their development philosophy. The second problem lies in the support of QoS in the Arrowhead Framework. The systems responsible for providing QoS as a service, must be capable of providing configuration and monitoring operations, independently of the underlying network technology.

2.1.1 Communication Robustness

Today's networks face several challenges, including communication robustness, mainly because of an exponential increase of network deployed devices and services. Consequently, these environments become unpredictable which in the case of real-time systems it is unthinkable, since only one failure can provoke a catastrophe.

Furthermore, certain services require various QoS guarantees for their fruition, which constitutes more challenges for these networks. In some cases, certain scenarios can guarantee only a subset of the QoS capabilities. For instance, in wider networks such as the Internet, where the in-between network is not under control, real-time objectives are not feasible and either prioritization can be provided.

In order to distinguish each service and to prioritize important services, QoS serves as a quantity measurer of quality of service. By managing certain QoS parameters as delay, jitter and bandwidth, networks can guarantee predictable behaviours avoiding traffic congestion and data losses [4].

QoS is often essential in many applications, a typical example of QoS parameters is latency, safety and bandwidth. Though the use of QoS solves the problems refered above, it´s development also brings the following challenges [5]:

- Heterogeneous networks: The diversity of different technologies and cyber-physical systems present in automation systems such as Wi-Fi, and its geographical dispositions poses challenges for the support of QoS.
- Performance: Common IoT devices were not developed for resource-constrained devices and consequently have to be adapted and simplified to work properly on those devices.
- Scalability: The continuous addition of new devices to existing systems implies extra care on the sharing of resources, like bandwidth and CPU processing time.

The support of QoS on the Arrowhead Framework addresses these challenges.

2.1.2 Integrating QoS in Arrowhead

The support of QoS in the Arrowhead Framework, in local clouds, is a fundamental functionality in some automation applications. To this purpose, an architecture has been developed that outlines the roles of involved parties in supporting QoS between a service producer and a service consumer. For this purpose, it is possible to foresee the involvement of network elements that mediate data transfers in the system (i.e. switches, routers) and the devices that are hosting the services.

The integration must be scalable and highly adaptable, since the Arrowhead Framework is deployed in many different networks, using different technologies and devices dispositions. There are multiple implementations of Arrowhead, therefore the solution has to be generic enough to be used in many of the implementations.

Another very important feature of the architecture is that the interfaces are exactly the same independently of the underlying network protocols and design on a Service Oriented manner.

2.2 Business Areas

Internet not only changed every person’s life but also marked a milestone on how industries work. It improved their efficiency, reducing costs by deploying all sort of sensor devices

connected to each other. This new era of Internet, IoT, instead of connecting people to people, it connects devices to devices. It consists in a system with multiple attached physical devices which are connected to each other via wireless or wired connections. Between these devices occurs huge amounts of data transmission that is used for powerful intelligence gathering. One area of IoT is Industrial Internet of Things (IIoT), where there is manufacturing focus in order to increase productivity gains and increasing profit margin. Back in 2000, manufacturing facilities were effectively managing only 10-12 Ethernet devices, however ten years later they are working with hundreds or more of these devices [6].

Recent studies have predicted that IoT devices usage will grow as 35 billion devices in 2019 [7], from all physical devices, IoT is the one with the most expected growth.

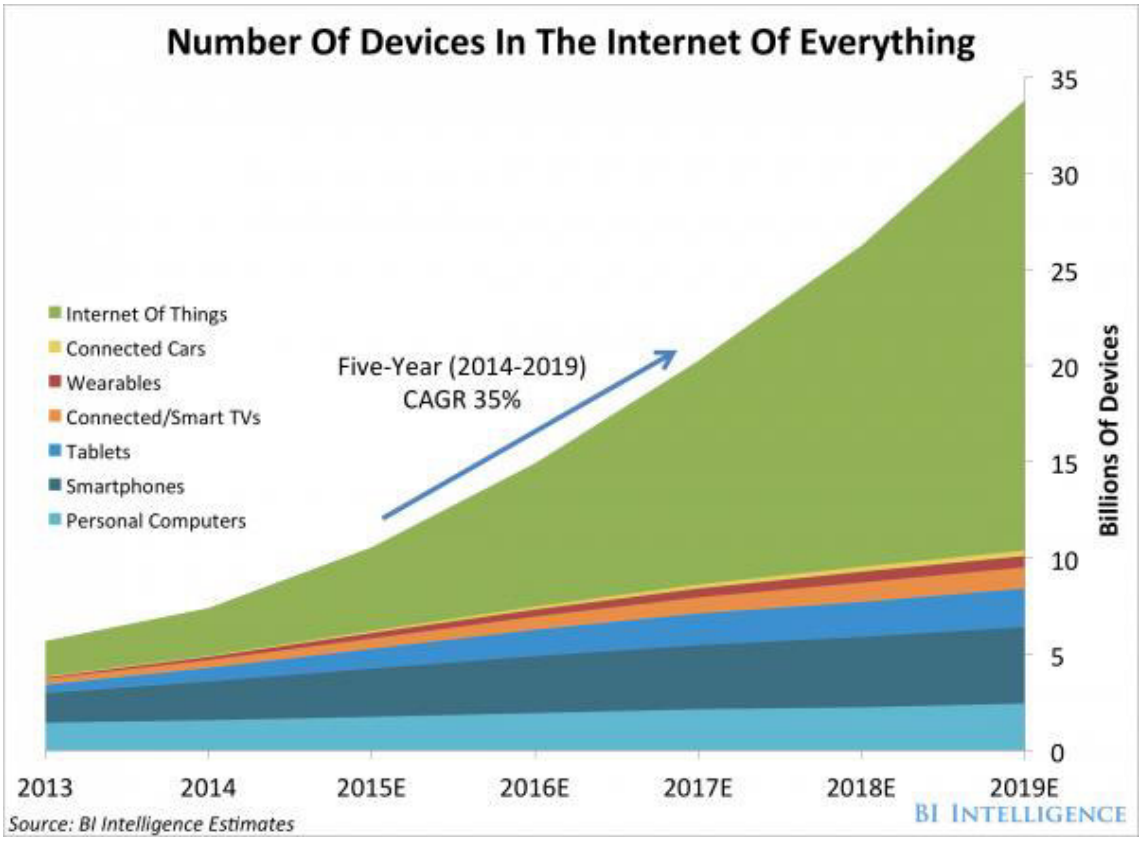


Figure 1 - Prediction for the IoT market expansion through 2019 [7].

The areas on which IoT has the most influence are [8]:

- **Automotive**

The automotive industry has been suffering profound changes, in both its manufacturing and transportation areas. Automotive manufacturing, in order to respond to the market demands of high customization and time high quality, is constantly improving, increasing its level of agility and responsiveness. With IoT, manufacturing workers have now more control over equipment and operator safety, and can easily identify machine faults or, in some cases, predict some [9]. Regarding the transportation area, according to Gartner [10] more than 250 million vehicles in 2020 will be globally connected. Vehicles would transmit and receive all sorts of data via the Internet to a service centre. This connectivity enables management and control of traffic,

optimising drive energy usage and reducing traffic accidents [11]. Furthermore, IoT promises to revolutionize automotive industry with Autonomous driving. Although it still is in a prototype phase due to the challenges of the interaction between the vehicle and the environment, self-driving cars with the deployment of various sensors (Vision chips) will be present in the foreseeable future.

- **Energy**

In the last century mankind acknowledged its low efficiency energy creation, prejudicing the planet and consequently all living being's health. IoT can have an important function on the Energy consumption, allowing a more intelligent and efficient management. Some solutions have already been developed such as outside public lightening [12], where sensors on street lamps receive on the luminosity of the street so that the intensity of the light that the lamp is giving can be related to current natural lighting. It could also relate the luminosity emanating from the street lamp with the time and day of the week: On week days, people are less active during the earlier hours of the day in contrast with the night life during the weekends.

- **Healthcare**

In medicine, knowledge is essential to the patient's life. By connecting all kinds of monitoring devices, capable of tracking all the user health status in real time, an effective treatment can improve life quality and in some cases save people's life. In a more futuristic environment, there could be small devices that monitor bodily functions: heart rate, glucose levels or even physical activity. One example would be: a man has some kind of dysfunction in his heart, before going to the cardiologist, a small device has already collect any kind of alteration in his heart. The doctor can now act immediately but before the patient would have to undergo some additional exams to check if he had any kind of problem. The treatment is applied sooner and therefore may have saved a life.

- **Industrial / Smart Manufacturing**

Since the first industrial revolution in the nineteenth century, three more phases have happened. The most recent and fourth revolution is called "Industry 4.0". This new phase brings more advanced and efficient industrial systems with self-optimization, self-configuration, self-diagnosis methods. With IoT monitoring systems can take place. Real time dashboards showing machine utilization, its performance and much more gives the manufacturing managers more control and efficiency perception, allowing them to make smarter decisions.

IoT promises to revolutionize many working areas, and currently industrial, transportation, oil & gas, and healthcare are where IoT is most used. According to Cisco last reports, it is expectable that the health segment will have the fastest growth, increasing from 144 million devices in 2015 to 729 million in 2020. This is due to the improved healthcare and industrial infrastructures, growing geriatric population, and growing prevalence of chronic and lifestyle associated diseases [13].

Arrowhead Industry Examples

Arrowhead has already been used in five applications verticals: industrial production, smart buildings and infrastructure, electro mobility, energy production and end-user services and, finally the virtual market of energy. One of these application verticals which can mostly benefit from the existence of QoS support is the industrial production, specifically on the manufacturing pilot. The following example describes an automotive manufacturing pilot in which the Arrowhead Framework was tested.

The automotive industry is becoming extremely competitive and the current evolution of the automotive industry towards hybrid and fully electric vehicles is further positioning the sector in a potentially risk averse phase because of the introduction of news powertrain technologies and new manufacturing processes. This phase of uncertainty requires an increased level of agility and responsiveness from the automotive manufacturing industry in order to integrate the constant changes that occur in the design of powertrain components. In particular, the manufacturing of electric powertrain components requires additional control over the equipment and operator safety due to the handling of dangerous chemical content.

All the above requires a radical increase in the ability to monitor and control the manufacturing processes very closely. This implies the deployment at a large scale of so-called Industrial IoT (IIoT). In this context, the Arrowhead Framework provided a key element in achieving connectivity and integration between various layers of the manufacturing systems and organisations, and in facilitating the management of data.

One of the functions that Arrowhead assumed was the maintenance of automation systems. It had three main responsibilities: detecting faults, as soon they occur in order to minimise their impacts; accurately assessing the stat of the system in order to anticipate faults; to enable rapid and effective intervention in a shop floor after a fault is identified. The pilot was developed in a ZigBee [14] protocol using REST servers in the University of Warwick.

2.3 State of the Art

State of the art section approaches the three principal scientific areas that required studying and analysis for the project development, IoT and Cooperative Automation, Flexible Time Triggered and Network Monitoring.

2.3.1 IoT and Cooperative Automation

IoT is a network of real-time, physical, embedded devices capable of producing an output or even receive input. These devices, called as “things”, are deployed on a host, and are dispersed on different places, all interconnected to processing units. An entity, such as a company or a person, can use these devices to control and monitor its host, i.e. building, machine, or person, and improve its performance based on the collected data [15] .

In certain systems, the network devices cooperate between themselves. This integration is called cooperative automation. In order to reach a faster solution, the solution is divided in tasks

and each task is assigned to a device, this happens in some real scenarios, in which devices perform different tasks, cooperating in teamwork to reach the same solution.

The next sub-section describes the Industry evolution during time:

- **Industrial Generations**

Modern industrial production and manufacturing systems have evolved in four different generations. The first one that enabled the industrial revolution dates back to the mid-1800s. Mass production of goods such as clothes, cars and many other products was made possible by the use of steam-powered machines in the beginning of the 20th century.

The second generation saw efficient pneumatic systems emerge as a widely employed solution for mass-production. The use of pneumatic valves combined with sensors enabled automatic production systems to be used in industrial applications.

Pneumatic motors evolved to electrical motors in the third generation. Using electricity as the energy source made it so that even newer types of automatic control systems were created. Sensors and actuators were now connected to new types of monitoring and control systems like Distributed Control Systems, DCS [16] and Supervisory Control And Data Acquisition, SCADA [17] using technologies such as field buses. The hierarchical approach of device-level, DCS, and SCADA (known as ISA-95 [18]), soon became the architectural style in effect for how industrial manufacturing systems were designed and set up. Eventually DSC and SCADA systems became networked, which enabled solid integration between control systems and Enterprise Resource Planning Systems (ERP) and Manufacturing Execution System (MES). Nowadays this is the most generally used approach in the industry, and has been so for at least the last 20-30 years. The current state of the art architecture ISA-95 [19] was established in the 90's. Apparently, the size of ISA-95 based automation systems seems to be narrow to nearly 100.000 I/O points, thus becoming a technology bottleneck in the perspective of the upcoming smart cities and smart energy grids.

In 2011, the concept of Industry 4.0 was born in Germany. This idea builds upon the last generation of monitoring and control systems, yet allows an even thinner level of interaction between shop-floor devices and high-level enterprise systems. In Industry 4.0, state of the art technologies like IoT and Cyber-Physical Systems (CPS) are used in order to be able to break the classical rigorous hierarchical approach of ISA-95 with a more flexible approach without hurdles and sealed systems. By basing all communication on standard-based protocols, like the TCP/IP protocol suite, it is now feasible to have information exchange between (nearly) any systems in a manufacturing facility. This gives room for new strategies in terms of safety and security, minimized energy consumption, global plant optimization, etc.

2.3.2 Flexible Time Triggered (FTT)

Beginning from its definition, a real time system must process the input and produce an outcome within a specified time, else it will fail [20]. These systems were made and are mainly used for the industry area, a perfect example of this system is Anti-Lock Braking System [21] on cars, which guarantees the passenger safety. If this system fails it will have severe consequences.

Industrial processes must be integrated on specific networks, connecting multiple systems, working with each other, where monitoring, control and predictability are essential to guarantee the avoidance of any failure. In this cases predictability is favoured against average throughput, and message transmission is typically characterized by time and precedence constrains.

Communication systems that support real-time applications have to accomplish several requirements, the most relevant are predictability, and QoS dynamic management.

Sensors are the hands and legs of the industrial automation system that monitor the industrial operation conditions, inspections, and measurements in real-time. They are an integral part of the industrial automation systems and provide feedback for system control [22].

Communication is the backbone of all the industrial components for efficient automation production systems. There are multiple solutions for this real time communications, however since Ethernet is a very used architecture it is not ready for this type of communications. Due to Ethernet easy deployment and low cost, it is mandatory to develop real time solutions using Ethernet.

Ethernet has a feature to avoid packets collision named CSMA/CD which is at the same time an arbitration mechanism, and the fact of being randomness makes it the main obstacle of supporting real-time applications in Ethernet. Several techniques have been developed for applying the real time behaviour, some of the most relevant methods are listed below:

- **Master/Slave techniques:**

A hierarchy is established by dividing all network nodes into two groups, the master and salves. The master controls the traffic in the network among slave nodes, deciding when and which slave has the permission to send data.

- **Switched Ethernet**:

Using switches in the network reduces the non-deterministic behaviour of Ethernet. Basically, a switch buffered the arrival message and checks the destination address of the message. The output ports have output buffers and the order of message sending is based on priority level.

The FTT-SE protocol therefore proposes a solution using an Ethernet network which accomplishes all the requirements of a real time network.

- **Concept**

Flexible Time Triggered – Switched Ethernet is a real-time communication protocol, and it is the last development of the FTT-Ethernet paradigm.

Proposed in 1998, the FTT paradigm is a framework which has an ability to handle time-triggered and event-triggered messages, timelines guarantee, temporal isolation support. Its master/salve architecture allows a centralized message scheduling by a single node in the network called master. The master schedules the traffic in Elementary Cycles (EC) and propagates the respective triggers throughout the system, using Trigger Messages (TM). Whenever a slave receives the trigger, it has the permission to transmit its message to the message receiver slave.

- **FTT-SE - Architecture**

The FTT paradigm to overcome the Ethernet real time limitations it uses a centralized scheduling and master/multislave transmission control. The centralized scheduling allows a dynamic QoS management and the master/multislave control makes the network more deterministic, capable of enforcing a notion of time and therefore avoid collisions.

The master is the system coordinator and it is responsible for building the elementary cycle and the trigger message. The slave nodes execute the tasks required by the user, requesting services delivered by the communication system.

FTT-SE is based on the FTT paradigm, and brings another advantage, the absence of collisions. Due to its micro-segmented switch-based structure, as Figure 2 depicts, each port in the switch is a private domain collision, avoiding traffic collision and capable of getting parallel transmission in the network.

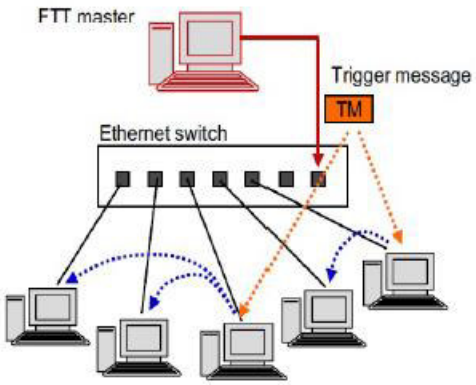


Figure 2 - FTT-SE architecture [23].

- **Elementary Cycle (EC)**

The master internal scheduling for all nodes is a critical process on the FTT-SE protocol, because it´s where it's decided which messages can be exchanged within a certain elementary cycle.

An elementary cycle is a fixed duration timeslot used to allocate traffic on the network. There can be several windows dedicated to specific types of messages [23]. On each elementary cycle, there are two timeslot windows, synchronous for the time-triggered messages and asynchronous for all event-triggered messages.

As Figure 3 depicts, each elementary cycle starts with a broadcast Trigger Message (TM) by the master. The TM synchronizes the network and identifies all the messages, synchronous or asynchronous, that must be processed on the same elementary cycle. The synchronous messages can have priorities, and are the first transmitted on the EC. The asynchronous traffic occurs on the remaining time of the EC, and there are no guarantees that the real time requirements will be accomplished.

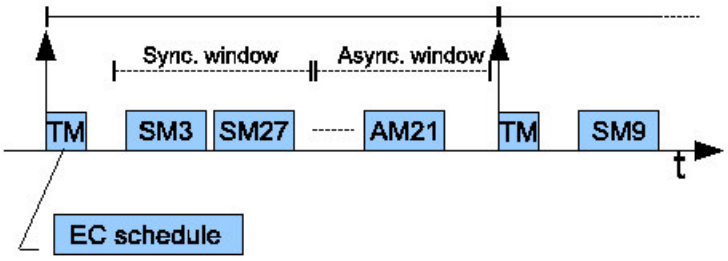


Figure 3 - Elementary Cycle Structure [24].

- **Trigger Message (TM)**

A TM is a message used only by the master node at the beginning of each EC. The master propagates the TM in broadcast to be received by all slave nodes. In the trigger message payload there is information about all the messages or tasks that must be sent from the correspondent nodes during the EC, as Table 1 lists.

Table 2 - Trigger Message structure.

Field	Size (bytes)	Description
FTT Type	2	Message Type
Sequence Number	1	Sequence Number
Flags	1	Flags set
Number of Synchronous Message	2	Message number
Number of Asynchronous Messages	2	Message number
Message Index	4	Message information
ID	2	Message Identification
Fragmentation Number	2	Fragmentation Number

- **Traffic Transmission**

In FTT-SE protocol, there can only be two types of traffic, synchronous and asynchronous. Each one is decided by the master scheduling, and are completely opposite between each other, the synchronous is time-triggered while asynchronous is event-triggered. Both have a relevant influence on the network communication.

- **Synchronous**

The master node saves all the synchronous streams on the Synchronous Requirements Table (SRT), including the stream sender and receiver. Each stream is defined by its Worst-Case Message Length (WCML), a deadline, period and an offset. These 4 properties will be used by the master to decide which stream has a higher priority, depending on the scheduling policy used.

The synchronous transmission only takes place when the timeslot for the TM ends. A bandwidth is also dedicated to synchronous messages to be transmitted in the EC.

- **Asynchronous**

All the asynchronous messages are saved on the Asynchronous Requirements Table (ART) on the master node. Each stream is defined by a Worst-Case Message Length, a minimum inter-arrival time (which replaces the deadline on the synchronous messages), and a minimum period between two asynchronous messages (which replaces the period slot on the synchronous messages).

For handling the asynchronous traffic, a signalling technique is used by the master. This procedure takes advantage of full-duplex connections due to the use of Ethernet switches, having the possibility to receive and transmit simultaneously. After the TM timeslot window, the slave nodes communicate their status regarding the queue of asynchronous messages to the master. After that the master records and processes the transmitted slave status using the scheduling policy to decide which message should be transmitted at the next EC. Thus the response time of synchronous message is never less than two ECs [24].

2.3.3 Network Monitoring

Network monitoring is the use of a system that is constantly monitoring a network, including its system, services, machines, etc. This system can be event-based, notifying any interested part if something fails, slows down or is not working properly. Hence, it is a different form of intrusion, which monitors networks for threats from inside or outside. There are some highly known monitoring tools available, with a couple being named here:

- **NAGIOS** [25]

  It's the most recognizable and used tool in the monitoring industry, an open source software application that monitors everything from systems, networks, infrastructures, devices, applications and services. It is highly customizable, making it very easy to anyone create plugins, and can alert its users when something is wrong or not supposed to happen. It uses agents for a common way of communication and can handle many protocols as SSH, SNMP, WMI.

- **CACTI** [26]

  Cacti was initially designed as a front-end application for the data logging tool RRDtool, focusing more on visual graphing. Much like Nagios, it polls services at predetermined times and graphs the data returned but has inferior protocols compatibility than Nagios.

Monitoring besides warning real time events, with current technologies can also make predictions as enabling prognostics through the ability of calculating useful life of monitored components. On the other hand, it can also even predict machines mal-functioning, automatically ordering for spare part providers, for a fast response. These processes are becoming more relevant in order to keep the production machinery running at high efficiency.

2.4 Arrowhead Key Definitions

The objective of Arrowhead Framework architecture is to facilitate the creation of local automation clouds enabling local real time performance and security, paired with simple and cheap engineering, while enabling scalability. Here the concept of local cloud takes the view that a specific geographically local automation should be encapsulated and protected. The local cloud idea is to let the local cloud include the devices and systems required to perform the desired automation tasks, providing a local "room" which can be protected from outside activities [5].

Devices in such local clouds are considered to be IoT devices speaking at least one SOA protocol. The capability of building automation systems requires a number of local cloud properties to be enabled. Furthermore, both intra and inter cloud information service exchange capabilities are necessary for enabling IoT devices to interoperate and to be integrated with others to become an automation System of Systems.

To discuss and define a local cloud architecture, the following definitions are important to understand the Arrowhead Framework properly. Note that these keywords may have other definitions in different domains, but for the usage of Arrowhead the following definitions were made.

- **Service**

  A Service is what is used to exchange data between a providing System to a consuming System. A Service can be implemented to use a number of different SOA protocols, some examples being REST or XMPP. A Service is produced by a software System. A Service can have related metadata and can be able to support non-functional requirements such as security, real-time operation or different levels of reliability – among others [5].

- **System**

  A System is what provides and/or consumes services, and must be able to be the Service provider of one or more services and in the meantime the Service consumer of one or more services. A System is a software implementation and runs on a Device.

- **Device**

  An Arrowhead compliant Device is a piece of hardware, with computational, memory and communication capacities that hosts one or more Systems and can be set up in an Arrowhead Local Cloud.

- **Local Cloud**

  In the Arrowhead context, a Local Cloud, as represented in Figure 4, is defined as a self-contained network with the three required core systems deployed and, at least, one application system deployed. A Local Cloud must host only one ServiceRegistry system.

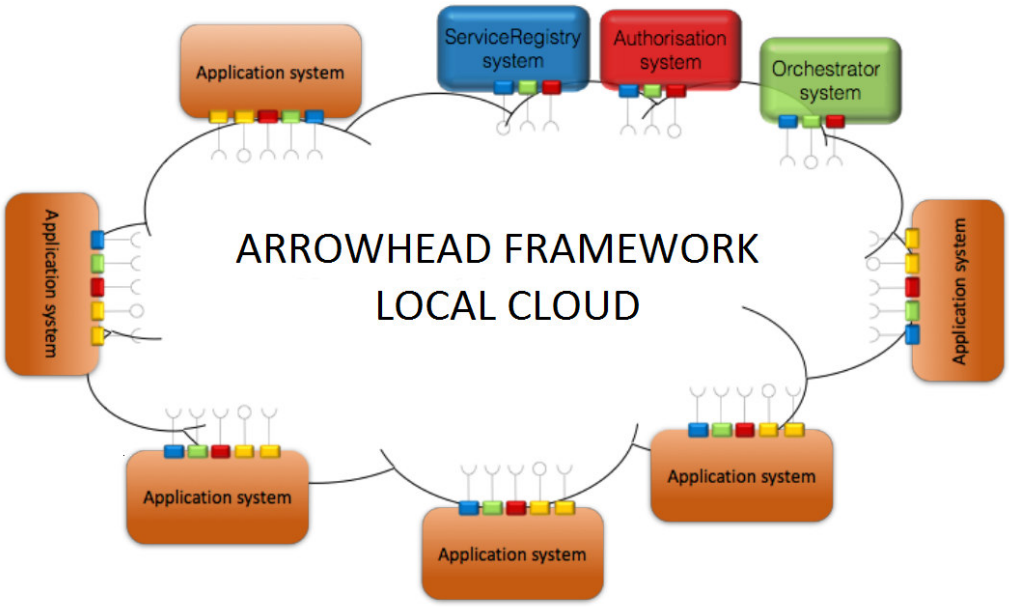


Figure 4 – Local Cloud representation [27]

- **System of Systems**

  Inside the Arrowhead Framework, a System of Systems is defined as a group of Systems that exchanges information between themselves by means of Services. These systems are administrated by the Arrowhead core systems, such as the Orchestrator. Therefore, a Local Cloud becomes a System of Systems in the Arrowhead Framework's definition. Such as Figure 5 represents, if two Systems hosted by different Local Clouds are administrated by Arrowhead core systems to exchange services, it also is a System of Systems. When Arrowhead compliant Systems work together, they become a System of Systems. Seeing that two or more such System of Systems can also work together, the Arrowhead Framework becomes a natural enabler of further, complex solutions.

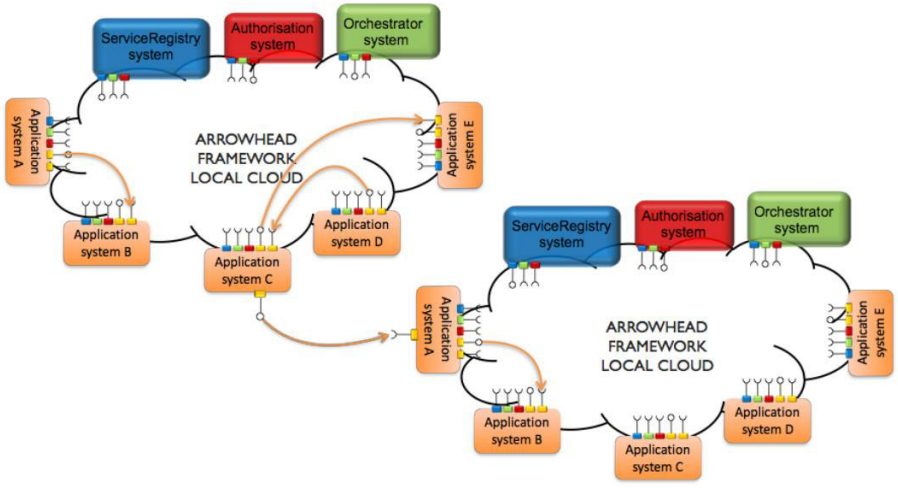


Figure 5 – Systems exchanging Services, thus creating a System of Systems in a Local Cloud and between Local Clouds [27].

2.5 Vision for the Solution

This chapter approaches the solutions of the problems already described in the Section 2.1, explaining them with the support of diagrams. The Section 2.5.1 describes the Arrowhead solution, in which explains the architecture and the protocols used. The Section 2.5.2 is relative to the support of QoS in the Arrowhead Framework, describing a proposed integration architecture.

Note that the Arrowhead solution was already developed in this project, and consisted only of the framework without any QoS support.

2.5.1 Arrowhead Solution Architecture

The Arrowhead architecture is composed by a set of Systems, which provide a number of Services. The objective of the framework is to provide an architecture, from which a self-contained local automation cloud can be created. These clouds shall further be capable of providing certain automation support services and provide support for bootstrapping, security, suitable metadata, protocol and semantics transparency and inter-cloud service exchanges [1].

The architecture features three types of services:

- Mandatory core services
- Automation support core services
- Application services

These are provided by mandatory and support core systems, as well application systems.

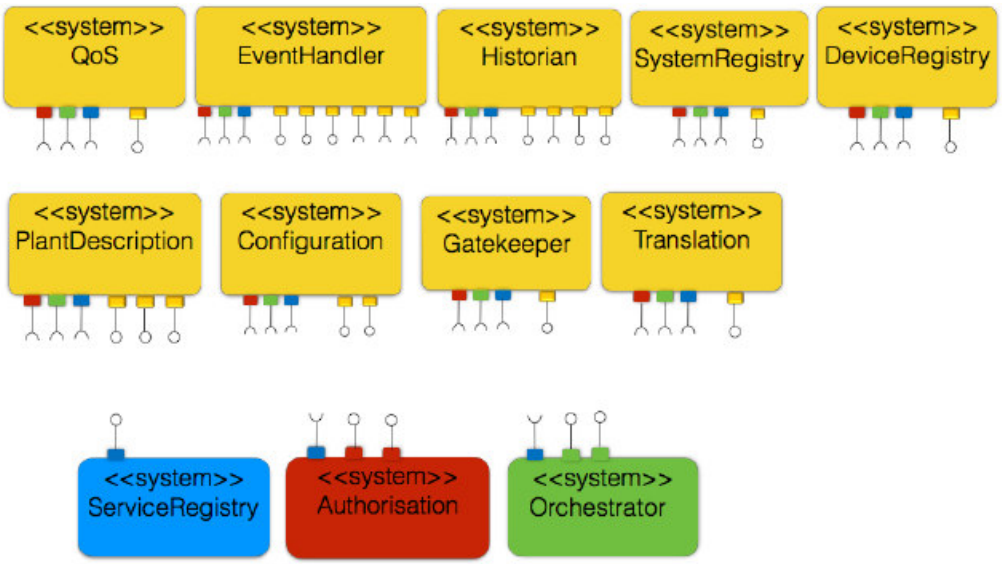


Figure 6 - Core Systems of the Arrowhead Framework [1].

Figure 6 depicts the cores systems defined with the Arrowhead Framework. These mandatory core services will enable the basic properties of a local cloud, such as service exchange between

a service producer and a service consumer with a desired level of security and autonomy. There are three independent core systems present in Arrowhead, which are the following:

- **ServiceRegistry System**: The ServiceRegistry goals are to provide a storage of all active services and enable the discovery of these. Since the Arrowhead Framework is a domain-based infrastructure its service registry functionality is based on DNS-SD standard [1].
- **Authorisation System**: The Authorisation System provides Authentication and Authorisation of services, working with a set of rules that allow a consumer to use a service resource or not. Two different Authorisation systems are defined within the Arrowhead Framework, an AA – Authorisation Authentication system and an AAA – Authorisation Authentication and Accounting system. The AA system is better suited for local clouds enrolling Systems hosted on Devices with sufficient computational power. While the AAA systems is better suited for local clouds enrolling Systems hosted in resource constrained devices.
- **Orchestrator System**: The Orchestrator is a central component for Arrowhead, it is utilised to dynamically allow the re-use of existing services and systems in order to create new services and functionalities. From an architectural point of view, the Orchestrator is responsible for finding and pairing service consumers and providers.

To facilitate automation application design, the Arrowhead Framework contains a number of automation services hosted on supportive core systems, on the contrary of the core services these services are optional. There are nine support core systems:

- PlantDescription system
- Configuration system
- DeviceRegistry system
- SystemRegistry system
- EventHandler system
- QoS system *(see section 4*)
- Historian system
- Gatekeeper system
- Translation system

From these nine systems, with the exception of the QoS, only the EventHandler was used in this project.

EventHandler

The EventHandler supports publish/subscribe communication and filtering of events, and storage of information regarding events. In the usual workflow, the EventHandler receives events from event producers, dispatches them to registered event consumers, logging these events to persistent storage, registers producers and consumers of events, and applies filtering rules configured by Event Consumers [28].

Regarding the events flow, event producers declare they consume the Publish service, event consumers declare that they produce the Notify service. Furthermore, producers and consumers

create rules on the plant description service regarding which event they produce/consume. The Orchestrator system retrieves the information, computes the matchings, pushes the matching to consumers, and to EH instances.

2.5.2 Supporting QoS in Arrowhead

Since the problem focused on configuring and monitoring QoS, two systems were defined, one is the QoSManager, responsible for configuring QoS, the other, not less important, is the QoSMonitor, which guarantees the success of QoS configuration by monitoring both network and devices in real-time. Hence, these two systems are considered as supportive systems of the Orchestrator since they only add optional functionalities to the Arrowhead Framework.

Concerning the QoSManager, it is responsible for configuring QoS, working with, not only, the systems that consume and provide services, but also, with network actives. The functionalities are supported by keeping track of network actives and system configurations and by managing reservation of computational and communication capabilities over them. With the support of drivers and algorithms, the QoSManager is capable of working under different network technologies. These drivers are called as QoSDrivers, and they interact with the network actives and systems using custom protocols that depend on the network active, its vendor, etc.

A Service Level Agreement (SLA) mechanism is used in both QoSManager and QoSMonitor systems for setting up QoS parameters. In the field of embedded computing, critical applications typically require stricter timing requirements and in mainstream embedded applications, the focus is on energy saving and low cost. Regarding other scenarios, the SLA can specify the amount of data that must be offered by service providers [5].

Concerning the QoSMonitor, its purpose is to make sure that given QoS Requirements (expressed using the SLA) are being respected. The system must also warn of any given critical events that occur. It accomplishes these objectives by monitoring the performance of services directly, by having modules running in systems and accessing log information of network actives. This allows the service to detect if any QoS requirements (for example monitored parameters like delay or bandwidth, or periodicity of communication) cannot be guaranteed anymore by the current orchestrated service.

2.6 Arrowhead Documentation Methodology

The Arrowhead Framework has the goal of addressing the technical and applicative issues associated with cooperative automation, based on a SOA architecture. The problems of developing these systems is the lack of adequate development methodologies, which would facilitate the reusing of services on different applications.

As consequence, every Arrowhead partner must document and describe its developed solutions using an Arrowhead compliant documentation method. This has the purpose of accomplishing a common understanding of the systems developed by every Arrowhead partner. The Arrowhead compliant methodology includes design patterns, documentation templates and guidelines that aim at helping systems to conform to Arrowhead Framework specifications.

The Arrowhead compliant documents consist in three levels: System-of-Systems, System and Service levels, as Figure 7 depicts.

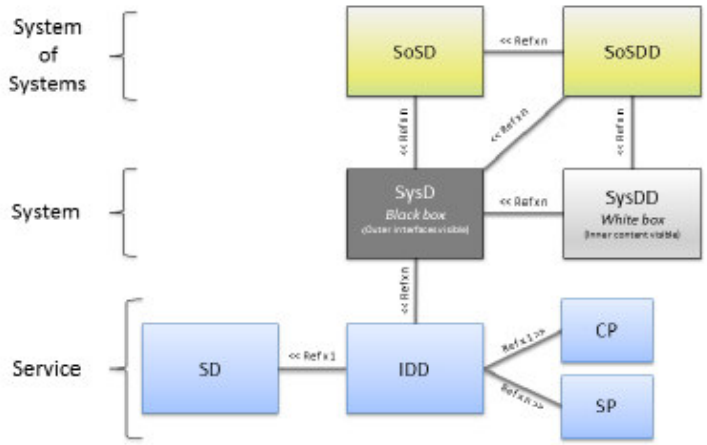


Figure 7 - The Arrowhead documentation relationships [29].

The approach is to apply terms “black-box” and “white-box” only in the System Level, in the sense of writing an abstract high-level description of a system approaching only its behaviour, and on the other case in the sense of writing with detail the implementation done.

This documentation system has been used within the aims of this report, therefore the structure of this report is in accordance with this kind of documentation and slightly different from more “traditional” PESTI reports.

2.6.1 System-of-Systems Level

At the System-Of-Systems (SoS) there are two types of documentation, System-of-Systems Description (SoSD) document and System-of-System Design Description (SoSDD).

- **System-of-Systems Description (SoSD) Template**

This document should contain an abstract high level view, describing the main functionalities and generic architecture, without referring any specific technology. Such document must include use-cases to help understand the expected behaviour. Based on these use-cases, the document should include behaviour diagrams. It is also recommended the support of UML diagrams, mainly component and activity diagrams [29].

In this document, it is also important to include information about non-functional requirements, in which the security must be treated separately. This includes the definition of security principles that SoS needs to follow on a non-technical generic level, the security objectives and the assets which need to be protected.

- **System-of-Systems Design Description (SoSDD) Template**

This document describes how a “System-of-Systems Design Description” has been implemented on a specific scenario, describing the technologies used and its setup. The document starts with an abstract high-level view of the SoS realization, describing how its main functionalities can be

logically implemented. Specific use-cases are described next, supported by structure and behaviour diagram.

The non-functional requirements implemented by this realization must be listed along with its security features. To support the validation of the security attributes of this SoS realization, it is also necessary to include information identifying the data flows in the system as well as its threads and vulnerabilities [29].

2.6.2 System Level

At the system level there are two different representations, the “SysD Template” consists in a “black-box” design, while the “SysDD Template” consists in a “white-box” design.

- **System Description (SysD) Template**

This document provides the main template for the System Description of Arrowhead compliant systems. As a “black-box”, there should be a description of the main services and interfaces of a system without describing its internal implementation where all the system produced/consumed services are listed. It is recommended the use of component diagrams to represent the interoperability of different systems. This structural view can be complemented with a high-level behavioural view such as sequence diagrams [29].

- **System Design Description (SysDD) Template**

This document provides the main template for the description of Arrowhead Systems, technological implementations, describing in detail the proposed solution. Here it is encouraged the usage of formal or semi-formal models in order to enable the automation generation of code from the specifications as much as possible. When automation is not possible, the document should be precise enough to guide developers towards an implementation that matches these specifications [29].

2.6.3 Service Level

The service level consists of four documents: the SD Template, the IDD Template, the CP Template and the SP Template.

- **Service Description (SD) Template**

A service description document provides an abstract description of what is needed for systems to provide and/or consume a specific service. SD´s for Application Service are created (specified) by the developers of any Arrowhead compliant system and by the developers of the Core Arrowhead Framework services. The SD shall make it possible for an engineer to achieve an Arrowhead compliant realization of a provider and/or consumer of description of how the service is implemented by using the Communication Profile and the chosen technologies [29].

The document starts by describing the main objectives and functionalities of the service and follows on defining the Abstract Interfaces and an Abstract Information Model. On Abstract Interfaces section all interfaces should be detailed using a UML sequence diagram. The Abstract Information Model section must provide a high level description of the information model with

types, attributes and relationships, based on UML Class diagram. Finally, non-functional requirements must be described for each service.

- **Interface Design Description (IDD) Template**

An IDD provides a detailed description of how a service is implemented by using a specific Communication Profile and specific technologies. This document describes each of the interfaces in a separate sub-section, and the functions included in each interface. To support the descriptions, the use of UML sequence, class and components diagrams is recommended. There must be an Information Model section present in the document, containing detailed information about the data formats used by the interface along with metadata information [29].

- **Communication Profile (CP) Template**

The CP document describes the types of message patterns, defining in detail how the CP handles security issues, regarding authentication and encryption based on the protocol specifications. For instance, in the use of Constrained Application Protocol (CoAP), Datagram Transport Layer Security (DTLS) is enabled. This document can be identified by three characteristics: transfer protocol (e.g. CoAP); security mechanism (e.g. DTLS); data format (e.g. XML).

- **Semantic Profile (SP) Template**

The SP describes the data format by pointing out its type (e.g. JSON; XML) and how that data is encoded.

Chapter 3. Working Environment

3 Working Environment

This chapter describes how the project developers worked during the project, along with a timeline. First, it introduces the work methodologies used. Then, with the support of a Gantt diagram, the planning of the overall work is shown along with the follow up meetings. Finally, in Section 3.4, we list all the used technologies and what systems each technology was used for.

3.1 Work Methods

The work here developed consisted in a team work involving local and foreign partners. Since other research centres should use the developed code, the supervisors of the project had a considerable participation in the analysis and documentation phases. This was done to ensure a high-quality software that meets the needs of all, having also a predictable schedule. Therefore, it was considered that the best work method should be an agile one, specifically Rational Unified Process (RUP).

As RUP proposes, the project timeline consisted in the following four phases:

1. **Inception** – In this phase, the tasks consisted more in research and the studying of the technologies that were planned to be used.
2. **Elaboration** – During this phase, several project architectures and technologies were proposed and discussed with the project supervisors. Both functional and non-functional requirements along with the pilot project were defined.
3. **Construction** – In this phase, all the elaborated use cases were implemented. We integrated all the implemented systems between themselves, with the framework and finally with the pilot project.
4. **Transition** – Some prototypes were made and posted on practice, resulting on code testing and improvements.

Since this project was more research-related, it consequently involved considerable theoretical tasks, and new technologies usage. The analysis phase has a more relevant role than in typical software developments, and consequently was more time consuming. After each research task, a power point would be written containing the captured ideas, and presented to the supervisors for comments.

Some of RUP best practices [30] were adopted, such as:

- Develop Software Iteratively. In each iteration, the developing team presented the work and received feedback from the supervisors. This occurred during the RUP construction phase, in two weeks periods or less.
- Control changes to software. Each system was developed in different branches to isolate any changes made by another partner.
- Visually Model Software. With the support of UML diagrams, every code design and description was documented in graphical presentations in order to ease its understanding by all members of the team.

All the source code of the project and its design was managed using the Bitbucket [31] system, using Git [32] as a revision control system. Regarding the tasks, the team used the Issue Tracker system of Bitbucket for the project tasks/issues. Each task consisted in three different types: code, analysis and tests. These work methods allowed a rigorous code design documentation and the possibility to see what every one on the development team was doing, had done and what was going to do. Any error or bugs that emerged, would be immediately reported.

Along with the internship responsible it was decided to do weekly report that would describe the work done in the last seven days, so there would be a historical documentation with an evolution timeline of the project. There would be also a weekly meeting, at minimum, with the project responsible, where it would be discussed the tasks status, the code design and planned future tasks. Never less than thirty minutes, every meeting participant would show what had done, and what problems they were facing. These meetings were very important to overcome any of the internship problems, and constantly improve the project solution.

3.2 Work Planning

The project planning since its beginning was divided in six different phases: Project Requirements, Software Analysis, Development, Tests, Documentation, and Meetings. Unlike typical project planning, it was best to separate the Meetings phase from the Documentation since it had an important role. This was done because during the project, meetings occurred very often, more than normal with both supervisors and team supporters given that they had an enormous influence in the work planning.

Additionally, we decided that the initial phases would be the Project Requirements gathering and the Project Meetings. In this stage of work, the team would discuss the objectives and functionalities that were considered essential to be developed. When the team decided it had enough information to evolve from Project Requirements to Software Analysis, the Meetings still occurred until the end of the project.

Regarding the Software Analysis phase, it consisted in a more rigorous and detailed description about the project application. The team, using UML [33], studied and proposed several architectures, detailing the technologies to be used. During this stage, the team had weekly meetings to reach a consensus about what and how certain solutions should be implemented.

Furthermore, both Development and Test phases occurred at the same time with only a few days of difference. During Development the team implemented all the previously designed systems along with the respective functionalities. In Testing phase, the team integrated all the developed systems and consequently would corrected any captured failure. Two types of tests were used, Unit [34], Acceptance [35].

Lastly, the Documentation phase involved the writing of PESTI report and the Arrowhead compliant technical documentation. Table 3 shows a representation of the Project Planning.

Table 3 - Project Planning.

ID	WBS	TASK	PREDECESSOR	RESPONSIBLE	DURATION	START	FINISH
#R	1	**Project Requirements**		**Team**	15 weeks	1 March	31 May
#A	2	**Software Analysis**	**#R**	**Team**	10 weeks	2 May	30 June
#D	3	**Development**	**#A**	**Team**	14 weeks	1 June	31 August
#T	4	**Tests**	**#D**	**Team**	13 weeks	6 June	31 August
#D	5	**Documentation**	**#I**	**Team**	20 weeks	30 May	23 September
#M	6	**Meetings**	**none**	**Team**	34 weeks	1 March	30 September

3.3 Follow up Meetings

The meetings occurred throughout the project are here presented and explained. First, as Table 4 depicts, there were three types of meetings: Briefings, Tele-Conferences and Demo Presentations.

The Briefings consisted in formal meetings with the supervisors, using Power Point [36] presentations to discuss the project, more specifically its status, the tasks and solution strategies. During the code implementation, the Briefings were also used to present the developed work and discuss functionalities.

The Tele-Conferences occurred only with the foreign team supporters. The purpose of these meetings was to resolve any technical problems and plan the joint work.

Regarding the Demo-Presentations, these meetings occurred less often that the others since they took place in the final stage of the project. In this meetings the integration of all developed systems and their functionalities were presented, and a proof of concept video was recorded.

In addition, it must be noted that other meetings, of more informal nature, were held daily, in a more informal environment. Table 4 shows information regarding the meetings held throughout the project in chronological order.

Table 4 - Meetings Agenda.

DATE	SUBJECT	PARTICIPANTS
16/02/2016 15:00 – 16:00	**Briefing**: Project presentation.	Luis Lino, Michele, Joss, Pedro, Paulo Barbosa
25/02/2016 10:00 – 11:00	**Briefing**: Definition of tasks, elaboration of planning.	Luis Lino, Michele, Renato, Paulo Barbosa.
29/02/2016 16:30 – 17:30	**Briefing**: Work strategy definition.	Luis Lino, Renato, Paulo Barbosa.

DATE	SUBJECT	PARTICIPANTS
1/03/2016 15:30 – 16:30	**Briefing**: Project Status.	Luis Lino, Renato, Joss, Paulo Barbosa.
04/03/2016 15:00 – 16:00	**Briefing**: Work strategy definition. Project planning redefinition.	Luis Lino, Michele, Renato, Joss, Pedro, Paulo Barbosa.
07/03/2016 14:30 – 15:30	**Briefing**: Project Status.	Michele, Paulo Barbosa.
08/03/2016 14:30 – 15:30	**Briefing**: Project Status. Difficulties exposure.	Luis Lino, Michele, Paulo Barbosa.
11/03/2016 10:00 – 12:00	**Briefing**: Code Design Discussion.	Luis Lino, Michele, Joss, Renato, Pedro, Paulo Barbosa.
15/03/2016 15:00 - 16:00	**Briefing**: Pilot-Project Discussion.	Luis Lino, Michele, Paulo Barbosa.
17/03/2016 18:00 – 19:00	**Briefing**: Project Status Project planning redefinition.	Luis Lino, Paulo Barbosa.
21/03/2016 16:00 – 17:00	**Briefing**: Project Status Project planning redefinition	Luis Lino, Paulo Barbosa.
24/03/2016 15:30 – 16:30	**Briefing**: Project Status	Luis Lino, Michele, Renato, Paulo Barbosa.
30/03/2016 15:00-16:00	**Briefing**: Project Status	Luis Lino, Michele, Renato, Paulo Barbosa.
06/04/2016 11:30-12:30	**Briefing**: Project Status	Luis Lino, Michele, Renato, Paulo Barbosa.
08/04/2016 14:00 – 15:00	**Tele - Conference**: Discussion about integration problems.	Renato, Paulo Barbosa.
02/05/2016 14:00 – 15:00	**Briefing**: Project Status Project work methodology redefinition.	Luis Lino, Michele, Renato, Paulo Barbosa.
03/05/2016 14:30 – 15:30	**Tele - Conference**: Problems discusses. Shared thoughts about solutions.	Renato, Paulo Barbosa.
10/05/2016 15:30 – 16:00	**Briefing**: Project Status. Code design redefinition.	Luis Lino, Paulo Barbosa.
20/05/2016 18:00 – 20:00	**Briefing**: Demonstration of features. Project Status.	Luis Lino, Michele, Bruno, Renato, Paulo Barbosa.
27/05/2016 15:00-16:00	**Briefing**: Demonstration of features.	Luis Lino, Michele, Bruno, Renato, Paulo Barbosa.
30/06/2016 15:00-16:00	**Briefing:** Project Status.	Luis Lino, Michele, Bruno, Joss, Paulo Barbosa.
03/06/2016 16:30 – 17:00	**Briefing**: Project Status Project planning redefinition.	Luis Lino, Bruno, Renato, Paulo Barbosa.
09/06/2016 14:15 – 16:00	**Briefing**: Demonstration of features. Project planning redefinition.	Luis Lino, Bruno, Joss, Pedro, Renato, Paulo Barbosa.
15/06/2016 18:00-18:30	**Briefing**: Demonstration of features. Project planning redefinition	Luis Lino, Bruno, Paulo Barbosa.
5/07/2016 11:00-11:15	**Briefing**: Project Status.	Michele, Luis Lino, Renato, Bruno, Paulo Barbosa.
18/07/2016 12:00-12:30	**Briefing**: Project Status Project planning redefinition	Michele, Luis Lino, Renato, Roberto, Paulo Barbosa.

DATE	SUBJECT	PARTICIPANTS
29/07/2016 17:00-17:30	**Demo Presentation:** Demonstration of all features – 1st demo.	Michele, Bruno, Renato, Roberto, Paulo Barbosa.
1/08/2016 17:00-18:00	**Demo-Presentation:** Discussed the demo. Demonstration of features.	Luis Lino, Michele, Bruno, José Pedro (Event Handler), Renato e Paulo Barbosa.
13/08/2016	**Demo-Presentation:** Demonstration of all features – 2nd demo	Luis Lino, Michele, Bruno, José Pedro (Event Handler), Renato e Paulo Barbosa.
19/08/2016	**Tele – Conference:** Discussed the developed changes. Planning of future tasks.	Michele, Paulo Barbosa.
29/08/2016	**Briefing:** Demonstration of all features – final demo	Luis Lino, Renato, Joss, Paulo Barbosa.
23/09/2016	**Briefing:** Documentation Overview	Luis Lino, Bruno, Renato, Paulo.

3.4 Technologies

Table 5 summarizes which technologies were used, their version and the justification of their employment. In the following sub-sections, each technology is explained.

Table 5 - Employed technologies.

Technology	Version	Where Was Used
Java	JAVA SE 8	Arrowhead Applications
Maven	2.5.1	Arrowhead Applications
Jersey	2.23.1	Arrowhead Applications
NetBeans	8.0.1	Arrowhead and FTT-SE Application
NS-3	3.25	For QoS and FTT-SE research and analysis
C	ANSI-C	FTT-SE Application
MySQL	5.1.6	Arrowhead Database
MongoDB	3.2.2	QoSMonitor Database

3.4.1 Languages & Libraries

This sub-section describes each programming language and library used during the project.

- **Java**

Java is an object-oriented programming language first released by Sun Microsystems in 1995. It is fast, secure, and reliable. From laptops to datacentres, game consoles to scientific supercomputers, cell phones to the Internet, Java is widely used [37]. Java language is very based on C and C++ languages, many of Java´s defining characteristics come from this two predecessors, which are refinements and responses to the predecessor's limitations.

What really defines Java is its portability, because to make C and C++ work in different CPUs it is needed a compiler for each type of CPU, and compilers are expensive and time-consuming to create. Therefore, back then Java founders decided to work on a portable, platform independent language that could be run every type of CPUs, leading to the creation of Java [38].

Currently the latest Java version is Java SE 8 and represents another very significant upgrade with the introduction of lambda expression. The purpose of lambda expressions is to simplify and reduce the amount of source code needed to create any functions.

- **C**

C is an imperative programming language, a very powerful tool, capable of providing low-level access the computer's memory. Developed by Bell Labs in 1972 it has been most used to convert from assembly language programs and operating systems without any performance losses. The main advantage of C language during its release was its high-level and easiness to program that could replace assembly code when creating software.

C works best for small projects where performance is important and the programmers have the time and skill to make it work in C. In any case, C is a very popular and influential language. This is mainly because of C's clean (if minimal) style, its lack of annoying or regrettable constructs, and the relative ease of writing a C compiler [39].

- **Jersey**

In order to simplify development of RESTful Web services and their clients in Java, a standard and portable JAX-RS API has been designed. Jersey RESTful Web Services framework is open source, production quality, framework for developing RESTful Web Services in Java [40].

Although Jersey is not the only JAX-RS API [41], it is the only officially developed by Oracle. Available for Java 6 and higher versions, it gives all Java web programmer's an essential tool reducing programming time and facilitating code comprehension. JAX-RS uses annotations to define the REST relevance of Java classes [42]. These annotations define the behaviour of interfaces, and facilitate the implementation.

This RESTful Web Service consists in resources, that are accessed via a common interface based on HTTP methods. There are four different HTTP methods used in REST: Get, Put, Delete and Post.

- **NS-3**

For all students and investigators with limited budget NS-3 offers a free and open network simulation platform for networking research. It is an open-source platform written in C++ capable of working in all the three main operative systems, Linux, Mac OS X and Windows. Currently is in its third version that was released in mid-2008.

In brief, NS-3 provides models of how packet data networks work and perform, and provides a simulation engine for users to conduct simulation experiments. Some of the reasons to use NS-3 include performing studies that are more difficult or not possible to perform with real systems, studying system behaviours in a highly controlled, reproducible environment, and learning about how networks work [43].

All the NS-3 simulation are based on discrete events, meaning that between consecutive events, no change in the system is assumed to occur, differentiating from continuous events which are event-based systems where the program is continuously tracking the program events over time. Therefore, discrete events systems have more performance because they have only to simulate between time slices [44].

3.4.2 Databases

This sub-section describes the two databases technologies used for the project.

- **MongoDB**

MongoDB is an open-source database developed by MongoDB, Inc. MongoDB stores data in JSON-like documents that can vary in structure. Related information is stored together for fast query access through the MongoDB query language. MongoDB uses dynamic schemas, meaning that you can create records without first defining the structure, such as the fields or the types of their values [45].

Classified as NoSQL database, MongoDB eliminates the complex object relational mapping (ORM) and instead it provides a flexible and scalable data modelling. Unlike MySQL MongoDB has the capability of working with large quantity of data without any performance cost.

- **MySQL**

MySQL offers reliable, high-performance and scalable Web-based and embedded database applications [46].
Released in 1995 MySQL is the second most used Relational Database Management System, after Oracle (RDBMS) [47]. Developed by a Swedish company and acquired in 2008 by the giant software company “Oracle”. Written in C and C++, it is an open-source software capable of working on most of existing operative systems such as Linux, Windows and OSX.

A relational database stores all the data in separate tables, which are then organized and associated using foreign keys. This means that a relational database must have a logical structure, setting up rules between different data fields, avoiding duplicate and inconsistent data. MySQL derives from SQL, which is a standardized language for users to access the databases.

The main characteristics that MySQL has are its portability, capable of working on almost every platform system, compatibility, it can work in all the top 10 most used programming languages [48] and many more, high performance.

MySQL is the world’s most popular open source database, enabling the cost-effective delivery.

3.4.3 Development

This sub-section describes the tools, such IDE’s and project management technologies, used for the development of the project solutions.

- **Maven™**

Maven is a project management tool which encompasses a project object model, a set of standards, a project lifecycle, a dependency management system, and logic for executing plugin goals at defined phases in a lifecycle. When using Maven, a description of the project must be made using a well-defined project object model, Maven can then apply cross-cutting logic from a set of shared (or custom) plugins.

- **NetBeans**

NetBeans is the official IDE for Java 8. With its editors, code analysers, and converters, it can quickly and smoothly upgrade applications to use new Java 8 language constructs, such as lambdas, functional operations, and method references [49].

Initially developed by two young students in 1996, and acquired by Sun Microsystems in 2000, Netbeans is one of most the most used IDE's [50]. More importantly than its features, Netbeans has a vast community due to its open-source platform.

Written completely in Java, it is a portable system capable of working on any machine, with Java Virtual Machine installed. Many other programming languages, such as C/C++, JavaScript and PHP are supportable by Netbeans, and its community plugins expands the IDE capabilities.

- **Eclipse**

Eclipse is a community for individuals and organizations who wish to collaborate on commercially-friendly open source software. Its projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. The Eclipse Foundation is a not-for-profit, member supported corporation that helps cultivate both an open source community and an ecosystem of complementary products and services [51].

Eclipse is an integrated development environment capable, it is free and open-source, and can be used development of application on Java, C, C++, JavaScript, PHP, Prolog, Python, R and on many other languages.

Developed by IBM in 2001, Eclipse has a very large community contributing with the development of plugins. Configurability and extensibility are the main features that Eclipse offers.

- **Git**

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with unlimited number of developers [52]. It was created by in 2005 for assisting the Linux kernel development project in order to control all the changes and therefore avoid files corruption. The time when Git was developed there were already some version control systems such as BitKeeper [53] but all of them had some flaws, like weak performance, so Git corrected all the other systems weaknesses.

Nowadays version control systems are widely used in software development and are a fundamental tool. They are essentially a code repository where all the project workers can

access and change, with monitored access. Every source code changes are tracked, along who made the change, and why they made it. Three main characteristics from Git and other system version controls are version tracking, versions restoring and team coordination.

1. **Tracking all project versions.**

Version tracking allows the recording and analysing of all the project changes, who made it and why, when a new functionality was implemented or when a bug was introduced even fixed. The project timeline is automatically made by the version control system, reducing the project manager work.

2. **Restoring previous versions.**

Being able to restore older versions of the project effectively means when the last project changes crash it, a simply undo can be done with few clicks. Knowing this makes all the project workers a lot more relaxed when working on important bits of a project [54].

3. **Coordinating Teams.**

Teams, either co-located or distributed, usually carry out resource development. Version control is central for coordinating teams of contributors. It lets one contributor work on a copy of the resources and then release their changes back to the common core when ready. Other contributors work on their own copies of the same resources at the same time, unaffected by each other's changes until they choose to merge or commit their changes back to the project. Any conflicts that arise whenever two contributors independently change the same part of a resource are automatically flagged. Such conflicts can then be managed by the contributors [55].

Chapter 4. Arrowhead Documentation/ Analysis and Implementation

4 Arrowhead Documentation/ Analysis and Implementation

4.1 Introduction

As partner of Arrowhead Framework, the team is obliged to write the technical documentation following the Arrowhead templates and guidelines. Since most of this documentation approaches both project analysis and implementation, it was decided to put in this chapter all the written documents. The distribution of the documents per section is depicted in Figure 8.

The overall Arrowhead documentation is stored in a shared blog, located in the following URL https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page. All written documentation presented in this Chapter is in the process of approval by the Arrowhead Partners, until the publication of this report.

Section 4.2 describes both QoSManager and QoSMonitor systems in a “black-box” approach. This Section contains two SysD documents. Section 4.3 describes both QoSSetup and Monitor services provided by QoSManager and QoSMonitor systems, respectively. This Section contains two SD documents. Section 4.4 describes all the interfaces provided by QoSManager and QoSMonitor systems. This Section contains six IDD documents. Section 4.5 describes all the semantic profiles of each interface messages. This Section contains six SP documents. Section 4.6 describes, in detail, the implementation of the QoSManager and QoSMonitor systems. This Section contains two SysDD documents. Section 4.7 describes the implementation of the pilot project done in a FTT-SE network. This Section contains one SoSDD document.

In overall, Sections 4.2 and 4.3 can be considered as an analysis of the project since their referred documents focus on a higher-level description. The Sections 4.4 to 4.7 can be considered as a technical description because the referred documents focus on the description of the used technologies and its implementations.

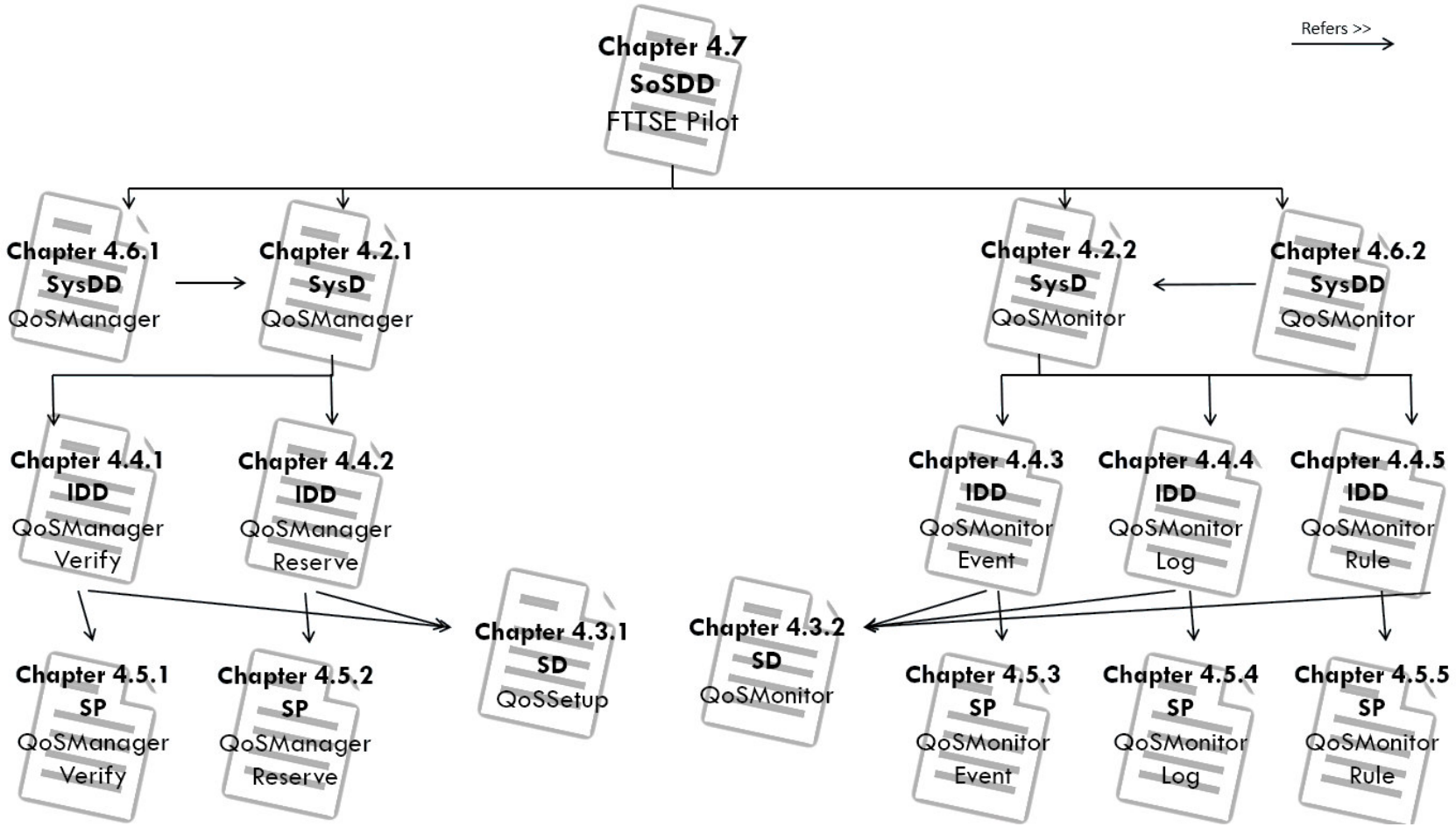


Figure 8 - Tree view of the Arrowhead written documents and their associations.

4.2 Systems Description

This section lists the goals, main services and interfaces of both QoSManager (Section 4.2.1) and QoSMonitor (Section 4.2.2) systems without describing any technology.

4.2.1 QoSManager System Description

A. System Description Overview

The QoSManager purpose is to verify and manage QoS for service fruition.

This document regards a design considering that the QoSSetup and Monitor services are produced by two different systems. Thus, QoSManager system produces the QoSSetup service, QoSMonitor produces the Monitor service.

Acting as a support system for the Orchestrator system, the QoSManager provides services to enable the configuration of systems and network actives. To this aim, the QoSManager system produces the QoSSetup service, which allows two functions to be invoked. First, a verification of a requested QoS is done with the support of a specific communication protocol algorithm. Second, the QoSManager can configure all the necessary network actives and devices to guarantee the selected QoS with the support of specific communication protocol drivers. To assure the fulfilment of the QoS during execution time, the QoSManager consumes the Monitor service provided by the QoSMonitor system.

A high level of the QoSManager system is depicted in Figure 9.

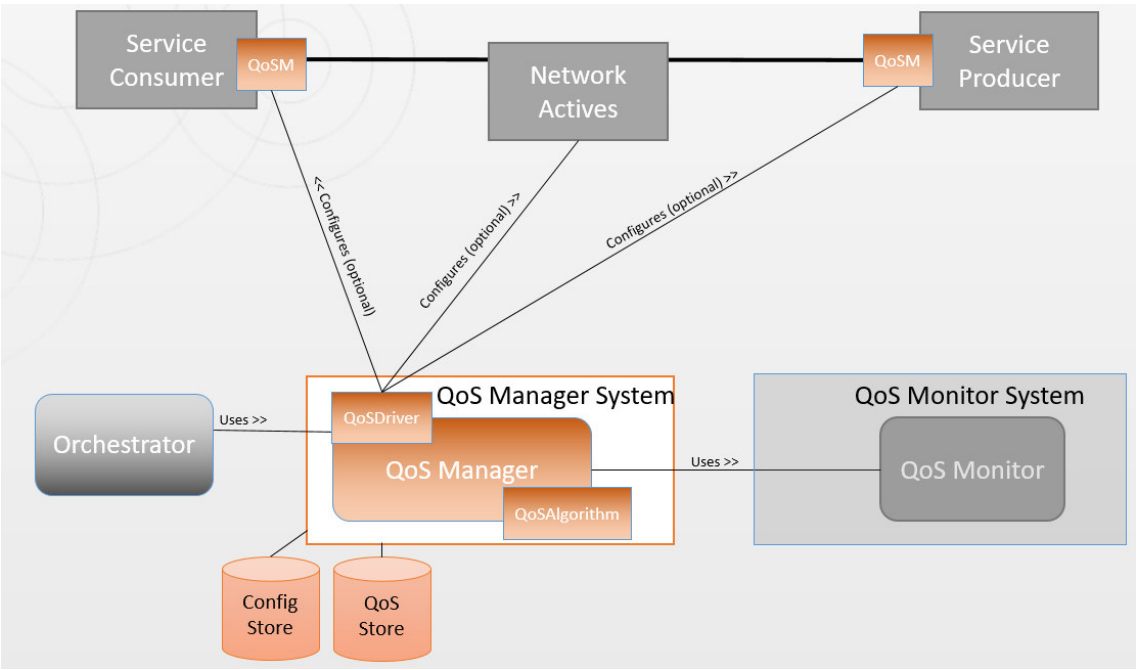


Figure 9 - Overview of the QoSManager System.

a. Domain Model

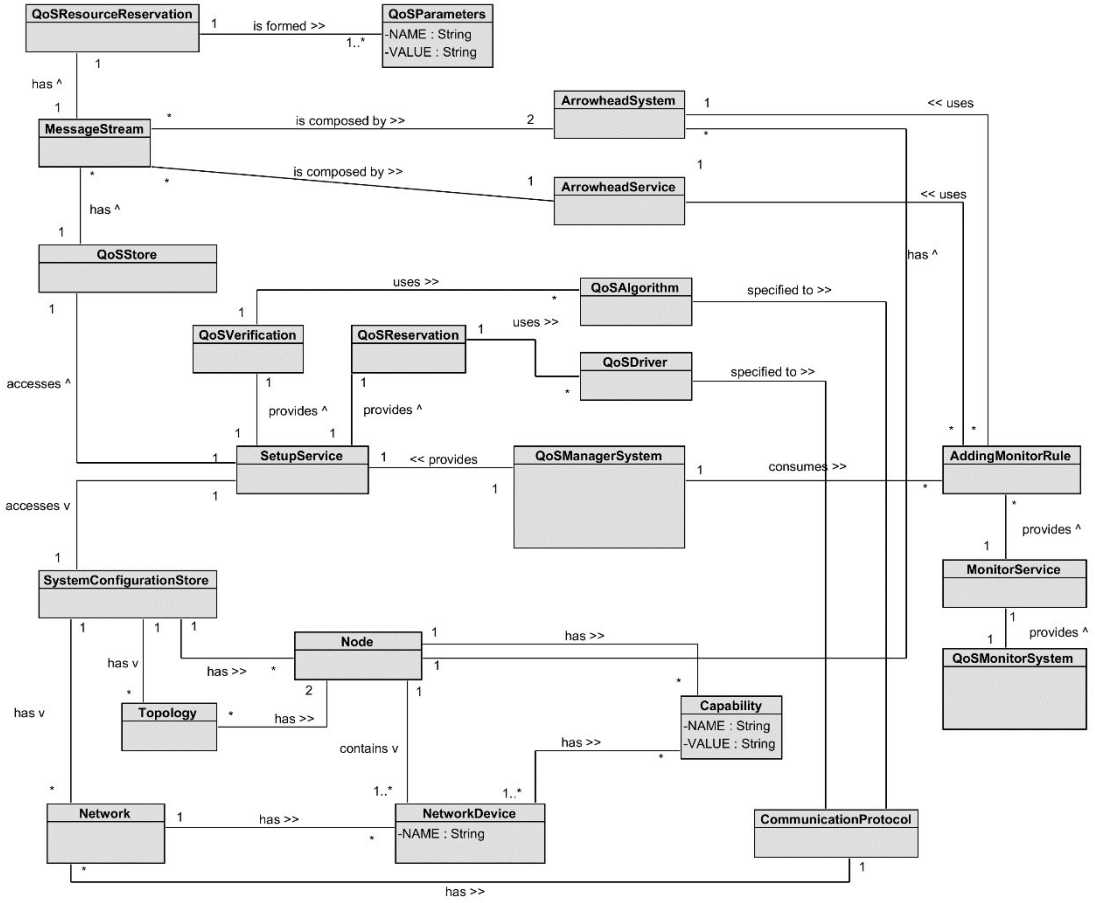


Figure 10 - Domain Model of the QoSManager system.

Using Figure 10 has a guideline, the Domain Model for the QoS Manager can be described as:

The `QoSManagerSystem` provides only one service, the `QoSSetupService`. This service provides two functionalities, the `QoSVerification` and `QoSReservation`.

The `QoSVerification` uses multiple `QoSAlgorithms`, each one is specific to a communication protocol. The `QoSReservation` is similar to `QoSVerification` and instead of using `QoSAlgorithms` it uses `QoSDrivers`.

To guarantee enough information to its operations, the `QoSSetup` accesses two stores, `QoSStore` and `SystemConfigurationStore`. The `QoSStore` has various `MessagesStreams`, each one is composed by two `ArrowheadSystems`, one `ArrowheadService` and one `QoSResourceReservation`. The `QoSResourceReservation` is formed by multiple `QoSParameters`, and each parameter has a name and a respective value (i.e. “bandwidth” as name and “1500” as value).

Regarding to the `SystemConfigurationStore`, it has various `Networks`, `Nodes`, and `Topology`. A `Node` contains multiple `NetworkDevice`, `ArrowheadSystem`, `Capability`. The `Topology` entity contains two “neighbour” `Node` objects that have a direct communication. Such as the `Node`, a `NetworkDevice` can also have multiple

`Capability` objects and is grouped in a `Network` entity. A `Network` has a `CommunicationProtocol` that is specific to each `QoSDriver` and `QoSAlgorithm`.

The `QoSManagerSystem` consumes the `AddingMonitorRule` operation, provided by the `MonitorService` that in turn is provided from the `QoSMonitorSystem`. `AddingMonitorRule` uses two `ArrowheadSystem`s and one `ArrowheadService`.

b. Mandatory Properties Files

For storing QoS reservations and configurations of network actives and devices, the QoSManager uses a relational database, for example a MySQL database. In order to function properly, the QoSManager must be configured with information regarding the databases and the QoSMonitor Uri.

The database properties file named hibernateQoS.properties, contains one `databaseurl` parameter which refers to the URL of the QoSStore database, e.g. "jdbc:mysql://localhost:3306/qos_store":

```
connectionQoS.url=[databaseurl]
```

The database properties file named hibernateSCS.properties contains one `databaseurl` parameter which refers to the URL of the SystemConfigurationStore database, e.g. "jdbc:mysql://localhost:3306/system_configuration_store":

```
connectionSCS.url=[databaseurl]
```

The Monitor properties file named monitor.properties contains one `url_to_monitor` parameter, which refers to the URL of the QoSMonitor system, e.g. "http://192.168.1.1:8080/qosmonitor":

```
monitor.uri=[url_to_monitor]
```

B. Use-Cases

QoSManager provides two functionalities, the Verification of a QoS and the Reservation of a QoS.

The first use-case, Verification of QoS requirements, involves computing whether certain QoS requirements are feasible, with the use of an algorithm. The same algorithm, which is specific to a communication protocol, must take into account the configurations and capabilities of the SoS and the current reservations over the network actives and devices.

The second use-case, Reservation of a QoS, regarding configuring a stream connecting the producer and consumer accordingly to the requested QoS. Using a set of QoSDrivers, each one specific to a communication protocol, the QoSManager configures all the network actives and devices to accommodate the new service.

a. Functional Requirements

The QoSManager has two major functional requirements:

- Verification of a QoS: By receiving a set of orchestrated services and QoS requirements, using a QoS algorithm, the QoSManager must decide if the

requirements are feasible. The QoS algorithm, which is specific to a communication protocol, verifies a request by taking into account the configurations and capabilities of the system of systems and the current reservations over the network devices.

- Reservation of a QoS: Whenever a service is requested with QoS requirements, the stream connecting the producer and consumer must be configured by the QoSManager, to accommodate the new service. This is accomplished using the QoSDriver, which is specific to a communication protocol.

b. Non-Functional Requirements

Regarding the non-functional requirements, five must be highlighted:

- Availability: The system must be online and accessible as long as possible, 24 hours per day and 365 days per year.
- Integrity: Dealing with sensible industrial requests the system must always report any execution error into its database for further analysations and improvements.
- Interoperability: The developed system must be able to be easily migrated to other Arrowhead Frameworks, since there are more than one (ex. Hungary [56], BNearIT). At best during the migration of frameworks, there should be no adaptation or even logic model changes.
- Performance: The system and its algorithms must have the shortest execution time therefore an advanced hardware and good programming code should be adopted.
- Extensibility: The System must support new communication protocols and different algorithms therefore it should be generic.

c. Use-Cases Execution Flow

Table 6 - Use Case 1 execution flow.

Use-Case 1: Verification of QoS
ID: 1
Brief description: The use-case describes the sequence of steps for the verification of the service consumer requested QoS.
Primary actors: Orchestrator
Secondary actors:
Preconditions: - The Service Consumer and Service Provider network information must already be stored at the System Configuration Store.
Main flow: 1- A Service Consumer contacts the Orchestrator, orchestrating a service, located on a Local Cloud, with a Quality of Service. 2- The Orchestrator requests the QoSManager to verify the feasibility of the QoS on the consumer and producer stream. 3- Using a specific network algorithm the QoSManager verifies if the requested QoS is or not possible giving a reject motivation back to the Orchestrator. 4- The Orchestrator gives all possible producers that can provide the requested service with QoS.
Post conditions: -
Alternative flows: 3*- There is no sufficient information on the System Configuration Store to verify if the requested QoS is feasible and therefore the QoSManager sends a warning.

Table 7 - Use Case 2 execution flow.

Use-Case 2: Reservation of QoS
ID: 2
Brief description: The use-case describes the sequence of steps for the storage of events into a database or a local file.
Primary actors: Orchestrator
Secondary actors: Producer System, Consumer System
Preconditions: - The Service Consumer and Service Provider network information must already be stored at the System Configuration Store. - This use-case comes only after UC1.
Main flow: 1- A Service Provider registers a service. 2- A Service Consumer contacts the Orchestrator, orchestrating a service, located on the Local Cloud, with a requested Quality of Service. 3- The Orchestrator requests the QoSManager to reserve a message stream between the Service Consumer and the Service Provider with the QoS desired. 4- The QoSManager, using the QoSDriver, setups the necessary configurations between the Service provider and consumer to meet the requested QoS. 5- After the configuration the QoSManager responds to the Orchestrator if the configuration was or not successful.
Post conditions:
Alternative flows: 4** - There is no sufficient information on the System Configuration Store to the QoS setup the QoSManager sends a warning.

C. Diagrams

Verification of QoS

This diagram is already illustrated in the Section 4.3.1.

Reservation of QoS

This diagram is already illustrated in the Section 4.3.1

D. Application Services

Figure 11 depicts a representation of the set of services provided (produced) and consumed by the QoSManager system. This system produces one service, the QoSSetup.

The QoSManager system consumes two services, namely: the QoSMonitor Monitor service, the Authentication core system and the Service Registry (SR).

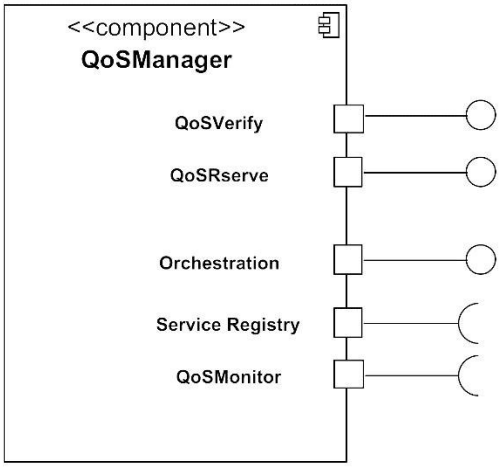


Figure 11 - Services provided and consumed by the QoSManager.

a. Produced Services

Table 8 - Pointers to IDD documents..

Service	IDD Document Reference
QoSVerify	Section 4.4.1
QoSReserve	Section 4.4.2

With the support of an algorithm, the QoSVerify service calculates if a certain QoS is feasible depending on the network topology, capabilities and current QoS reservations. On the other hand, the QoSReserve manages the reservations used to guarantee QoS to service fruition with the support of a driver.

b. Produced Services

Table 9 - Pointers to IDD documents..

Service	IDD Document Reference
QoSMonitor-Monitor	Section 4.4.5
ServiceRegistry	https://forge.soa4d.org/svn/arrowhead/WP7/Task%207.3/Working/AITIA/Arrowhead_G3.2_QuickStart.zip
Orchestration	https://forge.soa4d.org/svn/arrowhead/WP7/Task%207.3/Working/AITIA/Arrowhead_G3.2_QuickStart.zip

The description of the QoSMonitor, Service Registry and Orchestrator can be found in their respective references.

E. Security

This chapter defines high-level security principles the system needs to follow on a non-technical, generic level.

a. Security Objectives

The QoSManager system provides secure HTTP communications using SSL/TLS as s security protocol. This is to keep sensible information only readable by a restricted group of recipients with the usage of certificates.

4.2.2 QoSMonitor System Description

A. System Description Overview

The QoSMonitor system provides functionality for the monitoring of performance between two systems, one consuming a service provided by the other, in a given Arrowhead compliant installation.

This document regards a design considering that the QoSSetup and Monitor services are produced by two different systems. Thus, QoSManager system produces the QoSSetup service, QoSMonitor produces the Monitor service. It uses a set of plugins (extensions of the QoSMonitor system) deployed in producer and consumer systems to capture communication information. The QoSManager *(section 4.2.1)* sends rules that are used to specify QoS requested by the service consumer when the orchestration process is performed. The QoSMonitor *(section 4.2.2)* uses the Monitor database to store rules sent by the QoSManager. Afterwards, the QoSMonitor receives monitor logs from the plugins and uses the information to present communication state over time in graphic form. The monitor logs are stored in the Monitor database, with each having a reference to a specific monitor rule. Furthermore, QoSMonitor uses the same data to verify QoS against a rule identified by the same systems as given by the monitor log, thus sending events to the EventHandler system if QoS requirements are not met. The QoSMonitor uses the Monitor database to access the rules defined in each monitor log. In addition, it gives the possibility to an Arrowhead compliant system to send errors, which are then transformed into a maximum level severity event. A high-level view of the QoSMonitor system is depicted in Figure 12.

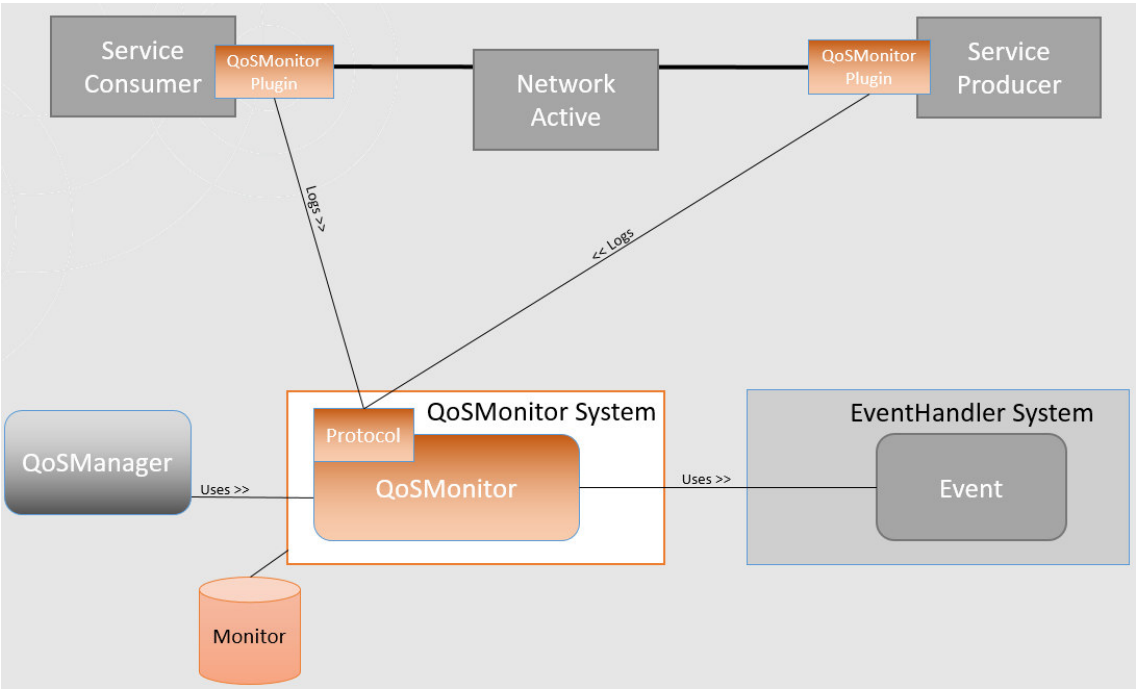


Figure 12 - QoSMonitor High Level Component Diagram.

a. Domain Model

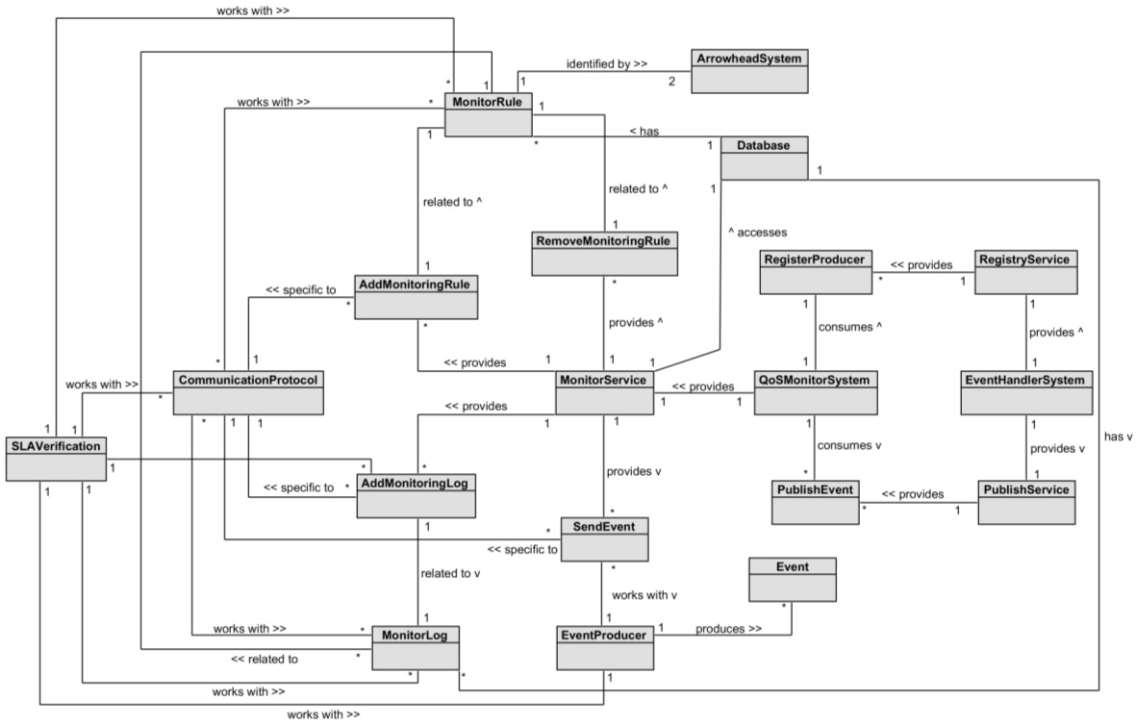


Figure 13 - Domain Model of the QoSMonitor system.

The domain model of the QoSMonitor system, shown in Figure 13 , can be described as follows:

The `QoSMonitorSystem` provides the `Monitor` service that in turn provides four functionalities, namely `AddMonitoringRule`, `RemoveMonitoringRule`, `AddMonitoringLog` and `SendEvent`. Since the `QoSMonitorSystem` must be capable of working with various `CommunicationProtocol` like "FTT-SE" or even "ZigBee", all of these functionalities use various types of `CommunicationProtocol` to execute its logic. The `CommunicationProtocol` is defined by the user of the functionality each time it uses it.

A `Database` is used to store `MonitorRule` and `MonitorLog` and to support the `Monitor` service in the various use cases. The `SLAVerification` (in section 4.4 there is an example of an SLA) is responsible for verifying the QoS of a `MonitorLog`, against a `MonitorRule` using a specific `CommunicationProtocol`. It also works with the `EventProducer` to create an event whenever information of a `MonitorLog` does not comply with that specified in the respective `MonitorRule` (for example, if a `MonitorRule` exists specifying that a consumer can use 100 MB/s of bandwidth but the `MonitorLog` shows that 110MB/s were used then this is a breach in QoS). The `QoSMonitorSystem` consumes the `RegisterProducer` and `PublishEvent` functionalities from the `RegistryService` and `PublishService` services respectively. The `EventHandlerSystem` provides both services.

b. Mandatory property files

The database properties file named mongodb.properties contains a `database_connectionString` **parameter, which refers to the connection path**

of the Monitor database, e.g. "mongodb://192.168.60.74:27017"; the `database_name` **parameter which refers to the name of the database, e.g. "mongodb**"

```
connectionString=[database_connectionString]

database=[database_name]
```

The ServiceRegistry properties file named serviceregistry.properties, contains a `comma_separated_list_of_ServiceRegistry` **parameter, which refers to the Arrowhead Frameworks where the user wants to register the QoSMonitor as a service provider, e.g. "Hungary"**

```
registry.option=[comma_separated_list_of_ServiceRegistry]
```

The EventHandler services properties file named eventhandler.properties contains a `url_to_orchestration` **which refers to the Orchestrator System where the EventHandler system is located, e.g. http://localhost:8080/core/orchestrator; the** `eventhandler_serviceGroup` **parameter refers to the service group of the Event Handler system, e.g. "supportsystems"; the** `registry_service_definition` **parameter refers to the definition of the registry service, e.g. "registry"; the** `publish_service_definition` **refers to the definition of the publish service, e.g. "publish".**

```
orchestrator.orchestration.uri=[url_to_orchestration]

eventhandler.servicegroup=[eventhandler_serviceGroup]

eventhandler.registryservicedefinition=[registry_service_de
finition]

eventhandler.publishservicedefinition=[publish_service_defi
nition]
```

B. Use-cases

The QoSMonitor is registered and authenticated as an Arrowhead compliant system in the ServiceRegistry. It is considered as given that the systems being monitored were also registered.

Table 10 - Add Monitor Rule Use-Case Description

Use-Case 1: Add Monitor Rule
ID: 1
Brief description: Add monitor rule about requested Quality-of-Service between two systems.
Primary actors: QoSManager *(Section 4.2.1).*
Secondary actors: MongoDB Manager.
Preconditions: At least one monitor parameter (for example bandwidth).
Main flow: 1- QoSManager (*Section 4.2.1)* sends a monitor rule to the system. 2- System validates the monitor rule. 3- Saves monitor rule in the database, identified by the given systems.
Post conditions: Monitor rule stored in the database.
Alternative flows: 2 The payload is not valid. 2.1 Returns bad request as response. 3 A rule identified by the same given systems already exists in the database. 3.1 The rule is deleted. 3.2 The new rule is saved.

Table 11 - Remove Monitor Rule Use-Case Description

Use-Case 2: Remove Monitor Rule
ID: 2
Brief description: Removes monitor rule about requested Quality-of-Service between two systems.
Primary actors: QoSManager *(Section 4.2.1).*
Secondary actors: MongoDB Manager.
Preconditions: -
Main flow: 1- QoSManager *(Section 4.2.1*) sends a monitor rule to the system. 2- System checks existence of rule in the database. 3- Removes monitor rule in the database, identified by the given systems.
Post conditions: Monitor rule deleted in the database.
Alternative flows: 2 A rule identified by the given systems does not exist in the database. 2.1 Returns not found as response.

Table 12 - Add Monitor Log Use-Case Description

Use-Case 3: Add Monitor Log
ID: 3
Brief description: Add monitor log with information regarding communications between two systems, service producer and service consumer.
Primary actors: MonitorPlugin of service prosumer.
Secondary actors: MongoDB Manager
Preconditions: At least one monitor parameter (for example bandwidth). Rule identified by the given systems must exist in the database.
Main flow: 1- MonitorPlugin sends monitor log.

2- System validates the payload. 3- Checks for a monitor rule identified by the given systems. 4- Saves monitor log in the database, identified by the given timestamp. 5- Validates Quality-of-Service by comparing monitor log information against rule specifications.
Post conditions: Monitor log stored in the database
Alternative flows: 2.1- The payload is not valid. 2.2- Returns bad request as response. 3.1- A rule identified by the given systems does not exist. 3.2- Returns not found as response. 4.1- Checks that the Quality-of-Service requirements were not met. 4.2- Sends event to the EventHandler system.

Table 13 - Send Event Use-Case Description

Use-Case 4: Send Event
ID: 4
Brief description: Forwards service error descriptions as events to the EventHandler system. Normally, these events are not related to Quality-of-Service violations.
Primary actors: Arrowhead compliant system
Secondary actors: -
Preconditions: Valid payload
Main flow: 1- Arrowhead compliant system sends a service error to the system. 2- System validates the payload. 3- Creates an event with information received. 4- Sends event to the EventHandler .
Post conditions: -
Alternative flows: 2.1- The payload is not valid. 2.2- Returns bad request as response.

C. Diagrams

Add Monitor Rule Sequence Diagram

This diagram is already illustrated in the Section 4.3.2.

Remove Monitor Rule Sequence Diagram

This diagram is already illustrated in the Section 4.3.2.

Add Monitor Log Sequence Diagram

This diagram is already illustrated in the Section 4.3.2.

Send Event Sequence Diagram

This diagram is already illustrated in the Section 4.3.2.

D. Application Services

Figure 14 depicts a representation of the service provided and consumed by the QoSMonitor system. This system produces one service, the Monitor service. The QoSMonitor system consumes four services, namely: the EventHandler Registry and Publish services, the ServiceRegistry and the Orchestration.

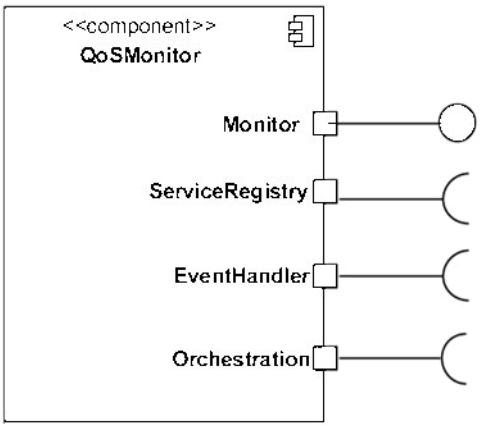


Figure 14 - Component Model.

Produced Services

Table 14 - Pointers to IDD documents

Service	SD Document Reference
Monitor	Section 4.3.2.

The Monitor service is used to add and delete monitor rules, add monitor logs and send service errors as events to the EventHandler system. Monitor rules specify Quality-of-Service

(QoS) requested in the Orchestration process. Adding monitor logs also validates QoS against monitor rule specifications. A service error is an occurrence that an Arrowhead compliant system is interested in sending to the EventHandler.

Consumed Services

Table 15 - Pointers to IDD documents

Service	IDD Document Reference
EventHandler Registry	https://forge.soa4d.org/svn/arrowhead-f/3_Core%20Systems%20and%20Services/2_Support%20Core%20Systems%20and%20Services/5_Eventhandler%20system/Documetation/Arrowhead%20IDD%20EventHandlerRegistry%20REST_WS-TLS-XMLv1.0.docx
EventHandler Publish	https://forge.soa4d.org/svn/arrowhead-f/3_Core%20Systems%20and%20Services/2_Support%20Core%20Systems%20and%20Services/5_Eventhandler%20system/Documetation/Arrowhead%20IDD%20EventHandlerPublish%20REST_WS-TLS-XMLv1.0.docx
ServiceRegistry	https://forge.soa4d.org/svn/arrowhead/WP7/Task%207.3/Working/AITIA/Arrowhead_G3.2_QuickStart.zip
Orchestration	https://forge.soa4d.org/svn/arrowhead/WP7/Task%207.3/Working/AITIA/Arrowhead_G3.2_QuickStart.zip

The description of the EventHandler Registry, EventHandler Publish, ServiceRegistry and Orchestration services can be found in their respective references.

E. Security

This chapter defines high-level security principles the system needs to follow on a non-technical, generic level.

a. Security Objectives

Objectives for this system cover the well-known *AIC* [57]-triad (availability, integrity, confidentiality). The attribute *availability* ensures that information is available when it is needed, and thus the system must be always on. *Integrity* refers to the authorized modification of data within a given system, and it is granted by limiting to this system the write capabilities on the NoSQL database of the log data. *Confidentiality* seeks to ensure that information can only be read by authorized subjects, and must be applied to all interactions with the QoSMonitor system.

4.3 Services Description

This section provides an abstract description of what is needed for systems to provide and/or consume the two services, QoSSetup (Section 4.3.1) and QoSMonitor (Section 4.3.2), provided by the QoSManager (Section 4.2.1) and QoSMonitor (Section 4.2.2) systems, respectively.

4.3.1 QoSSetup Service Description

A. Overview

This document describes the QoSManager QoSSetup service, including its abstract interfaces and its abstract information model. The QoSSetup is the only service provided by the QoSManager system. The purpose of the QoSSetup service is calculating if a certain QoS request is feasible by taking into account the configurations and capabilities of the system of systems and the current reservations over the network devices. If a service consumption is compatible with the requested QoS, the QoSSetup can be used to configure a stream connecting that consumer and a producer according to the requested QoS.

If a consumer wants to consume a service with QoS guarantees, it must make a request to the Orchestrator system with both the functional requirements for the service – which defines which services the Orchestrator has to put together – and with the non-functional requirements – which will be used by the Orchestrator in its interactions with the QoS Manager. The QoSSetup service is consumed by the Orchestrator only, and in this sense, the QoSSetup acts as a plugin for the Orchestrator.

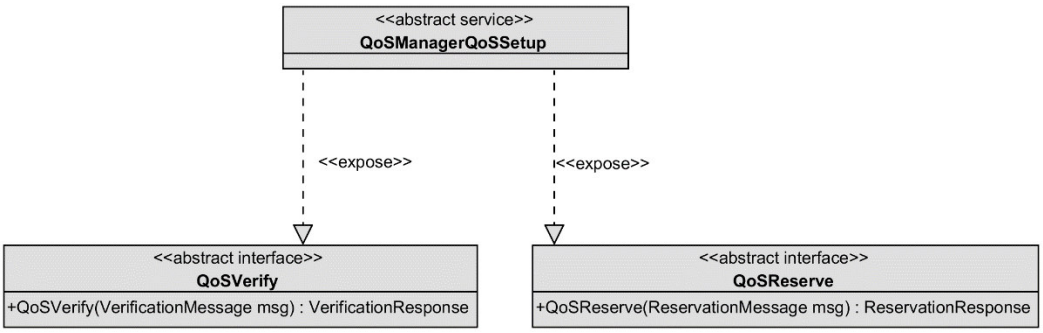


Figure 15 - QoSManager QoSSetup Overview.

The QoSManager QoSSetup service is a core service.

B. Abstract Interfaces

The QoSManager QoSSetup service exposes two interfaces, namely the QoSVerify and QoSReserve interfaces.

a. QoSManager QoSVerify

During a service orchestration request with QoS, the Orchestrator core system calls the QoSVerify to determine if the requested QoS is feasible. The interface QoSVerify contains one single function.

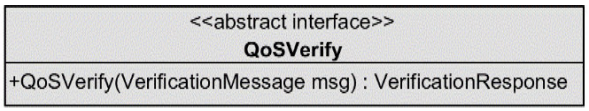


Figure 16 - QoSVerify Interface.

Function:

- **QoSVerify:** To verify if a requested QoS is feasible the QoSManager calls the QoSVerify function. The function will return true or false with a reason, true means that is possible to consume a service with QoS. Whenever the service consume is not possible the function will return false along with a reason, which is a parameter that can have three values: Always means that the requested QoS is not possible under no circumstance, since it would be in excess even if the SoS was not executing any other service; Temporary means that the condition is temporary and it depends on current resource reservations in the SoS; Combination means that the QoS is feasible, but only with a different orchestration of the services, for example by swapping two service producers that are serving two service consumers. This latter condition is usually related to the network topology.

Sequence Diagram:

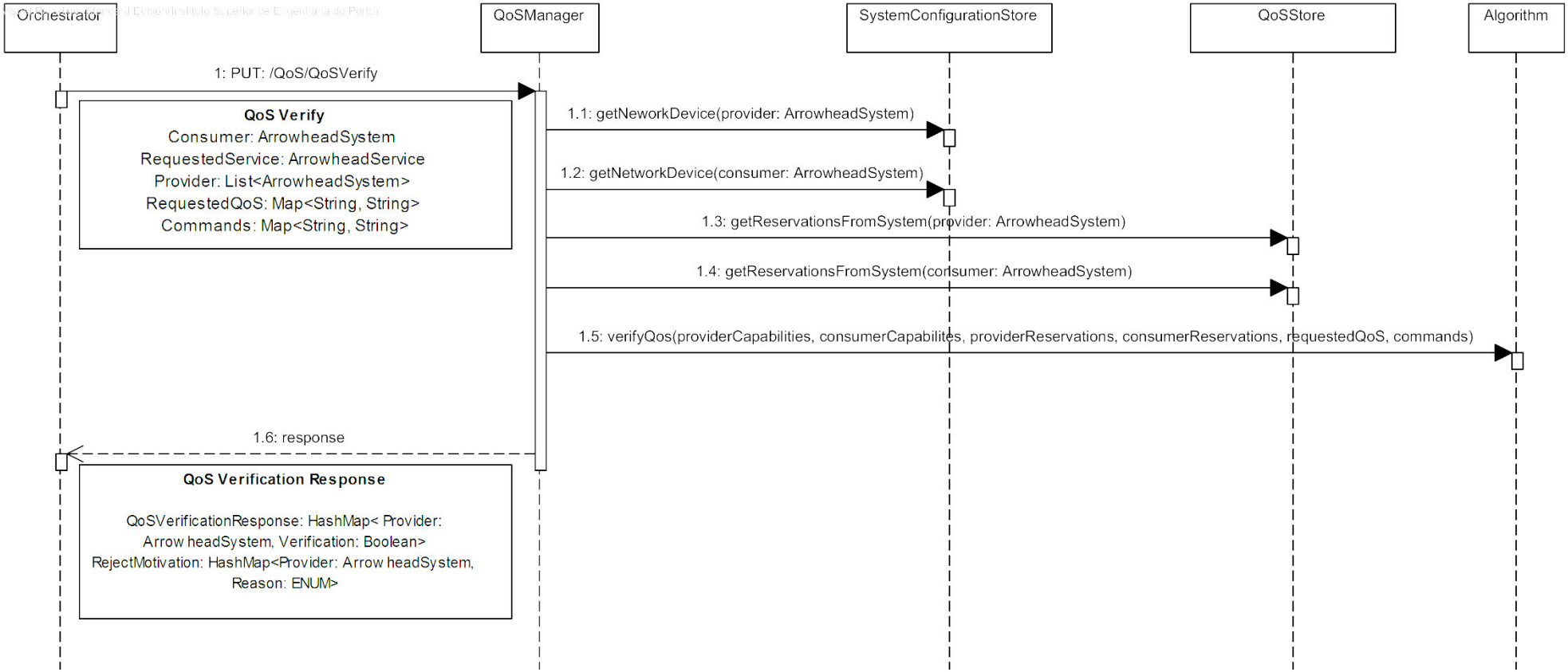


Figure 17 - High Level Sequence Diagram of QoSSetup Service QoSVerify interface.

b. QoSManager QoSReserve

After the QoS verification, the Orchestrator core system can call the QoSReserve to configure a stream between the consumer and the producer. The QoSReserve interface contains one single function.

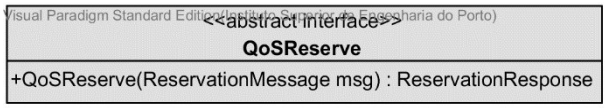


Figure 18 - QoSReserve Interface.

Function:

- **QoSReserve**: The QoSReserve function is used by the Orchestrator to set up QoS. It receives an orchestrated service and a consumer, it will update the reservations that are active in the SoS, and it will deploy QoS configuration to devices and network actives involved in the service fruition.

Sequence Diagram:

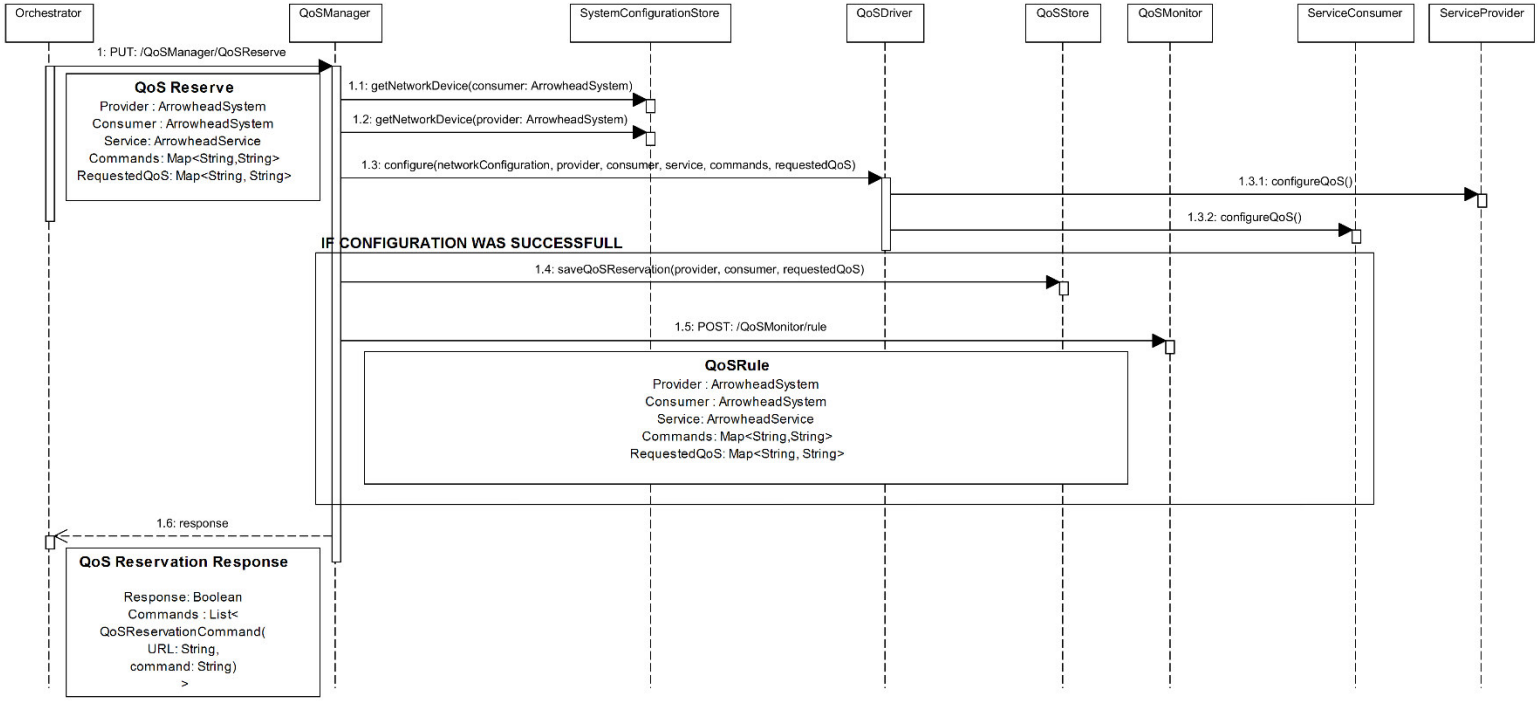


Figure 19 - High Level Sequence Diagram of QoSSetup Service QoSReservation interface.

C. Abstract Information Model

a. Service Information Data

Table 16 - Data type description

Field	Description
VerificationMessage	It contains a list of system providers, one consumer, a requested service, a requested QoS and a map of commands.
VerificationResponse	It contains a Boolean per a service provider, and map containing the reasons why in some cases the QoS is not possible.
ReservationMessage	It contains one service consumer, provider, a requested service, a requested QoS and a map of commands.
ReservationResponse	It contains a Boolean and a list of the configured stream parameters.

VerificationMessage

consumer is the system that consumes a service.
requestedService is the service to be consumed by the consumer and provided by the producer.
providers is a list of providers, each element is a system like the consumer , that is a possible producer of the selected service.
requestedQoS is a map containing a set of QoS parameters (i.e. bandwidth).
commands is a map containing a set of configuration parameters.

VerificationResponse

qosVerificationReponse is a map containing a Boolean per a service provider.
rejetcMotivation is a map containing a reason per a service provider.

ReservationMessage

provider is the system that provides a service.
consumer is the system that consumes a service.
service is the service to be consumed by the consumer and provided by the producer.
commands is a map containing a set of configuration parameters.
requestedQoS is a map containing a set of QoS parameters (i.e. bandwidth).

ReservationResponse

response is a Boolean about the success of the configuration.
commands is a map containing a set of the configurations done by the QoSReserve.

b. Non-functional Requirements

The QoSManager QoSSetup must satisfy five non-functional requirements:
Availability: The system must be online and accessible as long as possible, 24 hours per day and 365 days per year.
Integrity: Since it deals with sensible industrial requests, the service must always report any execution error into its database for further analysis and improvements.
Interoperability: The developed service must be able to be easily implemented by new systems, and must be easily migrated to other Arrowhead Frameworks (ex. Hungary [56], BNearIT). At best, during the migration of frameworks, there should be no adaptation nor logic model changes.

Performance: The system and its algorithms must have the shortest execution time possible. Thus, the service must rely on advanced hardware and good programming code.
Extensibility: The service must be able to support large number of requests, thus its implementations must be able to leverage deployment into computational clouds and other elasticity enablers.

4.3.2 Monitor service Description

A. Overview

This document describes the QoSMonitor Monitor service, including its service interfaces and its abstract information model. The Monitor is the only service provided by the QoSMonitor. The purpose of the Monitor is to compare performance values of communication between two Arrowhead compliant systems, one service producer and one service consumer, against Quality-of-Service (QoS) contracts previously defined by the QoSManager system. It sends events to the EventHandler system if such obligations are not fulfilled.
It also allows the possibility for an Arrowhead compliant system to notify errors, which semantics is expressed by means of maximum severity level events.

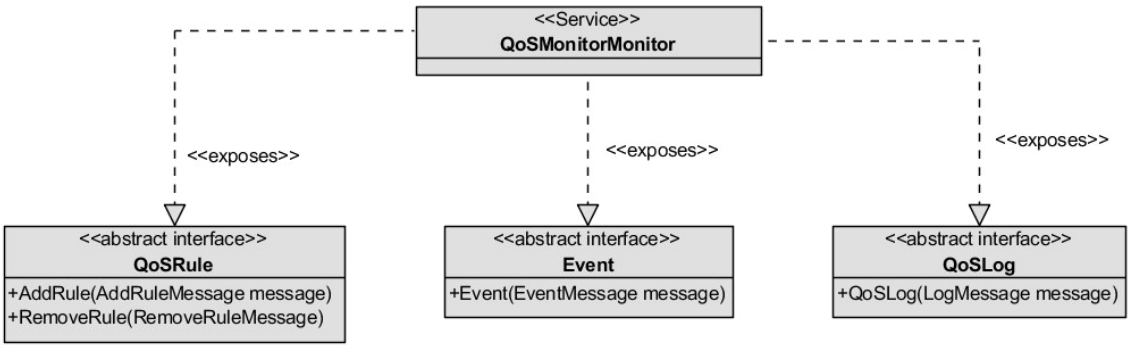


Figure 20 - QoSMonitor Monitor Overview.

The QoSMonitor Monitor service is a core service.

B. Abstract Interfaces

The QoSMonitor Monitor service exposes three interfaces, namely the QoSRule, the QoSLog and the Event interfaces

a. QoSMonitor QoSRule

During a service orchestration request with QoS, the Orchestrator core system calls the QoSVerify to determine if the requested QoS is feasible. The interface QoSVerify contains one single function.

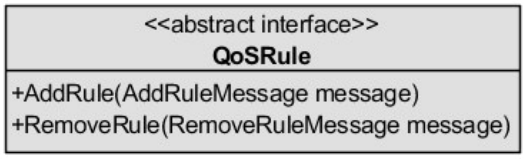


Figure 21 - QoSRule Interface

Functions:

- **AddRule:** The AddRule function is used to add a monitor rule. It receives the communication protocol, a service producer and a service consumer, a check value for soft real time monitoring as well as a map

specifying what needs to be monitored with the requested value (i.e. bandwidth = 100 Mbps). A rule must be associated to a unique id.

- **RemoveRule:** The RemoveRule function exists for removing rules. It receives a service producer and a service consumer.

Sequence Diagram:

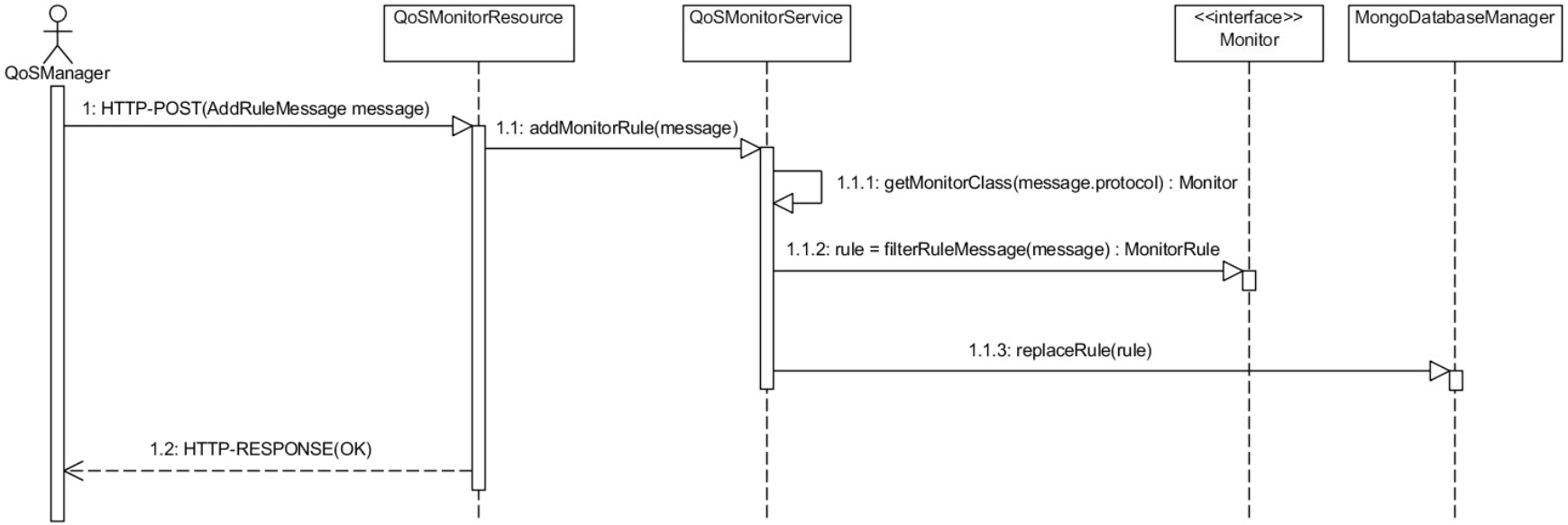


Figure 22 - High Level Sequence Diagram of Monitor Service AddRule Interface.

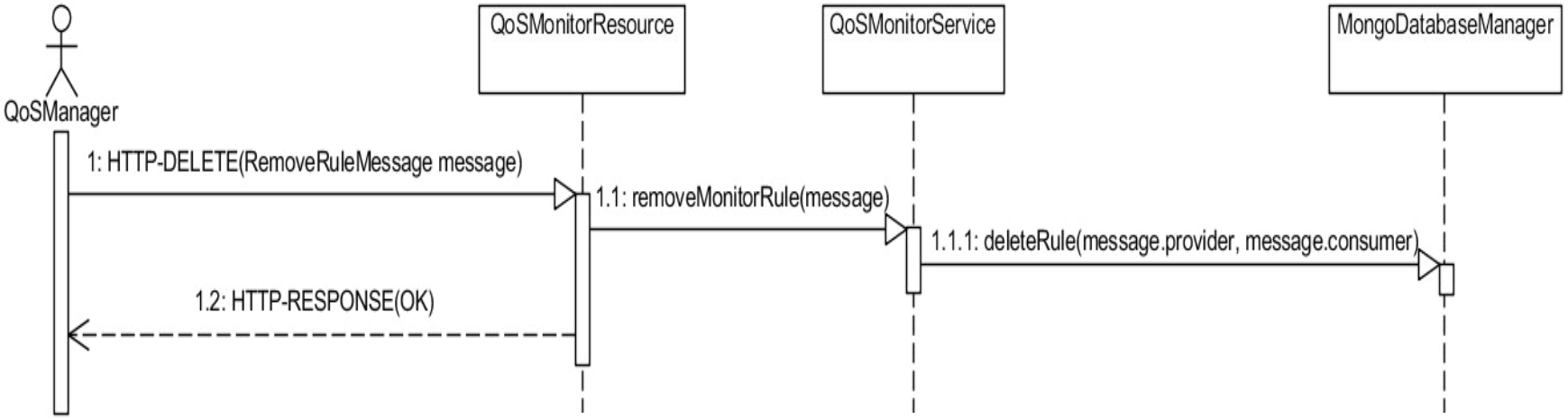


Figure 23 - High Level Sequence Diagram of Monitor service RemoveRule Interface.

b. QoSMonitor QoSLog

After This interface is used to send messages containing logs regarding communication monitoring to the QoSMonitor. The log activities are intended to be performed in the producer and the consumer systems. Upon the usage of this interface, the system saves the log as well as verifies if the respective rule specifications are being met. If not, an event is sent to the EventHandler with this information.

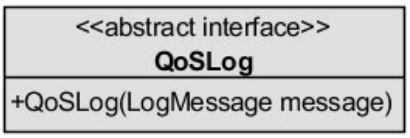


Figure 24 - QoSLog Interface

Function:

- **AddLog**: The AddLog function is used by the MonitoringPlugins in the provider and consumer systems to add monitor log data to the QoSMonitor. The function receives the communication protocol being used, the service producer and the service consumer, the timestamp as well as a map specifying what was monitored and the data value (i.e. bandwidth = 152 Mbps).

Sequence-Diagram:

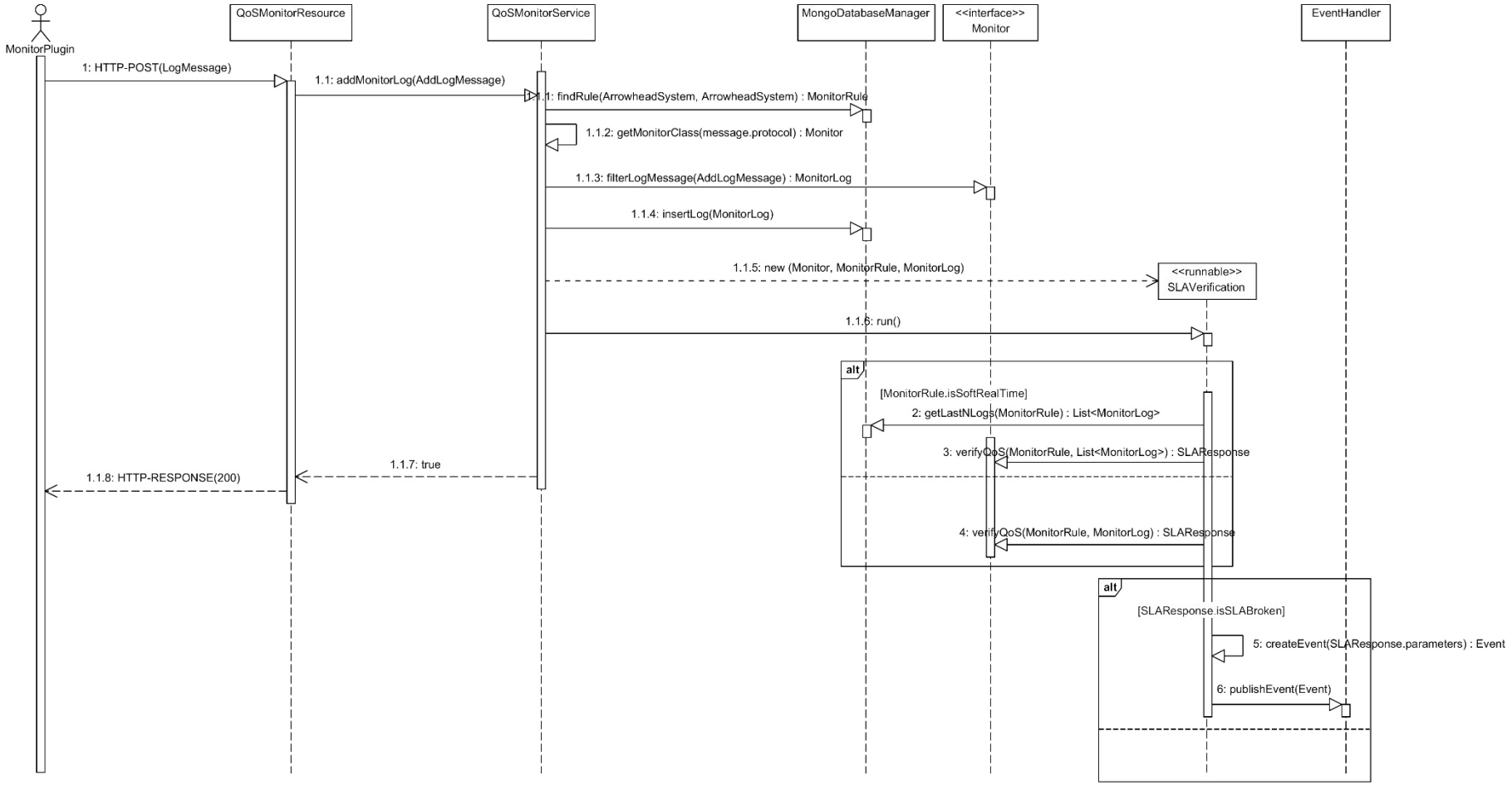


Figure 25 - High Level Sequence Diagram of Monitor service AddLog Interface

c. Event

This interface gives the possibility of sending maximum severity level events to the EventHandler. An error message is transformed into an event and sent to the aforementioned system.

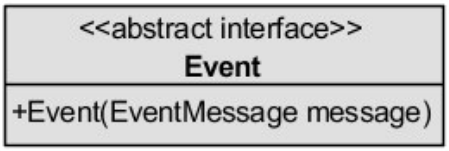


Figure 26 – Event Interface.

Functions:

- **SendEvent**: The SendEvent functions is used by Arrowhead compliant services as a means of sending events to the EventHandler system. It receives the communication protocol, the system using the function, an error message and a map of protocol specific handling information.

Sequence-Diagram:

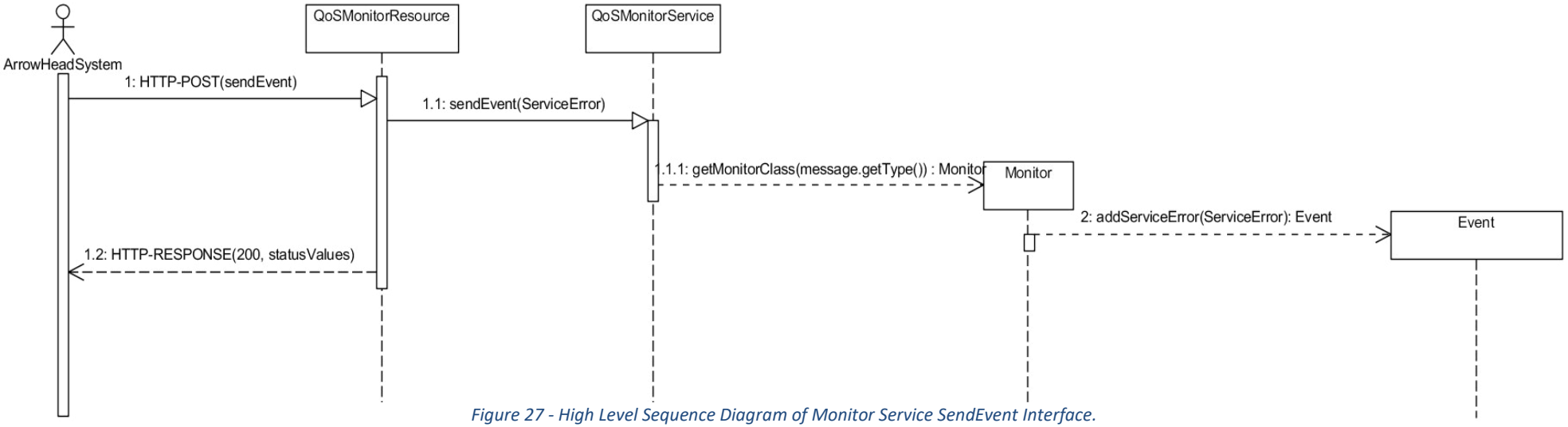


Figure 27 - High Level Sequence Diagram of Monitor Service SendEvent Interface.

C. Abstract Information Model

a. Service Information Data

Table 17 - Data type description

Field	Description
AddRuleMessage	It contains the communication protocol, a service producer and a service consumer, a check value for soft real time monitoring as well as a map specifying what needs to be monitored with the requested value.
RemoveRuleMessage	It contains a service producer and a service consumer.
LogMessage	It contains the communication protocol being used, the service producer and the service consumer, the timestamp as well as a map specifying what was monitored with a designated value.
EventMessage	It contains the communication protocol, the system using the function, an error message and a map of protocol specific handling information.

AddRuleMessage

consumer is the system that consumes a service.
requestedService is the service to be consumed by the consumer and provided by the producer.
providers is a list of providers, each element is a system like the consumer , that is a possible producer of the selected service.
requestedQoS is a map containing a set of QoS parameters (i.e. bandwidth).
commands is a map containing a set of configuration parameters.

RemoveRuleMessage

qosVerificationReponse is a map containing a Boolean per a service provider.
rejetcMotivation is a map containing a reason per a service provider.

LogMessage

provider is the system that provides a service.
consumer is the system that consumes a service.
service is the service to be consumed by the consumer and provided by the producer.
commands is a map containing a set of configuration parameters.
requestedQoS is a map containing a set of QoS parameters (i.e. bandwidth).

EventMessage

response is a Boolean about the success of the configuration.
commands is a map containing a set of the configurations done by the QoSReserve.

b. Non-functional Requirements

Regarding the non-functional requirements, five must be satisfied by this service:

- Availability: The system must be online and accessible as long as possible, 24 hours per day and 365 days per year.

- Integrity: Dealing with sensible industrial requests, the systems implementing this service must always report any execution error into its database for further analysis and improvements.
- Interoperability: The developed service must be able to be easily implemented by new systems, and must be easily migrated to other Arrowhead Frameworks (ex. Hungary [56], BNearIT). At best, during the migration of frameworks, there should be no adaptation nor logic model changes.
- Performance: The system and its algorithms must have the shortest execution time possible. Thus, the service must rely on advanced hardware and good programming code.
- Extensibility: The service must be able to support large number of requests, thus its implementations must be able to leverage deployment into computational clouds and other elasticity enablers.

4.4 Interface Design Description

This section provides a detailed description, describing functions and exchanged messages of five interfaces, two relative to the QoSSetup (Section 4.3.1) service, QoSVerify (Section 4.4.1) and QoSReserve (Section 4.4.2), and the remaining three are relative to the QoSMonitor (Section 4.3.2) service, QoSEvent (Section 4.4.3), QoSLog (Section 4.4.4), QoSRule(Section 4.5.5).

4.4.1 QoSManager QoSVerify Interface Design Description

A. Interface Design Description Overview

This document describes how to realize the QoSManager QoSVerify interface.

Table 18 - Pointers to SD documents

Service description	Path
Arrowhead SD QoSManagerQoSSetup	Section 4.3.1.

Table 19 - Pointers to CP documents

Communication Profile	Path
Arrowhead CP REST_WS-TLS-JSON	`https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page`

Table 20 - Pointers to SP documents

Semantic Profile	Path
Arrowhead SP QoSManagerQoSVerify-JSON	Section 4.5.1.

B. Interfaces

a. QoSManagerQoSVerify_JSON

Table 21 - List of Functions provided by the QoSVerify service.

Function	Service	Method	Input	Output
qosVerify	/QoSVerify	PUT	VerificationMessage	VerificationResponse

Table 22- QoSManager web application description language.

Interface Description	QoSManager.wadl
Location	Appendix
Version	1.0

b. Information Model

See Arrowhead QoSManager QoSSetup Service SD, in Section 4.3.1, for the abstract information model

VerificationMessage

VerificationMessage is the abstract data type described in the SD document, cited in Section 4.3.1.

VerificationResponse

VerificationResponse is the abstract data type described in the SD document, cited in Section 4.3.1.

4.4.2 QoSManager QoSReserve Interface Design Description

A. Interface Design Description

This document describes how to realize the QoSManager QoSReserve interface.

Table 23 - Pointers to SD documents

Service description	Path
Arrowhead SD QoSManagerQoSSetup	Section 4.3.1.

Table 24 - Pointers to CP documents

Communication Profile	Path
Arrowhead CP REST_WS-TLS-JSON	`https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page`

Table 25 - Pointers to SP documents

Semantic Profile	Path
Arrowhead SP QoSManagerQoSReserve-JSON	Section 4.5.2.

B. Interfaces

a. QoSManagerQoSReserve_JSON

Table 26 - List of Functions provided by the QoSReserve service.

Function	Service	Method	Input	Output
qosReserve	/QoSReserve	PUT	ReservationMessage	ReservationResponse

Table 27 - QoSManager system web application description language

Interface Description	QoSManager.wadl
Location	Appendix
Version	1.0

b. Information Model

ReservationMessage

ReservationMessage is the abstract data type described in the SD document, cited in Section 4.3.1.

ReservationResponse

ReservationResponse is the abstract data type described in the SD document, cited in Section 4.3.1.

4.4.3 QoSMonitor Event Interface Design Description

A. Interface Design Description

This document describes how to realize the QoSMonitor Event interface.

Table 28 - Pointers to SD documents

Service description	Path
Arrowhead SD QoSMonitorMonitor	Section 4.3.2.

Table 29 - Pointers to CP documents

Communication Profile	Path
Arrowhead CP REST_WS-TLS-JSON	`https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page`

Table 30 Pointers to SP documents

Semantic Profile	Path
Arrowhead SP QoSMonitorEvent-JSON	Section 4.5.3 .

B. Interfaces

a. QoSMonitorEvent_JSON

Table 31 - List of Functions provided by the QoSEvent service

Function	Service	Method	Input	Output
sendEvent	/Monitor	POST	EventMessage	OK

Table 32- QoSMonitor web application description language

Interface Description	QoSMonitor.wadl
Location	Appendix.
Version	1.0

b. Information Model

EventMessage

EventMessage is the abstract data type described in QoSMonitor Monitor SD, cited in Section 4.3.2.

4.4.4 QoSMonitor QoSLog Interface Design Description

A. Interface Design Description

This document describes how to realize the QoSMonitor QoSLog interface.

Table 33 - Pointers to SD documents

Service description	Path
Arrowhead SD QoSMonitorMonitor	Section 4.3.2

Table 34 - Pointers to CP documents

Communication Profile	Path
Arrowhead CP REST_WS-TLS-JSON	https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page

Table 35 Pointers to SP documents

Semantic Profile	Path
Arrowhead SP QoSMonitorQoSLog-JSON	Section 4.5.4.

B. Interfaces

a. QoSMonitorQoSLog_JSON

Table 36 - List of Functions provided by the QoSLog service

Function	Service	Method	Input	Output
addMonitorLog	/Monitor	POST	LogMessage	OK

Table 37 - QoSMonitor system web application description language

Interface Description	QoSMonitor.wadl
Location	Appendix.
Version	1.0

b. Information Model

LogMessage

LogMessage is the abstract data type described in QoSMonitor Monitor SD, cited in Section 4.3.2 .

4.4.5 QoSMonitor QoSRule Interface Design Description

A. Interface Design Description

This document describes how to realize the QoSMonitor QoSRule interface.

Table 38 - Pointers to SD documents

Service description	Path
Arrowhead SD QoSMonitorMonitor	Section 4.3.2

Table 39 - Pointers to CP documents

Communication Profile	Path
Arrowhead CP REST_WS-TLS-JSON	https://forge.soa4d.org/plugins/mediawiki/wiki/arrowhead-f/index.php/Main_Page

Table 40 Pointers to SP documents

Semantic Profile	Path
Arrowhead SP QoSMonitorQoSRule-JSON	Section 4.5.5 .

B. Interfaces

a. QoSMonitorQoSRule_JSON

Table 41 - List of Functions provided by the QoSRule service

Function	Service	Method	Input	Output
addMonitorRule	/Monitor	POST	AddRuleMessage	OK
removeMonitorRule	/Monitor	DELETE	RemoveRuleMessage	OK

Table 42 - QoSMonitor system web application description language

Interface Description	QoSMonitor.wadl
Location	Appendix.
Version	1.0

b. Information Model

AddRuleMessage

AddRuleMessage is the abstract data type described in QoSMonitor Monitor SD, cited in Section 4.3.2 .

RemoveRuleMessage

RemoveRuleMessage is the abstract data type described in QoSMonitor Monitor SD, cited in Section 4.3.2 .

4.5 Semantic Profile Description

This section describes the data forma by pointing out its type (e.g. JSON; XML) and how that data is encoded of the five interfaces already described in section 4.4 .

4.5.1 QoSManagerQoSVerify Semantic Profile Description

A. Overview

The QoSManager QoSVerify-JSON profile is used to represent the data of the QoSVerify interface.

B. Data Format

VerificationMessage

The description regarding this data type is in Section 4.4.1.

```
{
    "requesteService":{
        "interfaces":[ "RESTJSON" ],
        "serviceMetaData":[{"key":"location",
"value":"Portugal"} ],
        "serviceDefinition":"C1",
        "serviceGroup":"Cs"
    },
    "consumer":{
        "address":"192.168.60.23",
        "authenticationInfo":"noAuth",
        "port":"9999",
        "systemGroup":"Cs",
        "systemName":"C1"
    },
    "provider":[
                {"address":"192.168.60.69",
                "authenticationInfo":"noAuth",
                "port":"9999",
                "systemGroup":"Ps",
                "systemName":"P1"}
            ],
     "requestedQoS":{
        "entry": [
              {
                "key": "delay",
                "value": "300"
              },
              {
                "key": "bandwidth",
                "value": "2"
              }
            ]
    }
}
```

Each parameter of the VerificationMessage is described in Table 43.

Table 43 - VerificationMessage parameters description.

ID	Parameter	Description
1	requestedService:interfaces	List of String containing all available interfaces protocols to access the service.
2	requestedService:serviceMetaData	Map of Strings containing a description of the service.
3	requestedService:serviceDefinition	String containing the name of the service.
4	requestedService:serviceGroup	String containing a name for the group where the services belongs.
5	consumer:address	String containing the IP address of the system.
6	consumer:authenticationInfo	String containing information about the Authorisation procedure of the system.
7	consumer:port	String containing the port of the system where users establish a connection.
8	consumer:systemGroup	String containing the name of the group where the system belongs.
9	consumer:systemName	String containing the name of the system.
10	provider:*	See descriptions 5,6,7,8.
11	requestedQoS:delay	Integer (milliseconds) containing the maximum delay of the message stream. This parameter is Optional.
12	requestedQoS:bandwidth	Decimal (Bps) containing the maximum bandwidth for the message stream. This parameter is Optional.

VerificationResponse

The description regarding this data type is in Section 4.4.1.

```
{
      "qosVerificationResponse":{
         "entry": [
                {
                  "key": {
                  "systemName":"provider1",
                  "systemGroup":"providers"
                  },
                  "value": "true"
                },
                {
                  "key": {
                  "systemName":"provider2",
                  "systemGroup":"providers"
                  },
                  "value": "false"
                }
              ]
     },
      "rejectMotivation":{
         "entry": [
```

```
                {
                   "key": {
                   "systemName":"provider2",
                   "systemGroup":"providers"
                   },
                   "value": "ALWAYS"
                 }
               ]
    }
}
```

Each parameter of the VerificationResponse is described in Table 44.

Table 44 -. VerificationResponse parameters description.

ID	Parameter	Description
1	qosVerificationResponse	Map of systems and Booleans as values. This map lists the systems that are capable of providing QoS (Boolean as true) and the ones that cant (Boolean as false).
2	rejectMotivation	Map of systems and Strings as values. This map lists the rejection causes for each system that cannot provide the QoS.

4.5.2 QoSManagerQoSReserve Semantic Profile Description

A. Overview

The QoSManagerQoSReserve-JSON profile is used to represent the data of the QoSReserve interface.

B. Data Format

Data received by the QoSManager QoSReserve interface will have the following format:

ReservationMessage

The description regarding this data type is in Section 4.4.2.

```
{
    "service":{
        "interfaces":[ "RESTJSON" ],
        "serviceMetaData":[{"key":"location",
"value":"Portugal"} ],
        "serviceDefinition":"Camera1",
        "serviceGroup":"Cameras"
    },
    "consumer":{
        "address":"192.168.60.23",
        "authenticationInfo":"noAuth",
        "port":"9999",
        "systemGroup":"Consumers",
```

```
        "systemName":"Consumer1"
    },
    "provider":[
                {"address":"192.168.60.69",
                "authenticationInfo":"noAuth",
                "port":"9999",
                "systemGroup":"ProcessingUnits",
                "systemName":"Unit2"}
            ],
     "requestedQoS":{
        "entry": [
              {
                "key": "delay",
                "value": "20"
              },
              {
                "key": "bandwidth",
                "value": "2"
              }
            ]
    }
}
```

Each parameter of the ReservationMessage is described in Table 45.

Table 45 - ReservationMessage parameters description.

ID	Parameter	Description
1	service:interfaces	List of String containing all available interfaces protocols to access the service.
2	service:serviceMetaData	Map of Strings containing a description of the service.
3	service:serviceDefinition	String containing the name of the service.
4	service:serviceGroup	String containing a name for the group where the services belongs.
5	consumer:address	String containing the IP address of the system.
6	consumer:authenticationInfo	String containing information about the Authorisation procedure of the system.
7	consumer:port	String containing the port of the system where users establish a connection.
8	consumer:systemGroup	String containing the name of the group where the system belongs.
9	consumer:systemName	String containing the name of the system.
10	provider:*	See descriptions 5,6,7,8.
11	requestedQoS:delay	Integer (milliseconds) containing the maximum delay of the message stream. This parameter is Optional.
12	requestedQoS:bandwidth	Decimal (Bps) containing the maximum bandwidth for the message stream. This parameter is Optional.

ReservationResponse

The description regarding this data type is in reference 4.4.2.

```
{
      "response": false,
      "commands":[
       {
            "url": "127.0.0.1/entrypoint",
            "comand":"size:1,period:3,type:0"
       }
       ]
}
```

Each parameter of the ReservationResponse is described in Table 46.

Table 46 - ReservationResponse parameters description.

ID	Parameter	Description
1	response	Boolean relative to the success of the configuration of the QoS.
2	commands	Map containing the configuration data sent to all devices.

4.5.3 QoSMonitorEvent Semantic Profile Description

A. Overview

The QoSMonitorEvent-JSON profile is used to represent the data of the Event interface.

B. Data Format

Data received by the QoSMonitor Event interface will have the following format:

EventMessage

The description regarding this data type is in Section 4.4.3.

```
{
    "protocol":"ftt-se ",
    "system":{
        "address":"192.168.60.50",
        "authenticationInfo":"authinfo",
        "port":"9996",
        "systemGroup":"group",
        "systemName":"name"
    },
    "parameters":{
        "entry": [
              {
                "key": "stream_id ",
                "value": "1"
              }]
     },
     "errorMessage":"message"
}
```

Each parameter of the EventMessage is described in Table 47.

Table 47 - EventMessage parameters description

ID	Parameter	Description
1	protocol	String containing the communication protocol name (i.e. “ftt-se”).
2	system:address	String containing the IP address of the system.
3	system:authenticationInfo	String containing information about the Authorisation procedure of the system.
4	system:port	String containing the port of the system where users establish a connection.
5	system:systemGroup	String containing the name of the group where the system belongs.
6	system:systemName	String containing the name of the system.
7	parameters:stream_id	Map element. Integer containing a message package identifier where the event occurred.
8	errorMessage	String containing an error message.

4.5.4 QoSMonitorQoSLog Semantic Profile Description

A. Overview

The QoSMonitorQoSLog-JSON profile is used to represent the data of the Monitor interface

B. Data Format

Data received by the QoSMonitor QoSLog interface will have the following format:

AddLogMessage

The description regarding this data type is in Section 4.4.4 .

```
{
    "protocol":"ftt-se",
    "provider":{
        "address":"192.168.60.50",
        "authenticationInfo":"authinfo",
        "port":"9996",
        "systemGroup":"group",
        "systemName":"name"
    },
    "consumer":{
        "address":"192.168.60.69",
        "authenticationInfo":"authinfo",
        "port":"9996",
        "systemGroup":"group",
        "systemName":"name"
    },
    "parameters":{
        "entry": [
              {
                "key": "delay",
                "value": "90"
              },
              {
                "key": "bandwidth",
                "value": "120"
              }]
     },
     "timestamp":"1474384344"
}
```

Each parameter of the AddLogMessage is described in Table 48.

Table 48 - AddLogMessage parameters description

ID	Parameter	Description
1	protocol	String containing the communication protocol name (i.e. “ftt-se”).
2	system:address	String containing the IP address of the system.
3	system:authenticationInfo	String containing information about the Authorisation procedure of the system.
4	system:port	String containing the port of the system where users establish a connection.
5	system:systemGroup	String containing the name of the group where the system belongs.
6	system:systemName	String containing the name of the system.
7	parameters:delay	Map element. Integer containing communication delay in milliseconds.
8	parameters:bandwidth	Map element. Integer containing communication bandwidth in Mbps.
9	timestamp	Long containing the time when the log was sent.

4.5.5 QoSMonitorQoSRule Semantic Profile Description

A. Overview

The QoSMonitorQoSRule-JSON profile is used to represent the data of the QoSRule interface.

B. Data Format

Data received by the QoSMonitor QoSRule interface will have the following format:

AddRuleMessage

The description regarding this data type is in Section 4.4.5 .

```
{
    "protocol": "ftt-se",
    "provider": {
        "systemGroup": "a1",
        "systemName": "s24",
        "address": "192.168.60.144",
        "port": "8080",
        "authenticationInfo": "noAuth"
    },
    "consumer": {
        "systemGroup": "tr3",
        "systemName": "temp2",
        "address": "192.168.60.190",
        "port": "8081",
        "authenticationInfo": "noAuth"
    },
    "parameters": {
        "entry": [{
```

```
                "key": "delay",
                "value": "20"
            },
            {
                "key": "bandwidth",
                "value": "1500"
            },
            {
                ”key”: ”stream_id”,
                ”value”: ”2”
            },
            {
                ”key”: ”NLogs”,
                ”value”: ”10”
            ]
    },
    "softRealTime": "false"
}
```

Each parameter of the AddRuleMessage is described in Table 49.

Table 49 - AddRuleMessage parameters description

ID	Parameter	Description
1	protocol	String containing the communication protocol name (i.e. “ftt-se”).
2	provider:address	String containing the IP address of the system.
3	provider:authenticationInfo	String containing information about the Authorisation procedure of the system.
4	provider:port	String containing the port of the system where users establish a connection.
5	provider:systemGroup	String containing the name of the group where the system belongs.
6	provider:systemName	String containing the name of the system.
7	consumer:*	See descriptions 2,3,4,5.
8	parameters:delay	Map element. Integer (milliseconds) containing the maximum delay of the message stream. This parameter is optional.
9	parameters:bandwidth	Map element. Decimal (Bps) the maximum bandwidth for the message stream. This parameter is optional.
10	Parameters:stream_id	Map element. Integer identifying the stream_id being worked with. Not optional.
11	parameters:NLogs	Map element. Integer with the number of logs to work with if softRealTime time parameter is set to true.
12	softRealTime	Boolean containing the type of stream to be monitored (true for soft real-time and false for hard real-time).

RemoveRuleMessage

The description regarding this data type is in reference 4.4.5.

```
{
     "provider": {
        "systemGroup": "a1",
        "systemName": "s24",
        "address": "192.168.60.144",
        "port": "8080",
        "authenticationInfo": "noAuth"
    },
    "consumer": {
        "systemGroup": "tr3",
        "systemName": "temp2",
        "address": "192.168.60.190",
        "port": "8081",
        "authenticationInfo": "noAuth"
    }
}
```

Each parameter of the RemoveRuleMessage is described in Table 50.

Table 50 - RemoveRuleMessage parameters description

ID	Parameter	Description
1	type	String containing the communication protocol name (i.e. “ftt-se”).
2	provider:address	String containing the IP address of the system.
3	provider:authenticationInfo	String containing information about the Authorisation procedure of the system.
4	provider:port	String containing the port of the system where users establish a connection.
5	provider:systemGroup	String containing the name of the group where the system belongs.
6	provider:systemName	String containing the name of the system.
7	consumer:*	See descriptions 2,3,4,5.

4.6 System Design Description

This section describes in detail the implementation of QoSManager and QoSMonitor Systems, approaching the used technologies and software patterns.

4.6.1 QoSManager System Design Description (SysDD)

A. System Design Description Overview

Table 51 – QoSManager System Information.

Name	QoSManager (*see Section 4.2.1*).
Owner	ISEP

The Arrowhead QoSManager system has been developed by CISTER/ISEP, for the Arrowhead project with the goal of managing all requests with Quality of Service (QoS) by verifying the feasibility and its configuration to any involved party, such as network actives and devices. Acting as a support system for the Orchestrator, the QoSManager also works with the QoSMonitor system to guarantee the fulfilment of the requested QoS during the life of a message stream.

This system produces one service only, the QoSSetup. This service provides two interfaces, the QoSVerify to verify the feasibility of a QoS; and the QoSReserve for the configuration of the network.

A more abstract description of the QoSManager system can be found on the document referenced in Table 52.

Table 52 – QoSManager SysD Documentation Pointer.

System name	Path
QoSManager	Section 4.2.1 .

B. Use-cases

a. Non-Functional Requirements

To guarantee the non-functional requirements described in the document referenced in Table 52, this section lists the proposed solutions to its corresponding requirement:

- Availability: Deployment on a dedicated server.
- Integrity: Usage of a Log system, reporting any considerable code instruction execution. This allows to create a historical of all the application interactions between users or systems.
- Interoperability & Extensibility: Usage of SOLID [58] software principles, developing a high cohesion and low coupling code.
- Performance: Usage of high performance technologies, specifically MySQL for the databases operations.

b. List of Use-Cases

The QoSManager is registered and authenticated on the Arrowhead system as any other Arrowhead compliant system. The Orchestration Service is the only system to make use of the QoSManager in order to configure a QoS between the service consumer and provider.

As Figure 28 shows, there are two possible use-cases: The Verification of QoS (UC1) and Reservation of QoS (UC2).

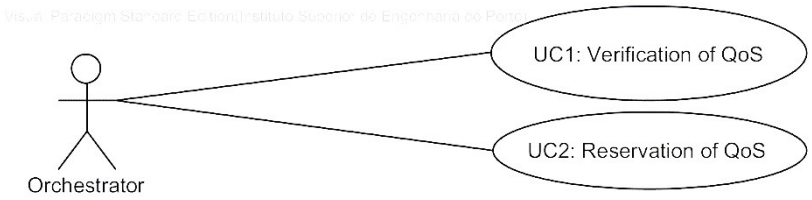


Figure 28 – QoSManager System Use-Cases List.

As Table 53 depicts, the Verification of a QoS verifies if a requested QoS is feasible depending on the devices capabilities and on the active QoS reservations. This use-case receives a list of providers with a requested QoS and returns a response using its QoS algorithm, containing the approved providers and the reasons why on the rejected providers the QoS wasn´t possible.

Concerning the implementation of the Use-Case 1, as Figure 29 shows, it starts whenever the `QoSManagerResource` class receives a `QoSVerify` REST object. The `QoSManagerService` class is responsible for the core operations of the use-case, providing the `qosVerify()` method to verify the received QoS requirements. Both `SCSFactory` and `QoSFactory` classes, which are singleton [59], create DTO [60] objects. In addition, these two factories also use the database repositories (`SystemConfigurationStoreRepository` and `QoSStoreReposiroty`) to get `NetworkDevice` objects from an `ArrowheadSystem` object, and to get from an `ArrowheadSystem` its corresponding QoS reservation. Additionally, the `VerifierAlgorithmFactory` has the goal of locating the `IQoSVerifierAlgorithm` implementation classes, using the reflector [61] pattern. The `IQoSVerifierAlgorithm` interface receives in its `verifyQoS()` method the capabilities, reservations of each `ArrowheadSystem` consumer and provider, the QoS parameters requested by the consumer, and the configuration commands that are optional. In the end, the interface method returns which systems are capable of sustaining the QoS.

Concerning the implementation of the Use-Case 2, as Figure 30 shows, it starts whenever the `QoSManagerResource` class receives a `QosReserve` REST object. The `QoSManagerService` class, for this Use-Case, provides the `qosReserve()` method to reserve the received QoS requirements. Both `SCSFactory` and `QoSFacory` classes, which are singleton [59], create DTO [60] objects. Each factory invokes the respective database operations located in the `SystemConfigurationStoreRepository` and `QoSStoreRepository`. Particularly, the operations that get a `NetworkDevice` object from its deployed `ArrowheadSystem` object, and that save a `MessageStream` object. Additionally, the `DriverFactory` class has the goal of associating a communication type name to its respective `IQoSDriver` implementation, using the reflector [61] pattern. The

`IQoSDriver` provides the configure method, in which receives the configuration of a network, each consumer and provider `ArrowheadSystem`, the requested QoS and the configuration commands that are optional. In the end, the interface methods returns a success status indicating if the configuration was or not successful.

Table 53 - Execution Flow of Use-Case 1 of QoSManager System.

Use-Case 1: Verification of QoS
ID: 1
Brief description: The use-case describes the sequence of steps for the verification of the service consumer requested QoS.
Primary actors: Orchestrator
Secondary actors: -Provider Systems, Consumer System.
Preconditions: - The Service Consumer and Service Provider network information must already be stored at the System Configuration Store.
Main flow: 1- A Service Consumer contacts the Orchestrator, orchestrating a service, located on a Local Cloud, with a Quality of Service. 2- The Orchestrator requests the QoSManager to verify the feasibility of the QoS on the consumer and producer stream. 3- Using a specific network algorithm, the QoSManager verifies if the requested QoS is or not possible giving a reject motivation back to the Orchestrator. 4- The Orchestrator gives all possible producers that can provide the requested service with QoS.
Post conditions: -
Alternative flows: 3.1- There is no sufficient information on the System Configuration Store to verify if the requested QoS is feasible and therefore the QoSManager sends a warning.

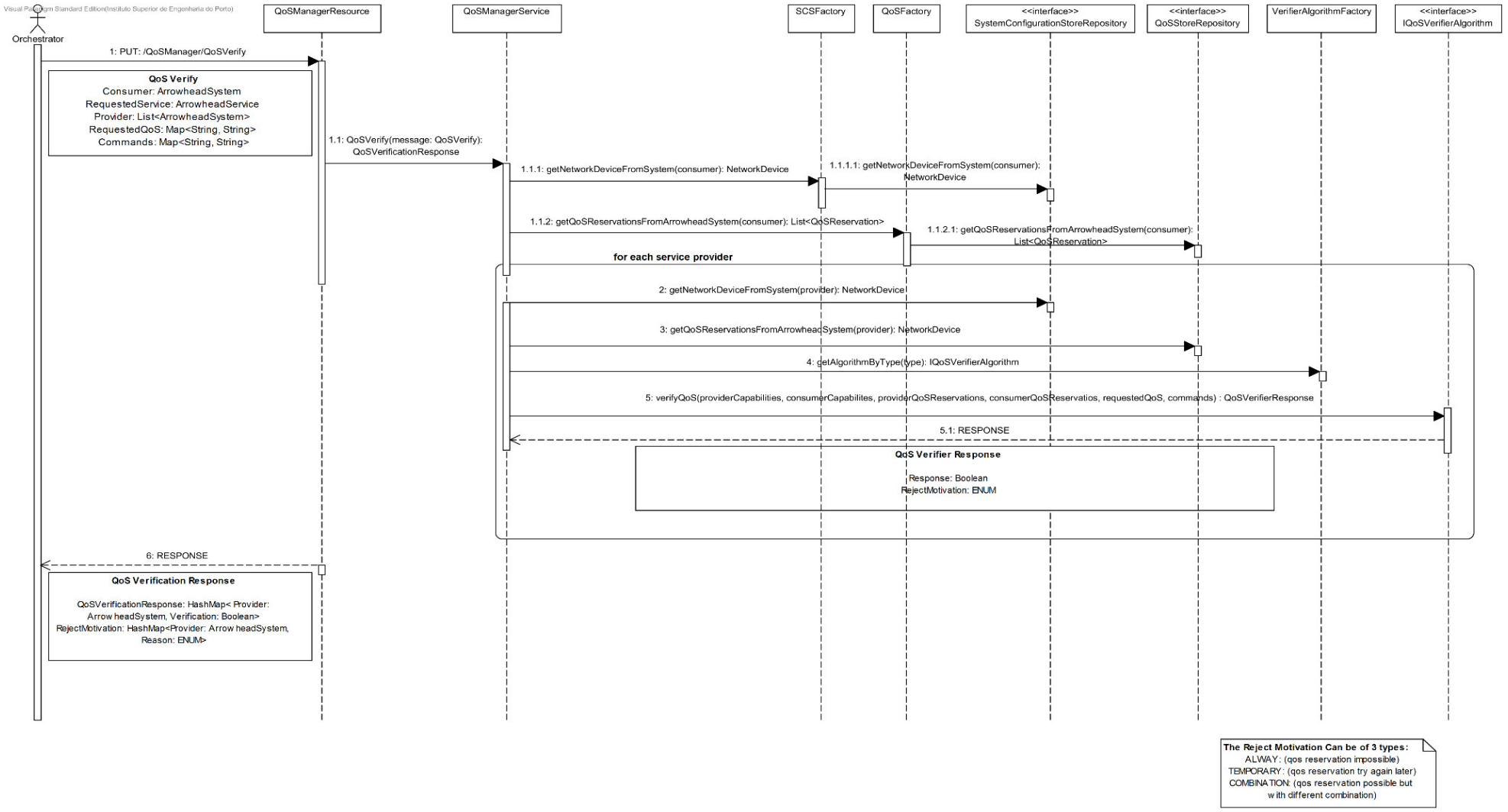


Figure 29 - Sequence Diagram of UC1.

As Table 54 depicts, the QoSManager also provides the Reservation of QoS use-case.

It receives a provider and a consumer system between which the QoSDriver must configure a connection. Note that it is recommendable that this use-case is executed only after the QoS verification to have full assurance of the success of the QoS reservation, however this is not mandatory.

Table 54 - Execution Flow of Use-Case 2 of QoSManager System.

Use-Case 2: Reservation of QoS
ID: 2
Brief description: The use-case describes the sequence of steps for the storage of events into a database or a local file.
Primary actors: Orchestrator
Secondary actors: Provider System, Consumer System.
Preconditions: - The Service Consumer and Service Provider network information must already be stored at the System Configuration Store. - This use-case comes only after UC1.
Main flow: 1- A Service Provider registers a service. 2- A Service Consumer contacts the Orchestrator, orchestrating a service, located on the Local Cloud, with a requested Quality of Service. 3- The Orchestrator requests the QoSManager to reserve a message stream between the Service Consumer and the Service Provider with the QoS desired. 4- The QoSManager, using the QoSDriver, setups the necessary configurations between the Service provider and consumer to meet the requested QoS. 5- After the configuration the QoSManager responds to the Orchestrator if the configuration was or not successful.
Post conditions: QoS reservation logged in the database.
Alternative flows: 4.1 - There is no sufficient information on the System Configuration Store to the QoS setup the QoSManager sends a warning.

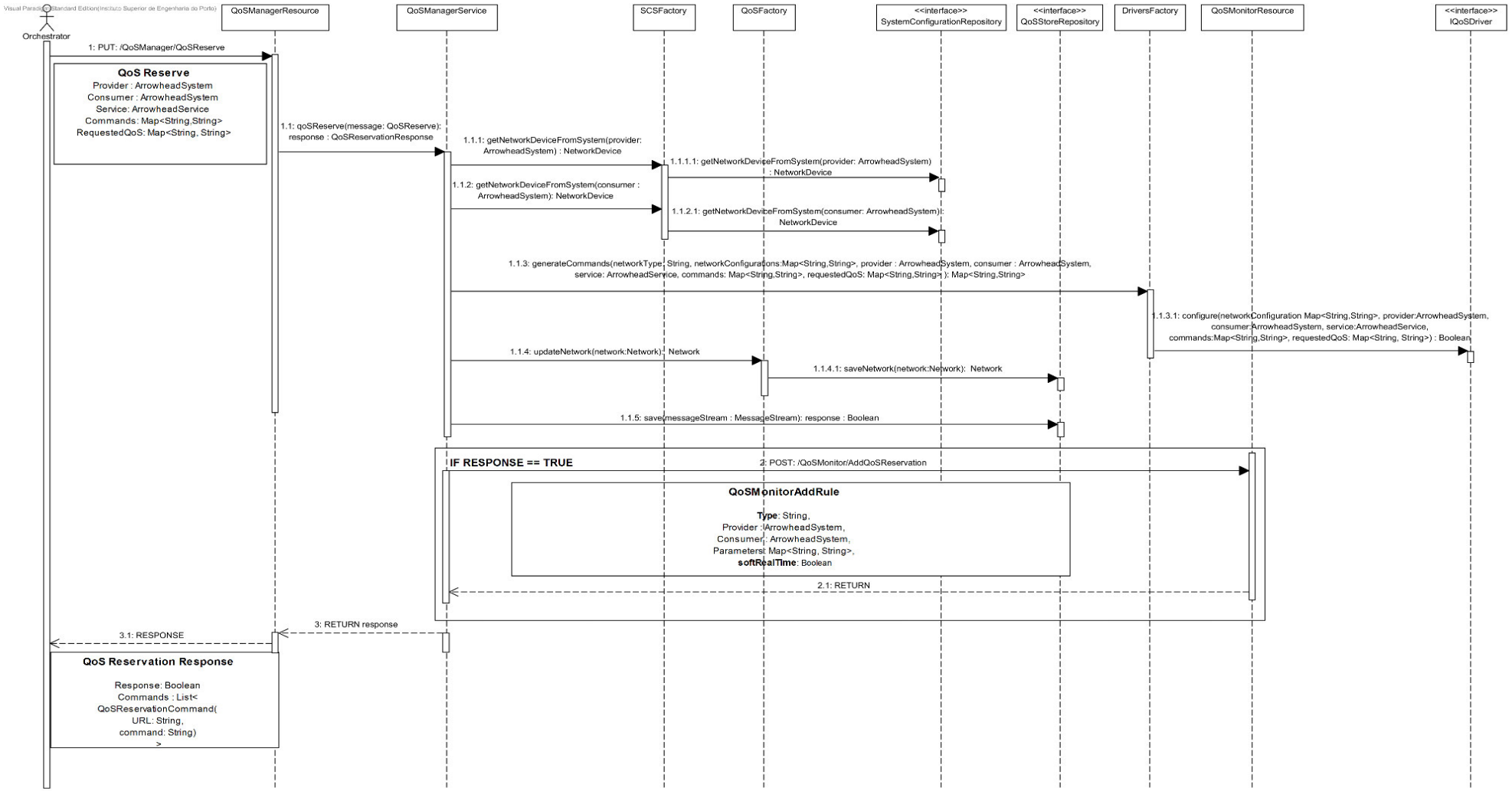


Figure 30 - Sequence Diagram of UC2.

C. Security

This chapter describes how security is implemented in the QoSManager System.

a. Decomposition of the System

None.

b. Technical Security Requirements

Any network exploit in IoT systems can cause both physical and economic damages, particularly in smart-cities, manufacturing, and transportation. The developed solution prevents these security problems.

Therefore, the QoSManager system provides a secure HTTP protocol (HTTPS) using a specific Java KeyStore (JKS) file. To interact with the QoSManager the user must know a private password, preventing unauthorized systems from using the QoSManager.

c. Data Flow Diagram

None.

d. Threats and Vulnerabilities

None.

D. Solution Description

The purpose of this chapter is to describe the implementation of the solution. Initially an overview of the system architecture is described with the support of a component diagram, then all the core classes used on the code implementation are explained along with a class diagram. Since the QoSManager must be able to interface with custom communication protocols, the following chapter explains what the user must do to create a new adapter. Finally, the two databases which the system works with, are explained with the support of a database model diagram.

a. Components Diagram

As Figure 31 depicts, the QoSManager is divided into three major components: the QoSSetup is where the core logic is implemented; the QoSDriver and QoS Verifier are developed to a specific communication protocol.

While the QoSSetup component manages all the core operations that QoSManager has assigned to, both QoSVerifier and QoSDriver have the responsibility of verifying QoS on a network, and configure a network according to the made request, respectively. Regarding to interaction with other components, the Orchestrator uses the QoSManager for the management of QoS. The QoSMonitor is used by the QoSManager whenever a configuration is made, with the aim of providing online monitoring of QoS. Both QoSStore and SystemConfigurationStore are used to store all the data regarding to QoS reservation and network configurations respectively.

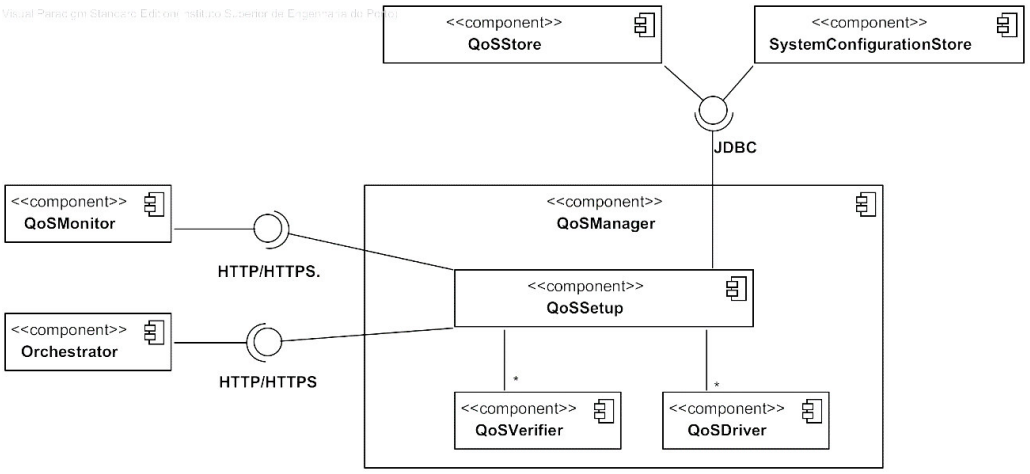


Figure 31 - Components Diagram of the QoSManager system.

b. Classes Structure, and Design Patterns under use

To accomplish both *Interoperability* and *Scalability* requirements, the code was developed according to best practices, as explained next.

Since the QoSManager has to work with an unpredictable number of custom communication protocols, the programming patterns *Reflection* [61] and *Factory* [62] were used on the QoSDriver and QoSVerifier components.

The *Reflection* pattern is a mechanism that allows the lookup and loading of software modules at runtime, for example to extend a software structure and behaviour dynamically [61]. In this project, it was used to avoid code recompilation every time an adapter with a new custom communication protocol was added. The *Factory* pattern is a creational pattern that hides the logic creation of an object, acting as an interface. It is used on the QoSFactory and SCSFactory classes, to create both Data Transfer Object (DTO) and specific database objects. It is also present on the QoSDriverFactory and VerifierAlgorithmFactory to assign them the correspondent driver of a specific communication protocol.

Another used pattern is *Repository* [63], which is used to isolate all database related operations in a software, in order to avoid duplication of code and to simplify the business model logic. The IQoSRepository and ISCSRepository are the classes where this pattern is implemented, and they have the responsibility of managing the interaction with the QoS Store and System Configuration Store, respectively. Even in relation to the databases, the classes QoSFactory and SCSFactory are responsible of facilitating all the databases operations by receiving DTO objects and converting them to the databases models and vice-versa.

Figure 32 depicts how the QoSManager main classes are structured and what operations they offer.

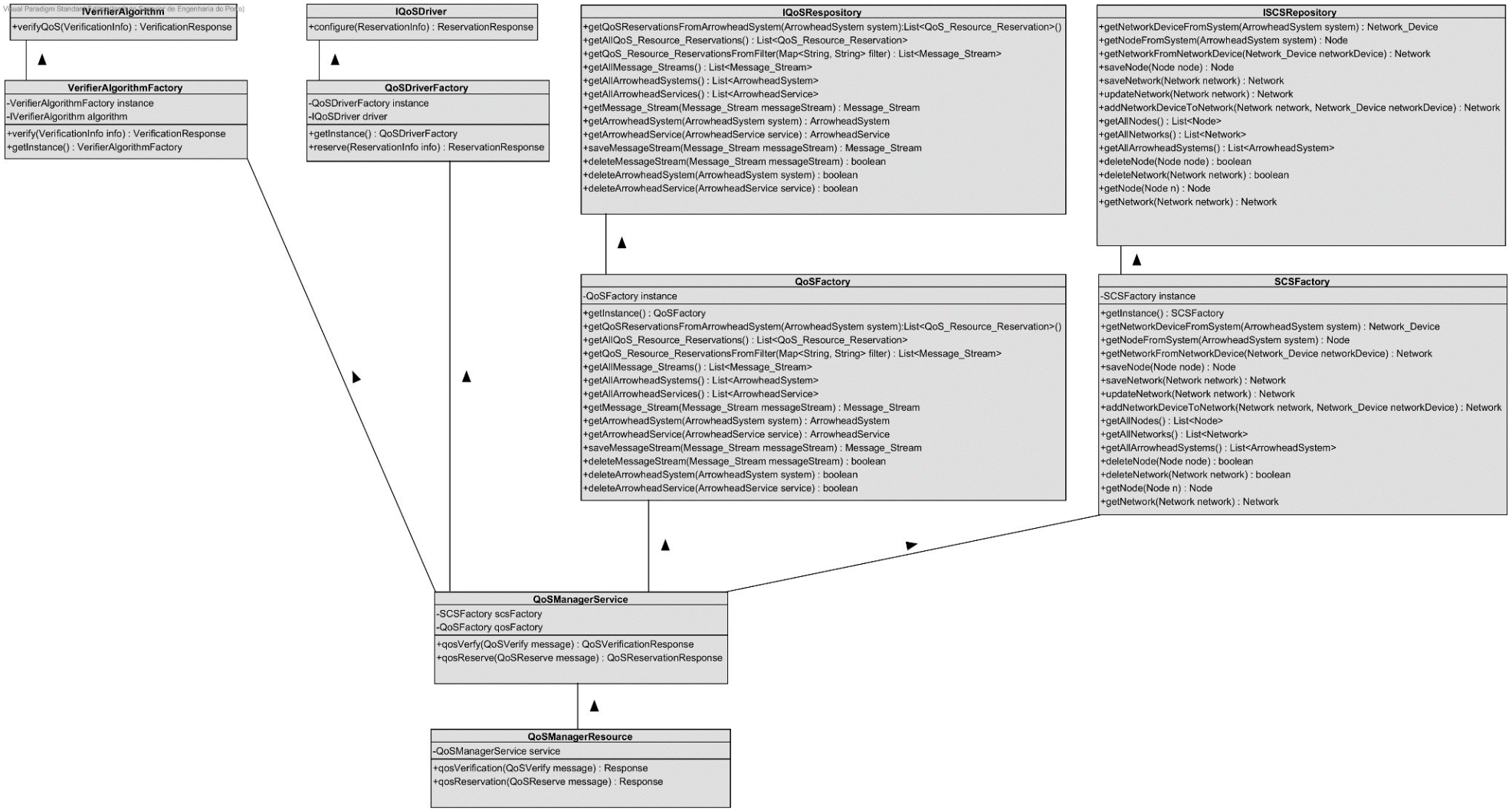


Figure 32 - Class Diagram of the QoSManager system.

c. Adapting to new communication protocols

Each time a new communication protocol is added, a QoSVerifier algorithm and a specific QoSDriver adapter must be developed. In the current implementation, both algorithm and driver must be contained in a class whose name is the same of the novel protocol, to ease the usage of the *Reflection* pattern to load it in at runtime, and they must be placed into the package "qosmanager.verifieralgorithms" in case of the QoS verifier algorithms and in the case of the QoS driver in the package "qosmanager.drivers".

Regarding the QoS algorithm, shown in Figure 33, the developed class must be an implementation of the interface IVerifierAlgorithm. The function `verifyQoS()` verifies the feasibility of the QoS parameters to be set up, depending on the available network capabilities and current QoS reservations.

In relation to the QoS Driver, shown in Figure 34, it must be an implementation of the interface IQoSDriver. Its `configure()` function receives the network and its devices including the requested QoS to set-up the stream between the service provider and consumer.

Figure 33 - IVerifierAlgorithm interface.

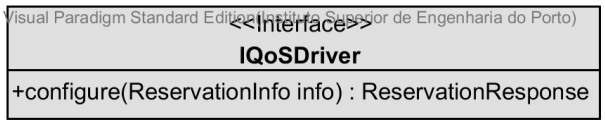


Figure 34 - IQoSDriver interface.

d. Databases

To support the QoSSetup service that the QoSManager provides, the system must keep track of the network devices configuration and the QoS reservations of computational and systems. In particular, the QoSManager accesses two stores:

The System Configuration Store, containing the configuration of the system of systems, thus providing information regarding network topologies, capabilities of the network actives and devices, configuration of both network actives and systems.

The QoSStore, which keeps track of resource reservations over the network actives and systems.

Both databases represented on Figures 35 and 36 are deployed on a MySQL server, which guarantees data consistency and fast execution time.

Regarding the QoSStore, its main table is the `Message Stream` table, shown in Table 55. It contains all the information regarding the stream service, both provider and

consumer systems, and the respective QoS reservations. Another important parameter is the stream type, which expresses the used communication protocol.

Table 55 - Message Stream table parameters.

Field	Data Type	Key	Unique	Description
qualityOfService_id	QoSResourceReservation	Foreign	No	QoS Reservation Parameters.
type	Varchar	No	No	Communication Protocol type of the stream. (ex. “fttse”)
consumer_id	ArrowheadSystem	Foreign	Yes	Service Consumer.
provider_id	ArrowheadSystem	Foreign	Yes	Service Provider.
service_id	Integer	Foreign	Yes	Service that is both consumed and produced.
qualityOfService_id	Integer	Foreign	No	Stream parameters related to its configuration.
message_stream_id	Integer	Primary	Yes	Identifier of the Message Stream.

The other used by the QoSManager system is the System Configuration Store. The central table is “Node” where all the network devices and actives are saved. This table, shown in Table 56 is logically divided into two table, the `Node_deployedsystem` and `Node_processingcapabilties`, containing all the systems and network devices that are deployed on that node, and its processing and networking capabilities.

Table 56 - Node table parameters

Field	Data Type	Key	Unique	Description
id	Integer	Primary	Yes	Identifier of the Node.
Device_model_code	String	No	Yes	Code containing the device brand, model and, If possible, its code.

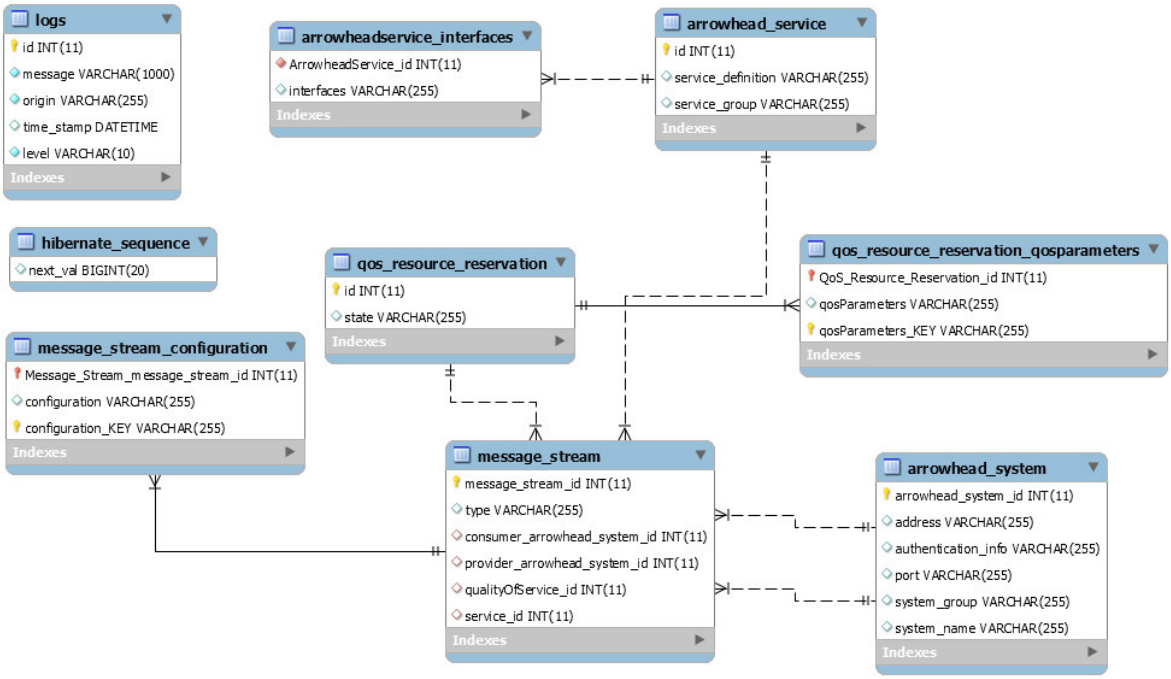


Figure 35 - Database Model of the QoSStore schema.

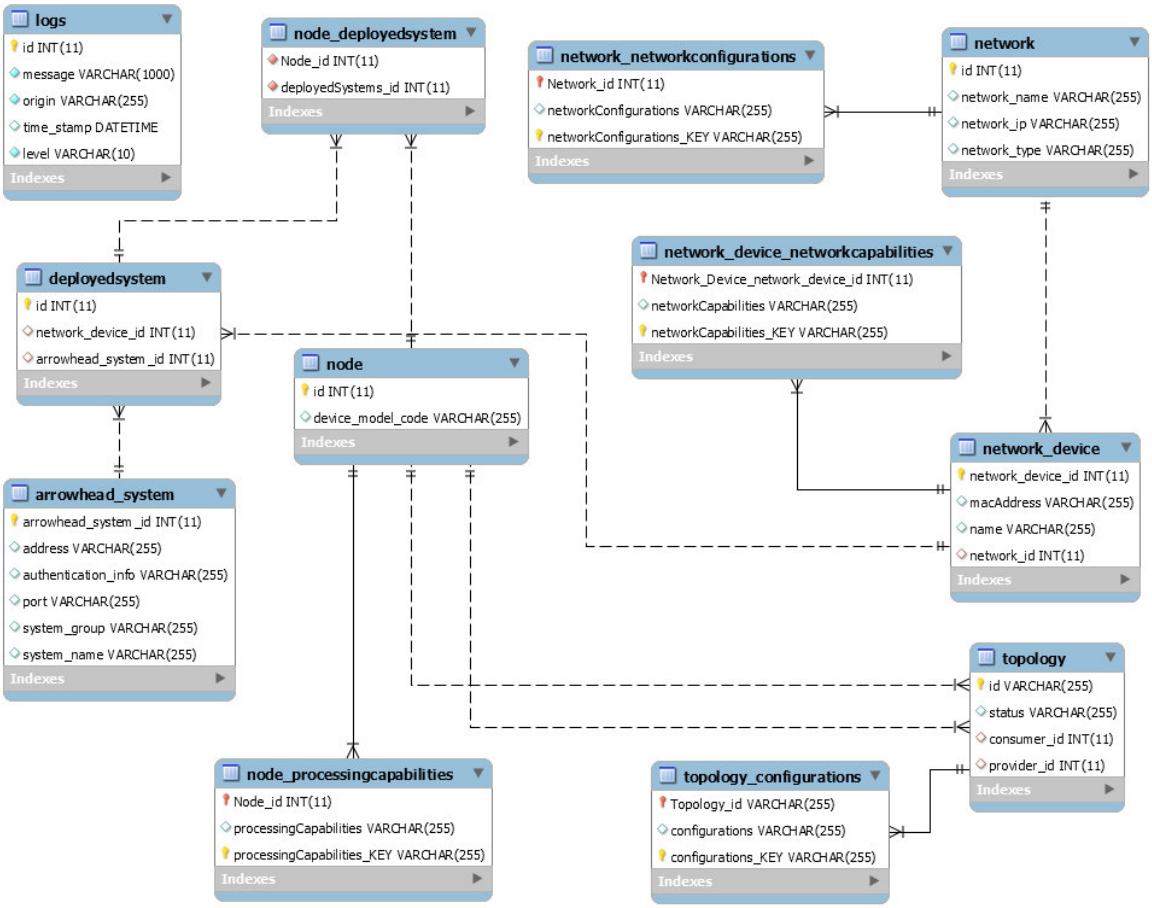


Figure 36 - Database Model of the SystemConfigurationStore schema

e. Deployment Diagram

Regarding the deployment, shown in Figure 37, three machines were used to deploy all the necessary components for the scenario. The three core systems were deployed on the same machine, although they could be on separate machines. The OS used on the first machine was Windows 10 and the web server was built on the Grizzly framework. Regarding to the developed QoSManager and QoSMonitor, both were deployed in a different machine with the same Windows 10 OS and same web server framework. The two databases, which the QoSManager works with, were installed in a Windows 10 machine on a MySQL server.

The applications that were used during the deployment are stored on the following repository (https://bitbucket.org/cister_pt_arrowhead/), including the QoSManager and QoSMonitor systems. The two systems were developed using the Netbeans IDE [64].

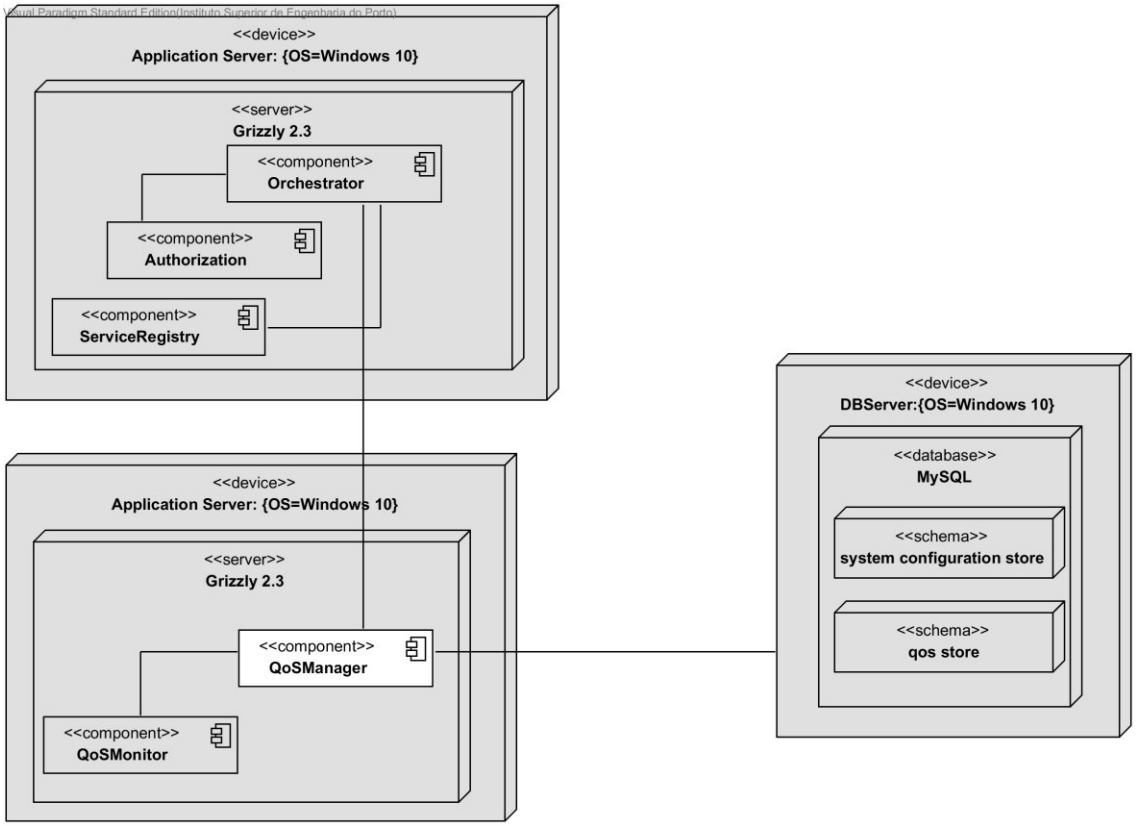


Figure 37 - Deployment Diagram of the QoSManager system.

4.6.2 QoSMonitor System Design Description (SysDD)

A. System Design Description Overview

Table 57 - System Information of QoSMonitor

Name	QoSMonitor (*see Section 4.2.2*).
Owner	ISEP

The Arrowhead QoSMonitor system has been developed by CISTER/ISEP for the Arrowhead project with the goal of monitoring communication performance between systems, usually two systems, composed by a service producer and a service consumer, in an Arrowhead compliant installation.

This system is also supported by a plugin, an extension of the QoSMonitor deployed in each of the two last mentioned systems, responsible for capturing information regarding communications between them and sending it to the QoSMonitor, responsible for QoS examination. It uses previously defined rules to compare the QoS requirements against the data received. If any of these rules is not fulfilled an event is created which is sent to consuming nodes using the EventHandler [65] system. It also allows sending events to the aforementioned system by Arrowhead compliant systems.

A black box description of the QoSMonitor and the EventHandler systems can be found on the documents referenced in Table 58 and Table 59 respectively.

Table 58 – QoSMonitor SysD Documentation Pointer.

System name	Path
QoSMonitor	Section 4.2.2

Table 59 - EventHandler SysD Documentation Pointer.

System name	Path
EventHandler	`https://forge.soa4d.org/svn/arrowhead-f/3_Core%20Systems%20and%20Services/2_Support%20Core%20Systems%20and%20Services/5_Eventhandler%20system/Documetation/Arrowhead%20SySD%20EventHandlerSystem%20v1.0.docx`

B. Use-Cases

a. Non-Functional Requirements

Regarding the non-functional requirements there are 5 that must be highlighted:

- Availability: Deployment on a dedicated server.
- Integrity: Usage of a Log system, reporting any considerable code instruction execution. This allows to create a historical of all the application interactions between users or systems
- Interoperability & Extensibility: Usage of SOLID software principles, developing a high cohesion and low coupling code.

- Performance: Usage of high performance technologies, specifically MongoDB for the database operations

b. List of Use-Cases

The QoSMonitor is registered and authenticated on the Arrowhead system as an Arrowhead compliant system. The QoSManager is the only system, developed so far that makes use of the QoSMonitor functionalities. Any Arrowhead compliant system can exploit the sending of events functionality.

As Figure 38 shows, there are four possible use-cases.

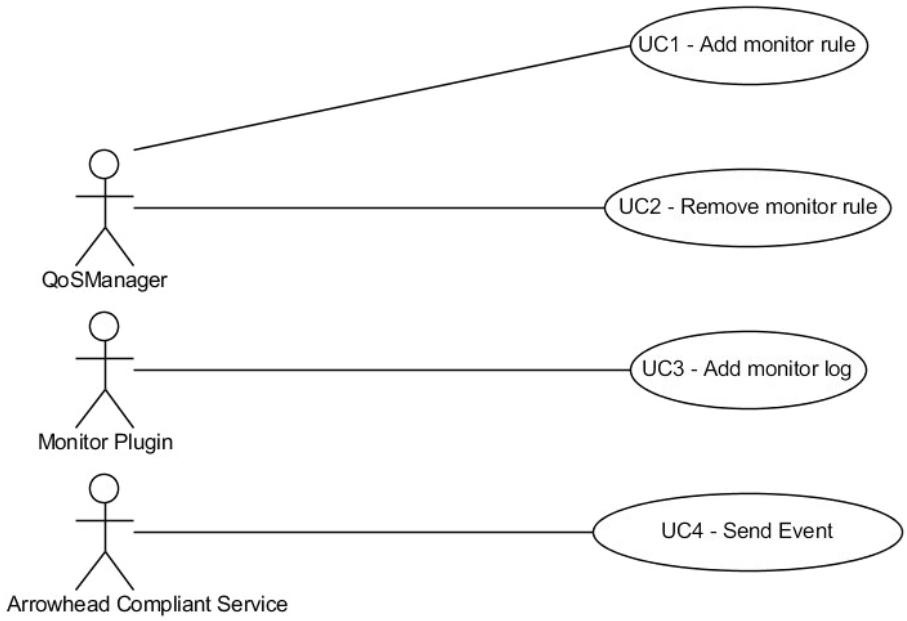


Figure 38 - QoSMonitor Use Cases List.

As detailed in Table 60, adding a monitor rule implies that the QoSManager sends a payload with the communication protocol, identification of the two involved systems, QoS parameters with respective requested values and a check value for soft real time or real time monitoring.

Table 60 - QoSMonitor Use-Case 1 Execution Flow.

Use-Case 1: Add Monitor Rule
ID: 1
Brief description: Add monitor rule about requested Quality-of-Service between two systems.
Primary actors: QoSManager
Secondary actors: MongoDB Manager.
Preconditions: At least one monitor parameter.
Main flow: 1- QoSManager sends a monitoring rule to the QoSMonitor. 2- QoSMonitor validates the payload. 3- QoSMonitor saves monitoring rule in the database, identified by the given systems.
Post conditions: Monitor rule stored in the database.
Alternative flows: 2.1- The payload is not valid. 2.2- Returns bad request as response. 3.1- A rule identified by the same given systems already exists in the database. 3.2- The rule is deleted. 3.3- The new rule is saved.

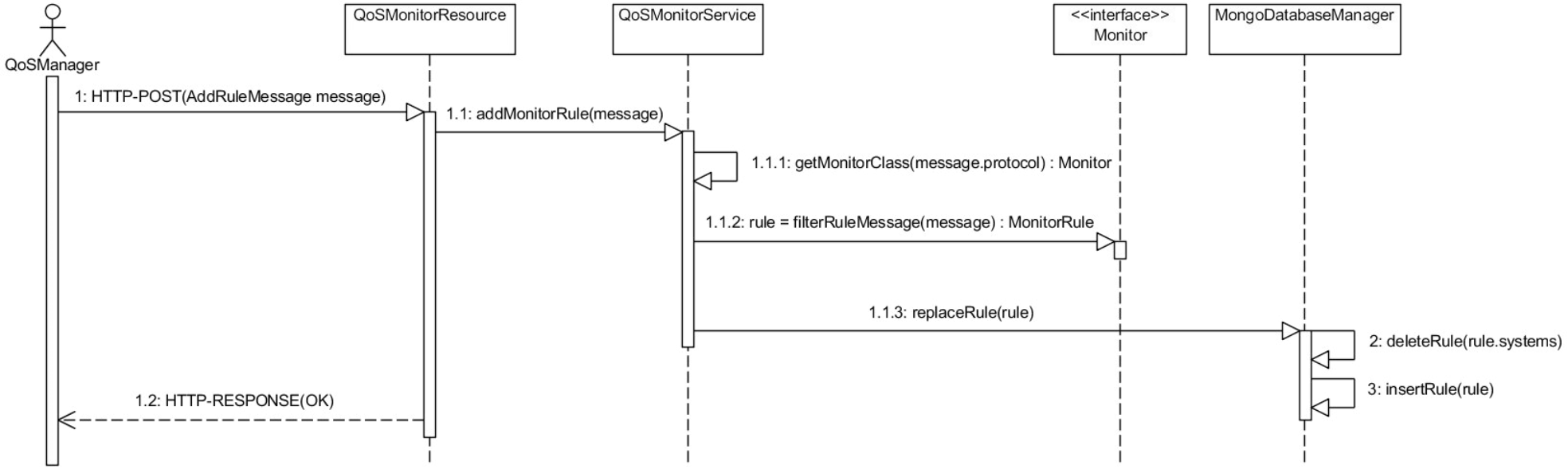


Figure 39 - QoSMonitor Sequence Diagram of UC1.

As shown in Table 61, to remove a monitoring rule, the QoSManager only needs to send the identification of the rule, therefore the service producer and service consumer.

Table 61 - QoSMonitor Use-Case 2 Execution Flow.

Use-Case 2: Remove Monitor Rule
ID: 2
Brief description: Removes monitor rule about requested Quality-of-Service between two systems.
Primary actors: QoSManager.
Secondary actors: MongoDB Manager.
Preconditions: -
Main flow: 1- QoSManager sends a monitor rule to QoSMonitor. 2- QoSMonitor checks existence of rule in the database. 3- Removes monitor rule in the database, identified by the given systems.
Post conditions: Monitor rule deleted in the database.

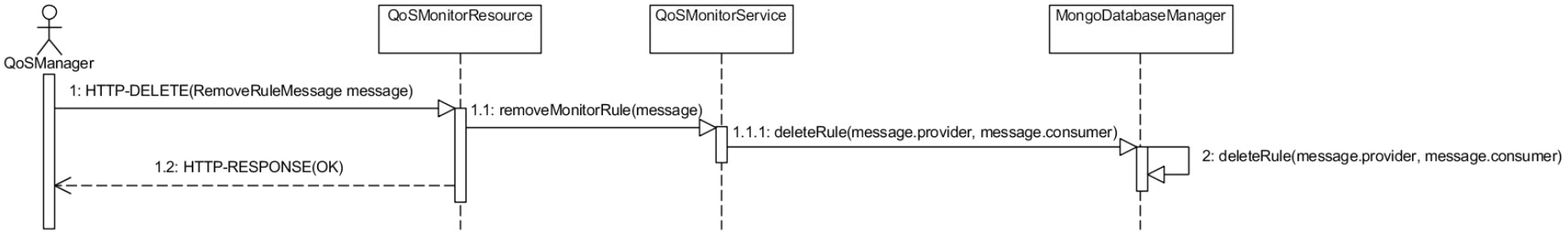


Figure 40 - QoSMonitor Sequence Diagram of UC2..

A monitor log is a set of information regarding performance monitoring of services or systems in an Arrowhead compliant installation in a given moment. To add a monitor log, a monitor plugin needs to send a payload with the communication protocol, identification of the two involved systems, QoS parameters with respective monitored values and a timestamp to work with soft real time if it's enabled. After the log is saved in the database, the corresponding rule is retrieved by using the given systems and the parameters are compared against each other. The rule requested values versus the log monitored values. If QoS inconsistency is found, then a maximum severity level is sent to the EventHandler. Table 62 shows the execution flow of this use case.

Table 62- QoSMonitor Use-Case 3 Execution Flow.

Use-Case 3: Add Monitor Log
ID: 3
Brief description: Add monitor log with information regarding communications between two systems, service producer and service consumer.
Primary actors: MonitorPlugin of service prosumer.
Secondary actors: MongoDB Manager
Preconditions: At least one monitor parameter. Rule identified by the given systems must exist in the database.
Main flow: 1- MonitorPlugin sends monitor log. 2- System validates the payload. 3- Checks for a monitor rule identified by the given systems. 4- Saves monitor log in the database, identified by the given timestamp. 5- Validates Quality-of-Service by comparing monitor log information against rule specifications.
Post conditions: Monitor log stored in the database
Alternative flows: 2.1- The payload is not valid. 2.2- Returns bad request as response. 3.1- A rule identified by the given systems does not exist. 3.2- Returns not found as response. 4.1- Checks that the Quality-of-Service requirements were not met. 4.2- Sends event to the EventHandler system.

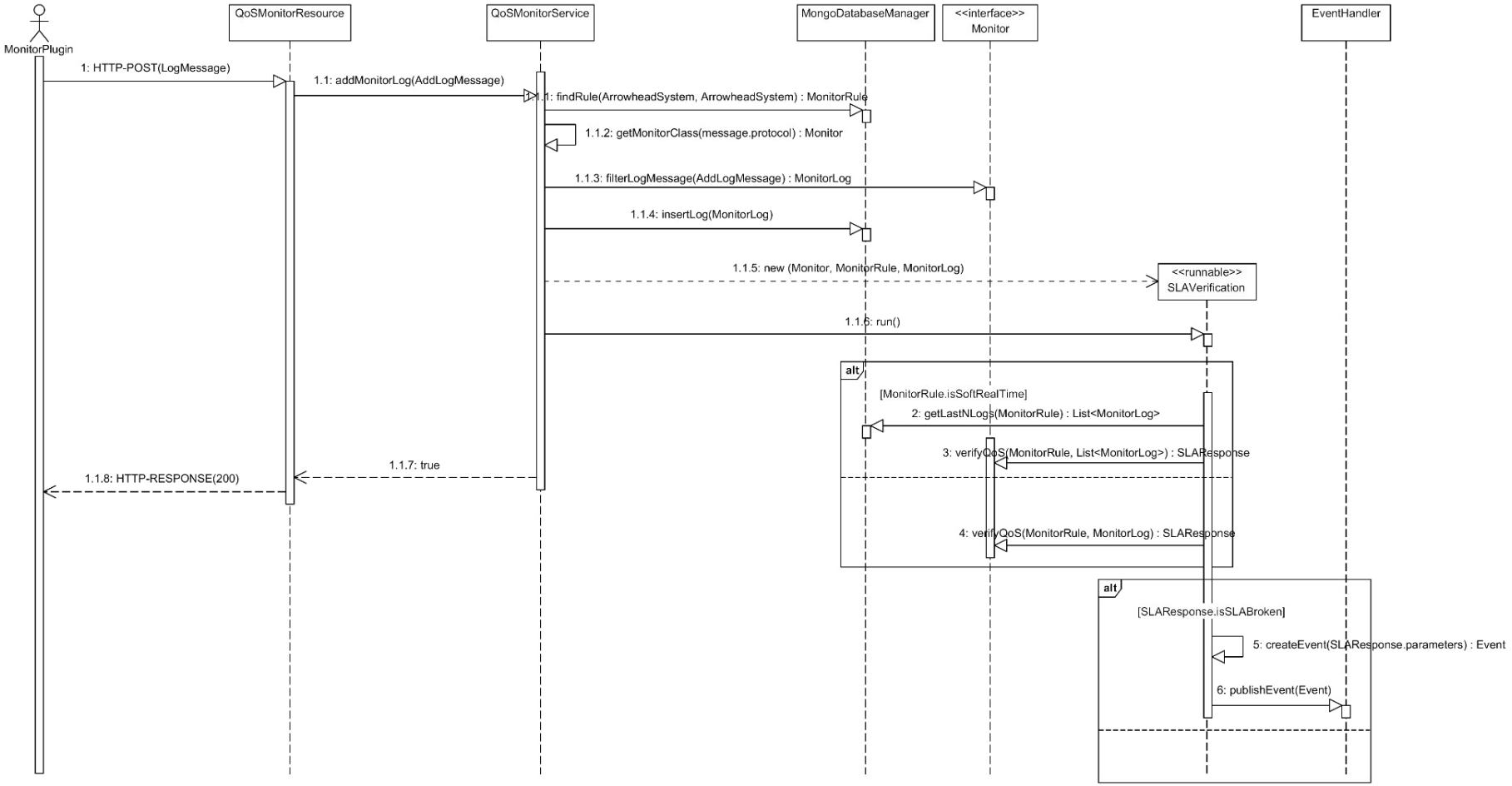


Figure 41 - QoSMonitor Sequence Diagram of UC3.

As supported by Table 63, to send an event to the EventHandler system, an Arrowhead compliant system needs to send a payload with the communication protocol and specific parameters, the source of the error, and an arbitrary error message. Normally, these events are not related to Quality-of-Service violations.

Table 63 - QoSMonitor Use-Case 4 Execution Flow.

Use-Case 4: Send Event
ID: 4
Brief description: Forwards service error descriptions as events to the EventHandler system. Normally, these events are not related to Quality-of-Service violations.
Primary actors: Arrowhead compliant system
Secondary actors: -
Preconditions: Valid payload
Main flow: 1- Arrowhead compliant system sends a service error to the system. 2- System validates the payload. 3- Creates an event with information received. 4- Sends event to the EventHandler.
Post conditions: -
Alternative flows: 2.1- The payload is not valid. 2.2- Returns bad request as response.

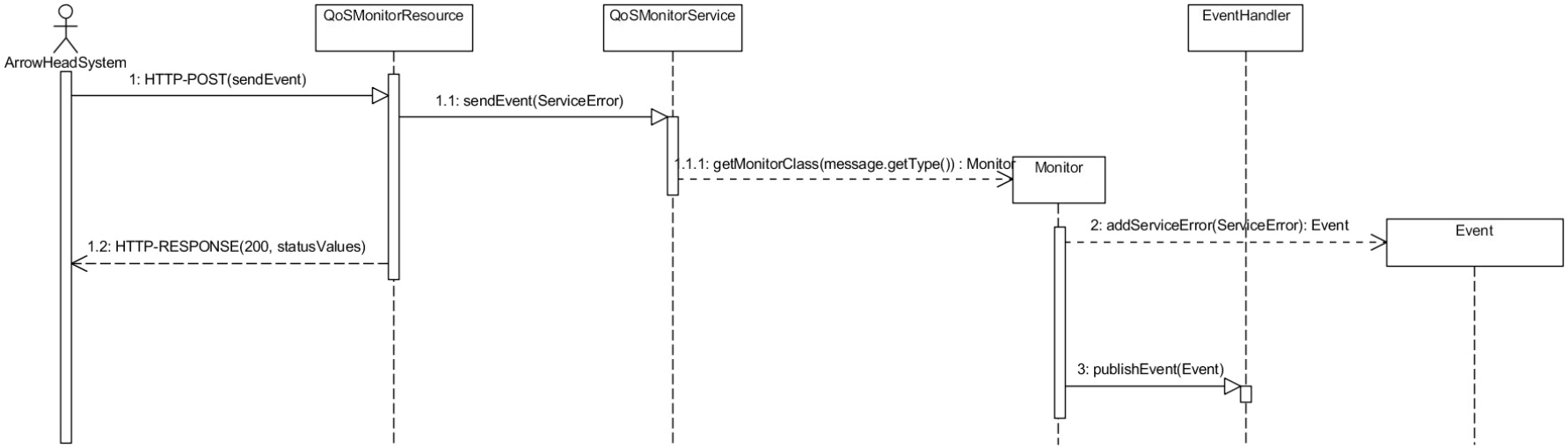


Figure 42 - QoSMonitor Sequence Diagram of UC4..

C. Graphical Interface

The QoSMonitor system also provides a way to show graphics of monitoring information. It works by defining what information goes into the graphical interface, and the QoSMonitor system works with it. Figure 43 is an example of the window that is shown when the process of receiving MonitorLog begins. Two areas stand out and are identified by the letters A and B. In the A area, information regarding the logged monitor parameters is shown in the form of area graphs. The logged parameters are the same requested in the QoSReserve process of the QoSManager system and stored in the MonitorRule. Each MonitorRule at some point has a window associated. The title of the window is the identification of the MonitorRule. In the B area, every event regarding breaks in Quality-of-Service is shown, as well as any event received by a system in the context of this window, through the SendEvent functionality. For example, in FTTSE implementation streams are used as a mean of communication between a service provider and a service consumer. The stream is identified by a number, so when using the SendEvent functionality, the stream id is sent so that the correct MonitorRule can be found and the specific window is updated.

Figure 43 - QoSMonitor system log information

D. Security

This chapter describes how security is implemented in the QoSMonitor System.

a. Decomposition of the System

None.

b. Technical Security Requirements

Any network exploit in IoT systems can cause both physical and economic damages, particularly in smart-cities, manufacturing, and transportation. The developed solution prevents these security problems.

Therefore, the QoSManager system provides a secure HTTP protocol (HTTPS) using a specific Java KeyStore (JKS) file. To interact with the QoSMonitor the user must know a private password, preventing unauthorized systems from using the QoSMonitor.

c. Data Flow Diagram

None.

d. Threats and Vulnerabilities

None.

E. Solution Description

The purpose of this chapter is to describe the implementation of the solution. Initially an overview of the system architecture is presented, supported by a component diagram, afterwards the core classes used on the code implementation are explained along with a class diagram.

Since the QoSManager must be ready to work with multiple communication protocols the Section 4.2.1 explains what the user must do to create a new one. Finally, the database the system works with is explained with the support of a database model diagram.

a. Component Diagram

As Figure 44 depicts, the QoSMonitor is divided in three major components: the Monitor and the Protocol are where the core logic is implemented, with the latter being an abstraction for a specific communication protocol. At last the DatabaseManager is responsible for all database related operations as well as storing all rules and logs. Furthermore, the major responsibility of the Monitor component is the delegation of tasks, to successfully reach the goal requested by the QoSMonitor, namely the use cases previously mentioned.

In regards to the exterior components, the QoSManager uses the system to create and delete monitoring rules after the QoSReserve process.

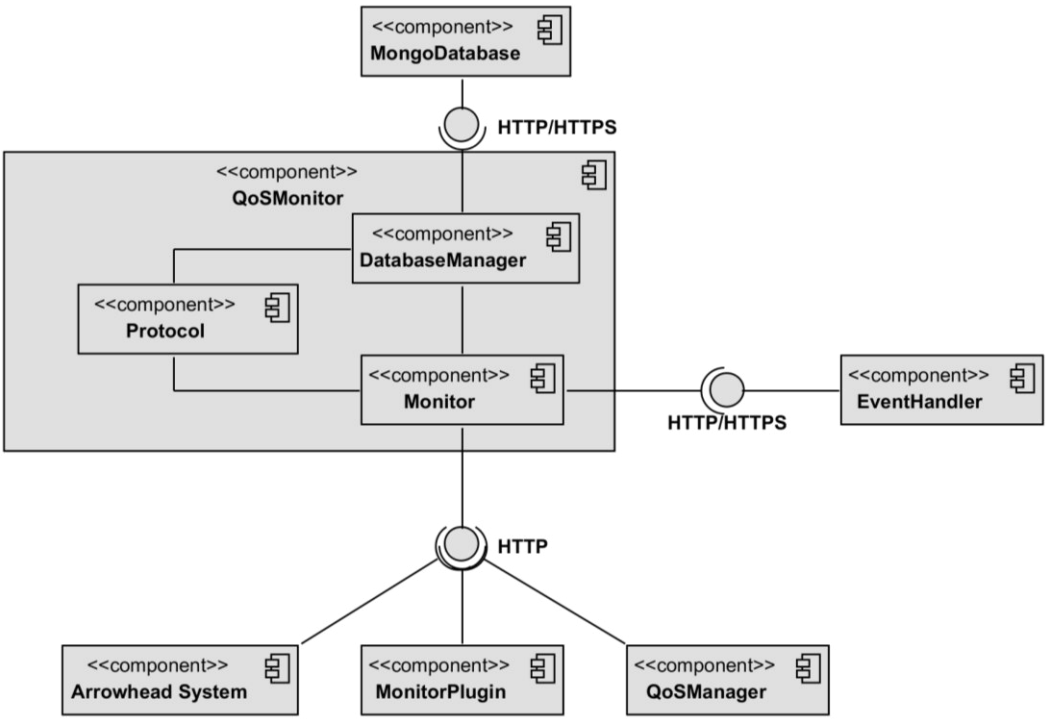


Figure 44 - Component Diagram of the QoSMonitor System.

b. Classes Structure

To accomplish both *Interoperability* and *Extensibility* requirements the code was developed according to the best practices to achieve these ends, as explained next.

Since the QoSMonitor had to work with an unknown number of communication protocols, the programming pattern *Reflection* [61] was used on the Protocol component.

The *Reflection* [61] pattern is a mechanism that allows changing a software structure and behaviour dynamically. In this project it was used to avoid code recompilation every time a new communication protocol was added, regardless the performance cost.

Another used pattern is *Repository* [63] which is related to database operations. This pattern is meant to isolate all database related operations, in order to avoid duplication of code and to simplify the business model logic. The `MongoDatabaseManager` is the class where this pattern is implemented, and it has the responsibility of managing the `MongoDatabase` operations. It also converts the database object files into model data, for further manipulation and from model into persistent data for storing purposes.

Figure 45 show a representation of how the QoSMonitor main classes are structured and what operations they offer.

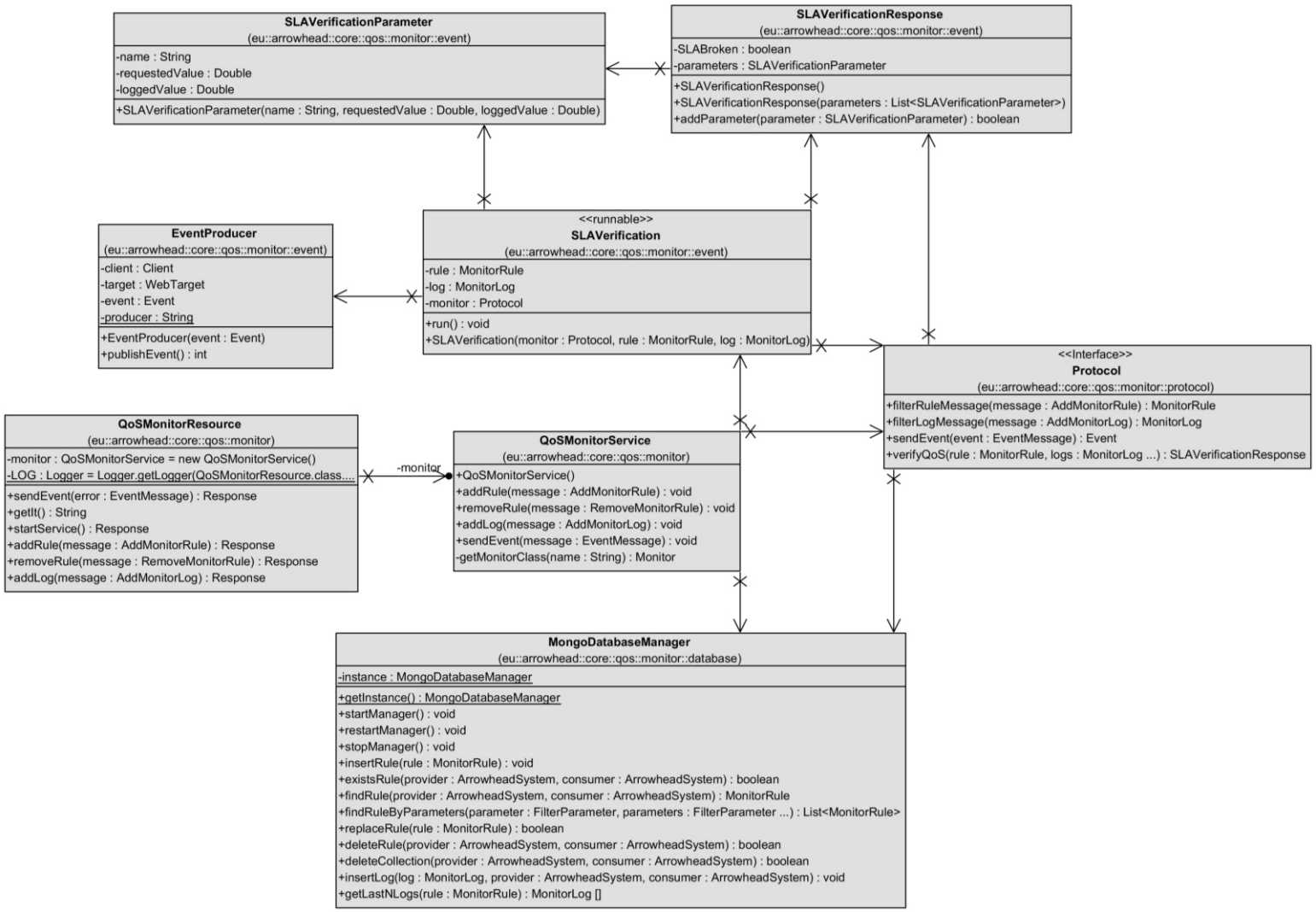


Figure 45 - Class Diagram of the QoSMonitor system.

c. Supporting new communication protocols

Each time a new communication protocol is added, a `Protocol` class be implemented. The file must have the same exact name as the protocol due to the use of the *Reflection* pattern, and it must be located in the package `qos.monitor.protocol`.

The developed class must be an implementation of the *Protocol* interface, shown in Figure 46. The functions `filterRuleMessage` and `filterLogMessage` transform `AddRule` and `AddLog` messages into `MonitorRule` and `MonitorLog` respectively, for database storage and Quality-of-Service verification.

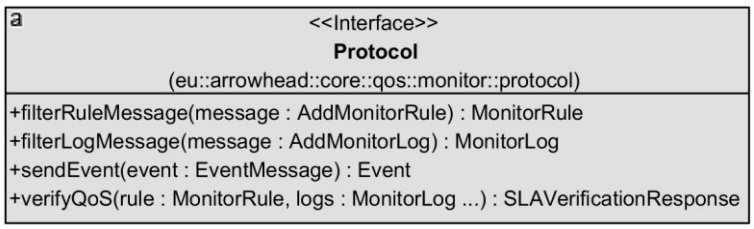


Figure 46 - Protocol interface

d. Databases

To support rules and logs functionalities that the QoSMonitor provides, the system must keep track of the configurations made in the network (i.e. rules) and logs to enable soft real time monitoring. To do this a MongoDB instance is used: a NoSQL database that stores data in BSON [66] documents, a specific type based in JSON. It guarantees great data consistency and performance. It uses Collections instead of Tables, but with the same basic purpose.

A representation of the information being stored is shown in Figure 47 and 48 as well as an explication in the respective following tables.

MonitorRule:

A MonitorRule is identified by the provider and consumer parameters. The combination of all four is unique. Rules collection saves MonitorRule.

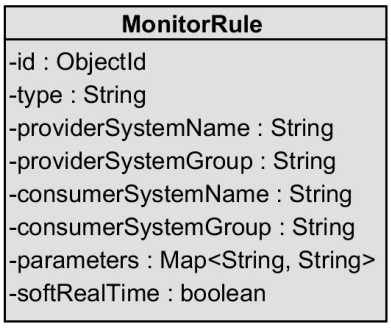


Figure 47 - Database Model of the Rule document.

Table 64 - MonitorRule collection parameters.

Field	Data Type	Unique	Description
id	ObjectId	Yes	MongoDB document identification
type	String	No	Communication Protocol type. (ex. “fttse”)
providerSystemName	String	No	Service Provider
providerSystemGroup	String	No	Service Provider
consumerSystemName	String	No	Service Consumer
consumerSystemGroup	String	No	Service Consumer
parameters	Map<String, String>	No	Requested Monitor Parameters
softRealTime	Boolean	No	Check value for soft real time or real time monitoring

MonitorLog:

A MonitorLog collection is created for each MonitorRule, and is named using the provider and consumer parameters. For each log received in `addLog` function, a MonitorLog document is saved in the respective collection.

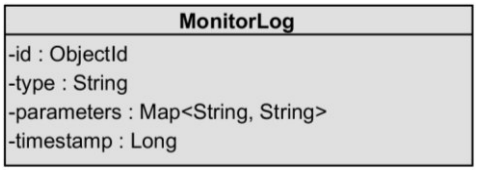


Figure 48 - Database Model of the Log document.

Table 65 - MonitorLog collection parameters.

Field	Data Type	Unique	Description
id	ObjectId	Yes	MongoDB document identification
type	String	No	Communication Protocol type. (ex. “fttse”)
parameters	Map<String, String>	No	Logged Monitor Parameters
timestamp	Long	Yes	Timestamp of monitor

e. Deployment Diagram

Regarding the deployment, shown on Figure 49, three machines were used to deploy all the necessary components to a properly functioning system. The QoSMonitor was deployed in a machine with the Windows 10 OS in a Apache Tomcat servlet container. The database instance which the QoSMonitor works with was installed in a Linux server

machine. The EventHandler system was deployed in a machine with Windows 10 OS in a Grizzly server. A deployment diagram is depicted in figure 49.

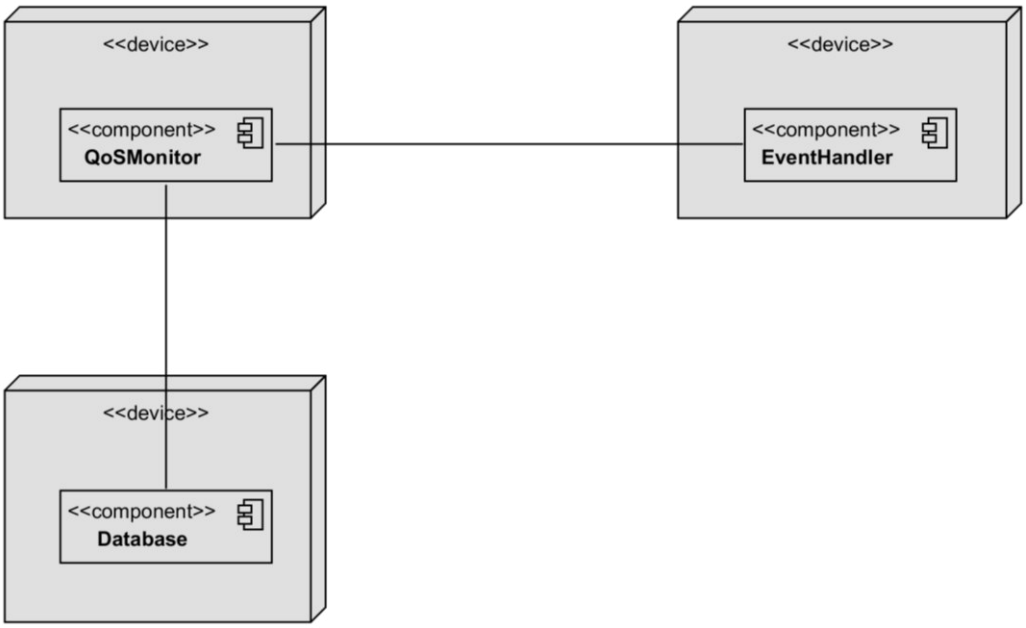


Figure 49 - Deployment Diagram of the QoSMonitor system

4.7 System-of-Systems Design Description/Pilot Project

This section describes how the Arrowhead with QoS support solution has been implemented on an FTT-SE scenario, describing the technologies used and its setup.

A. Overview

The QoS-as-a-Service in the Local Cloud was implemented on the Flexible Time Triggered Switched Ethernet (FTT-SE) [67] which is a Time Triggered Model capable of providing a "real time" network on the *Ethernet*. The goal of this pilot project was to prove the QoS functionality of the Arrowhead Framework.

FTT-SE is implemented over Raw Sockets to access the MAC layer of the Ethernet hardware, and it uses these sockets to transmit and receive data between all nodes. The solution does not make use of the Internet Protocol (IP), and instead it connects the nodes with a switch in a layer 2 topology, using MAC addresses to address the nodes and establish communications.

During the integration of all components, several issues emerged, and the two most important are discussed in this section.

a. Address Incompatibility

The addresses used in the REST-based Arrowhead communication are the ones of the IP protocol stack. On the other hand, FTT-SE uses the MAC addresses of the nodes. Since Arrowhead only works with TCP/IP, two possible solutions were proposed.

1) The first one was the use of the TunTap [68] technology to create a generic interface allowing the use of TCP/IP over FTT-SE. This solution makes possible TCP/IP transmissions between internal and external devices in FTT-SE, since TunTap would receive them and retransmit them over FTT-SE.

2) Another proposed solution to this problem was to use multiple network interface on the nodes. Every node had an IEEE 802.3 interface managed by the FTT-SE protocol for the service fruition, and another interface, wireless or not, to perform TCP/IP communications with Arrowhead.

The second solution was chosen due to the remaining time of the project since it was obvious that the first solution would require more development and analysis time. However, in future work, the first solution must be developed instead of the second one, since it avoids the use of unnecessary technologies, mainly Berkeley Sockets, and demands less performance from the nodes**.**

b. Deployment of the Network

Another issue with the topology of the demonstrator was that FTT-SE needed to be deployed on a dedicated network to work properly, since interferences with nodes, not respecting the FTT-SE protocol, would impair the protocol.

It was decided to deploy a local network of nodes providing and consuming services, plus a node managing the FTT-SE (FTT-SE Master node, as per the FTT-SE specifics); nodes talk with each other through the FTT-SE interface. Moreover, all the mentioned nodes and an EntryPoint node interact through the IP interface (see Section a). The EntryPoint node has also got a public IP address to reach (and be reached by) the Arrowhead Framework, and act as the internet gateway for all other nodes.

Figure 50 presents an overview of the implemented topology. Relatively to the FTT-SE network, three FTT-SE nodes (master node, consumer node and provider node) are connected to a switch. The EntryPoint, the service producer and consumer nodes are also connected via their secondary interfaces, a wireless one, to avoid interferences with the FTT-SE network.

The Arrowhead network consists in three core systems: the Orchestrator, Service Registry and Authentication. The Orchestrator is used to create the matching between service producers and service consumers, to allow service fruition. The Service Registry allows the registration of systems and services in the Arrowhead Local Cloud. The authentication is used to authenticate and provide Authorisation for connections between services. These systems are vital for the Arrowhead operations.

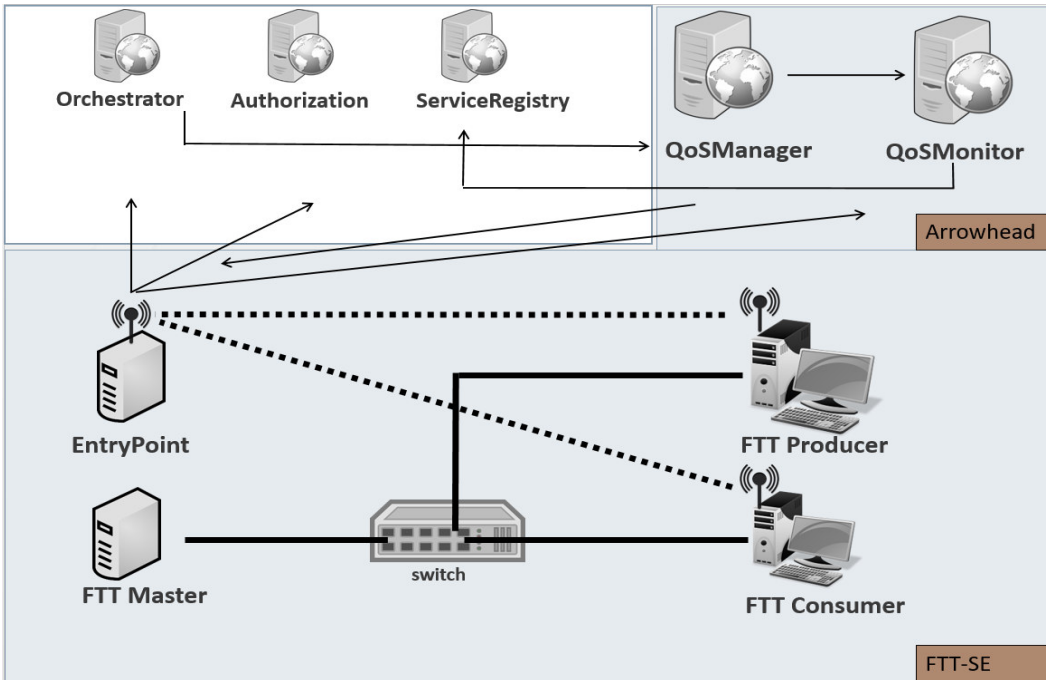


Figure 50 - Disposition of all devices used on the FTT-SE and Arrowhead integration.

On a component perspective, as Figure 51 depicts, the Arrowhead network contains the Orchestrator, QoSManager and QoSMonitor systems. The QoSManager interacts with two databases, the SystemConfigurationStore and QoSStore. The QoSMonitor interacts only with the Monitor Store. Relatively to FTT-SE network, both consumer and producer nodes must have a plugin to work with Arrowhead. In addition, the EntryPoint must also contain an application capable of retransmitting any request to or from Arrowhead.

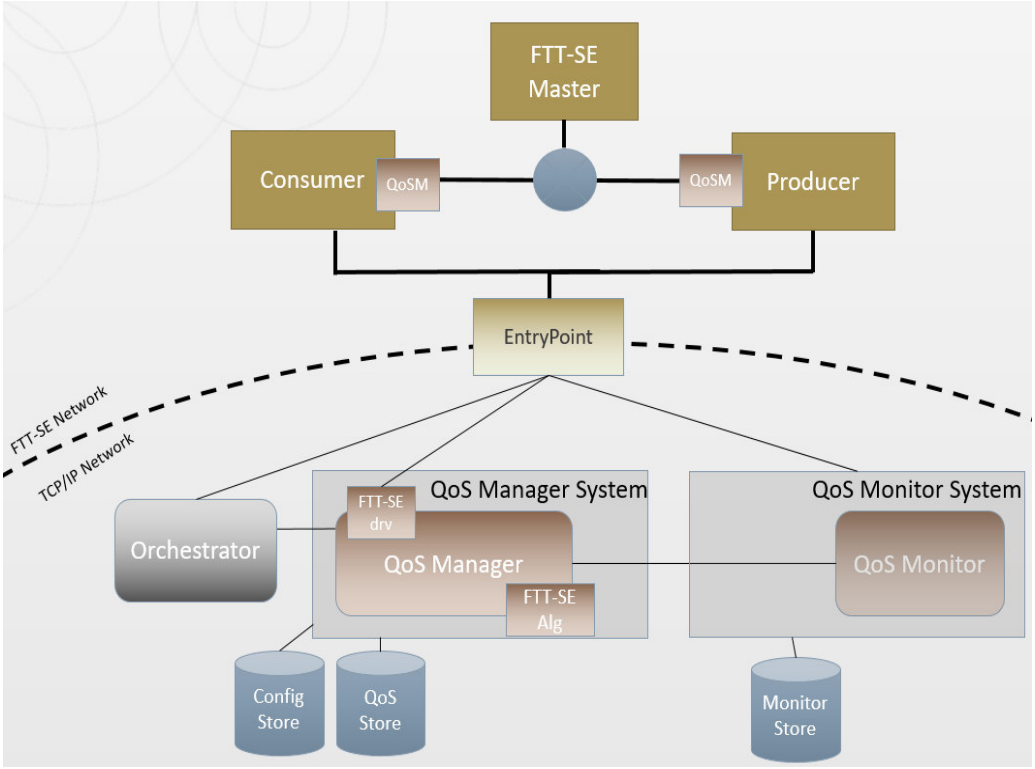


Figure 51 - Component Diagram of the integration of Arrowhead with FTT-SE.

B. Deployment Architecture

FTT-SE requires at least one Master node to manage all communications, and every node must share the same 100Mbps switch to exchange both data and control messages. Since the FTT-SE application only works on Linux Operative System (OS), all used machines had installed the Debian GNU/Linux 7.7 (wheezy) OS [69], as shown in Figure 53. Each machine had two network interfaces, one Ethernet interface dedicated to FTT-SE communications, and a wireless interface responsible for the communications with the Arrowhead Framework.

In the deployment of FTT-SE, each node was installed on a quad core laptop, all connected by a 100Mbps switch, as per Table 66.

Table 66 - Description of the used devices along and its usage.

Devices	Used For
HP Probook 6460b [70]	FTT-SE application as a Master/Consumer/Producer Node
Switch TP-LINK SF1008D (100Mbps Full-Duplex) [71]	Connect the Master and producer/consumer nodes for FTTSE

Since the FTT-SE code is written on C language (ANSI-C [72]) it was decided to deploy on each consumer and producer node a Berkeley Socket server to communicate with the EntryPoint machine. Therefore, any service request or registry is communicated to the EntryPoint via Berkeley Sockets.

The EntryPoint is responsible for converting any socket message to HTTP and vice-versa, making possible establishing a connection between any FTT-SE producer/consumer and the Arrowhead Framework. Therefore, in the EntryPoint it was developed a Web Server, communicating towards the Arrowhead Framework, and a Socket Server, interacting with the local nodes. The OS used is Windows 10 [73].

The full deployment features four machines located on the FTT-SE network to guarantee communication with the Arrowhead Framework, those are the EntryPoint, the Consumer, the Producer and the Master.

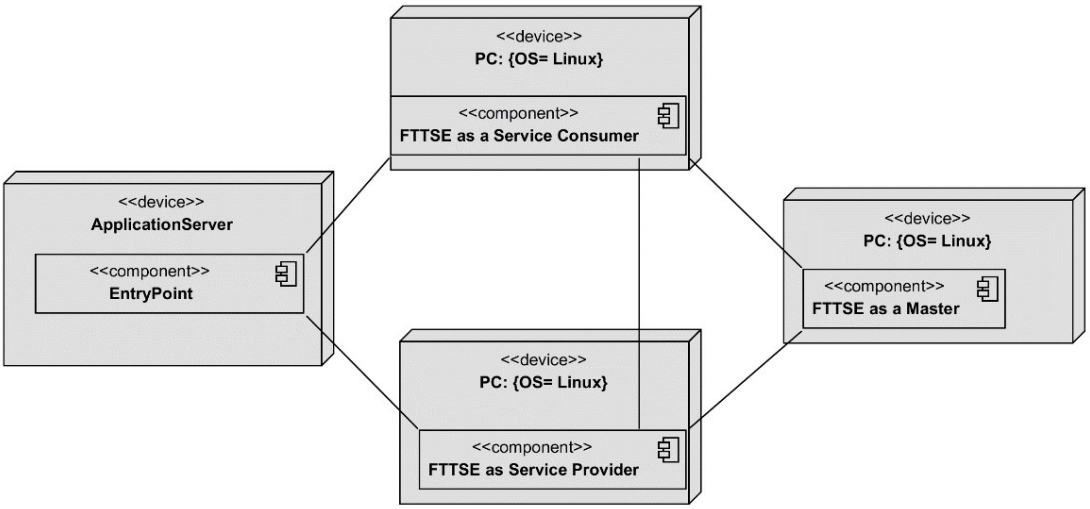


Figure 52 - Deployment Architecture on FTT-SE

C. Components Architecture

This section is divided into three Sections; it starts by describing the components deployed on the FTT-SE node including the monitoring capabilities and operations. Further, it also details the EntryPoint responsibilities, operations and communications. Finally, the section ends by explaining the monitoring capabilities and how the message streams are monitored.

In an overall view, the only node that did not need any development work was the master node. The existing FTT-SE application was modified with new components named monitoring, core and services requester. As described on Figure 53, any client that wishes to register a service, or look up one, must be running an application named “FTTSE_Arrowhead_Plugin”.

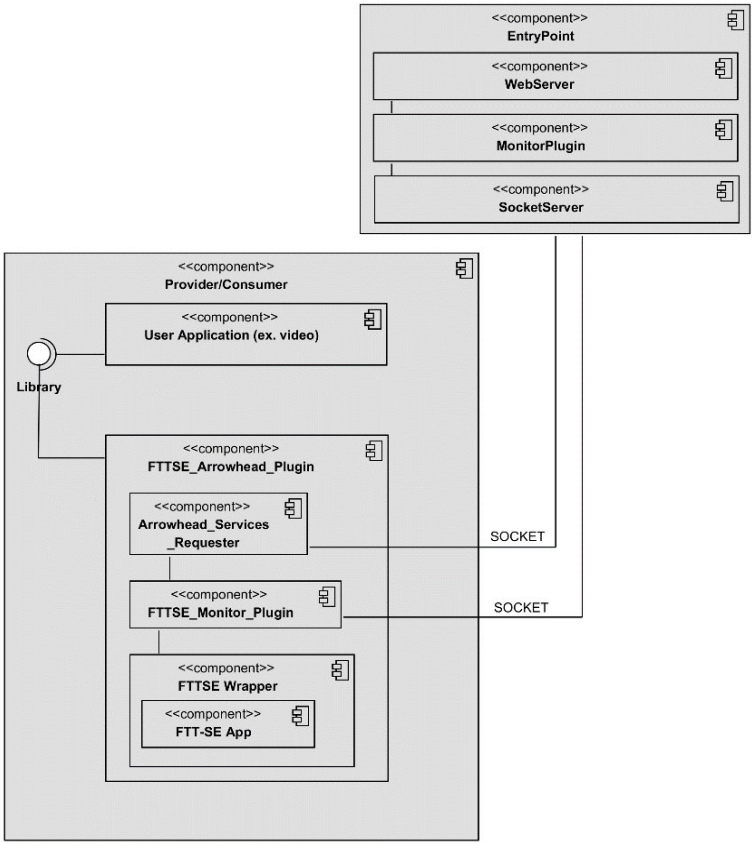


Figure 53 - Component Diagram of FTT-SE.

a. FTT-SE node

The `FTTSE_Arrowhead_Plugin` is divided in three independent components. The first one is `FTT-SE Wrapper`, which contains the original FTT-SE application incorporated on an IEEE 802.3 interface. The `FTSSE Wrapper` component had been developed in a past project, and its goal is to simplify all possible interactions with the original FTT-SE app by working as a wrapper and providing a simplified API. The remaining components were developed specifically for this pilot project.

The first developed component was the `FTTSE_Monitor_Plugin` which has all the logic components necessary for the application to send/receive messages and monitor them. Another responsibility of this component is monitoring, during the message streams, all the streams delay, bandwidth and critical events.

The `ArrowheadServicesRequester`, which is used by the `FTTSE_Monitor_Plugin`, is responsible for services registering and for the requests made to the Arrowhead via the EntryPoint node.

To use the `FTTSE_Arrowhead_Plugin` the header `core_public.h` must be included. Through the `core_public.h` the user can provide or request a service to Arrowhead and, when a stream is configured, it can send and receive messages. Figure 54 shows that there are two important parameters that are set when a stream is configured, the application parameter that contains all the registered streams, and the

information for the monitoring operations. The *application stream* parameter contains all the configuration parameters for the stream, such as the identifier and the message size. It is through the application stream that the user can put the data message that he wants to send.

The application provides twelve functions, as Figure 54 describes, to guarantee the necessary operations of the application. How to use these functions is described next.

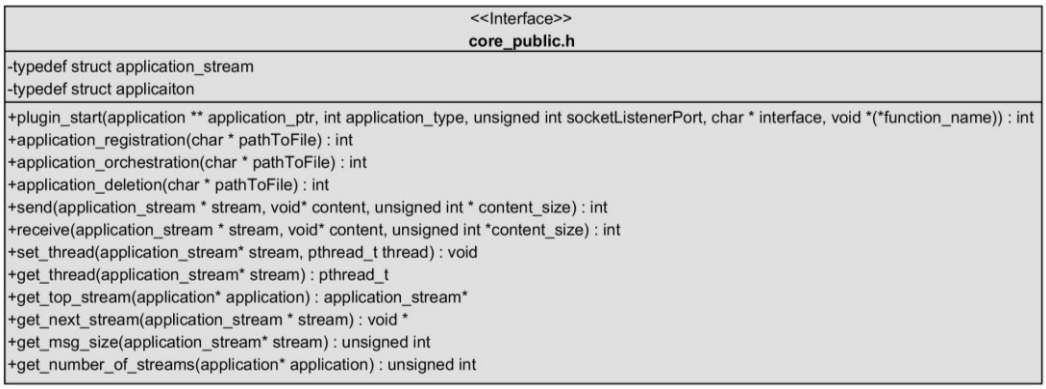


Figure 54 - Class Diagram of the FTT-SE interface.

To use the `FTTSE_Arrowhead_Plugin`, both consumer and producer must call the function `plugin_start()`, as shown in Figure 55. The function initiates a slave node in FTT-SE and creates the socket server to listen for any Arrowhead request. The required parameters are the following: `consumer_application` that will be initialized by FTT-SE and has all the streams and informations; the `TYPE_CONSUMER` is a number that will define the type of the slave, it can also be `TYPE_PRODUCER`; the `SOCKET_PORT` is the port where the socket server will listen; the interface is the name of the network interface where FTT-SE will operate (ex. “eth0”); the `receives()` is the function that will be called whenever a stream is created.

```
application * consumer_application;

plugin_start(consumer_application, TYPE_CONSUMER,
SOCKET_PORT, INTERFACE, receives);
```

Figure 55 - Execution of the FTT-SE plugin.

The developed plugin creates a stream whenever a consumer requests a service. Therefore, a stream has only one consumer and producer. The plugin also allows a producer to have multiple consumers. Figure 56 and 57 shows basic transmitting and receiving functions that the user must create in the FTT-SE plugin.

```
void * receives(void * stream) {
    application_stream * message_stream = (application_stream*) stream;

    unsigned char rec[get_application_stream_stream_size(message_stream)];
    unsigned int received_msg_size = -1;
    int ret = 0;
    while (ret != -1) {
        ret = receive(message_stream, &rec, &received_msg_size);
    }
}
```

Figure 56 - Basic receive function of the FTT-SE plugin.

```
void * transmits(void * stream) {
    application_stream * message_stream = (application_stream*) stream;

    char * content[] = "Hello World";
    unsigned int received_msg_size = -1;
    int ret = 0;
    while (ret != -1) {
        int ret = send(message_stream, &content, sizeof(content));
    }
}
```

Figure 57 -- Basic transmit function of the FTT-SE plugin..

b. EntryPoint

In addition to the FTT-SE components, it was necessary to deploy a new device named EntryPoint. This System is not Arrowhead compliant, and it was developed as a workaround for the FTT-SE limitations. The EntryPoint is responsible for retransmitting all the messages originated by any producer/consumer node to the Arrowhead Framework and vice-versa. It is divided into 3 components, the `WebServer` which receives and processes any input from the Arrowhead Framework; the `MonitorPlugin` that receives all the statistics from the `FTTSE_Monitor_Plugin` and retransmits to the Arrowhead; the `SocketServer` which receives all the inputs from any producer/consumer node via socket.

During the retransmissions, the EntryPoint must change the messages protocol from socket to HTTP or the other way around, since it acts as gateway between the FTT-SE messages sent via socket, and the REST based world of the Arrowhead requests sent via HTTP.

There are six use cases associated to the EntryPoint, which connects to any Producer/Consumer FTT-SE node, the QoSDriver (that is integrated on the QoSManager system), the Orchestrator, the ServiceRegistry and the QoSMonitor, as Table 67 shows.

Table 67 - EntryPoint Use-Cases.

Use Case	From	To	Protocol	Interface
Configure Stream	QoSDriver	Service Producer/Consumer	REST-JSON	http:/<ip address>:<port>/entrypoint/configure
Service Registration	Service Producer	Service Registry	SOCKET-JSON	port: 9999
Service Deletion	Service Producer	Service Registry	SOCKET-JSON	port: 9999
Critical Event Notification	Service Producer/Consumer	QoSMonitor	SOCKET-JSON	port:9999
Monitoring Logs	Service Producer/Consumer	QoSMonitor	SOCKET-JSON	port:9999
Service Request	Service Consumer	Orchestrator	SOCKET-JSON	port:9999

c. Monitoring

This section describes the monitoring plugin in two different sections: its capabilities; and the functioning of the plugin.

Capabilities

Currently, monitoring has the capability to track the message streams delay, bandwidth and any critical event. On each message transmission and reception, the time and size of the message are logged and sent to the Arrowhead framework to verify if the QoS is being fulfilled.

Three characteristics are important for the Monitoring configuration. “delay” is the time interval between send messages, meaning that if the second message was sent 30ms after the first one, the “delay” will be 30ms. “Bandwidth” is the maximum message size sent per unit time. Time to time the QoSMonitor will check if the last recorded throughputs are superior to the defined bandwidth, if so, notifications and warnings are triggered. A third characteristic is the message stream consistency, which is also being monitored in real time by the plugin, detecting any eventual critical event. A critical event is considered as an abnormal situation that prejudices or even stops the stream between producer and consumer (ex. a deadline that was not fulfilled).

Monitoring data are accumulated onto a queue and the Monitor component transmits its logs to the QoSMonitor system every 300 milliseconds. The monitoring data has thus a maximum delay of 300ms with respect to data collection.

One more QoS parameter that was initially considered is “response-time”. Due to the FTT-SE specifics (communications are half-duplex, meaning that the producer cannot get

a response of the consumer whenever it sends a message), this QoS parameter immediately was discarded.

Monitoring Plugin Functioning

The Monitor Plugin takes care of sending message logs to the QoS Monitor from the monitored nodes.

One issue was raised during the monitoring tests: it was not possible to send message log reliably every 20ms. This caused data inconsistency and in worst cases the loss of some log data. Since one major non-functional requirement of FTT-SE was the capability of providing a message with a minimum delay of 20ms, and each FTT-SE message was monitored, a solution to the monitoring problem ought to be found.

The solution was the use of a queue, to save all incoming logs, and periodically a specific thread would flush the queue content and send it to the QoSMonitor via a message. After some tests, it was concluded that the minimum delay that would guarantee content consistency in our deployment was of 300ms.

Whenever a user decides to transmit or receive data, the `FTTSE_Monitor_Plugin` component, that provides an interface to the user (producer or consumer), will add to the queue all the monitoring statistics relative to that sent/received message stream. A specialized thread, named monitor thread, flushes the queue content every 300ms and sends the data to the QoSMonitor via the EntryPoint node. Both queue and thread are responsible for one stream only, therefore each stream will have its dedicated monitor thread and queue.

Since the used programming language was ANSI-C [72] and natively it doesn't provide dynamic lists, dedicated wait-free queues were studied and tested, among the several queues available on GitHub [74]. One main requirement about the queue implementation was that it would be wait-free, to avoid any interruption that would compromise the FTT-SE operations.

D. Use-Cases

The `FTTSE_Arrowhead_Plugin` has three possible use-cases, as Figure 58 shows:

1- Service Registration – a user service is registered and made available to provide services to other consumers by means of the Arrowhead Framework

2- Request of a Service – a consumer requests a service with or without QoS, and the Arrowhead Framework provides it by configuring a FTT-SE stream between that consumer and the provider

3- Service Deletion – a service selected by the user is deleted, and will be no longer provided by the Arrowhead Framework.

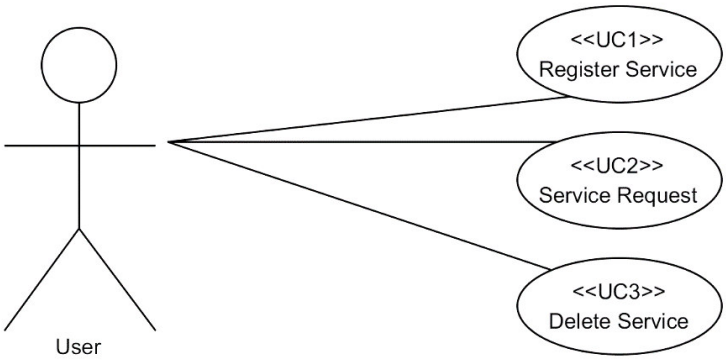


Figure 58 - FTTSE Use Cases List.

All three use-cases require a properties file to point the service and system that the user wants to register/request/delete. All files contain a JSON structure.

In use-cases 1 and 3, a properties file described on Figure 59 is used. In use case 2, the property file is described in figure 60.

Several important details related to the property file content should be highlighted. Firstly, all file parameters must be present, and the values must obey the rules that Table 68 describes.

Table 68 - Description of the parameters contained on the properties files.

ID	File Parameter	Necessary in	Description
1	serviceGroup	UC1,2,3	String containing a name for the group where the service belongs.
2	serviceDefinition	UC1,2,3	String containing the name of the service.
3	interfaces	UC1,2,3	List of strings containing all available interfaces protocols to access the service.
4	serviceRegistryEntry: provider:systemGroup	UC1,3	String containing the name of the group where the system belongs.
5	serviceRegistryEntry: provider:systemName	UC1,3	String containing the name of the system.
6	serviceRegistryEntry: provider:address	UC1,3	String containing the address of the system.
7	serviceRegistryEntry: provider:port	UC1,3	String containing the port of the system where users establish a connection.
8	serviceRegistryEntry: provider:authenticationInfo	UC1,3	String containing information about the Authorisation procedure of the system.
9	serviceRegistryEntry: serviceURI	UC1,3	String containing the URI of the service.
10	serviceRegistryEntry: serviceMetadata	UC1,3	Map of strings containing a description of the service.

11	serviceRegistryEntry: tSIG_key	UC1,3	String containing the access key of the DNSSD server allowing to register/delete a service
12	serviceRegistryEntry: version	UC1,3	String containing the version of the service.
13	orchestrationFlags	UC2	Map of strings containing the selected options to the service request. For this project one necessary string was needed, triggerInterCloud. TriggerInterCloud must be false, because the QoS on Arrowhead only works in Local Clouds.
14	requestedService: *	UC2	See descriptions 3, 10, 2, 1.
15	requestedQoS: delay	UC2	Integer (milliseconds) containing the maximum delay of the message stream. This parameter is Optional.
16	requestedQoS: bandwitdh	UC2	Decimal (Bps) containing the maximum bandwidth for the message stream. This parameter is Optional.
17	requesterSystem: *	UC2	See descriptions 4, 5, 6, 7, 8.

```
{
"serviceGroup":"ServiceGroupA",
"serviceDefinition":"ServiceDefinitionA",
"interfaces":["RESTJSON"],
"serviceRegistryEntry":{
        "provider":{
                "systemGroup":"SystemGroupA",
                "systemName":"SystemNameB",
                "address":"127.0.0.1",
                "port":"8080",
                "authenticationInfo":"authinfo"
        },
        "serviceURI":"/video/3",
        "serviceMetadata":[{"key":"location","value":"Portugal"}],
        "tSIG_key":"AAABBBCCCDDD=",
        "version":"1.0"
        }
}
```

Figure 59 - Properties file necessary to register/delete a service.

```
{
"serviceGroup":"ServiceGroupA",
"serviceDefinition":"ServiceDefinitionA",
"interfaces":["RESTJSON"],
"serviceRegistryEntry":{
        "provider":{
                "systemGroup":"SystemGroupA",
                "systemName":"SystemNameB",
                "address":"127.0.0.1",
                "port":"8080",
                "authenticationInfo":"authinfo"
        },
        "serviceURI":"/video/3",
        "serviceMetadata":[{"key":"location","value":"Portugal"}],
        "tSIG_key":"AAABBBCCCDDD=",
        "version":"1.0"
        }
}
```

Figure 60 - Properties file necessary to request a service

Table 69 - Use Case 1 Execution Flow.

Use Case 1: Service Registration
ID: 1
Brief description: The user registers a service.
Primary actors: Client
Secondary actors: EntryPoint, Service Registry (SR).
Preconditions: The User must have a properties file with the service information, containing the service description and the security key that allow him to register a service on the Service Registry (SR).
Main flow: 1- A Service Provider requests registration of a service. 2- The "FTTSE_Arrowhead_Plugin" sends the registration request to the EntryPoint. 3- The EntryPoint will validate and retransmits via REST to the Service Registry System (SR) of the Arrowhead Framework. 4- The SR will notify of the success of the registration. 5- The EntryPoint will return the same SR response back to the node where the user made the request.
Post conditions: -
Alternative flows: 2.1-If the file was not found an error message will be shown. 3.1- If the properties file has errors, the entry point will give an error back to the user.

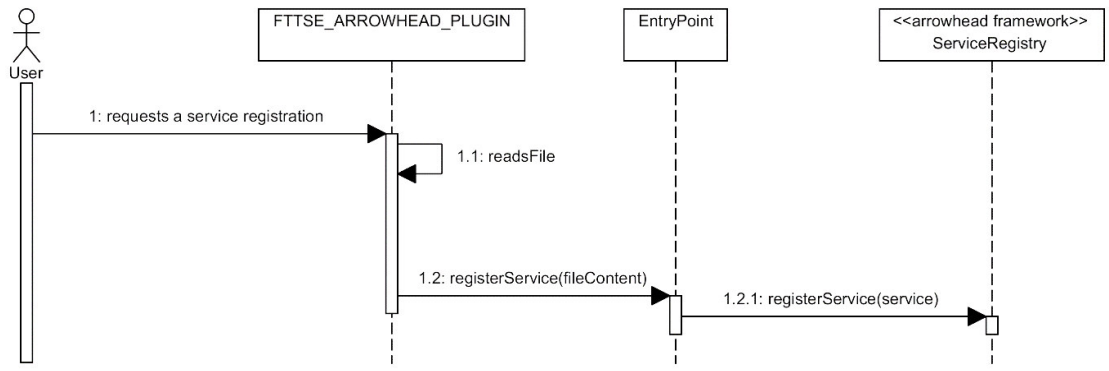


Figure 61 - Sequence Diagram of UC1.

Table 70 - Use Case 2 Execution Flow.

Use Case 2: Request of a Service
ID: 2
Brief description: The user requests a service.
Primary actors: Client
Secondary actors: EntryPoint, Orchestrator System.
Preconditions: The User must have a properties file with the service request information.
Main flow: 1- A Service Consumer requests a service. 2- The "FTTSE_Arrowhead_Plugin" sends to the EntryPoint the request. 3- The EntryPoint validates the request and if successful sends to the Orchestrator System of the Arrowhead Framework the request made. 4- The Arrowhead Framework will configure the streams between a producer and a consumer nodes on FTT-SE, sending the configuration to EntryPoint. 5- The EntryPoint will send to both producer and consumer the configuration message. 6- The FTTSE_Arrowhead_Plugin will receive the stream configuration and if successful will return success back to the EntryPoint. 7- The EntryPoint will communicate back to the Arrowhead the success. 8- The Orchestrator will finally send a response of the service request. 9- The EntryPoint will return the message to both producer and consumer nodes. 10- The consumer can now receive the data, and the producer transmit it.
Post conditions: -
Alternative flows: 2.1 - If the file was not found an error message will be shown. 3.1 - If the properties file has errors, the entry point will give an error back to the user. 4.1- If the Orchestrator doesn't have any service like the one that was requested, an error message will be sent to the EntryPoint and consequently back to the user.

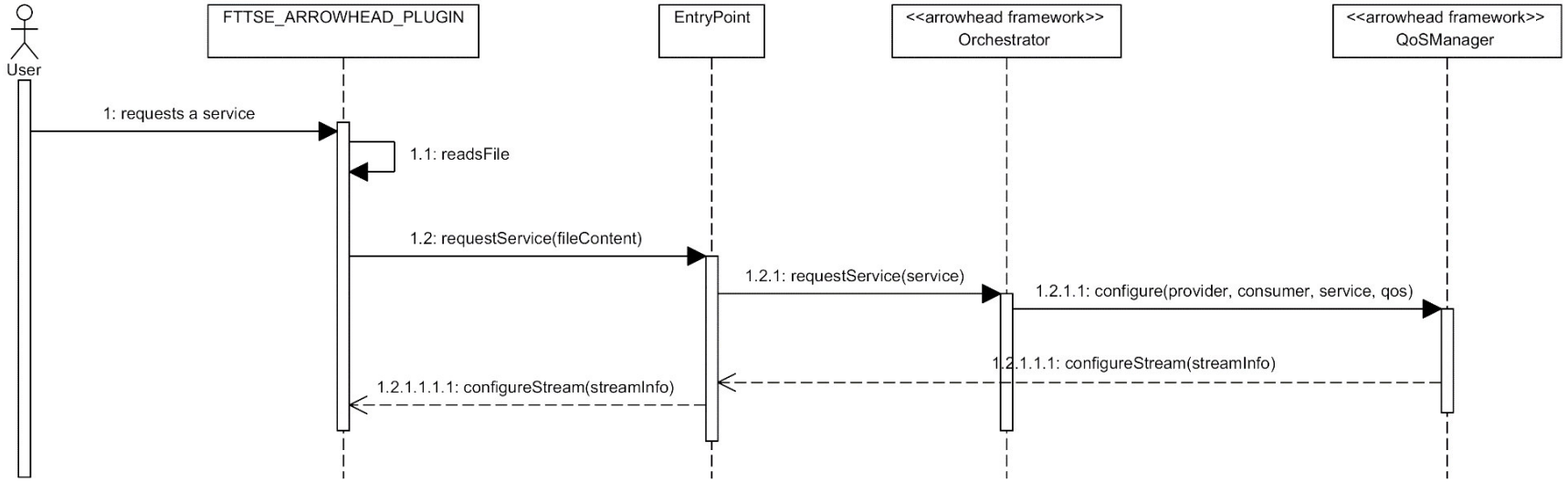


Figure 62 - Sequence Diagram of UC2.

Table 71 - Use Case 3 Execution Flow

Use Case 3: Service Deletion
ID: 3
Brief description: The user deletes a service.
Primary actors: Client
Secondary actors: EntryPoint, Service Registry (SR)
Preconditions: - The user must have a properties file containing the service description and a security key that allows him to delete a service on the SR. - The service must exist on the SR.
Main flow: 1- A Service Provider request a service deletion. 2- The "FTTSE_Arrowhead_Plugin" sends to the EntryPoint. 3- The EntryPoint will validate and retransmits via REST to the Service Registry System (SR) of the Arrowhead Framework. 4- The SR will notify of the success of the deletion. 5- The EntryPoint will return the same SR response back to the node where the user made the request.
Post conditions:
Alternative flows: 2.1 - If the properties file was not found, use case ends with warning. 3.1 - If the properties file fails the EntryPoint validation, the use case ends with warning.

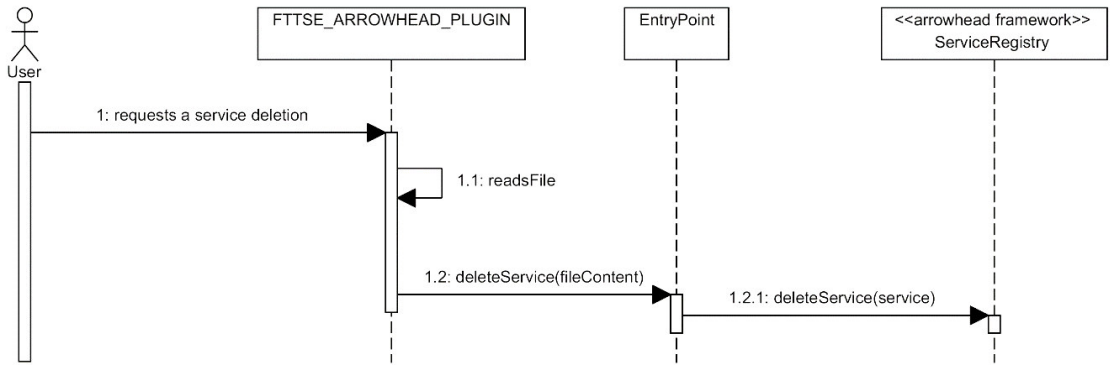


Figure 63 -- Sequence Diagram of UC3.

E. Systems

Beyond the SR and the Orchestrator Systems that are the principal systems of the Arrowhead Framework, there are two more Systems, which deserve a description, as Table 72 depicts. These two systems communicate with the EntryPoint whenever a stream is created, and are support systems for the Orchestrator.

The QoSManager has a considerable responsibility on the Use Case 2 because its QoSDriver sends to the EntryPoint the configuration of the stream between the service provider and consumer, according to the requested QoS.

The QoSMonitor will act as a receiver since it only receives the message streams monitoring data regarding delay, bandwidth and events, all sent from the service consumer and provider nodes.

Table 72 - Systems involved.

System name	Path
QoSManager	Section 4.2.1.
QoSMonitor	Section 4.2.2.

Although the QoSDriver is not a system, it is worth mentioning because it is instrumental to allow Arrowhead to interact with the FTT-SE protocol, whose configuration is not REST-based. For each stream, it is necessary to set the FTT-SE network configurations, in particular, the current stream period, id and the EntryPoint URL.

In FTT-SE, the EC (ms) consists in a time interval where all FTT-SE messages occur. The stream id is a number that identifies a stream, therefore to avoid any inconsistence/duplication, the stream id is incremented whenever a new service request is made. The EntryPoint URL gives the location where the QoSDriver makes the streams configuration requests, since the interaction with the QoSDriver is initiated by this last component and thus the EntryPoint must be reachable as a server.

Currently the QoSDriver is capable of creating a stream in accordance with two QoS goals, delay and bandwidth.

The FTT-SE parameters that allow the management of the delay and bandwidth are the stream period and size. The stream period is the number of ECs between each message transmission, corresponding to the time interval between messages transmission. If the EC is 20ms and the period is three, that means a message will be sent every 60ms (20*3). Note this time interval varies according to the stream type, which can be Synchronous or Asynchronous. If the stream is asynchronous the period value will be the maximum interval time, and in the case of synchronous will be the exact time interval. This happens because FTT-SE acts as Event-Triggered for asynchronous communication, and Worst Case Communication Time must be considered. For synchronous communication, FTT-SE is Time-Triggered. The size (Bytes) corresponds to the message size.

To impose the delay, a period must be calculated by dividing the requested delay by the EC, as Figure 64 represents. If the delay is 80ms and the EC 20ms, the period value will be four ECs. If no delay is chosen, the period will have the default value of five.

Relatively to the bandwidth, as Figure 65 shows, the stream message size is calculated by multiplying the message delay (ms) by the requested bandwidth (Bps), then dividing by 1000 (ms) due to the seconds and milliseconds unit differentiation. If the requested bandwidth is 30 Bps and the message delay (EC*PERIOD) is 20ms, the message size will be 1.5 B.

```
int EC = getEC();
int PERIOD = delay/EC; //number of ECs
```

Figure 64 - Period Calculation

```
int EC getEC(); //ms
int PERIOD = getPERIOD(); //number of ECs
float SIZE = EC*PERIOD*BANDWITH/1000; // Bytes
```

Figure 65 - Bandwidth calculation

F. Non-Functional Requirements Realization

- The FTT-SE must be capable of providing messages streams with a minimum delay of 20ms.
- The QoS parameters monitoring must have a maximum delay of 500ms.
- Regarding security, all the data exchanged between the producer/consumer nodes with the EntryPoint via socket is not encrypted. However the messages between the QoSManager system and the EntryPoint are protected with a SSL/TLS protocol that uses Java Key Stores.

G. Proof of Concept/ Acceptance Test

After the development of all components, to test the project, it was decided to transmit a file, specifically a video, between an Arrowhead Service Producer with a Consumer. To prove that the QoS was being accomplished, two scenarios were created. On scenario 1 a user requests a service with QoS (20ms of delay and 300KBps of bandwidth) under a very congested network; on scenario 2, on the same congested network, the user requests the same video service without QoS.

To reproduce the video, it was used the video player MPlayer [75] because it had the capability of reading from a pipe. The goal of the use of a pipe was to play the video while is still being transmitted by the producer. It was expected that during the reproduction of the video of the service without QoS the video would stop and be inconsistent; in the service request with QoS, the reproduction of the video would be fluid. Relatively to monitoring, on scenario 1 the captured delay and throughput must be similar to the ones requested. However, on scenario 2 it was expected the capture of inconsistent values both on delay and bandwidth, due to its best-effort transmission along with the stressful network.

During testing, both scenarios had the expected results, proving the success of the QoS implementation in the Arrowhead Framework. Figures 66 to 69 show the monitoring statistic collected during the message streams transmission, making possible to observe the consistency of both QoS parameters on scenario 1 and the QoS values compliance. Regarding the values obtained during scenario 2, it is also possible to observe the expected inconsistency and values irregularity on both delay and throughput QoS parameters.

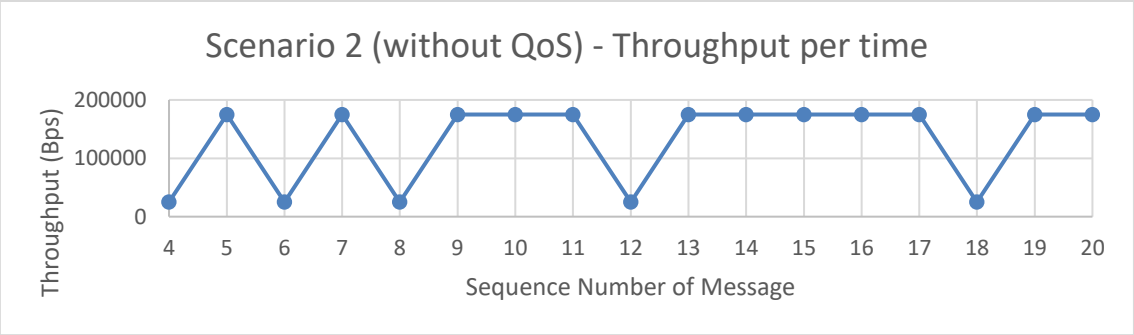


Figure 66 - – Monitoring of the capture of the throughput during scenario 2

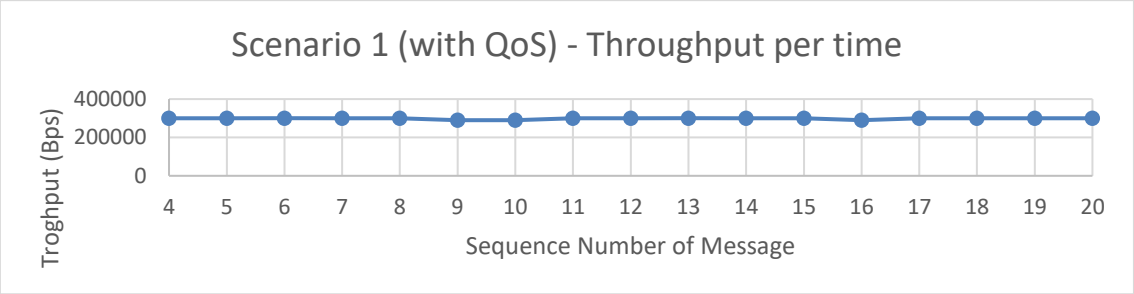


Figure 67 - Monitoring of the capture of the throughput during scenario 1

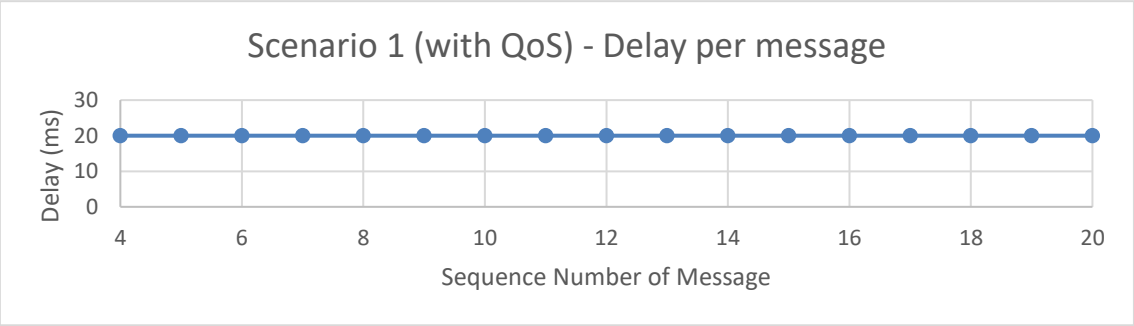


Figure 68 - Monitoring of the capture of the delay during scenario 1.

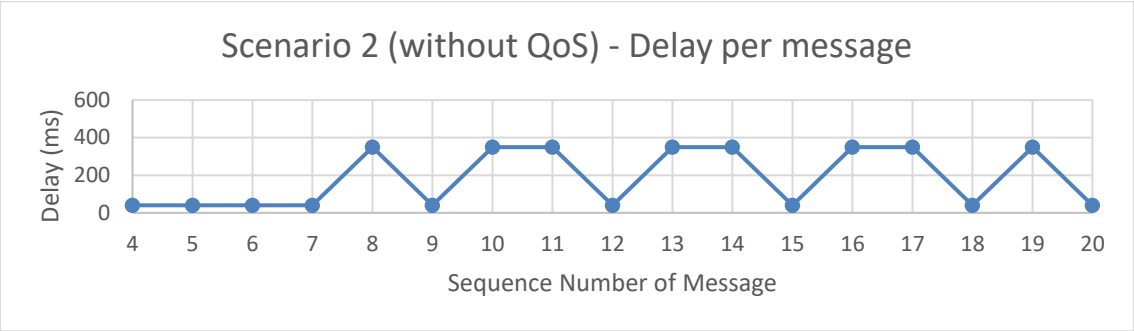


Figure 69 - Monitoring of the capture of the delay during scenario 2.

Chapter 5. Tests Description

5 Tests Description

This section provides a description of the software tests performed, first introducing the theoretical concepts in section 5.1 . Finally, in section 5.2, some of the developed unit tests are explained.

5.1 Introduction

The realization of software tests is fundamental to guarantee that an application behaves as expected and accomplished successfully all its requirements. This process should occur during a project development to eliminate any construction errors, that otherwise could in result in avoidable problems.

In fact, there are two different approaches of development tests, “Black-Box” and “White-Box” testing. “Black-Box” testing is done without knowing the internal parts of the software, the tester knows what the program should do but does not know how it works. This testing approach applies in different levels of software testing such as integration testing, system testing or acceptance testing. Whereas the White-Box testing focus on the internal parts of the software, forcing the tester to know the structure of the program. Most commonly, the White-Box testers have programming skills and had already studied the implementation code. This testing approach applies in three different level of software testing: integration, unit and system [76].

Since this project had two developed systems, several “White-Box” unit tests were implemented, using the framework JUnit (version 4.12). These tests focused in the core functionalities of both systems, which were of extreme importance to guarantee the accomplishment of each system requirements.

It is important to note that as section G describes, the team, after the development of the pilot project, performed a “Black-Box” acceptance test. This test consisted in two different scenarios, one with configuration of QoS and the other with best-effort transmission. Each scenario was carried out in the same environment conditions, including the network traffic, and demanded a different behaviour of the developed systems. At the end, all systems behaved as expected.

Therefore, this Chapter only describes some of the “White-Box” units tests performed during development, while the “Black-Box” tests are described in Section 4.7.

5.2 Unit Tests

A unit test is code that exercises a specific portion of a codebase in a particular context. Typically, each unit test sends a specific input to a method and verifies that the method returns the expected value, or takes the expected action. Unit tests prove that the code does in fact do what you expect it to do [77].

During development, several unit tests were implemented to both QoSManager and QoSMonitor systems. In order to have an organized testing structure the team adopted the AAA

[78] testing pattern. AAA, or Arrange, Act and Assert pattern consists in these three stages, one preceding the other. The first stage is the Arrange, which is where the tester makes the necessary set up prior to invoking the method of interest. It can consist in simple instantiations of objects or it can even consist in more complex set ups, depending on the application. The second stage is the Act, where the tester invokes the method of interest. Finally, in the Assert stage, the tester ensures that his expectations were met, by comparing the obtained result with the expected one.

Regarding the QoSManager (Section 4.2.1) the classes that demanded more testing were the ones responsible of verifying certain QoS requirements, named as QoSAlgorithm, and the ones responsible of configuring a network depending on certain QoS requirements, named as QoSDriver. These classes are specific to a communication protocol, and since in this project only one communication protocol was worked (FTT-SE) there were only two classes available. Moreover, only the FTT-SE QoSDriver was implemented at the expense of the QoSAlgorithm. Therefore most of the unit tests focused in the FTT-SE QoSDriver.

Table 73 describes a unit test performed for the `generateCommand()` method of the class `FTTSE` in a success scenario perspective. As well, Figure 70 shows how the same unit test was implemented. This method receives a list of QoS requirements along with the network information, including a stream identifier, the elementary cycle of the network and the maximum transmission unit of the switch. After processing this input, it is expected that the method returns a list of commands containing the configuration parameters of the stream to be configured.

Table 73 - Test case 1.

Purpose	Evaluate function return.
Setup	None.
Test data	Three integer objects corresponding to the network configuration: a contour of the streams ID, the elementary cycle value and the maximum transmission unit. One map containing the following QoS requirements: “bandwidth” with value of 1500 and “delay” with a value of 20.
Expected result	The method must return a map containing the following instructions: “period” with value of 1, “size” with value of 7500, “id” with the value of 1, and the “synchronous” value as 0.
Steps	Create a map object, containing certain QoS requirements. Create three integer objects to use as parameters as invoking. Invoke the
Actual result	The method should return a map containing the following instructions: “period” with value of 1, “size” with value of 7500, “id” with the value of 1, and the “synchronous” value as 0.
Remarks	The method returned the expected configuration commands and it is considered successful.

```
/**
         * Test of generateCommands method, of class FTTSE. With
QoS, bandwitdh
         * 1500B/s, delay 20 ms.
         */
        @Test
        public void testGenerateCommands1() {
                //ARRANGE
                Integer streamID = 0;
                Integer elementaryCycle = 20;
                Integer mtu = 1500;
                Map<String, String> requestedQoS = new HashMap<>();
                requestedQoS.put("bandwidht", "1500");
                requestedQoS.put("delay", "20");
                FTTSE instance = new FTTSE();
                Map<String, String> expResult = new HashMap<>();
                expResult.put("PERIOD", "1");
                expResult.put("SIZE", "7500");
                expResult.put("ID", "1");
                expResult.put("SYNCHRONOUS", "0");

                //ACT
                Map<String, String> result = instance.
                        generateCommands(streamID, elementaryCycle,
mtu, requestedQoS);

                //ASSERT
                assertEquals(expResult, result);
        }
```

Figure 70 - Implementation of Test Case 1.

As for the QoSMonitor system, the same scenario of the QoSManager applies given that the logic is conditioned by the communication protocol being used. Therefore, only one class was in need of testing, the `FTTSE` class. A generic interface, `IProtocol`, defines the behaviour that each communication protocol representative class must implement, thus supporting the use cases described in section 4.2.2.B. Five methods are defined in the `IProtocol` interface, but only four were tested since the remaining one plays a part in the showing of graphics and is not important for this matter. They are:

- `filterRuleMessage(AddMonitoringRule message)`: takes an `AddMonitoringRule` as a parameter and creates a `MonitorRule` with the needed information, specific to the communication protocol.
- `filterLogMessage(AddMonitoringLog message)`: takes an `AddMonitoringLog` as a parameter and creates a `MonitorLog` with the needed information, specific to the communication protocol.
- `createEvent(SendEvent message)`: takes a `SendEvent` as a parameter and creates an EventHandler compliant event with the needed information.
- `verifyQoS(MonitorRule rule, MonitorLog… logs)`: takes a `MonitorRule` and at least one `MonitorLog` as parameters. As the name says, it compares logs information against that defined by the rule The number (N) of

`MonitorLog` parameters defines if soft real time (N > 1) or real time (N = 1) monitoring is to be executed. When executing a real time Quality-of-Service verification, an error of 10% is added to the bandwidth and 15 ms to the delay, to compensate for the processing time of the FTTSE protocol.

With that said, in this report only the `verifyQoS()` tests will be shown since it's the most important of the four.

Table 74 describes a unit test created for the `verifyQoS()` method of the `FTTSE` class in a success scenario perspective. In the `FTTSE` specification, only bandwidth and delay are monitored. After processing the inputs, it is expected that the return represents a violation in the requested Quality-of-Service, defined in the `MonitorRule`.

Table 74 - Test Case 1 of the QoSMonitor system.

Purpose	Evaluate function return.
Setup	MonitorRule and MonitorLog instantiations.
Test data	A MonitorRule and a MonitorLog with default information. The MonitorRule has three parameters, a stream_id with a value of 1, a requested bandwidth value of 200 Mbps and a requested delay value of 40 ms. The MonitorLog has two parameters, a logged bandwidth value of 120 and a logged delay value of 60.
Expected result	The method returns an instance of SLAVerificationResponse that is a representation of the output of the Quality-of-Service verification process. It must have information about the unmet requested value in the MonitorLog, in this case is the delay which exceeds the value specified in the MonitorRule.
Steps	Create a MonitorRule with default information and the following parameters: stream_id value of 1, requested bandwidth value of 200 Mbps, requested delay value of 40 ms. Create a MonitorLog with default information and the following parameters: logged bandwidth value of 120, logged delay value of 60. Create a SLAVerificationResponse with information about the unmet requested values and the respective logged values. In this case requested delay value of 40 ms and logged value of 60 ms.
Actual result	The method should return a SLAVerificationResponse with information of the unmet requested values. In this case a requested delay value of 40 and a logged value of 60.
Remarks	The method returned the expected SLAVerificationResponse and it is considered successful.

```
/**
 * Test of verifyQoS method in real time, not meeting QoS
 * requirements.
 */
@Test
public void testVerifyQoSNotMetRealTime() {
        System.out.println("verifyQoSNotMetRealTime");

        //ARRANGE
        /* Creates a MonitorRule with default information, a
        stream_id value of 1, requested bandwidth and delay values
        of 200 Mbps and 40 ms respectively */
        MonitorRule rule = createMonitorRule("1", "200", "40");

        /* Creates a MonitorLog with default information,
        logged bandwidth and delay values of 120 Mbps and
        60 ms respecively. */
        MonitorLog log = createMonitorLog("120", "60");

        //Output of a QoS verification process.
        SLAVerificationResponse expResult
                = new SLAVerificationResponse();
        //Information about an unmet requested value.
        expResult.addParameter(
                new SLAVerificationParameter("delay", 40.0, 60.0));

        //ACT
        //Variable instance represents a FTTSE instance.
        SLAVerificationResponse result
                = instance.verifyQoS(rule, log);

        //ASSERT
        assertEquals(expResult, result);
}
```

Figure 71 - Test Case 1 of the QoSMonitor system.

Table 75 describes a unit test created for the `verifyQoS()` method of the `FTTSE` class in a success scenario perspective. In the FTTSE specification only bandwidth and delay are monitored After processing the inputs, it is expected that the return represents a violation in the requested Quality-of-Service, defined in the `MonitorRule`. In this case, soft real time is enabled, and 20 `MonitorLog` instances are used.

Table 75 - Test Case 2 of the QoSMonitor system.

Purpose	Evaluate function return.
Setup	MonitorRule and 20 MonitorLog instantiations.
Test data	A MonitorRule and 20 MonitorLog with default information. The MonitorRule has four parameters, a stream_id with a value of 1, a requested bandwidth value of 200 Mbps, a requested delay value of 40 ms and the number of last logs (NLogs) to use in the monitoring process set as 20. Also has a Boolean value for soft real time set as true.

	The 20 MonitorLog instances have two parameters each, a logged bandwidth value and a logged delay value and each one is represented next:
	MonitorLog 1: bandwidth value: 112, delay value:58 MonitorLog 2: bandwidth value: 162, delay value:75 MonitorLog 3: bandwidth value: 158, delay value:65 MonitorLog 4: bandwidth value: 164, delay value:45 MonitorLog 5: bandwidth value: 100, delay value:20 MonitorLog 6: bandwidth value: 780, delay value:100 MonitorLog 7: bandwidth value: 12, delay value:90 MonitorLog 8: bandwidth value: 1200, delay value:10 MonitorLog 9: bandwidth value: 192, delay value:48 MonitorLog 10: bandwidth value: 241, delay value:40 MonitorLog 11: bandwidth value: 243, delay value:39 MonitorLog 12: bandwidth value: 351, delay value:20 MonitorLog 13: bandwidth value: 121, delay value:17 MonitorLog 14: bandwidth value: 801, delay value:45 MonitorLog 15: bandwidth value: 709, delay value:38 MonitorLog 16: bandwidth value: 125, delay value:85 MonitorLog 17: bandwidth value: 251, delay value:21 MonitorLog 18: bandwidth value: 199, delay value:37 MonitorLog 19: bandwidth value: 177, delay value:49 MonitorLog 20: bandwidth value: 120, delay value:60
Expected result	The method returns an instance of SLAVerificationResponse that is a representation of the output of the Quality-of-Service verification process. It must have information about the unmet requested value in the MonitorLog, in this case both the bandwidth and the delay exceed the respective values specified in the MonitorRule.
Steps	Create a MonitorRule with default information and the following parameters: stream_id value of 1, requested bandwidth value of 200 Mbps, requested delay value of 40 ms. Create 20 MonitorLog instances with default information and the values described in the test data section of this table. Create a SLAVerificationResponse with information about the unmet requested values and the respective calculated values of the last 20 logs. In this case requested bandwidth value of 50 Mpbs and calculated of 310.9 of the last 20 logs and requested delay value of 40 ms and calculated value of the last 20 logs of 47.85 ms.
Actual result	The method should return a SLAVerificationResponse with information of the unmet requested values. In this case a requested bandwidth value of 250 Mbps and a calculated value of 310.9 of the last 20 logs and a requested delay value of 40 and calculated value of 47.85 of the last 20 logs.
Remarks	The method returned the expected SLAVerificationResponse and it is considered successful.

```
/**
 * Test of verifyQoS method in soft real time, not meeting QoS
 * requirements.
 */
@Test
public void testVerifyQoSNotMetSoftRealTime() {
        System.out.println("verifyQoSNotMetSoftRealTime");

        //ARRANGE
        /* Creates a MonitorRule with default information, a
        stream_id value of 1, requested bandwidth and delay values
        of 200 Mbps and 40 ms respectively */
        MonitorRule rule = createMonitorRule("1", "250", "40");
        //Enables soft real time monitoring
        rule.setSoftRealTime(true);
        rule.getParameters().put("NLogs", "20");

        /* Creates 20 MonitorLog with default information,
        bandwidth and delay values */
        MonitorLog[] logs = create20Logs();

        //Output of a QoS verification process.
        SLAVerificationResponse expResult
                = new SLAVerificationResponse();
        //Information about unmet requested values.
        expResult.addParameter(
                new SLAVerificationParameter("bandwidth", 250.0,
310.9));
        expResult.addParameter(
                new SLAVerificationParameter("delay", 40.0, 60.0));


        //ACT
        //Variable instance represents a FTTSE instance.
        SLAVerificationResponse result
                = instance.verifyQoS(rule, logs);

        //ASSERT
        assertEquals(expResult, result);
}
```

Figure 72 - Test Case 2 of the QoSMonitor system.

Chapter 6. Conclusion

6 Conclusion

This chapter is divided in three sections. Section 6.1 summarizes the project problem and the proposed solution. After, Section 6.2 describes the planned objectives, the ones that were accomplished and the others that were not. The project limitations and its future work are described in Section 6.3. Finally, Section 6.4 we present a final assessment on the developed solution and the research centre that made this project possible.

6.1 Summary

The developed project consisted of the design and implementation of an architecture that would allow QoS support in the Arrowhead-framework. The Arrowhead-framework was developed under the Arrowhead Project which focused on allowing collaborative automation by devices embedded in the network. Furthermore, to test the architecture and prove its validity, a pilot project was also developed which applied the concept to real-time video application running over a FTT-SE network.

The implemented solution consisted in two independent systems, the QoSManager and QoSMonitor. The QoSManager is responsible of configuring QoS parameters in networks, and the QoSMonitor is responsible of monitoring all the QoS requested requirements. These two systems interact with each other and also with the other Arrowhead systems such as the Orchestrator, the Service Registry and EventHandler.

Regarding the pilot project, we developed an application on top of the FTT-SE code, with the purpose of integrating it with the Arrowhead Framework. During the integration, we designed an architecture, which proposed the addition of an EntryPoint device that could retransmit the messages coming from the Arrowhead and FTT-SE network. FTT-SE is a real-time protocol and as such must process the input and produce an outcome within a specified time, else it will fail In FTT-SE protocol A hierarchy is established by dividing all network nodes in two groups, the master and salves. The master controls the traffic in the network among slave nodes, deciding when and which slave has the permission to send data.

In all, the developed project allowed the consume of services with two QoS requirements: delay and bandwidth. After the integration, a proof of concept was also implemented, with the recording of a video demo. To prove the success of the work, the team made a test recurring to a video transmission between a consumer and a producer node on a FTT-SE network. This test occurred in two scenarios, one with QoS support and other without it (best-effort transmission). On both scenarios the network had a considerable usage of bandwidth by other application, thus allowing to prove that the required QoS levels were being accomplished.

As expected, the video transmitted in the best-effort scenario was inconsistent. However, on the other scenario, the video transmitted with the requested QoS requirements had the required quality, proving the success of the project.

6.2 Accomplished Objectives

As Chapter 1 explained, the primary objective of this project was to design an architecture that could support QoS in the Arrowhead Framework. This architecture had to be abstract enough to be capable of working with different network technologies and QoS requirements. Since this architecture was designed, developed and tested with the support of the pilot project it is fair enough to say that this objective was successfully accomplished.

Regarding the developed architecture, both QoS configuring and monitoring goals were achieved with the development of the QoSManager and QoSMonitor systems. Only one functionality of the QoSManager, checking if certain QoS requirements were feasible in FTT-SE, was not implemented due to insufficiency of time and high mathematical complexity.

The extensibility objectives regarding to the communication protocols and SLA parameters were also accomplished in both QoSManager and QoSMonitor systems recurring to software design patterns. These two developed systems are capable of working with multiple communication protocols without the need of recompiling their source codes. As long, there is a QoSDriver and a QoSAlgorithm class for each communication protocol, both systems can operate in various protocols. Regarding the SLA parameters, the same objective was accomplished since both systems are capable of working with unlimited number of parameters. It is important to note that this depends on the capability of the monitoring plugins. If they are capable of monitoring only delay, obviously both QoSManager and QoSMonitor systems can solely operate with this parameter.

Furthermore, the developed monitoring graphical interface allowed the user to visualize the monitored parameters, like delay or bandwidth, and any given critical event, such as a loss of a packet. Therefore, all the monitored data is displayed in graphics using the JavaFX [79] platform, allowing a user-friendly and easily comprehensible graphical interface. The visualisation of the monitored data can only be done in the node where the QoSMonitor system is deployed.

Another planned objective consisted in the implementation of a pilot project. The team successfully implemented the pilot project, proving the well functioning of all the developed systems. Considering the limitations of the FTT-SE network, the team studied different architecture solutions and come to a final one, which imposed the addition of one network interface per node and the addition of a computer node, named as EntryPoint. The EntryPoint had the purpose of converting socket messages to HTTP messages and vice-versa.

After the implementation of the proposed architecture, the QoS support of the Arrowhead could finally be tested in the FTT-SE network. The team decided to transmit a video from a producer to a consumer and during that transmission the video would be reproduced in the consumer node, to prove the robustness of the stream connecting both producer and consumer. Using MPlayer, as section 4.7 describes, the team tested and recorded the system behaviour. As expected the video reproduction was consistent and fluid with the QoS support, and in the other hand, it was inconsistent without the QoS support.

As a final point, it is fair enough to say that all the objectives were achieved with success, except the QoSAlgorithm of the FTT-SE network.

6.3 Limitations and future work

Despite the developed architecture had been successfully tested, some of its non-functional requirements such as security and performance could not be implemented in a proper way. Furthermore, due to lack of documentation, the integration with the FTT-SE protocol proved to be more complicated than expected. As a result, the pilot-project was unstable, so in the future work it must be corrected. Still on the pilot-project, it is expected as future work to develop a different architecture that avoids the use of two interfaces. Such architecture could be supported using the Tun/Tap technology, enabling the transport of TCP packets in FTT-SE.

Although most of the objectives were accomplished, both systems can be improved in several scenarios. First, the QoSAlgorithm for the FTT-SE network must be implemented, since it is important for the QoSVerify functionality of the QoSManager.

Regarding the QoSMonitor, its graphical interface could, as future work, be deployed as a web service, allowing its visualization via browser, outside the local cloud where the QoSMonitor system is deployed. Also, in order to avoid overload in the service providers, a “load-balancing” like mechanism should be employed by the QoSMonitor system and the respective plugins deployed in the provider systems (e.g. whenever a provider can’t relay all the monitoring information, in MonitorLog form, to the QoSMonitor system, the frequency of transmission of MonitorLog can be halved).

In order to increase the consistency and integrity of the developed systems, as future work, other pilot-projects must be made, specially in other communication protocols than the FTT-SE, such as ZigBee [14].

Another asset for the foreseeable future work would be the implementation of this architecture for other Arrowhead Frameworks. Since some of the partners use different implementations of the Arrowhead, it would be beneficial for all, the addition of the QoS support.

The developed work is being prepared to be included in a paper, which is under development. Since all primary objectives were achieved and the developed work has potential to be extended in the core functionalities (as referred throughout this report, more protocols) it can be a starting point, given that not only the Quality-of-Service core systems were implemented but also a pilot project was made to show all the framework’s practical uses.

6.4 Final Appreciation

The inherent problems in this project enabled a positive experience, especially enriching ones. The solution implementation required a large versatility by all developers, since various technologies were studied and used.

The solution development focused primarily on good programming techniques and in the usage of robust technologies in order to fulfil some of the project non-functional requirements. Although the solution complies with its main objectives, it still requires greater robustness, more specifically in the FTT-SE application.

Regarding the organization where the internship took place, CISTER facilities are of high quality where there was no lack of resources necessary to carry out the work. The working environment had been always been pleasant, and since the beginning, all CISTER colleagues made very easy the integration of every team member in the research centre. As CISTER focuses on various areas of work, the centre does not have in its work philosophy a fixed software development methodology. As a result, the team and the supervisors had almost daily contact, to guarantee that all were aware of the project status and problems.

At last, assessing the outcome and considering the complexity and the time of work, the team was pleased with the results achieved.

7 Bibliography

[1] Arrowhead, Arrowhead Book, 2013.

[2] C. R. Center, “Cister Info,” Cister Research Center, 5 June 2016. [Online]. Available: http://cister.isep.ipp.pt/info.

[3] “Cister Research Topics,” 5 June 2016. [Online]. Available: http://cister.isep.ipp.pt/research/.

[4] G. Branca, “Architectures and Technologies for Quality of Service,” Italy, 2012.

[5] L. L. Ferreira and M. Albano, “QoS-as-a-Service in the Local Cloud,” 2016.

[6] PANDUIT, “Modernizing the Industrial Ethernet Network,” PANDUIT, July 2016. [Online]. Available: http://www.panduit.com/ccurl/368/276/modernizing-the-industrial-ethernet-network-with-increased-visibility,0.pdf. [Accessed 3 October 2016].

[7] “Iron Paper,” Iron Paper, 5 March 2015. [Online]. Available: http://www.ironpaper.com/webintel/articles/internet-things-market-statistics-2015. [Accessed 3 June 2016].

[8] “Intel,” Intel, [Online]. Available: http://www.intel.com/content/www/us/en/internet-of-things/industry-solutions.html. [Accessed 4 June 2016].

[9] SAP, “IoT for the Automotive Industry,” SAP, 2014. [Online]. Available: https://www.sap.com/bin/sapcom/en_us/https://www.sap.com/bin/sapcom/en_us/downloadasset.2014-10-oct-31-20.ceo-perspective-the-interrnet-of-things-for-the-automo.2014-10-oct-31-20.ceo-perspective-the-interrnet-of-things-for-the-automotive-industry-pdf.html. [Accessed 2 October 2015].

[10] Gartner, “Gartner Technology Consultant - Official Site,” Gartner, [Online]. Available: http://www.gartner.com/technology/home.jsp. [Accessed 2 October 2016].

[11] N. Vej, “IoT - From Research and Innovation to Market Deployment,” River Publishers, Denmakr, 2014.

[12] Telensa, “IoT in smart cities project,” Telensa, [Online]. Available: http://www.telensa.com/. [Accessed 2 October 2016].

[13] “BusinessWire,” BussineWire, 7 June 2016. [Online]. Available: http:www.businesswire.com/news/home/20160607005875/en/Global-Internet-IoT-Healthcare-Market-Grow-37.6. [Accessed 8 June 2016].

[14] IGI, “Zigbee Wireless Description,” [Online]. Available: https://www.digi.com/resources/standards-and-technologies/rfmodems/zigbee-wireless-standard. [Accessed 25 September 2016].

[15] Raymond Hames, “The Internet of Things,” U.S. Research, 2014.

[16] YOKOGAWA, “DCS Description,” YOKOGAWA, [Online]. Available: http://www.yokogawa.com/solutions/products-platforms/control-system/distributed-control-systems-dcs/. [Accessed 26 September 2016].

[17] IBM, “SCADA Description,” IBM, [Online]. Available: http://www-935.ibm.com/services/us/iss/pdf/scada_whitepaper.pdf. [Accessed 29 September 2016].

[18] ISA, “ISA-95 Description,” [Online]. Available: https://isa-95.com/. [Accessed 26 September 2016].

[19] ISA, “ISA-95 Description,” ISA, [Online]. Available: https://www.isa.org/isa95/.

[20] R. Marau, “Real TIme-Communications over siwtched Ethernet supporting dynamic QoS Managment.,” 2009.

[21] Toyota, “Anti-lock Brake System,” Toyota, [Online]. Available: http://www.toyota-global.com/innovation/safety_technology/safety_technology/technology_file/active/. [Accessed 16 Sepetmber 2016].

[22] Texas Instruments, “An Inside look at industrial Ethernet communication protocols,” 2013.

[23] M. H. Ashjaei, “Extending FTT-SE Protocol for Multi-Master Networks,” Vâsterãs, Sweden, 2012.

[24] C. Gomes, “FTT-SE: Desenvolvimento de um dissector para um protocolo de tempo real.,” Universidade Aveiro, Aveiro, 2010.

[25] NAGIOS, “Nagios Home Page,” [Online]. Available: https://www.nagios.org/. [Accessed 29 September 2016].

[26] CACTI, “CACTI Home Page,” [Online]. Available: http://www.cacti.net/. [Accessed 29 September 2016].

[27] J. Delsing, P. Varga, J. Eliasson, F. Blomqvist, P. Olofsson, H. Derhamy, O. Carlsson, P. P. Pereira, T. S. Cinotti, A. Skou, L. Ferreira and M. D. S. Sanchez, IoT based Automation - made possible using Arrowhead Framework, 2013.

[28] M. Albano, L. L. Ferrreira and J. Sousa, “Extending publish/subscribe mechanisms to SOA applications,” 2016.

[29] L. L. Ferreira, “The Arrowhead Aproach for SOA Application Development and Documentation,” Arrowhead, EU, 2014.

[30] IBM, “Rational Unified Process - Best Practices for Software Development Teams,” IBM, 2011.

[31] Bitbucket , “Bitbucket,” Bitbucket, [Online]. Available: https://bitbucket.org/. [Accessed 16 September 2016].

[32] Git, “Git Official Page,” Git, [Online]. Available: https://git-scm.com/. [Accessed 16 September 2016].

[33] UML, “UML Description,” UML, [Online]. Available: http://www.uml.org/. [Accessed 27 September 2016].

[34] M. Fowler, “Unit Tests Description,” Martin Fowler, 5 May 2014. [Online]. Available: http://martinfowler.com/bliki/UnitTest.html. [Accessed 16 September 2016].

[35] Agile Alliance, “Acceptence Testing,” Agile Alliance, [Online]. Available: https://www.agilealliance.org/glossary/acceptance/. [Accessed 13 October 2016].

[36] Microsoft, “Power Point Tool,” Microsoft, [Online]. Available: https://office.live.com/start/PowerPoint.aspx. [Accessed 16 September 2016].

[37] Oracle., “Java Description.,” 5 June 2016. [Online]. Available: https://java.com/en/download/faq/whatis_java.xml.

[38] H. Schildt, Java The Complete Reference Ninth Edition Book, Oracle Press., 2014.

[39] N. Parlante, Essential C book, Standford CS Education Library, 2003.

[40] Jersey, “Jersey Java Net,” Jersey , [Online]. Available: https://jersey.java.net/. [Accessed 8 June 2016].

[41] Oracle, “JAX-RS API,” Oracle, [Online]. Available: https://jax-rs-spec.java.net/nonav/2.0/apidocs/. [Accessed 16 September 2016].

[42] Vogella, “Jersey Tutorial,” Vogella, 15 December 2015. [Online]. Available: http://www.vogella.com/tutorials/REST/article.html. [Accessed 16 September 2016].

[43] N.-3. Press., “NS-3 Documentation.,” 5 June 2016. [Online]. Available: https://www.nsnam.org/docs/tutorial/html/introduction.html#about-ns3.

[44] Wikipedia, “Discrete Event Simulation,” 6 June 2016. [Online]. Available: https://en.wikipedia.org/wiki/Discrete_event_simulation.

[45] MongoDB, “MongoDB,” MongoDB, [Online]. Available: https://www.mongodb.com/compare/mongodb-mysql. [Accessed 8 June 2016].

[46] P. Oracle, “Oracle MYSQl,” 7 June 2016. [Online]. Available: http://www.oracle.com/us/products/mysql/overview/index.html.

[47] TutorialPoints, “RDBMS Description,” TutorialPoints, [Online]. Available: https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm. [Accessed 29 September 2016].

[48] “MySQL,” 7 June 2016. [Online]. Available: http://www.tiobe.com/tiobe_index.

[49] O. Press, “Netbeans,” Netbeans, [Online]. Available: https://netbeans.org/features/. [Accessed 8 June 2016].

[50] C. A. Where, “Most Popular Desktop IDEs & Code Editors in 2014,” Code Any Where, [Online]. Available: https://blog.codeanywhere.com/most-popular-ides-code-editors/. [Accessed 8 June 2016].

[51] Eclipse, “Eclipse Description,” 7 June 2016. [Online]. Available: https://eclipse.org/org/.

[52] G. Website, “Git Description,” 7 June 2016. [Online]. Available: https://git-scm.com/.

[53] B. Website, “Bitkeeper,” 7 June 2016. [Online]. Available: http://www.bitkeeper.com/.

[54] G. Website, “Why Use Version Control,” 7 June 2016. [Online]. Available: https://www.git-tower.com/learn/git/ebook/en/command-line/basics/why-use-version-control.

[55] O. Website, “Version Control,” 7 June 2016. [Online]. Available: http://oss-watch.ac.uk/resources/versioncontrol.

[56] Csaba Miklós Hegedűs, “Arrowhead Hungarian Framewoek,” 2015.

[57] Pearson, “AIC,” Pearson, [Online]. Available: http://www.pearsonitcertification.com/articles/article.aspx?p=1708668. [Accessed 13 October 2016].

[58] Clean Coder, “SOLID description,” 2009. [Online]. Available: https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start. [Accessed 24 September 2016].

[59] Lokesh Gupta, “Singleton Description,” HowToDoInJava, 22 October 2012. [Online]. Available: http://howtodoinjava.com/design-patterns/creational/singleton-design-pattern-in-java/. [Accessed 2016 September 2016].

[60] Microsoft, “DTO Description,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ff649585.aspx. [Accessed 26 September 2016].

[61] Program Creek, “Reflection Pattern description,” Program Creek, [Online]. Available: http://www.programcreek.com/2013/09/java-reflection-tutorial/. [Accessed 9 September 2016].

[62] Program Creek, “Factory Pattern description.,” Program Creek, [Online]. Available: http://www.programcreek.com/2013/02/java-design-pattern-factory/. [Accessed 9 September 2016].

[63] Microsoft, “Repository Pattern description.,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/ff649690.aspx. [Accessed 9 September 2016].

[64] Oracle, “Netbeans Description,” Oracle, [Online]. Available: https://netbeans.org/. [Accessed 11 September 2016].

[65] “Event Handler System,” [Online]. Available: url. [Accessed 15 9 2016].

[66] MongoDB, “BSON Description,” MongoDB, [Online]. Available: https://www.mongodb.com/json-and-bson. [Accessed 26 September 2016].

[67] U. d. Porto, “FTT,” Universidade do Porto, [Online]. Available: https://paginas.fe.up.pt/~ftt/sections/Flavours/index.html#ftt_se. [Accessed 7 September 2016].

[68] Kernel, “TunTap Description,” Kernel, [Online]. Available: https://www.kernel.org/doc/Documentation/networking/tuntap.txt. [Accessed 7 September 2016].

[69] Debian, “Debian - The Universal Operating System,” Debian, [Online]. Available: https://www.debian.org/index.pt.html. [Accessed 7 September 2016].

[70] HP, “Support Centre,” HP, [Online]. Available: http://h20564.www2.hp.com/hpsc/swd/public/readIndex?sp4ts.oid=5045601. [Accessed 7 September 2016].

[71] TP-LINK, “TL-SF10080D Switch Descrption,” TP-LINK, [Online]. Available: http://www.tp-link.com/lk/products/details/cat-4763_TL-SF1008D.html. [Accessed 7 September 2016].

[72] Wikipedia, “C89,” Wikipedia, 16 August 2016. [Online]. Available: https://en.wikipedia.org/wiki/ANSI_C. [Accessed 7 September 2016].

[73] Windows, “Windows 10 Home Page,” Windows, [Online]. Available: https://www.microsoft.com/pt-pt/windows/get-windows-10. [Accessed 7 September 2016].

[74] supermartian, “Wait-Free Queue,” GitHub, [Online]. Available: https://github.com/supermartian/lockfree-queue. [Accessed 7 September 2016].

[75] MPlayer, “MPlayer Home Page,” [Online]. Available: http://www.mplayerhq.hu/design7/news.html. [Accessed 7 September 2016].

[76] British Computer Society, “Standard for Software Component Testing,” British Computer Society, 2014.

[77] Developer Salesforce, “Unit Tests description.,” 2014. [Online]. Available: https://developer.salesforce.com/page/How_to_Write_Good_Unit_Tests. [Accessed 24 September 2016].

[78] Microsoft, “AAA testing pattern,” Microsoft, 7 January 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh694602.aspx. [Accessed 26 September 2016].

[79] Oracle, “JavaFX - The Rich Client Platform,” [Online]. Available: http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html. [Accessed 11 October 2016].

[80] F. Oliveira, “Implementação da rede Flexible TIme Trigger para Switched Ethernet no SImulador NS-3,” ISEP, Porto, 2015.

[81] IDC, “IDC Market Intelligence Form,” IDC, [Online]. Available: https://www.idc.com/. [Accessed 15 Junho 2016].

[82] SmartBear, “Functional Tests Description,” SmartBear, [Online]. Available: https://smartbear.com/learn/automated-testing/introduction-to-functional-testing/. [Accessed 16 September 2016].

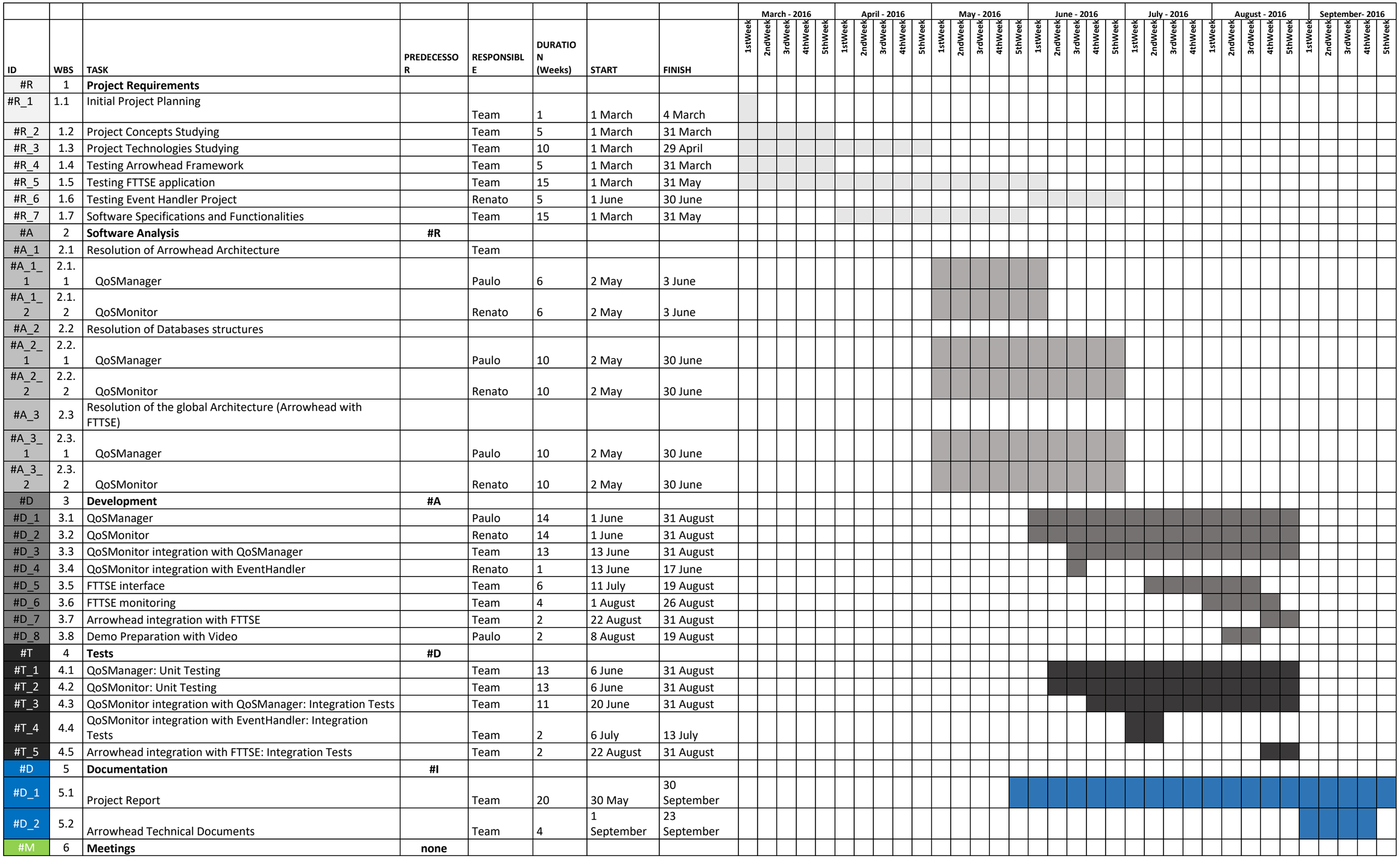
[83] MIcrosoft, “Integration Testing,” Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx. [Accessed 16 September 2016].

[84] Oracle, “DAO Description,” Oracle, [Online]. Available: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html. [Accessed 26 September 2016].

[85] I. MongoDB, “Community | MongoDB,” [Online]. Available: https://www.mongodb.com/community.

Appendixes

ID	WBS	TASK	PREDECESSOR	RESPONSIBLE	DURATION (Weeks)	START	FINISH
#R	1	**Project Requirements**					
#R_1	1.1	Initial Project Planning		Team	1	1 March	4 March
#R_2	1.2	Project Concepts Studying		Team	5	1 March	31 March
#R_3	1.3	Project Technologies Studying		Team	10	1 March	29 April
#R_4	1.4	Testing Arrowhead Framework		Team	5	1 March	31 March
#R_5	1.5	Testing FTTSE application		Team	15	1 March	31 May
#R_6	1.6	Testing Event Handler Project		Renato	5	1 June	30 June
#R_7	1.7	Software Specifications and Functionalities		Team	15	1 March	31 May
#A	2	**Software Analysis**	**#R**				
#A_1	2.1	Resolution of Arrowhead Architecture		Team			
#A_1_1	2.1.1	QoSManager		Paulo	6	2 May	3 June
#A_1_2	2.1.2	QoSMonitor		Renato	6	2 May	3 June
#A_2	2.2	Resolution of Databases structures					
#A_2_1	2.2.1	QoSManager		Paulo	10	2 May	30 June
#A_2_2	2.2.2	QoSMonitor		Renato	10	2 May	30 June
#A_3	2.3	Resolution of the global Architecture (Arrowhead with FTTSE)					
#A_3_1	2.3.1	QoSManager		Paulo	10	2 May	30 June
#A_3_2	2.3.2	QoSMonitor		Renato	10	2 May	30 June
#D	3	**Development**	**#A**				
#D_1	3.1	QoSManager		Paulo	14	1 June	31 August
#D_2	3.2	QoSMonitor		Renato	14	1 June	31 August
#D_3	3.3	QoSMonitor integration with QoSManager		Team	13	13 June	31 August
#D_4	3.4	QoSMonitor integration with EventHandler		Renato	1	13 June	17 June
#D_5	3.5	FTTSE interface		Team	6	11 July	19 August
#D_6	3.6	FTTSE monitoring		Team	4	1 August	26 August
#D_7	3.7	Arrowhead integration with FTTSE		Team	2	22 August	31 August
#D_8	3.8	Demo Preparation with Video		Paulo	2	8 August	19 August
#T	4	**Tests**	**#D**				
#T_1	4.1	QoSManager: Unit Testing		Team	13	6 June	31 August
#T_2	4.2	QoSMonitor: Unit Testing		Team	13	6 June	31 August
#T_3	4.3	QoSMonitor integration with QoSManager: Integration Tests		Team	11	20 June	31 August
#T_4	4.4	QoSMonitor integration with EventHandler: Integration Tests		Team	2	6 July	13 July
#T_5	4.5	Arrowhead integration with FTTSE: Integration Tests		Team	2	22 August	31 August
#D	5	**Documentation**	**#I**				
#D_1	5.1	Project Report		Team	20	30 May	30 September
#D_2	5.2	Arrowhead Technical Documents		Team	4	1 September	23 September
#M	6	**Meetings**	**none**				



#M_1	6.1	Hungary Partners		Team	23	2 May	30 September
#M_2	6.2	CISTER supervisors		Team	34	1 March	30 September
#M_3	6.3	Milestones Demonstrations		Team	13	6 June	31 August
#M_4	6.4	FTTSE partner		Team	25	15 March	19 August
#M_5	5.4	Event Handler partner		Team	9	23 May	11 July

QoSManager.wadl

```
<application xmlns="http://wadl.dev.java.net/2009/02">
<doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey:
2.23.1 2016-06-09 18:05:47"/>
<doc xmlns:jersey="http://jersey.java.net/" jersey:hint="This is simplified
WADL with user and core resources only. To get full WADL with extended
resources use the query parameter detail. Link:
http://localhost:8444/qos/application.wadl?detail=true"/>
<grammars>
<include href="application.wadl/xsd0.xsd">
<doc title="Generated" xml:lang="en"/>
</include>
</grammars>
<resources base="http://localhost:8444/qos/">
<resource path="QoSManager">
<method id="home" name="GET">
<response>
<representation mediaType="text/plain"/>
</response>
</method>
<resource path="/QoSVerify">
<method id="qosVerification" name="PUT">
<request>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns=""
element="qoSVerify" mediaType="application/json"/>
</request>
<response>
<representation mediaType="application/json"/>
</response>
</method>
</resource>
<resource path="/QoSReserve">
<method id="qosReservation" name="PUT">
<request>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns=""
element="qoSReserve" mediaType="application/json"/>
</request>
<response>
<representation mediaType="application/json"/>
</response>
</method>
</resource>
</resource>
</resources>
</application>
```

QoSMonitor.wadl (1/2)

```
<application xmlns="http://wadl.dev.java.net/2009/02">
<doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 2.23.1 2016-06-09 18:05:47"/>
<doc xmlns:jersey="http://jersey.java.net/" jersey:hint="This is simplified WADL with user and core resources only. To get full WADL with extended resources use the query parameter detail. Link: http://localhost:8144/qosmonitor/application.wadl?detail=true"/>
<grammars>
<include href="application.wadl/xsd0.xsd">
<doc title="Generated" xml:lang="en"/>
</include>
</grammars>
<resources base="http://localhost:8144/qosmonitor/">
<resource path="Monitor">
<resource path="/QoSRule">
<method id="addRule" name="POST">
<request>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="addMonitorRule" mediaType="application/json"/>
</request>
<response>
<representation mediaType="application/json"/>
</response>
</method>
<method id="removeRule" name="DELETE">
<request>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="removeMonitorRule" mediaType="application/json"/>
</request>
<response>
<representation mediaType="application/json"/>
</response>
</method>
</resource>
<resource path="/reload">
<method id="startService" name="GET">
<response>
<representation mediaType="application/json"/>
</response>
</method>
</resource>
<resource path="/online">
<method id="getIt" name="GET">
<response>
<representation mediaType="text/plain"/>
</response>
</method>
</resource>
```

QoSMonitor.wadl – (2/2)

```
<resource path="/Event">
<method id="sendEvent" name="POST">
<request>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02"
xmlns="" element="eventMessage" mediaType="application/json"/>
</request>
<response>
<representation mediaType="application/json"/>
</response>
</method>
</resource>
<resource path="/QoSLog">
<method id="addLog" name="POST">
<request>
<ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02"
xmlns="" element="addMonitorLog" mediaType="application/json"/>
</request>
<response>
<representation mediaType="application/json"/>
</response>
</method>
</resource>
</resource>
</resources>
</application>
```