QoS-as-a-Service in the Local Cloud

Luis Lino Ferreira
Michele Albano
Jerker Delsing

CISTER-TR-160612

2016/09/06

QoS-as-a-Service in the Local Cloud

Luis Lino Ferreira, Michele Albano, Jerker Delsing

*CISTER Research Center

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: llf@isep.ipp.pt, mialb@isep.ipp.pt

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

Abstract

This paper presents an architecture that supports Quality of Service (QoS) in an Arrowhead-compliant System of Systems (SoS). The Arrowhead Framework support local cloud functionalities for automation applications, provided by means of a Service Oriented Architecture (SOA), by offering a number of services that ease application development. On such applications the QoS guarantees are required for service fruition, and are themselves requested as services from the framework. To fulfil this objective we start by describing the Arrowhead architecture and the components needed to dynamically in run-time negotiate a system configuration that guarantees the QoS requirements between application services.

QoS-as-a-Service in the Local Cloud

Luis Lino Ferreira, Michele Albano
CISTER, ISEP/INESC-TEC
Polytechnic Institute of Porto
Porto, Portugal
{llf, mialb}@isep.ipp.pt

Jerker Delsing
EISLAB
Lulea University of Technology,
Lulea, Sweden
jerker.delsing@ltu.se

Abstract — **This paper presents an architecture that supports Quality of Service (QoS) in an Arrowhead-compliant System of Systems (SoS). The Arrowhead Framework supports local cloud functionalities for automation applications, provided by means of a Service Oriented Architecture (SOA), by offering a number of services that ease application development. On such applications the QoS guarantees are required for service fruition, and are themselves requested as services from the framework. To fulfil this objective we start by describing the Arrowhead architecture and the components needed to dynamically in run-time negotiate a system configuration that guarantees the QoS requirements between application services.**

Keywords— Quality of Service, Service Oriented Architectures

I. Introduction

The Arrowhead project [1] devoted considerable research and development efforts to normalizing all interactions involving embedded systems by means of a Service Oriented Architecture (SOA). The SOA paradigm is applied to both the interactions that serve the business logic of the applications, the support actions such as the authentication of devices, registration of the devices and the services they provide, the look-up of devices or service providers and to the orchestration of services for creation of more complex services. To this purpose, services are divided into Core Services and User Services. The first set is provided by the Arrowhead Framework and support Arrowhead applications by satisfying the non-functional requirements of the system. Three Core Services are mandatory and comprises the Authentication Service, the Registration Service and the Orchestration Service. User Services are the building blocks that provide the application functionalities on each particular scenario.

The Arrowhead Framework can be applied to multiple fields, like smart cities, industrial automation, smart grids, etc. The application of a SOA architecture to such scenarios allows for higher flexibility, scalability and evolvability [1 – 3].

The architecture of an Arrowhead-enabled system of systems is based on the concept of Local Cloud, which is a bounded set of computational resources used by stakeholders to attain a goal. The approach simplifies design and implementation, since it allows for greater control on access and operations. The local cloud also makes it possible to simplify QoS monitoring and to reduce the QoS managing complexity.

Arrowhead's aims at supporting the ISA-95 automation pyramid [4] at various levels, using IoT technologies and a System of Systems approaches. A particularly important non-functional requirement for SOA applications for automation systems is guaranteeing the Quality of Service (QoS) required by the application services, which varies with the application. Examples of QoS parameters are the reliability of communications (which allows guaranteeing that messages are delivered, and in which order and condition) and real-time constraints (which ensure that messages are delivered within their deadlines).

This work shows how the Arrowhead Framework has been expanded to implement the paradigm of QoS-as-a-Service. QoS is granted as a non-functional requirement for service fruition, and is offered by means of a Core Service itself. In this paper, we define an architecture enabling QoS management and monitoring for service connections. Detailed properties of a system for QoS management (QoSManager system) including service interfaces is proposed together with the anticipated interaction with a local cloud Orchestration system, which provides orchestrated services to service consuming systems. The proposed QoSManager system might be required to adapt to changing operating conditions, to this purpose the system should also be capable of monitoring activities to ensure that the required QoS parameters are being achieved or not, and inform the interested parties (application services) about QoS faults. The application services can then act according to the situation, eventually adapting to the new conditions (e.g. by asking the Orchestration system for a new system configuration).

The rest of the paper is organized as follows. Section 2 discusses background information comprising the Arrowhead Framework basics. Section 3 shows the motivation for the paper and discusses which QoS parameters will be tackled on the context of automation applications. Section 4 discusses the main architectural components that allow supporting service fruition with QoS requirements. Section 5 discusses the interactions between components. Section 6 draws some conclusions.

II. Background Information

A. *QoS for embedded systems*

It's clear that different applications have different QoS requirements in terms of latency, safety or bandwidth, just to name a few parameters. Nowadays, most automation applications are supported on closed systems with limited capabilities to evolve and usually it can only be done at high costs. The trend on applying Industrial IoT (IIoT) technologies and specifically a SOA Architecture to these systems requires changes on the philosophy applied to its development [5].

QoS is often required in many applications, for example on distributed control loops, while other applications are capable of adapting to environments where no QoS guarantees are possible. Nevertheless, it is possible to identify a set of challenges on the development of automation technologies and QoS in particular.

Performance: the technologies being used on IIoT were not developed for resource constrained devices and consequently have to be adapted and simplified to work properly on those devices.

Scalability: the addition of multiple things to the networks poses problems on the overall processing chain: sensor nodes, communication bandwidth and data processing.

Heterogeneous networks: automation systems are expected to span across multiple networks with different technologies and in geographically different areas, consequently this poses a new set of problems for guaranteeing security and QoS.

Security: automation systems are usually composed of many resource constrained devices, which are not be capable of using complex algorithms for implementing security functions.

The Arrowhead Framework, which is described in the rest of the section, tackles most of these challenges and in this paper we focus on how it can support QoS.

B. The Arrowhead Framework

The Arrowhead Framework is devoted to supporting local cloud automation functionalities by offering a number of services that ease application development. In Arrowhead, all interactions are mediated by services, which allow one element of the architecture to request information and actions from other elements of the architecture. Each element providing or consuming a service is called a system. Each physical or virtual platform providing computational resources in a local cloud is called a device.

The services offered in a local cloud comprise discovery of services, loosely coupled data exchange between producer and consumer services, security-related services and orchestration of services. The Arrowhead Framework offers the above mentioned functionalities through the definition of Core Services, among which three services are mandatory and present in each Arrowhead local cloud: ServiceDiscovery (SD), Authentication (AA) and Orchestration (O). The ServiceDiscovery, which is offered by DeviceRegistry, SystemRegistry and ServiceRegistry systems, allows devices, systems and services to be registered in the Arrowhead local cloud. The Authentication is used to authenticate and provide authorization for connections between services. The Orchestration service is used to create the matching between service producers and service consumers, to allow service fruition.

The process to allow service fruition starts with the proper registration of devices, systems, and services. Each device declares which systems it is hosting, and to which other devices it is connected. Each system declares which services it produces and consumes. Each service defines its semantics, by referring to a unique document for the service. A service instance is thus a triple (service, producing system, consuming system).

The Orchestration service is used to match service consumers and producers, taking care of satisfying further criteria such as localization of the systems, QoS, and identity of the interacting devices. For example, a system can request access to systems in its geographical area, with a maximum end-to-end delay, and on a given industrial machine. The Orchestration system, accesses all registries and computes the orchestration of services, which might comprise the computation of a chain of services that together compose a more complex service instance, and the matching between service consumers and producers.

The Orchestration system can respond to the pull or the push paradigm. Whenever a system decides that it needs to consume a service, it "pulls" an orchestrated service from the Orchestration service. Moreover, orchestration is repeated periodically, looking for changes in the system of systems, and the service instances are refreshed in front of the changes. The new set of orchestrated services are then "pushed" by the Orchestration system to the service consumers.

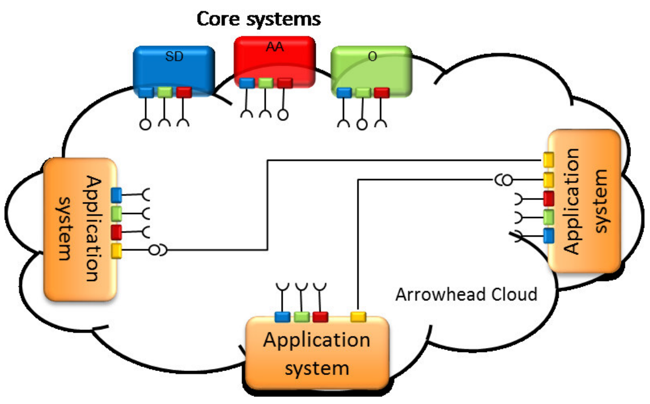

Figure 1 - Example of an Arrowhead application supported by the core services

Thus, the service orchestration responds to the declarative paradigm. The orchestration process is driven by the description of devices, systems and services in the Registry, and the Orchestration system is an engine that gets through the data, distils them into rules, and performs the matching between potential service producers and consumers. As explained in Section 4, the orchestration process is also responsible for the negotiation of QoS, and Section 5 will consider the pros and cons of the declarative paradigm to QoS.

Using this set of systems and their services, it is straightforward to design and implement a minimal local automation cloud. Figure 1 shows an example which only shows the connection between application services (depicted in yellow). The application services are also consumers of the core services, depicts in red, green and blue.

The Arrowhead framework already provides a large set of support Core Services, among which the Historian, Configuration Manager and Event Handler [6] are examples carrying obvious names. Arrowhead also enables the development of systems of systems supported on multiple protocols, like REST, MQTT and COAP.

III. Supporting QoS in Arrowhead

The support of QoS in the local cloud is of paramount importance to support some automation applications. This requires the definition of an architecture that outlines the roles of the involved parties in supporting QoS between a service producer and a consumer. For this purpose, it is possible to foresee the involvement of not only the producer and consumer services, but in some circumstances of network elements that are mediating data transfers in the system (switches, routers, gateways, etc) and the devices that are hosting the services.

The QoS service aims at covering the most common QoS dimensions for the most general automation scenarios. In particular, the QoS service should be able to impose constraints on the following non-functional parameters, and thus provide capabilities to work on the following QoS dimensions:

- End-to-end delay – hard/soft real-time guarantees;
- Data bandwidth;
- Communication semantics – delivery guarantees, and message ordering;
- Message prioritization;
- Local device parameters – on-device task scheduling;
- Service configuration parameters – buffer size, middleware parameters and prioritization of requests.

Two scenarios can be described, to highlight how QoS is a fundamental part of typical automation applications: i) an Ethernet network composed by several nodes interconnected by switches; and ii) a wide area network where parts of the network are not owned nor controlled by the service providers. For the sake of simplicity, let us assume that the objective is to setup the network parameters in order to guarantee communications latencies lower than a given deadline for the interaction between a service producer and a service consumer.

In our approach, a characteristic that is common between all scenarios is that, in order to provide QoS guarantees, a central entity must be able to monitor the QoS of relevance to a service consumer and compare it to the QoS requested. The mitigation of QoS can benefit from the knowledge of the current network topology and types of network actives (switches, routers, gateways, access points, etc) in the system. Additionally, the current status of involved devices. The status of the system comprises (complete or incomplete) information regarding all message streams in the system and the current configuration of the networks. The central entity must be able to collect required information by contacting the adequate registries and can leverage this information to run adequate algorithms to verify if all message streams fulfill their QoS and to determine a new system setup, which is then used to configure the involved devices and systems.

The main structural difference between the two scenarios described in the following involves the degree of control of the systems over the QoS parameters of the devices. In particular, the first scenario can feature an industrial Ethernet that may provide even hard real-time guarantees, while in the case of the wide area network, it is necessary to stick to solutions similar to Diffserv [8], which can only prioritize a message stream.

One of the main requirements for Arrowhead QoS Architecture is to be able to create a request for the needed QoS independently of the underlying network technology, taking for given that the QoS service will take care of feeding the network actives with a proper configuration to support the QoS.

A. *QoS Dimensions*

General QoS objectives can be articulated over many dimensions. This work aims at making the system more robust, both in terms of limiting the resources used by each message stream and in terms of mechanisms to protect communication against failures.

For each family of QoS objectives, we define a class, which is refined into types of QoS parameters. The QoS classes supported in this work are *Delay*, *Bandwidth*, *Resources Limits*, and *Communication Semantics*. Nevertheless, the actual implementation is capable of handling other parameters, just by adding such functionalities to the QoS-related components.

Delay is a very common non-functional requirement in distributed automation systems-of-systems. For the objective to be respected, it implies the execution of actions within a deadline. This class of objectives comprises time elapsed for a message delivery, and end-to-end delay of a service invocation. Moreover, this class of QoS objectives spans over both hard real-time and soft real-time constraints, the latter representing statistical guarantees on the communication delay.

Bandwidth refers to guarantees that sufficient communication and computational resources are allocated to the services, and it is quite common for service support in SOA applications and slightly less common in embedded applications. This class of objectives comprises both constraints on the minimum bandwidth for data produced / transmitted in a time unit, and on the number of service requests supported in a time unit. Usually, the requests bandwidth QoS requirement is applied to service producers only, since network actives do not track the number of requests and thus limit their vision over the data bandwidth being used. On the other hand, the data bandwidth is used on both networking elements, and service providers.

The *Resource Limits* class of QoS objectives is concerned with limiting the quantity of resources used by a service, to protect the system of systems against resource choking. Example of resources are CPU, memory, and any other resource used by an application in the case of nodes, and data bandwidth per time unit in case of both nodes and network actives.

While the previous class protects the system of systems against the services, the *Communication Semantics* protects the services interactions against events disrupting the communication infrastructure and the system of systems in general. This class spans over a set of capabilities that can be requested as part of the QoS. In particular, this QoS class is currently focused on reliability, and it is used to request assurance of receiving the message at least once, the assurance of not receiving duplicated messages, and the reception of messages in the same order they were produced. Moreover, this class comprises also the prioritization of services and message streams.

IV. Arrowhead QoS Architecture

We propose a QoS architecture centered on a QoSManager system that interacts with the Orchestration system to connect service instances respecting the specified QoS requirements. We consider both direct interaction between them through service fruition, and indirect interaction via the Registry, which hosts the result of the computation of the QoSManager system. The monitoring of QoS relies on a QoS Monitor system, which can run on the same device as the QoSManager system, or operate as an external device.

The QoSManager collects information regarding network topology, real-time device capabilities and QoS requirements from the ServiceRegistry, SystemRegistry and DeviceRegistry. The management of QoS is strictly related to the reservation of communication and computational resources, whose information is maintained on a QoS Store, accessible also through a SOA interface and under the control of the QoSManager system. This latter system is able to verify if the requested QoS requirements can be granted or not. Moreover, it can configure the involved active network elements (e.g. router and switches) and devices to grant a QoS request.

The set of elements that can be configured by the QoSManager comprises node's traffic smoothing filters on the output of service producers or consumers, parameters like traffic priority and delivery guarantees of message oriented middleware with QoS capabilities, like DDS [9], RabbitMQ [10] or XMPP [11]. Network actives, like switches, routers or gateways can also be configured in order to control the bandwidth of specific message streams.

The QoSManager might also be capable of configuring the device running the service producer and consumer in order to have response time guarantees for coding/decoding the request and providing a reply. To that purpose, the QoSManager system must be aware of the applications and threads running on the producer device and, if required, it must be able to configure the devices through a specific interface. More complex situations occur when services are composed by set of services running on different devices. Assuming that the application requires a specific response time, then, in both cases response time calculation tools, like holistic analysis [12] have to be applied in order to integrate communications with task scheduling.

Finally, some applications might also require to know the current status of the system and be able to adapt to changing conditions. As an example one of the Arrowhead pilots is capable of reducing its sampling rate and consequently the consumed bandwidth in order to support more devices in an IEEE 802.15.4 network, with very limited bandwidth. This can be achieved by monitoring the network status, using the QoSMonitor system, and informing the interested parties, using the Event Handler.

Even though the literature reports efforts of decentralized control [13], in Arrowhead we focus on a centralized solution, at least in terms of logical components. The main reasons are related to small number of message streams with QoS requirements on the envisaged applications for the Arrowhead framework and the existence of Arrowhead systems (QoS Store Service Registry, System Registry and Device Registry) which are able to capture the status and configuration of a system of systems.

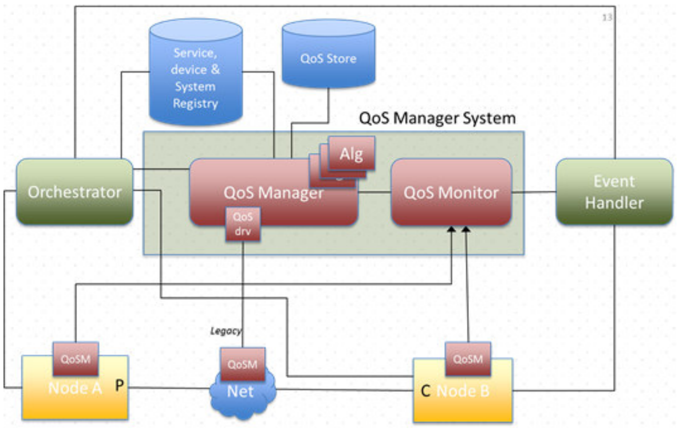

Figure 2 – Relevant components for the QoS support

Figure 2 presents the main building blocks of the Arrowhead QoS architecture, where a service producer and a service consumer are connected through a network active (e.g. a switch). The Orchestration system and the QoSManager system interact with the three registry systems (Service Registry, System Registry and Device Registry) to collect information regarding the system of systems. The QoSManager system accesses the QoS Store holding information regarding resource reservations, a module (Alg module) containing different algorithms for QoS verification and configuration, and a module (QoSDrivers module) with drivers for interaction with custom protocols. This latter module is used to configure network actives, since for the time being we must consider that network actives will not use the SOA approach of Arrowhead, and instead communicate using custom protocols. Finally, there is communication between the QoSManager and the Event Handler system, which is an extension to SOA of publish/subscribe communication and is using to notify systems of QoS faults. The details of this architecture are discussed in detail in the following sections.

A. *QoSManager system*

The main goal of the QoSManager system is to provide QoS-as-a-Service, aligned with Arrowhead Framework objectives, and this is done by means of offering a QoSSetup service. We consider that the QoS requirements are specified in a declarative manner, which can be achieved by adhering to Service Level Agreements (SLAs) mechanisms [14, 15, 16].

The usage of SLAs for setting up QoS parameters was already proposed in [17], specifically for the field of embedded computing, where the focus was on providing a common platform for both critical applications and mainstream embedded applications, the first being characterized by strict timing requirements, and the second by the need for energy saving and low cost. In other scenario, for example related to multimedia fruition, the SLA can specify the amount of data that must be offered by the service providers.

The QoS requirements are uploaded on the Registry at system startup time, or when a service consumer registers itself. The result of the verification process is a configuration of parameters on the devices and network actives. More information regarding the paradigm followed are given in Section V.

B. *QoS drivers*

The QoSDrivers represent the software modules responsible for providing a uniform interface, used by the QoSManager, for the configuration of QoS parameters on network actives and for the monitoring of all the devices. Their duty is to act as adapters between the custom protocol of the network actives and devices, and the protocols used in Arrowhead, and in particular to reach out to the non-Arrowhead compliant world, in particular when a configuration protocol in not natively REST-based.

As an example, these drivers could be used to configure a network switch, only accessible using proprietary protocols, in order to change the priority of the message streams. In some particular cases, it is possible that the QoS parameters of the system have to be setup manually by the system administrator (e.g. using proprietary software tools); in these cases the output of the driver is the configuration to be used on the network.

There are already some tools that allow the remote configuration of some network QoS parameters, such as Nagios [18] and OpenFlow [19]. Anyway, their integration into the Arrowhead environment has to be carefully evaluated since these tools are mostly used on common LAN/WAN network scenarios, and lack support for some wireless networks like IEEE 802.15.4 (ZigBee).

C. *QoS Store*

The QoS Store is a SOA database that holds information regarding the resource reservations active in the local cloud. The data in the QoS Store are kept aligned with the QoS configurations deployed onto network actives and devices. Should the system of systems host more than one QoSManager system, all of them will refer to the same QoS Store to gain a consistent vision of the resource reservations.

D. *Algorithms*

The information contained on the QoS Store is used by the Algorithms module to perform calculations to determine the system parameters which are capable of fulfilling the QoS requirements, taking into account the current status of the systems of systems. These algorithms can be based on mathematical models of the system of systems, which take into account all the data retrieved by the Registry regarding devices, systems and services. Since each service can be an orchestration of other services, particular care has to be taken for both verifying the kind of QoS constraints that can be satisfied by the orchestrated (composed) service (e.g.: should one of the orchestrated services not support hard real-time, all orchestrated services will not support hard real-time) and for the computation of the configuration that can satisfy the QoS levels.

As an example, the work in [20] proposes some algorithms to deal with the composition of real-time services, where it considers the real-time requirements in the context of Ethernet networks, using the Flexible Timer Triggered – Switched Ethernet (FTT-SE) protocol. This protocol can be modeled using a mathematical holistic analysis model proposed in [21], which accounts for the processing time of the nodes involved on a transaction and provides hard real-time guarantees. Similarly, the work in [2] is capable of providing real-time guarantees for beacon-enabled IEEE 802.15.4 networks.

E. *QoS Monitor*

The QoS Monitor main functionality is to monitor if the SLA between producer and consumer is not being violated. Additionally, some dynamic and adaptable QoS algorithms require the knowledge of the connection status during run-time in order to adapt.

The QoSManager system should be able to detect deviations from the performance requested through the QoSManager system. Therefore, the QoS Monitor is responsible for monitoring the behavior of devices and network actives in relation to QoS variables, and informing other systems regarding QoS faults. Violation of QoS requirements and its status is disseminated using the Event Handler system.

F. *Interaction with the Event Handler system*

The Event Handler system [6] provides functionality for the notification of events that occur in a given Arrowhead compliant system. Basically, the Event Handler receives the events from Event Producers and forwards them to subscribing Event Consumers. Two different communication workflows are envisaged.

If the system of systems administrator prefers the Orchestration system to push configurations to other systems, the Orchestration system is the subscriber to messages regarding QoS faults. On reception of the messages, the Orchestration system computes new orchestrated services, and pushes the new configuration to the systems involved in the service instance.

If the system of systems administrator prefers the pull approach to service orchestration, the service consumer is the subscriber of the messages regarding QoS faults. On reception of the message, the service consumer contacts the Orchestration system, and requests to pull a new orchestrated service to consume.

The mediation by the Event Handler system allows for different kinds of decoupling of the QoS Monitor from the message subscribers [23], thus it enables an easier development of QoS functionalities.

V. BEHAVIOR OF THE QOSMANAGER SYSTEM

A. *Interaction paradigm*

This section describes how systems interact with the QoSManager system, to allow the latter to realize its functions. As anticipated in previous sections, two approaches can be envisaged.

A more declarative approach considers that the QoSManager system processes periodically the knowledge base stored on the Registry service. The QoSManager system collects data on the devices, systems and services in the system of systems to compute constraints on the orchestrated services that are compatible with the QoS requirements. The constraints are distilled into rules that are uploaded onto the Registry. When computing orchestrated services, the Orchestration system will take into account the QoS-related rules. The interaction between the QoSManager system and the systems consuming and producing services is totally implicit, and mediated by the 3 registry systems.

In a more imperative paradigm, the Orchestration system, each time that computes orchestrated services, interrogates the QoSSetup service of the QoSManager system regarding the compatibility with the QoS objectives. The QoSManager receives an orchestrated service and QoS objectives from the Orchestration system, collects data on the system of systems from the Registry, and answer to the Orchestration system to allow the latter to push/pull the orchestrated service to the service consumer.

A comparison between the two approaches led to the definition of three issues with the declarative approach.

The first problem is related with race conditions in dynamic systems. When the system of systems is changed, the Orchestration system can compute a new matching before the QoS Monitor wakes up and updates the QoS-related rules. This would leave the system of systems in a state that cannot respect QoS objectives, which can be catastrophic in case of industrial machines and industrial processes.

The second issue regards the computational cost of QoS-related rules when resource reservations are considered. When the Orchestration system considers Registry information as rules, each orchestrated service instance can be considered at a time, since the capability of a service to satisfy functional requirements do not depend on consumption of other services. On the other hand, when the QoSManager considers to reserve resources for the consumption of a service, the fruition of all other services can be impacted. Thus, the computation of feasibility of QoS objectives must be computed against the system of systems. The net results is that the computation of QoS-related rules needs to consider all potential systems of systems, whose number is huge (exponential in the number of systems).

The third issue with the declarative approach is the reservation management. The operations to verify QoS feasibility and to reserve resources cannot be separated, since the QoS rules are put in place before the Orchestration system decides which service instances will be used. Thus, resources will be reserved even when the Orchestration system does not instruct a service consumer to consume the reserved resources.

The discussion at hand proved that the best paradigm for the interaction with the QoSManager is imperative, where the QoSManager behaves as a plug-in of the Orchestration service.

B. *A protocol for the QoSManager system*

Figure 3 depicts the most common interactions involving the verification of QoS objectives for service fruition, and the configuration of a system of systems to respect QoS.

The entry point for the process is the Orchestration service. The service consumer sends the SLA [16] to the Orchestration system, to ask at the same time for the functional and non-functional requirements for the service fruition. The QoSManager system is responsible for the setup of the involved services directly, once instructed by the Orchestration system, which builds up – and returns – one orchestrated service between a Service consumer (SC) and Service producer (SP). Should the returned orchestrated service not respect the QoS, the SC would have to restart the process an undefined number of times.

On the other hand, we assume that the Orchestration system is capable of determining one or a set of ordered configurations that establish this order, whose criteria are beyond the scope of this paper. The Orchestration system sends the description of the orchestrated service to the QoSManager, to allow it to verify the satisfiability of the QoS request and compute a proper configuration for the services and the devices.

The QoSManager system receives the set of possible orchestrated services and verifies, for each of them, if the required QoS level can be supported or not. To do that, the

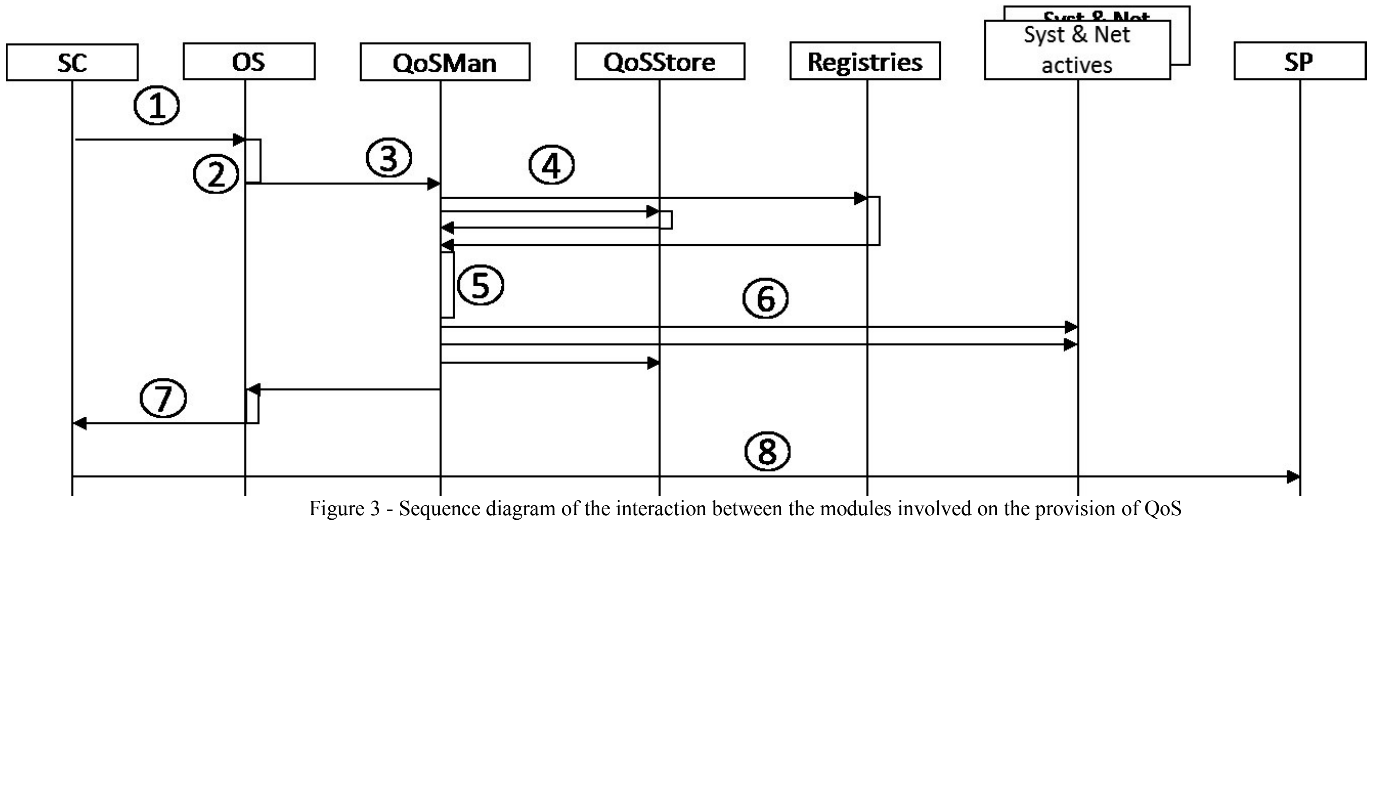

Figure 3 - Sequence diagram of the interaction between the modules involved on the provision of QoS

QoSManager system transforms the SLA into constraints on the non-functional requirements of the services to be orchestrated, and applies the mathematical or statistical models contained in the Algorithms modules. To this aim, the QoSManager system also retrieves information regarding the system of systems from the Registry service, and current status of resource reservation from the QoS Store. When a feasible configuration is found the QoSManager system contacts the involved systems and network actives to configure the QoS for the new service, thus reserving the required resources and modifying the data on the QoS Store to account for the new resource reservation status. Afterwards, the Orchestration system answers to the service consumer with a service configuration which complies with the functional and non-functional requirements of the service request.

The following list resumes the QoS provision sequence, which is depicted in Figure 3:

1) The SC sends request to OS, with functional requirements and a SLA;
2) The OS computes a set of possible service configurations;
3) The OS sends one service configuration at a time to the QoSMan;
4) QoSMan retrieves info from QoSStore and the Registries
5) QoSMan verifies if the QoS requirements contained in the SLA can be guaranteed for a configuration. This step can be repeated several times until a feasible configuration is found or no configuration is possible.
6) QoSMan communicates with systems and network actives to reserve resources to guarantee the required QoS; QoSMan also updates the QoSStore regarding the new reservation;
7) The OS answers to the CS;
8) The CS starts establishes connection with the SP and starts fruition of the service.

VI. QOS-AS-A-SERVICE ON AN FTT-SE NETWORK

The FFT-SE protocol makes use of the master/slave paradigm, where a dedicated node (the Master node) schedules messages on the network. The communications within a FTT-SE network are done based on Elementary Cycles (ECs), which uses fixed duration time slots for synchronous and for asynchronous messages, the remaining EC time can be used for conveying best effort traffic.

The scheduler applies a scheduling policy over these tables, generating the ready queues for transmission for that EC. Synchronous messages are scheduled autonomously by the master, without any petition/feedback from the slave nodes. Asynchronous messages are also scheduled by the master node, but asynchronous messages are activated in response to events that happen in the environment, thus, slave nodes must report its activation to the Master via a signaling mechanism. This process is repeated until no other message fits on the scheduling window for that EC (i.e., considering all messages from higher to lower priority).

For building the EC, it is important to consider: (i) the characteristics of the transmission links; switched Ethernet has fullduplex transmission links, namely the uplink and the downlink, connecting the ports exiting the switch to the nodes, (ii) the multiple switching delays: the switch relaying latency and the Store-and-Forward Delay (which depends on the message size and link speed), and (iii) the length of the specific transmission window for each type of traffic (e.g., synchronous or asynchronous window).

A simplified version of the instantiation of the Arrowhead QoS architecture on an FTT-SE network is depicted in Figure 4. In this figure the thick line represent physical connections between devices, while the thin lines represent logical connections. The circular object represent the switch that connects all devices.

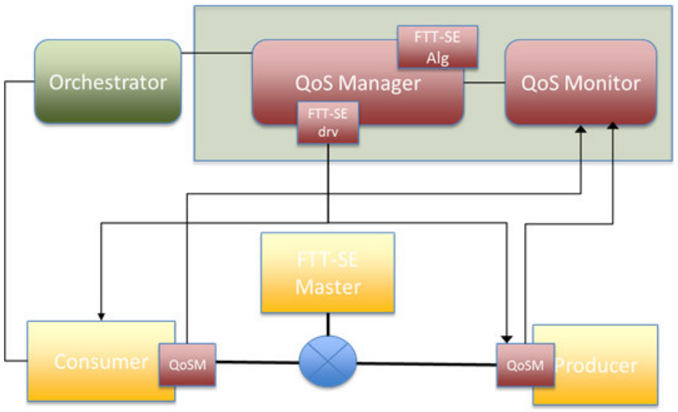

Figure 4 – QoS Architecture instantiation on an FTT-SE network

In this scenario it the responsibility of the FTT-SE slave to ask the orchestrator for a connection with a service producer using a specific set of QoS parameters. As an example of such a message assume a QoS parameter being requested is the delay time. An extract of the SLA is given next.

```
…
  "specifications": {
    "entry": [
      {
        "key": "delay",
        "value": {
          "value": "90.0"
          "period": "200.0"
          "msglength": "300"
        }
      }
    ]
 …
```

Code 1 – SLA extract: specifying delay QoS parameter

It is important to note that the SLA is defined in Janson, and the main objective of its structure is to allow it to be adapted for different QoS parameters, by allowing using adequate tuples of Key and Value.

The instantiation of the architecture also complies modules specific for FTT-SE networks. The QoSM module is responsible for monitoring the delay of the messages exchanged between producer and consumer and report any violation using the Event Handler systems (not represented in this figure) to interested parties. The QoS Driver is responsible for informing the producer and consumer about the parameters to be used on the connection, namely: stream ID, message size and period. After, it is the responsibility of both to establish the connection using the standard FTT-SE protocol. The FTT-SE Alg module takes care of performing complex calculations to determine if the new message stream can be admitted to the system or not. That can be done, supported by the mathematical model presented in [23]. This model tests if all message streams in the systems will be able to handle their delay requirements. If there is fail for one of the existing message streams or for the new one then the new message stream cannot be admitted.

VII. Conclusions

The paper presented the Arrowhead general approach, and went on describing how it can include services to support QoS requests for the interaction of service consumers and producers. An architecture was defined, up to the systems involved in QoS management and the services mediating their interactions. We advocate the use of SLA to define the QoS request, and we consider that the Orchestration System of Arrowhead takes care of bargaining with the QoSManager system the proper orchestrated services to support the requested QoS.

A discussion showed that, even in a SOA architecture that is heavily declarative, the reservation for QoS is better satisfied if the interactions are done explicitly and then responding to the imperative paradigm. Thus, the QoSManager system, which interacts with the Orchestration system only, is able to add QoS verification and setup to Arrowhead-compliant local cloud, without adding complexity to the architecture.

The architecture will act as container for mechanisms that will be studied in future work, for example algorithms that will “fill up” the *Algorithms* module and extend our approach to scenarios different from the FTT-SE one (Section VI). Future work will also discuss the impact of the approach on other characteristics of Arrowhead-compliant local clouds, such as security, scalability.

Acknowledgment

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234); also by FCT/MEC and the EU Artemis JU within project ARTEMIS/0001/2012 - JU grant nr. 332987 (ARROWHEAD).

References

[1] Ferreira, Luis Lino, et al. "Arrowhead compliant virtual market of energy", Emerging Technology and Factory Automation (ETFA), 2014 IEEE. IEEE, 2014.

[2] Koubâa, A., Alves, M., Tovar, E., Cunha, A., "An implicit GTS allocation mechanism in IEEE 802.15.4 for time-sensitive wireless sensor networks: theory and practice", Real-Time Systems Journal, Springer, Edited: Giuseppe Lipari. Aug 2008, Volume 39, Issue 1-3, pp 169-204.

[3] Valls, Marisol García, Iago Rodríguez López, and Laura Fernández Villar. "iLAND: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems." Industrial Informatics, IEEE Transactions on, vol. 9, no. 1 (2013): 228-236.

[4] Iiro Harjunkoski, Rasmus Nyström, and Alexander Horch. "Integration of scheduling and control—Theory or practice?." Computers & Chemical Engineering 33.12 (2009): 1909-1918.

[5] Da Xu, Li, Wu He, and Shancang Li. "Internet of things in industries: a survey." Industrial Informatics, IEEE Transactions on 10.4 (2014): 2233-2243.

[6] M. Albano, L.L. Ferreira, J. Sousa, “Extending publish/subscribe mechanisms to SOA applications”, 12th IEEE World Conference on Factory Communication Systems (WFCS 2016), May 3-6, 2016, Aveiro, Portugal

[7] Varga, Pál, and Csaba Hegedus. "Service Interaction through Gateways for Inter-Cloud Collaboration within the Arrowhead Framework." 5th IEEE WirelessVitae, Hyderabad, India (2015).

[8] Nichols, K., Blake, S., Baker, F. and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[9] Object Management Group, Inc. (OMG), “Data Distribution Service for Real-Time Systems Specification”, Version 1.1, December 2005

[10] AMQP Advanced Message Queuing Protocol, Protocol Specification, Version 0-9-1, 13 November 2008, http://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf

[11] P. Saint-Andre, K. Smith, and R. Troncon, “XMPP: The Definitive Guide”, O’Reilly, 2009

[12] J. Gutierrez Garcia, J. Gutierrez, and M. Gonzalez Harbour, “Schedulability analysis of distributed hard real-time systems with multiple-event synchronization,” in ECRTS’00, 2000, pp. 15–24.

[13] Li, Fei, Fangchun Yang, Kai Shuang, and Sen Su. Q-peer: A decentralized qos registry architecture for web services. Springer Berlin Heidelberg, 2007.

[14] Muthusamy, Vinod, et al. "SLA-driven business process management in SOA."Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research. IBM Corp., 2009

[15] Ding, Jianmin, and Zhuo Zhao. "Towards autonomic SLA management: A review."Systems and Informatics (ICSAI), 2012 International Conference on. IEEE, 2012.

[16] M. Albano, R. Garibay-Martínez, L.L. Ferreira, “Architecture to Support Quality of Service in Arrowhead Systems”, 7th INForum Simpósio de Informática (INForum 2015), 7-8 September, 2015, Covilhã, Portugal

[17] S. Girbal et al, “On the Convergence of Mainstream and Mission-Critical Markets”, In Proceedings of the 50th Annual Design Automation Conference, p. 185. ACM, 2013

[18] Nagios Enterprises, “Nagios”, www.nagios.com, last accessed in 04/2015.

[19] Open Networking Foundation, “OpenFlow Switch Specification, Version 1.4.0 (Wire Protocol 0x05)”, Oct/2013, available onlinde from: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[20] I. Estevez-Ayres, P. Basanta-Val, M. Garcia-Valls, J. Fisteus, and L. Almeida, “QoS-aware real-time composition algorithms for service-based applications,” IEEE Trans. Ind. Informatics, vol. 5, no. 3, pp. 278–288, Aug. 2009.

[21] Ricardo Garibay-Martínez, Geoffrey Nelissen, Luis Lino Ferreira, Paulo Pedreiras, Luís Miguel Pinho, “Holistic Analysis for Fork-Join Distributed Tasks supported by the FTT-SE Protocol,” in Proc. of the 11th IEEE World Conference on Factory Communication Systems, (to appear)

[22] Cardellini, Valeria, and Stefano Iannucci. "Improving SOA Applications Response Time with Service Overload Detection." 21st International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC'12). 2012.

[23] Michele Albano, Luis Lino Ferreira, Luís Miguel Pinho, Abdel Rahman Alkhawaja, “Message-oriented middleware for smart grids”, Computer Standards & Interfaces (2015), vol.38, pp. 133-143, Elsevier, DOI 10.1016/j.csi.2014.08.002

[24] R. Marau, L. Almeida, and P. Pedreiras, “Enhancing real-time communication over cots ethernet switches,” in WFCS’06, 2006, pp. 295–302.