A Reinforcement Learning Quality of Service Negotiation Framework For IoT Middleware

Udoh Itorobong Sunday

School of Computing and Communications
Lancaster University

Submitted in partial fulfilment of the requirements for the
Degree of Doctor of Philosophy
in Computer Science

Supervisor Dr Gerald Kotonya

June 2021

Declaration

This thesis is my work and has not been submitted in any form for the award of a higher degree elsewhere. The research has been carried out under the supervision of Dr Gerald Kotonya of the School of Computing and Communications at Lancaster University.

Udoh I. Sunday

3rd June 2021

Acknowledgements

Firstly, I would like to thank my supervisor, Dr Gerald Kotonya, for providing me with the opportunity to begin this PhD and for his invaluable guidance and support throughout my research at Lancaster University. His encouragement and enthusiasm have been priceless and a great source of motivation. I am grateful to the National Information Technology Development Agency (NITDA) for funding this research.

My most profound appreciation to my amazing friends, to name a few, Patrick Mendie, Victor Chiemela, Jenny Ratner, Onoriode Uviase and Zipporah Ebede, for their affection and heartwarming friendship.

This thesis is devoted to my parents, Mr and Mrs Sunday Udoh, and my siblings, Paul and Nsikak Udoh, for their unconditional love.

Finally, to my unborn child, may this work inspire you and make you proud of your Dad

For choosing to go to the moon…

Related Publications

I.S. Udoh and G. Kotonya, “Developing IoT applications: challenges and frameworks” in *IET Cyber-Physical Systems: Theory & Applications,* vol. 3, no. 2, pp. 65 -72, Jul. 2018.

I. Udoh and G. Kotonya, “A Dynamic QoS Negotiation Framework for IoT Services” in IEEE Global Conference on Internet of Things (GCIoT), pp. 1-7 Dec. 2019

I.S Udoh, and G. Kotonya, “A Reinforcement Learning QoS Negotiation Model for IoT Middleware” in *International Conference on Internet of Things, Big Data and Security (IoTBDS)*, pp. 205-212, May 2020.

Abstract

The Internet of Things (IoT) ecosystem is characterised by heterogeneous devices dynamically interacting with each other to perform a specific task, often without human intervention. This interaction typically occurs in a service-oriented manner and is facilitated by an IoT middleware. The service provision paradigm enables the functionalities of IoT devices to be provided as IoT services to perform actuation tasks in critical-safety systems such as autonomous, connected vehicle system and industrial control systems.

As IoT systems are increasingly deployed into an environment characterised by continuous changes and uncertainties, there have been growing concerns on how to resolve the Quality of Service (QoS) contentions between heterogeneous devices with conflicting preferences to guarantee the execution of mission-critical actuation tasks. With IoT devices with different QoS constraints as IoT service providers spontaneously interacts with IoT service consumers with varied QoS requirements, it becomes essential to find the best way to establish and manage the QoS agreement in the middleware as a compromise in the QoS could lead to negative consequences.

This thesis presents a QoS negotiation framework, *IoTQoSystem,* for IoT service-oriented middleware. The QoS framework is underpinned by a negotiation process that is modelled as a Markov Decision Process (MDP). A model-based Reinforcement Learning negotiation strategy is proposed for generating an acceptable QoS solution in a dynamic, multilateral and multi-parameter scenarios. A microservice-oriented negotiation architecture is developed that combines negotiation, monitoring and forecasting to provide a self-managing mechanism for ensuring the successful execution of actuation tasks in an IoT environment. Using a case study, the developed QoS negotiation framework was evaluated using real-world data sets with different negotiation scenarios to illustrate its scalability, reliability and performance.

Table of Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

The term *Internet of Things* (IoT) refers to a heterogeneous network of physical and objects embedded with electronics, software, sensors and connectivity to enable objects to achieve greater value and service by exchanging data with other connected objects via the internet [1]. In the context of IoT, a "thing" is a natural entity or man-made object that has a unique identifier and have the ability to transfer data and to interoperate within the existing Internet infrastructure. A *thing* can be a sensor module, a person with a heart monitor implant, a farm animal with a biochip transponder or a field operation robot that assists in a search and rescue mission or any [165].

The core idea of IoT is to integrate the physical world of things with the virtual world of the Internet [1]. IoT aims to blur the boundaries between digital and physical objects and enable seamless interaction between devices, machines, and humans. IoT promises to make it simple to incorporate the physical world into computer-based systems. In effect, real-world objects become integrated with the virtual world, allowing computing systems to remotely sense and act on the physical world [52]. IoT integrates a huge number of physical objects onto the Internet to transform high-level interactions with the physical world into a simple interaction with the virtual world [166]. By equipping physical objects with a technology stack, they become capable of interacting with each other over the Internet, resulting in a range of applications where tasks can be executed without human intervention.

IoT is an emerging paradigm and does not currently have a widely accepted definition. Different organisations and research communities have proposed several definitions of IoT. Some of the most commonly referenced definitions are:

- The International Telecommunications Union [2] describes IoT as: *"A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies."*
- The European Research Cluster for the Internet of Things [3] defines IoT as: *"A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network."*
- The IEEE IoT Initiative [4] provides its definition of IoT as: *"IoT is a network that connects uniquely identifiable "things" to the Internet. The "things" have sensing/actuation and potential programmability capabilities. Through the exploitation of unique identification and sensing, information about the "thing" can be collected, and the state of the 'thing' can be changed from anywhere, anytime, by anything."*

These definitions reflect the viewpoint and motivation of the stakeholders offering it. From these definitions, it is clear that IoT is characterised by the integration of real-world objects into the Internet. By merging the physical world with the digital world, IoT has the potential of significantly enhancing our comforts, automating and simplifying complex business operations. The promising benefits of IoT create an opportunity for creating domain-specific applications as well as applications that cut across several vertical IoT domains such as smart home, smart transportation, smart health and smart industry, as seen in Figure 1.1 The complete vision of IoT requires the integration of these vertical domains into a unified and horizontal domain, which is often referred to as the "smart life".

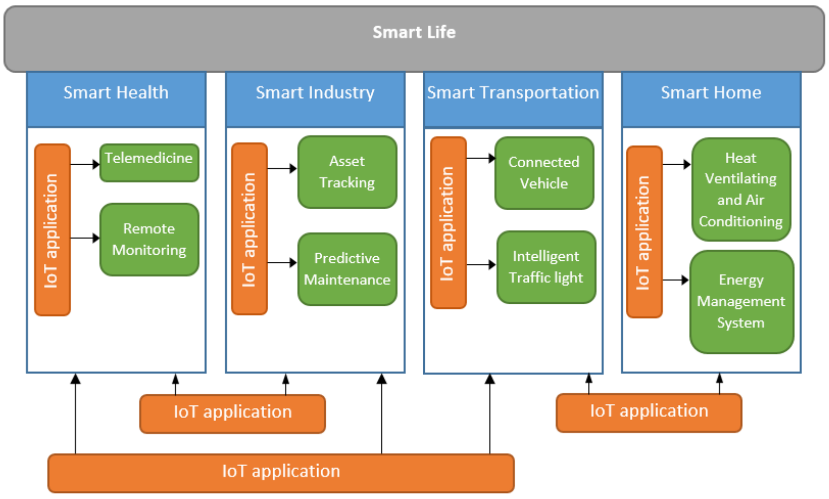


Figure 1.1: IoT application domains

The idea of IoT can be realised from the heterogeneous mix of numerous technologies. These technologies constitute the basic building blocks required to embed "intelligence" into "things" [95]. The RFID (Radio Frequency Identification) and other identification technologies, combined with the network of sensors and actuators, provide an interconnection between the physical world and the digital world. The communication network technologies connect heterogeneous devices to provide a set of specific IoT services. A review of several communication technologies and a detailed illustration of how objects with different communication protocols can be coupled together to provide the desired functionalities are presented in [6]. The middleware enables the development of IoT applications, the interoperability among heterogeneous devices and the management of resources and services. Several research initiatives such as [5][6][7] and [8] have structured these enabling technologies into n-layered technology architectures. Most of the proposed technology architectural models add more abstractions to the primary three-layer architecture, which consists of the perception/physical layer, communication/network layer and the application/service layer, as illustrated in Figure 1.2.

The physical/perception layer enables the interaction with physical objects through IoT devices such as sensors, actuators and RFID. It identifies, measures, gathers and processes data and states information associated with physical objects using the deployed IoT devices[8]. The communication/network layer is responsible for connecting the different objects together and transmitting data between these devices and end-user applications using various communication technologies(e.g WiFi, Bluetooth, Long-Term Evolution(LTE) and Zwave) and protocols(CoAP, MQTT, AMQP and HTTP REST) [6]. The application/service layer provides the software components for carrying out specific tasks using the data collected from the physical/perception layer and transmitted in the network provided by the communication/network layer. The software components can be used in providing data management (data aggregation, data analysis, data mining, data storage), service management(service discovery, composition, negotiation) and resources for the development of applications [7].

As an example to illustrate the various components in the three-layer architecture of IoT, consider an indoor vertical farming system consisting of several plant nodes. Each plant node uses a set of IoT enabling technologies to control environmental factors such as water and light to improve its yield quality. The sensors(e.g. humiture and moisture sensors) and actuators(e.g. the water pump and grow lights) that are connected to an IoT device (e.g. Raspberry Pi), which in turn is attached to a plant node, constitute the physical/perception layer. The Local Area Network and the WiFi technology through which the IoT device is connected to the Internet represent the communication/network layer. The software platform through which plant nodes can store their sensor readings and discover neighbouring plant nodes, the software interface through which plant nodes can interact with each other, and other complementary software systems built upon the software platform, constitute the application/service layer.

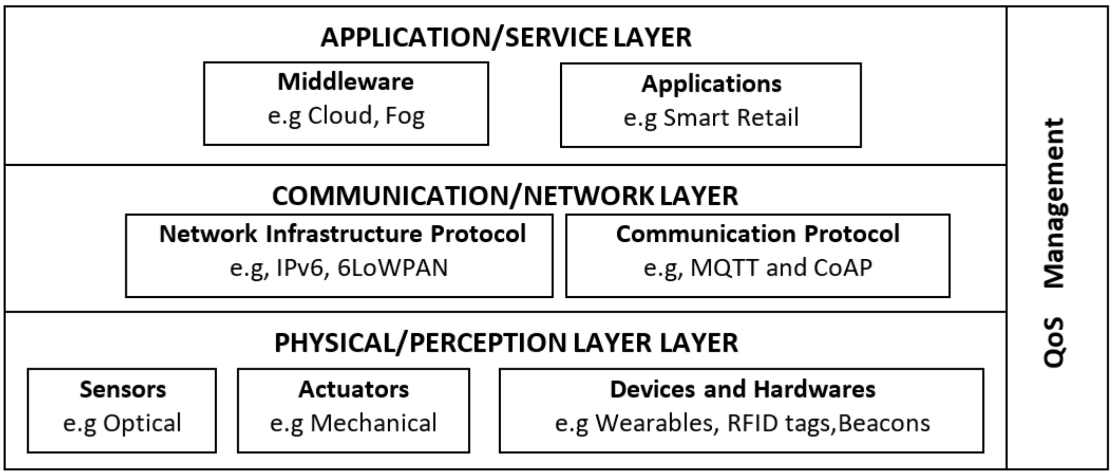


Figure 1.2: Technology architecture of IoT

In each layer of the technology architecture of IoT, it is critical to address QoS management concerns. This is to avoid situations that could lead to severe problems, particularly in IoT systems with stringent QoS requirements, such as embedded IoT medical devices and autonomous vehicle control systems or IoT systems for which providing best-effort QoS may not be sufficient for successful operation. To guarantee the successful operation of IoT systems, all layers of the IoT architecture must provide both effective and efficient QoS management strategies[167]. According to a recent survey by White et al.[103], current QoS research initiatives are primarily focused on the perception/physical and communication/network layers of the IoT architecture, with little attention paid to the application/service layer. The application/service layer, according to the authors, accounts for only 13% of all published works that investigate QoS in IoT.

This thesis focuses on the QoS in the application/service layer. In this layer, the notion of QoS, as stated in [169], refers to the different non-functionality characteristics of a service.(i.e. data provided by IoT devices). The non-functional characteristics of a service, such as response time and availability, are the constraint over its functionality [170]. QoS play an important role in the interactions that occur in this layer. QoS is a distinctive criterion in carrying out service management processes such as discovery, selection and negotiation. It allows services offering the same functionality to differ in terms of specific values in their non-functional properties. It enables service consumers to know in advance the quality of service they intend to use. From a service consumer perspective, QoS can be categorised into three classes: hard QoS, soft QoS and best-effort QoS. Hard QoS are QoS requirements that the service provider must guarantee. Soft QoS are QoS requirements that allow the service provider to provide some out-of-contract service quality. Best-effort QoS are QoS requirements that do not require any form of guarantee[59].To ensure a certain level of QoS, a Service Level Agreement (SLA) must be first established as a contract between the service provider and the service consumer via negotiation [168].

From a general perspective, the fundamental characteristics of the IoT are as follows:

- **Heterogeneity and Resource-constrained:** In IoT, heterogeneous devices interact by providing and consuming data in a variety of network topology. (such as fixed, wireless and mobile). This heterogeneity emanates not only from differences in features and capabilities but also for other reasons, including the manufacturer's and vendors' products and QoS requirements, since they do not always follow the same standards and protocols [9]. In addition, these devices are resource-constrained as they are characterised by limited computational, memory, power and connectivity capabilities.
- **Distributed and Large Scale:** The Internet of Things will include a vast number of devices that will communicate with one another over the Internet. The International Data Corporation estimates that there will be 41.6 billion connected IoT devices in 2025 [10]. Similarly, CISCO forecasts that 12 billion mobile IoT devices will be globally connected to the Internet by 2023 [11]. These devices will be distributed over several domains and locations at different scales.
- **Dynamic:** The resource-constrained devices connected in IoT can effectively manage their resources by dynamically changing their operation modes. Changes in the physical world can trigger these devices operation mode. In addition, given that these devices are mobile, they can leave or join a network anytime and be disconnected from a network due to poor connectivity or battery shortage. These factors make the IoT environment highly dynamic.

As illustrated in Figure 1.1, the Internet of Things can be viewed as a collection of vertical IoT systems whose mission is to improve the quality of our lives. To achieve this mission, IoT will require seamless integration and interaction between different IoT systems. Constituent IoT systems will have their own goal, function independently of other systems, and contribute to the overall mission of IoT. These IoT systems need to exhibit specific quality characteristics to achieve the vision of IoT. The leading feature is the capability of an IoT system to dynamically configure itself in response to changes in the physical world [3]. This characteristic introduces a number of challenges, which are the subject of the research problem addressed in this thesis.

1.1 PROBLEM STATEMENT

The IoT environment supports the discovery of IoT services to perform actuation tasks. This process allows IoT applications to carry out operations as their requirements evolve. The discovered IoT services are typically associated with IoT devices in different IoT systems, with each IoT system having its objective and can function independently [113]. Each of these devices in the different IoT systems can move around and interact with surrounding devices, leading to the spontaneous generation of events. Furthermore, they have their different IoT service configurations and constraints, which could change due to the changes in the physical world and operating environment.

In such a dynamic and complex environment, one of the problems is how to resolve the quality of service (QoS) contentions between IoT devices and across the different IoT systems as they provide and consume services. With service consumers with heterogeneous QoS requirements (e.g. hard QoS, soft QoS) interacting with service producers that often have varied QoS policy-driven behaviours (e.g. resource-conscious, performance-conscious) to execute an actuation task, it is necessary to provide a mechanism that allows service providers and consumers with different QoS objectives to reach a mutually agreeable QoS solution. If the IoT service provided by the service providers is forced on the service consumer and there is no opportunity for an agreement to be reached on the QoS parameters values of the IoT service, the service consumer task might be unsuccessful, leading to negative consequences. Hence, it becomes necessary to provide an automated negotiation framework that manages the QoS expectation and needs of the IoT service-oriented architecture users. However, despite the importance of a negotiation framework in the provisioning of IoT service, several factors make it challenging to manage and establish a QoS agreement effectively. These factors are due to the fundamental characteristics of IoT.

Firstly, IoT’s large scale and distributed nature will necessitate the negotiation framework components to be distributed across different locations at different scales. Consequently, some components of the negotiation framework will be implemented directly on IoT devices, while some components will be implemented in the IoT gateway/edge. Designing and implementing a distributed framework capable of taking consistent decisions from non-centralised resources is not always an easy task.

The second difficulty relates to the resource-constrained capacity of IoT devices. The limited computation and power capabilities of these devices introduce the challenging task of developing a lightweight negotiation framework with low communication cost.

Thirdly, the dynamic nature of the IoT environment poses a challenge to the performance of the negotiation framework. Changes in the physical world can affect the negotiation resources (e.g. memory allocation and CPU time and) and QoS constraints of IoT device. For instance, an increased workload on the CPU of the IoT edge node can change the CPU time of a negotiation process, and a deteriorating battery power can change the operational status of an IoT device.

1.2 KEY ISSUES AND RESEARCH QUESTIONS

Current research initiatives have a number of limitations that prevent them from providing an effective way for assuring QoS obligations of involved parties in the context of a particular IoT service. Consequently, they are inappropriate for addressing the QoS negotiation problem for the following reasons:

- **Insufficient support for managing negotiation uncertainties:** The majority of the existing negotiation frameworks struggle to manage the uncertainties in the negotiation environment, and as a result, they generate a QoS agreement with either a low social welfare (The sum of the utility gained by all the negotiating parties) or a low success rate (The number of successful negotiations) [63][62][39]. These uncertainties stem from the limited information about the negotiation state and the negotiating participants. If these uncertainties are not adequately managed, they can negatively affect the performance of the negotiation process.
- **Minimal support for proactive renegotiation:** In the event of a violation in the QoS agreement, it is required for the negotiation framework to initiate a prompt renegotiation as renegotiation is central to the reliability of a negotiation framework. However, only a few negotiation frameworks provide a renegotiation mechanism for QoS violation, and they are usually reactive as an IoT service failure must have occurred before the renegotiation process is initiated. There is a need for a negotiation framework to monitor and evaluate the QoS to detect a possible IoT service failure and proactively initiate and perform a quick service renegotiation. An adequate support for proactive renegotiation can increase the reliability of a negotiation framework.
- **Limited support for changing QoS preferences:** A number of negotiation framework implementations assume that the attributes of services are static[59][61][42]. However, the attributes of IoT services expressed as QoS parameters are deeply influenced by the real world, and as a result, the QoS preferences could change. This change could be triggered by the hosting IoT device inability to function correctly due to factors such as the gradual decline in its processing capabilities as its battery's voltage runs low. Consequently, it becomes necessary for negotiation frameworks to update the QoS profile of IoT devices as their external resources change. A negotiation framework that supports the continuous change of QoS preferences can reduce service failures in IoT systems.
- **Poor support for multilateral negotiation:** The execution of specific execution tasks underscores the need for initiating a negotiation process with more than two participants. The dynamic QoS preferences combined with the information uncertainty complicates the process of reaching a joint QoS agreement among several negotiating participants. A negotiation framework capable of performing multilateral negotiations is an indication that the framework can cope with scale.

These issues lead to four crucial research questions:

- How can a negotiation strategy be developed to manage the uncertainties in the negotiation environment effectively?
- How can the QoS negotiation framework support proactive renegotiation?
- How can the QoS profile of IoT devices be leveraged to mitigate service failure in IoT systems?
- How can the QoS negotiation framework be developed to support scalability?

1.3 OBJECTIVES

This thesis aims to address the challenges described in Section 1.2. by developing a negotiation framework capable of effectively establishing QoS agreements using a QoS negotiation strategy proposed in this research and proactively managing QoS violations. An experimental and heuristic methodology is used to evaluate the objectives of this study, which are as follows:

- Provide a reinforcement learning negotiation strategy for the generation and evaluation offers
- Provide proactive support for QoS violations through monitoring and renegotiation.
- Provide flexible support for the expression of QoS preferences
- Provide a solution that ensures that the framework can cope at scale.

1.4 CONTRIBUTIONS

The first contribution of this thesis is the identification of the challenges in developing QoS frameworks in IoT [9] and a literature review that provides a representative survey of state-of-the-art approaches that deal with the management of QoS during service provisioning. The survey spans from the approaches used in classical service-oriented environment to IoT service-oriented environment. It also includes discussing various aspects of the surveyed negotiation framework such as the negotiation model, technique, and architecture. In this discussion, the strengths and weaknesses of each negotiation framework are highlighted, with their approaches compared and contrasted with each other. Finally, a summary of how the existing negotiation frameworks perform against the QoS negotiation requirements in an IoT environment is presented.

The second contribution of this thesis is the novel modelling of the QoS negotiation process [12]. The QoS negotiation process is modelled as a Markov Decision Process(MDP) due to the dynamism and the uncertainties that characteristics the negotiating environment. MDP presents a standard formalism to describe multistage decision making in a dynamic environment [13]. Each concept in MDP is mapped to a corresponding negotiation element. By modelling the negotiation process as an MDP, negotiating parties can optimally make decisions taking into consideration uncertainties present in the IoT environment, leading to the execution of actions that makes the negotiation process efficient.

The third contribution is the proposed reinforcement learning (RL)-based negotiation strategy that enables negotiating parties to choose the appropriate negotiation tactic for any given negotiation state, leading to the generation of offers that maximises the chances of reaching a QoS agreement with high social welfare within the specific deadline [14]. It achieves this by computing the optimal policy using the value iteration method. The value iteration method was selected because it is less computationally expensive and takes less time to compute the optimal policy. This decision satisfies the requirement of generating a QoS agreement in real-time and in a resource-constrained environment.

The main contribution of this research is the reinforcement learning framework *IoTQoSystem*, which provides automated negotiation of QoS parameters at runtime for the invocation of IoT services [12]. The developed framework comprises two major components: a client and a service, collaborating to resolve and manage the QoS contentions among IoT devices in an IoT environment. Both components are implemented as microservices as they are designed to be lightweight and flexible and can be tested and deployed independently across the IoT Infrastructure.

The final contribution is the evaluation of the proposed framework. Using real-world data sets with different negotiation scenarios, the thesis presented a thorough assessment of the framework in terms of its scalability, reliability and performance.

1.5 THESIS STRUCTURE

Chapter 2 introduces the concept and architecture of service-orientation. It also provides an overview of the technology implementation of a service-oriented architecture (SOA). This chapter provides the framework for understanding QoS and insights into the QoS negotiation process in a service-oriented environment. It concludes by discussing the research initiatives in establishing QoS agreement via negotiation and how service orientation plays an essential role in the Internet of Things.

Chapter 3 describes the essential models in IoT and the remodelling of SOA for IoT services. It introduces the importance of IoT middleware and discusses the requirements of QoS negotiation in IoT middleware. Chapter 3 provides an in-depth review of the current QoS negotiation approaches in IoT middleware and concludes with a discussion of how the existing QoS negotiation frameworks perform against the QoS negotiation requirements.

Chapter 4 provides a description of the negotiation environment. It discusses how the characteristics of the negotiating environment influence the modelling of the negotiation process as a Markov Decision Process. It also presents the details of the proposed QoS negotiation model and concludes with the description of the novel reinforcement learning negotiation algorithm.

Chapter 5 introduces the design and implementation of *IoTQoSystem*, a QoS negotiation framework that attempts to manage the negotiation process of IoT services. It describes how the developed QoS negotiation framework addresses the objectives of this study. Within this description, key elements are identified and their processes explained, including an illustration of their architecture. This chapter discusses the design decisions and technologies adopted for the implementation of the framework and provide justifications for the choices made.

Chapter 6 describes the evaluation of the QoS negotiation framework and the set of experiments undertaken to demonstrate its feasibility and the effectiveness of the negotiation approach. It discusses the experimental results gained during the evaluation of the QoS negotiation framework, which was collected as it runs in a number of different negotiation scenarios using the selected case study.

Chapter 7 presents the final chapter of the thesis by first reviewing the achievement of the research objectives. It discusses the possible extensions to the research and highlights the potential research directions. This chapter concludes the thesis by summarising and reflecting on the findings of the research.

Chapter 2

SERVICE-ORIENTATION AND NEGOTIATION

Service-orientation, the recent paradigm for distributed computing, has changed how software systems are being designed, delivered, and consumed. At the heart of service-orientation are ***services*** that provide autonomous, platform-independent, computational elements that can be described, published, discovered, orchestrated and programmed to build networks of collaborating applications distributed within and across organisational boundaries [15]. Due to the principles of service orientation, service-based systems are intrinsic interoperable, architecturally composable, inherently reusable and easily extensible [16]. Realising the benefits service-orientation brings to software development requires a clear framework for establishing the quality of these services as the Quality of Service(QoS) offered reflects the software system usefulness.

This chapter discusses the concepts of Service-Oriented Computing (SOC) and provides an overview of the technology implementation of a Service-Oriented Architecture (SOA). It concludes with a review of the research initiatives in establishing QoS agreements via negotiation and the importance of service orientation in the Internet of Things and how it was adopted in this thesis

2.1 SERVICE-ORIENTATION CONCEPTS

One of the most important developments that have affected distributed software systems in recent times is the approach service-oriented computing (SOC). This approach allows access to the application system's functionality through a standard service interface, with a service for each discrete unit of functionality [17]. SOC is a computing paradigm that utilizes *services* as the basic constructs to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments. The promise of SOC is a world of cooperating services that are being loosely coupled to flexibly create dynamic business processes and agile applications that may span organizations and computing platforms, and can adapt quickly and autonomously to changing mission requirements. Achieving the SOC promise involves a Service-Oriented Architecture (SOA) that allows services to be discovered, combined and used to support any business processes [114]. SOA is a logical way of designing a software system to provide services to either end-user applications or other services distributed in a network through published and discoverable interfaces[119].Service-orientation has found application in many software systems, for example, enterprise computing [115], cloud computing[116], grid computing[117] and the Internet of Things (IoT) [118]. This section focuses on the fundamental concepts of service-oriented computing.

2.1.1 Services in Software System.

A service is a unit of logic that encapsulates a functionality within a given context [16]. According to [17], a service is a loosely coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed. The functionality provided by a service can be small or large. The size and scope of the logic represented by the service can vary from an atomic service to a complete software application. The delivery of a software application functionality to users is known as Software as a Service (SaaS). Figure 2.1 shows how a service can capture a varying amount of logic.

The essence of a service is to provide the service to an independent and different application or program [18]. For service to be delivered to a variety of service users, they must be aware of the service functionality and constraints. This awareness is achieved by service descriptions. A service description describes a service using a service description language such as the Web Application Description Language (WADL) [19]. A service description in its most basic structure defines the name of the service and the data expected and returned by the service. The manner in which service-users use service descriptions results in a relationship classified as *loosely coupled*.

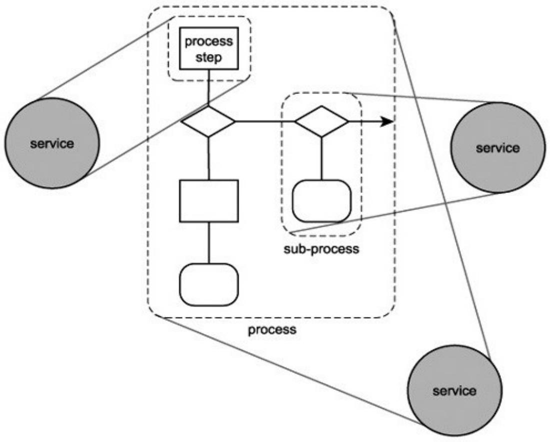


Figure 2.1: The varying amount of logic encapsulated by a service

For services to interact and perform a particular task, they must exchange information. The communications framework capable of preserving their loosely coupled relationship is called *messaging*. The communication framework defines the essential and optional components of messages passed between service users and providers [17]. An example of the communication framework is the Simple Object Access Protocol (SOAP) messaging protocol. Services that provide service descriptions and communicate via messages form a basic architecture known as the Service-Oriented Architecture (SOA).

2.1.2 Service Oriented Architecture

SOA is an element of SOC that enables service discovery, integration, and access, allowing application developers to overcome many distributed computing challenges. These challenges including designing and modelling complex distributed software systems,ensuring transactional integrity and QoS, and complying with agreements, while leveraging various computing devices (e.g., PCs, PDAs, cell phones, etc.) and allowing reuse of legacy systems [120]. As an SOC concept, SOA is a way of developing distributed systems where the system components are stand-alone services, executing on geographically distributed computers [17]. It is a software architecture where distinct components of a software system provide services to other components via a communications protocol over a network.

In the heart of SOA are three primary actors that play different vital roles: the service provider, the service consumer and the service registry. The service provider publishes the service description to the registry. The service registry serves as the repository for the available service description. The service consumer queries the service register for the service that meets its requirements. Figure 2.2 illustrates the interaction model between these components.

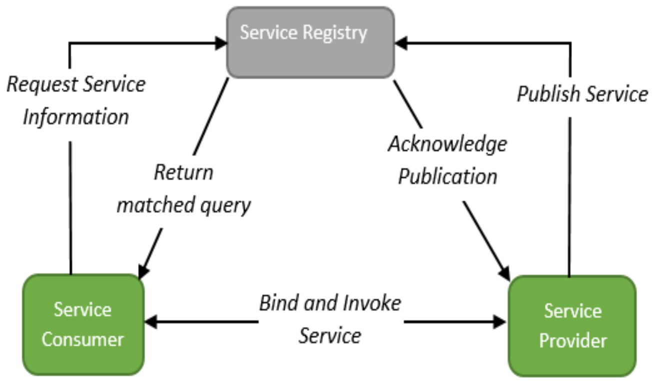


Figure 2.2: The conceptual model of SOA

The service registry is associated with the Enterprise Service Bus (ESB) that represents the backbone of SOA. The ESB is responsible for providing a centralised location for the access and discovery of services, fostering interoperability and maintaining compatibility by providing communication protocol independence to applications [20]. Given that the ESB promotes a centralised structure, resulting in a situation whereby a single point of failure can impact the entire software system, a new variant of SOA was introduced called ***Microservice.*** Microservice is a software architectural style that structures an application as a collection of services that are loosely coupled, independently deployable and communicate with each other via a language-agnostic protocol or API. Microservice is a style of engineering highly automated and evolvable software systems made of capability-aligned services [21]. Both SOA and microservice share the same set of principles, but they apply them differently. Both architectures decompose software system into services that cooperate to achieve an aim. However, both architectures accomplish this aim via different approaches. While SOA adopts the centralised approach of orchestration by using a middleware for the communication and integration of services, Microservice makes use of the decentralised approach of choreography for service integration and uses a simple messaging system for communication [22]. This difference is illustrated in Figure 2.3

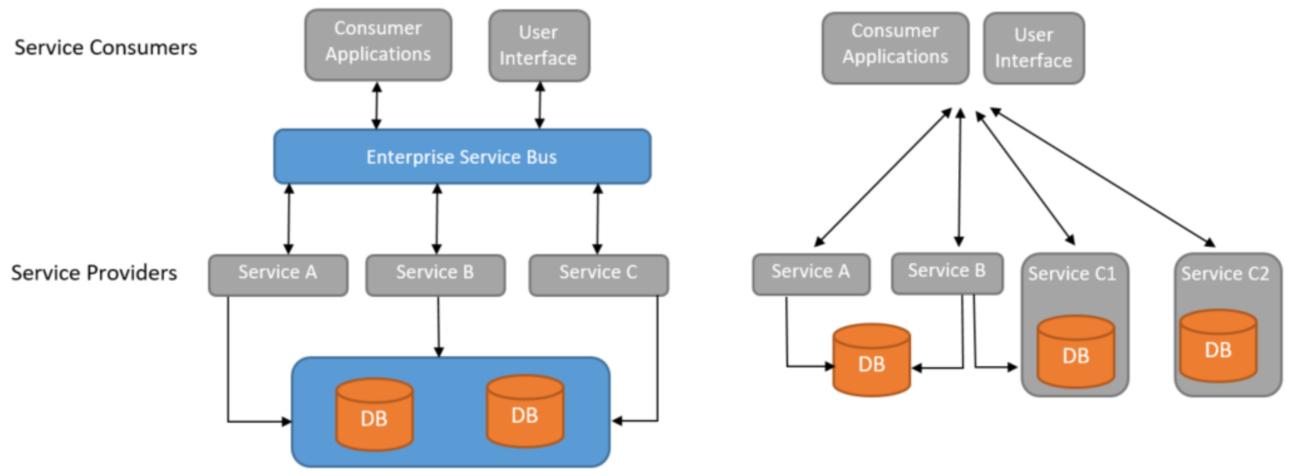


Figure 2.3:The architectural style differences between SOA and microservices

The adoption of either SOA or microservice depends on the purpose and type of application being developed. While SOA is better suited for large and complex business application environments that are characterised by the integration of multiple heterogeneous applications and services, microservice architecture is better suitable for smaller and well-partitioned software systems [23].

2.1.3 Technology Implementation of Service-Oriented Architecture

The service-oriented architecture paradigm can be realised through a number of technologies that promises a solution for the ever-growing complexity of software systems. Cloud services and web services are currently the most common forms of service for implementing SOA. A cloud service provides the foundation for the remote provisioning of scalable and measured IT resources such as a physical server, a virtual server, a network device or a network, a custom application, or a software program [24]. Depending on the IT resources being provided, cloud services are can exist in three forms: Software as a Service(SaaS), Platform as a Service(PaaS) and Infrastructure as a Service(IaaS) [25].

Webservices provide the basis for the development and execution of business processes that are distributed over the Web and accessible via standard interfaces and protocols. The World Wide Web Consortium [26] defines a web service as a "software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format(specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards."

Technically, these two types of service have been integrated to provide valuable solutions to users over the Internet. The functionalities of cloud services are typically encapsulated by web services technology architecture stack to deliver a set of business values. Standardisation bodies have recommended a technology stack for web services, and Figure 2.4 classify these technology recommendations based on their level of abstraction. These technologies such as the *Simple Object Access Protocol* (SOAP) [27], the *Web Service Description Language* (WSDL) [28], the *Universal Description, Discovery and Integration* (UDDI) [29] and the *Business Process Execution Language for Web Service* (BPEL4WS) [30], address how a web service is described, discovered, aggregated and accessed to provide high-level functionality.

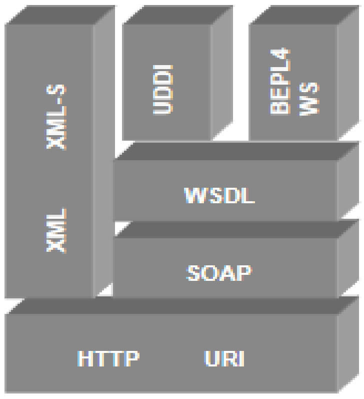


Figure 2.4: The webservice technology stack

SOAP provides an asynchronous mode for exchanging messages, and WSDL provides the standard for describing web services. While UDDI provides the mechanism for the registration and discovery of services, the BPEL4WS describes the control logic necessary for coordinating web services involved in a process model. The Web service specifications are based on XML standards which only define syntactic interoperability and characteristics.

The architectural pattern currently adopted by web services is the *Representation State Transfer (REST)* proposed by [31]. REST is a software architectural style for distributed systems that provide a set of design constraints defined by the REST architectural pattern. These constraints include stateless interaction, self-descriptive messages, uniform interface and resource-oriented. The REST architectural style defines three concepts: resource, representation and state. A resource is a physical object or an abstract concept that is important enough to be referenced and exposed for consumption. A representation is a view of the state of a response at a given time. This view can be encoded in any of the data formats such as XML, JSON and XHTML. The state of a resource can either be the information about a resource or the information about the path taken by a client to consume a resource.

Using the REST architectural style, a distributed system can scale well, exhibit loose coupling and high performance. Figure 2.5 depicts a RESTful web service architecture that captures the stateless interaction of the client and server exchanging resource representations. The interaction between the client and server is done through a uniform interface. The interface contains a resource identifier, *Uniform Resource Identifier* (URI) that identifies the resource and make it capable of being manipulated via HTTP method such as POST, GET and PUT

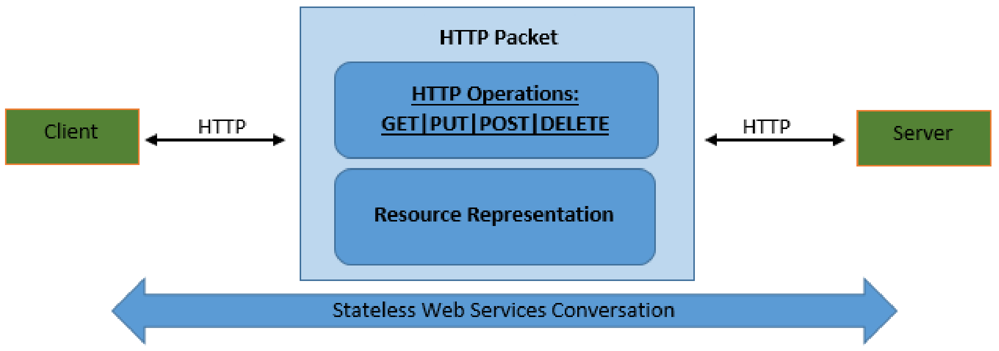


Figure 2.5: RESTful Web services architecture

2.1.4 Service Lifecycle

The idea of SOA is captured by the different activities required for running a service-oriented software system. From a service usage view, these activities define the life-cycle of a service. These activities include service publishing, goal specification and service discovery, service negotiation and agreement, service composition and execution and service monitoring and maintenance.

Service publishing: The service provider begins by offering a service by publishing the service description in a registry using a service description language. The publication of services will enable service consumers to discover services that are capable of achieving their objective.

Goal specification and service discovery: In this phase, the service consumer defines its objective, QoS requirement and search and selection criteria. The objective is decomposed into a set of tasks, with each task representing an abstract service with a specific function. Similarly, the QoS requirement is analysed and decomposed into a number of *QoS preferences* associated with each of the abstract services. The collection of these abstract services, together with the data flow and control among them, constitute the *Business Process* (BP). The discovery process begins when the service consumer queries the registry for a suitable service that meets the search and selection criteria for each of the abstract service. The criteria can be based on the following aspects of a service: functionality, QoS, interface, context and semantics. The criteria are executed against the service registry for services that meet the criteria. The execution of the query based on the criteria returns a candidate for each of the abstract services that best match the criteria, leading to the next phase.

Service Negotiation and Agreement: During this phase, service parameters values are negotiated and agreed on the definition of each abstract services that constitute the BP. Given that the service provider and consumer will typically have different preferences over the values of the service parameters, It is required they negotiate through the exchange of information and compromises to reach an agreement. This agreement is called the *Service Level Agreement* (SLA), and It serves as a *contract* binding both the service provider and the consumer.

Service Composition and Execution: Abstract services are mapped and binded to their respective concrete services to create a composite concrete service. The generated composite service is executed by the creation of a process instance.

Service Monitoring and Maintainance: This phase involves the continuous monitoring of the process instance for service failure or violation of the SLA, enforcing SLA compliance and supervising the compensation for service failures. When an SLA is violated, the process of renegotiation is initiated via the execution of service discovery for an alternative service or the modification of the existing agreement.

2.2 QoS NEGOTIATION FOR SERVICE LEVEL AGREEMENT(SLA)

In service-oriented infrastructure, the Quality of Service (QoS) of a service is a set of quality attributes that indicate the service's ability to satisfy a stated or implied requirement [32]. These attributes can be specified using two methods: *QoS models* and *Quality Specification Formalism*(QSF), as they provide a framework for understanding QoS. An agreed level of service quality attributes is usually defined in a contractual document called *Service Level Agreement* (SLA). The SLA represents the mutual understanding between service consumers and service providers, and the terms of this agreement are usually established through a *negotiation process*.

2.2.1 QoS Model

A QoS model is a hierarchically decomposed set of QoS attributes. It provides the basis for specifying QoS requirements, establishing QoS measures and establishing QoS evaluation. Each attribute in a QoS model belongs to a QoS group (i.e. QoS category or QoS dimension). For example, the response time QoS attribute belongs to the Performance QoS dimension in the QoS model proposed by OASIS [33]. While the majority of the QoS models describe domain-independent attributes, few QoS describe domain-dependent attributes. Domain dependent attributes are attributes that are only applicable to a particular application domain. A number of QoS models indicates if a QoS attribute is atomic or composite. Composite QoS attributes are attributes whose values are computed by evaluating two or more atomic QoS attributes. The attributes in a QoS model can be measurable or unmeasurable. A measurable QoS attribute, usually known as *QoS parameters,* are attributes that can be expressed in quantifiable measurements. An unmeasurable QoS attributes are expressed qualitatively and portray static information of a service. For instance, the QoS attribute can take the following values: {insecured, secured and very secured}. There exist several research proposals for QoS models that categorises QoS attributes in different ways. They differ in terms of structure, terminology and set of QoS attributes. Marc et al. [34] provide a systematic review of the current state of the art of QoS model for web services.

2.2.2 Quality Specification Formalism(QSF)

Given the inability of QoS models to specify all the QoS attributes in all the application domains, QSF provides a means of specifying extensible QoS models that can accommodate both generic and domain QoS attributes. In addition, QSF provides a mechanism for describing QoS parameters and their inter-relationship, and the agreed level of QoS as the popular service description language(i.e. WSDL) do not naturally support the description of non-functional requirements of a service. A Quality-Based Service Description (QSD) document, sometimes known as *QoS profile,* expresses a service's QoS capability by specifying the QoS parameter constraints. QSD is primarily used during the service publishing and negotiation phases of a service lifecycle. Apart from specifying QoS models, QSF can be used in specifying the QSD and SLA. By selecting a particular QSF approach, a negotiation framework that is independent of a specific set of QoS parameter can be designed and implemented. There exists three QSF approaches: QoS meta-models, QoS languages and QoS ontologies. QoS meta-model uses a model(e.g Unified Modelling Language(UML) and Object-Role Model(ORM)) to describe QoS parameters. QoS languages mostly use an XML-based schema as its QoS-meta model to describe QoS models. QoS ontologies specify QoS by providing both syntactic and semantic description of QoS terms, i.e. QoS parameters, QoS offers and QoS constraints. [35] reviews and compares the approaches of QoS description.

2.2.3 QoS Negotiation of Services

The QoS negotiation of services can be seen as the process by which an agreement can be reached over the QoS parameters values through compromises and the exchange of information [36]. The participants involved in this negotiation frequently have different preferences over QoS parameters, and they seek to reconcile these differences through negotiation. The participants involved in QoS negotiation are the service consumers, who request services with a certain range of quality, and service providers, who provide services with substantially varied quality. QoS negotiation can take two forms: *bilateral negotiation and multilateral negotiation*. Bilateral negotiation involves two participants(i.e. one service consumer and one service provider). Multilateral negotiation involves more than two participants (i.e. one service consumer and two or more service provider) negotiating over the values of QoS parameters to reach a joint agreement [70].

The negotiation process ensures a match between the request requirements of the service consumer and the service capability of the service provider(s) through the exchange of *offers*. An offer is a proposal that specifies the value of each QoS parameters that is beneficial to the negotiating participant that created it. Reconciling the preferences of both the service consumer and provider(s) requires that both participants automatically change their preferences within the specified QoS parameter constraints until an offer is accepted by all the participants. A negotiation framework can achieve such automation and flexibility. A negotiation framework is mainly characterised by the three elements of a negotiation model: the negotiation object, negotiation protocol and negotiation strategy. These three elements define the mechanism of automated negotiation. In addition, the integration and relative importance of these negotiation elements are determined by the negotiation context [37].

Negotiation Object: This defines the set of features of an element that the participants negotiate to reach an agreement over their values. In the service-oriented architecture, these features are the QoS parameters. QoS parameter values are usually negotiable, i.e. their values can be dynamically varied at runtime within a defined constraint. For a negotiation framework, the QoS parameters reference a QoS model that is either fixed or extensible. When QoS parameters reference a fixed QoS model, it means that the number of QoS parameter in a QoS profile is fixed, and their metrics, types and datatype values cannot be changed based on the domain in which the negotiation framework is deployed to. The referenced QoS parameters are usually performance-related attributes as they are usually not associated with a specific application domain. Conversely, when QoS parameters reference an extensible QoS model, the number of QoS parameters in the QoS profile can vary as the extensible QoS model typically relies on the ontological or declarative definitions of QoS parameters terms (i.e. metrics, datatypes etc.). It should be noted that the definition of a QoS parameter term can only be done before the negotiation framework is deployed. Since more than one parameters are usually involved during the negotiation process, It becomes necessary to allow the participants to know the relative importance of each parameter. Typically, this is achieved through a normalised weight for each QoS parameter.

Negotiation Protocol: Negotiation protocol describes the set of rules, which defines the limits to how the negotiating participants can communicate and exchange messages. It determines the form of QoS negotiation(bilateral or multilateral) a negotiation framework can support. It covers the permissible types of participants; the negotiation states (e.g. call for proposal, negotiation closed), the events that cause the change of negotiation states, and the valid actions of the agents in the different states of negotiation.

Negotiation Strategy: It defines the participants' decision model**.** The decision model is used for the generation of offers and the evaluation of counter-offers. It defines the decision rules followed by the negotiating participants to generate new offers, accept a negotiation solution and decline a negotiation solution. The participants' decision to carry out the aforementioned tasks can be influenced by the negotiation protocol adopted, the nature of the negotiation object, and the set operations that can be performed on it.

2.2.4 Approaches to QoS Negotiation of Web services

Negotiation represents one of the mechanisms for managing the QoS of web services. The QoS of web services is typically defined in a static quality document called WS-Policy. WS-Policy provides a framework for expressing and specifying web service capabilities and requirements as *policies* using the XML language [38]. It is usually adapted to define the negotiation participants decision model and negotiation preferences (i.e. the values of the QoS parameter referenced). This section presents a review of the vital research initiatives relating to the QoS automated negotiation of web services and cloud services. Each of the work attempts to address the QoS negotiation problem (i.e. the conflict in QoS preferences between service providers(s) and service consumer by implementing a selected negotiation paradigm.

Comuzzi and Pernici [39] present a QoS Negotiation architecture that supports the automated and semi-automated negotiation of web services QoS. It uses a negotiation broker, as shown in Figure 2.6, to store the values of the QoS parameters expressed by the negotiating participants and simulates the negotiation process. It uses the WS-policy to specify the negotiation preference and the WSLA to establish the negotiation outcome. The framework is designed to reduce the negotiation communication overhead. However, the negotiating parties are required to know the strategy models supported by the architecture before defining their selected strategy.

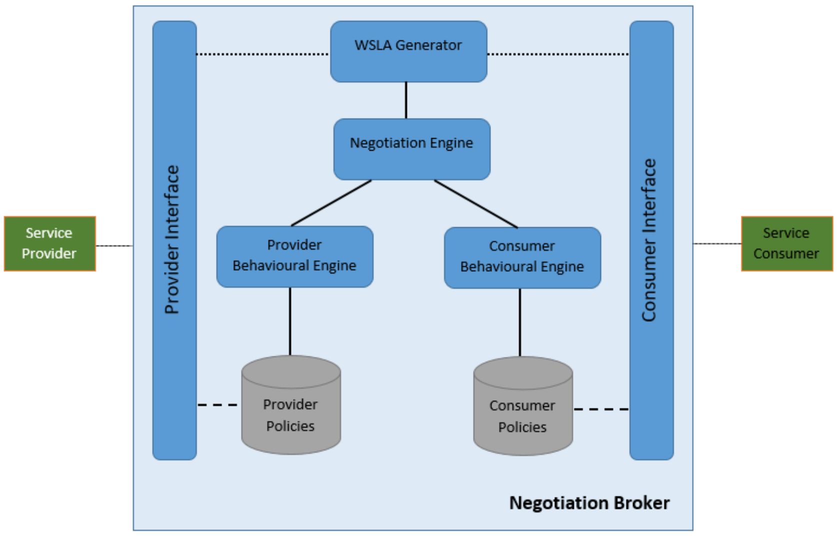


Figure 2.6: The architecture of the negotiation broker proposed by Comuzzi and Pernici [39]

Zulkernine and Martin [40] design a bilateral negotiation framework, shown in Figure 2.7, that uses a time-based decision function to generate SLAs. The framework uses the standard-based WS-Policy to express the negotiation parties preferences and a negotiation broker to execute the negotiation process using software agents. This approach ensures that an agreement is reached within the specified maximum negotiation time. However, adopting a time-based decision function usually leads to a non-optimal agreement at the end of the negotiation process.

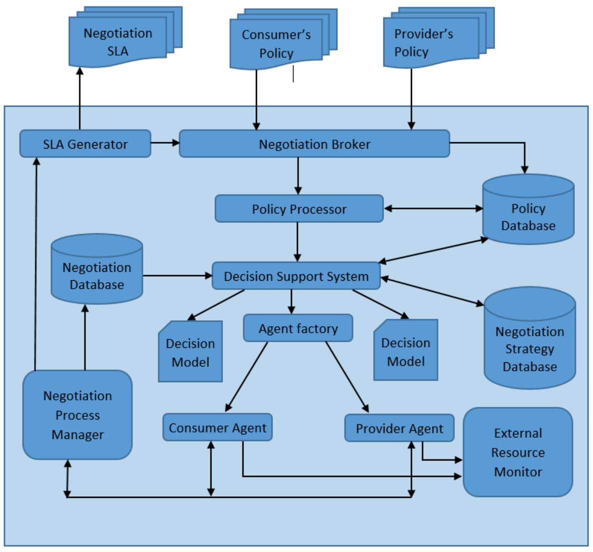


Figure 2.7: The architecture of the negotiation broker proposed by Zulkernine and Martin [40]

Hashmi et al. [41] have developed a QoS negotiation framework that uses a genetic algorithm (GA) to address the web service negotiation problem. It introduces the norm operator to the traditional genetic algorithm, making it possible for a service consumer to be privately involved in simultaneous negotiation with multiple providers. Although the framework supports multiple simultaneous negotiations for an optimised agreement, it increases the negotiation communication overhead.
Abdelatey et al. [42] describe a multilateral SLA negotiation framework using a time-based negotiation strategy. In this framework, multi-agents were used to generate an SLA among multiple negotiating participants. Since negotiation time was an essential issue in the proposed framework, a non-optimal contract can be created when the negotiation process terminates.

Chen et al. [43] propose a negotiation framework based on dynamic game theory for the bilateral SLA negotiation of cloud services using a broker. They introduced three models to find the degree of satisfaction for the service provider and consumer. However, the framework doesn't consider how to maximise the degree of satisfaction of the negotiation parties.

Anithakumari and Chandrasekaran [44] describe an automated negotiation framework that facilitated the bilateral bargaining of SLA between a service consumer and a provider in a cloud environment. As seen in Figure 2.8, the negotiation framework contains four main components. The service requisition component identifies and selects the service provider and consumer based on factors such as QoS values, provisioning interface, and negotiating parties' behaviour. The service listening component monitors the service registry for service changes. The SLA negotiating component finalises the interface for the service interaction, and the final component contains the information about the accepted SLA. With the use of pre-established SLA templates, negotiating parties have limited choice in generation offers during the negotiation process. For example, a negotiation party might want to generate offers with utility better than the utility associated with the predefined templates, but since it restricted to only selects one of the SLA templates, such actions may not be possible.

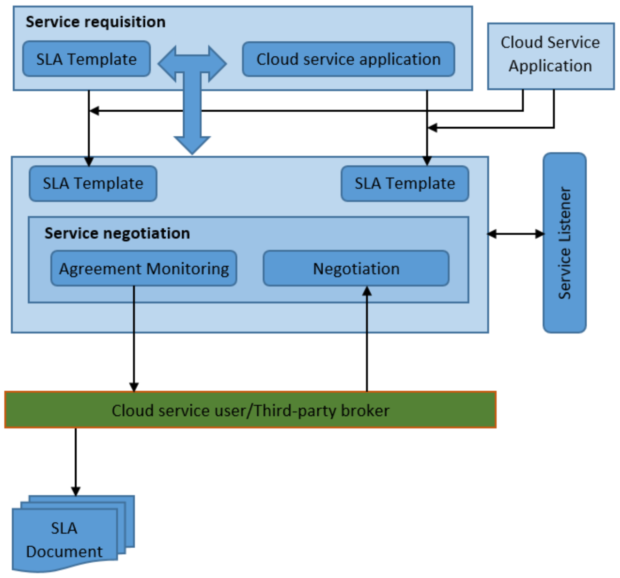


Figure 2.8: The architecture of the negotiation framework proposed by Anithakumari and Chandrasekaran [44]

Edu-yaw and Kuada [45] have developed *SLAnegmos*, an SLA negotiating and monitoring framework for cloud services. SLAnegmos primarily consists of two modules: a negotiation module and a monitoring module. The negotiation module, as shown in Figure 2.9, acts as a broker on behalf of both the service provider and the service user by simulating the negotiation process to generate the SLA while the monitoring module uses the SLA to monitor and check the service being provided for any QoS violations. When the SLA is violated, SLAnegmos only flags the violations and does not provide a means to alter the terms of the existing SLA.

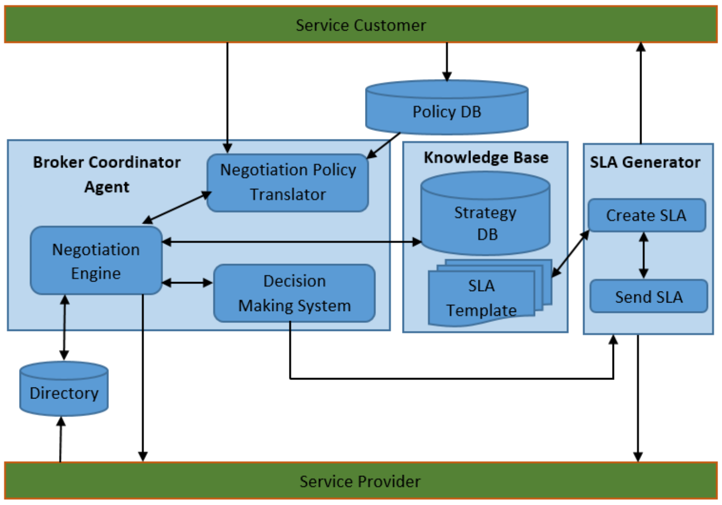


Figure 2.9: The architecture of the negotiation framework proposed by Edu-yaw and Kuada [45]

Khellaf et al. [46] describe an automated negotiation model aimed at quickly reaching an agreement on multiple QoS parameters through bilateral bargaining. It uses software agents to model both the service provider and service consumer behaviour and a mediator broker to resolve the QoS conflict between the negotiation parties. Agents are required to submit their proposal to the mediator, which then uses the bargaining strategy to produce an optimal proposal. This negotiation approach usually let both agents reach a mutual agreement quickly. However, this comes with a cost as agents may have to make significant compromises. For example, negotiating agents would need to reduce the utility of their offers to reach a mutual agreement quickly in situations where the negotiation space is small.

Table 2.1 provides a comparative summary of the paradigms adopted to address the QoS negotiation problem in web-based systems. The majority of the works focus on generating an SLA through an automated negotiation process. However, only a few addresses other aspects of the SLA management life cycle, such as SLA monitoring and SLA renegotiation. Due to the support of the rigid WS-Policy and the use of SLA templates, these negotiation frameworks may not be suitable in a dynamic environment as they may not respond effectively to the changes in the negotiating environment.

Table 2.1: Summary of QoS negotiation framework for web service.

QoS Negotiation Framework	**Negotiation Mode**	**Negotiation Technique**	**Negotiation Architecture**

Comuzzi and Pernici [39]	Bilateral	Concession	Broker-based
Zulkernine and Martin [40]	Bilateral	Time-based	Hybrid
Hashmi et al. [41]	Bilateral	Genetic Algorithm	Broker-based
Abdelatey et al. [42]	Multilateral	Time-based	Agent-based
Chen et al.[43]	Bilateral	Game theory	Broker-based
Anithakumari and Chandrasekaran [44]	Bilateral	Time-based	Broker-based
Edu-yaw and Kuada [45]	Bilateral	Game theory	Broker-based
Khellaf et al. [46]	Bilateral	Bargaining Strategy	Hybrid

2.3 SERVICE ORIENTATION IN INTERNET OF THINGS

The landscape of existing and future physical things in IoT is heterogeneous [52]. This device heterogeneity emerges from the differences in terms of features, capacity, operating platform and functionalities. To manage the inherent heterogeneity in IoT, the differences across these physical things are encapsulated using services. Services hide the complexities of accessing the functionalities of heterogeneous objects and, as a result, harmonises the interaction between them. The seamless interaction with a physical object is achieved through one or more services associated with its digital representation in the digital world.

The increasing number of networked heterogeneous devices in IoT will generate massive amounts of data [56]. The generated data is a mixture of structured, semi-structured, and unstructured data with various data formats such as text, binary, XML, CSV and JSON. This data might include analogue signals, discrete sensor readings, device health metadata, or large files for images or videos[148]. Given the non-uniformity of IoT data, services provide a well-defined and technical interface with which end-user applications can consume IoT data from different sources. The success of IoT relies on the seamless data exchange between heterogeneous devices [128]. Services offer a collection of protocols and standards to exchange data between the different devices and applications. IoT applications will often make critical decisions using the data fetched from the data providers. Services facilitate this decision by providing trust mechanisms that enforce privacy and confidentiality[150].

Services provide a convenient abstraction for developing large and systems, and they have been used as the building blocks for enterprise software systems [151]. In the same way, they played an essential role in enterprise systems; they are playing an important role in IoT as they are the primary components of the different architectural reference models proposed for IoT [152]. In IoT, a service can be regarded as a transaction between two parties, the service provider and the service consumer. Services enable interaction with the physical world by monitoring the physical state of entities and initiating actions that can change the state of physical entities[149].

Services can be classified according to the level of abstraction and how they are deployed [49]. A service can be a resource service, entity service, or an integrated service in terms of abstraction levels. A resource service(sometimes known as a low-level service) exposes the resource functionality of a device and deals with the QoS aspects such as availability and response time. An entity service provides the access point with which an object status and attributes can be read or updated. An integrated service is formed from the integration of resource and entity services. In terms of deployment options, services can be hosted on a device or in a network. On-device services are deployed on the devices they expose their resource functionality, and in-network services are deployed anywhere (e.g. the cloud, fog and IoT gateway) except on their associated devices. Figure 7 shows the different service abstraction levels and deployment options.

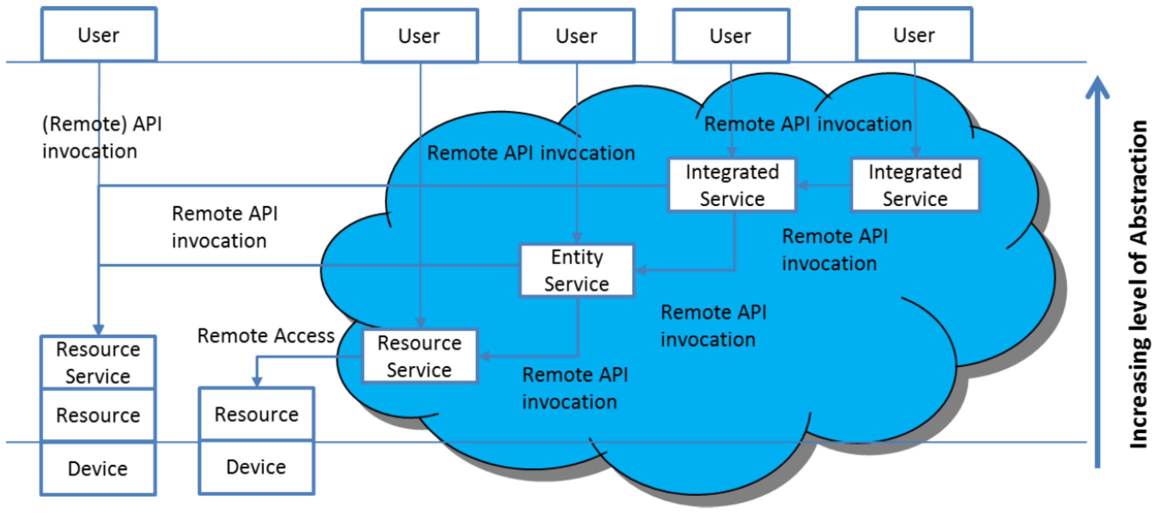


Figure 2.10: Service abstraction levels and deployment options by Bassi et al. [49]

Services abstract the resource functionalities of devices (for example, the physical quantity measurements of sensors) by wrapping the technical interface with well-known technologies such as SOAP and REST. Both technologies support the creation of Application Programming Interface (API), thus enabling the dynamic interaction(i.e. the transfer of data) between devices and applications [153]. The functionality of these heterogeneous devices accessed via these technologies is often called "*IoT service*" because they are provided by the devices that establish the connection between the physical world and the digital world. Unlike enterprise web services that are focused on mainly business entities, IoT service provides data about the physical world. IoT service encompasses heterogeneous entities, among which are mobile and resourced-constrained. In terms of availability, IoT services are very dynamic compared to enterprise web services, whose availability is relatively stable and reliable since they are hosted by stationary data-centres with immense computing capabilities. The availability of IoT services can be unstable and unpredictable due to the need for the resource-constrained devices to dynamically change state to conserve energy or the poor network connectivity of the new location of the mobile device [12]

2.3.1 IoT Service-Oriented Architecture.

Software architecture is necessary to provide access to and enable the sharing of services offered by IoT devices. Among the software architecture paradigms envisioned for IoT, Service

Oriented Architecture(SOA) is frequently proposed for IoT due to its inherent support for interoperability, composability and flexibility [51]. It has been argued that SOA is the most suitable architectural approach for modelling and implementing IoT based systems [154]. This is evident in the several service-oriented architectural reference models [152] and middleware platforms [155] developed for IoT. In this approach, services (low-level services) are used to expose the sensing and actuation capabilities of devices, and high-level services (which comprises two or more low-level services) are used for various decision making and data processing.

SOA provides a conceptual framework that allows software systems to be dynamically composed and reconfigured using services discoverable on the network at runtime. This architecture provides significant benefits over other design architectures. The dynamic nature of SOA means that it can support user application needs and expectations in a continuously changing environment such as IoT. In addition, services can be combined in different configurations and contexts to meet the needs of different service consumers. However, despite the benefits promised by the adoption of SOA in IoT, IoT inherent characteristics (described in Chapter1) make it difficult to deploy this architecture on IoT systems directly.

In addressing these challenges, a service-oriented approach that depends on mathematical models capable of providing estimation, approximation, prediction and conflict resolution can be used in remodelling the traditional SOA (described in Section 2.1.2) to satisfy the new requirements that IoT brings. Issarny et al. [52] defined a remodelled SOA that contains new components to satisfy IoT requirements. The proposed IoT service-oriented architecture introduces a service-oriented middleware that includes components that address the idiosyncrasies of IoT. Figure 3.4 depicts the IoT-service oriented architecture. The remodelled SOA addresses the issues of large scale, dynamism and heterogeneity associated with IoT.

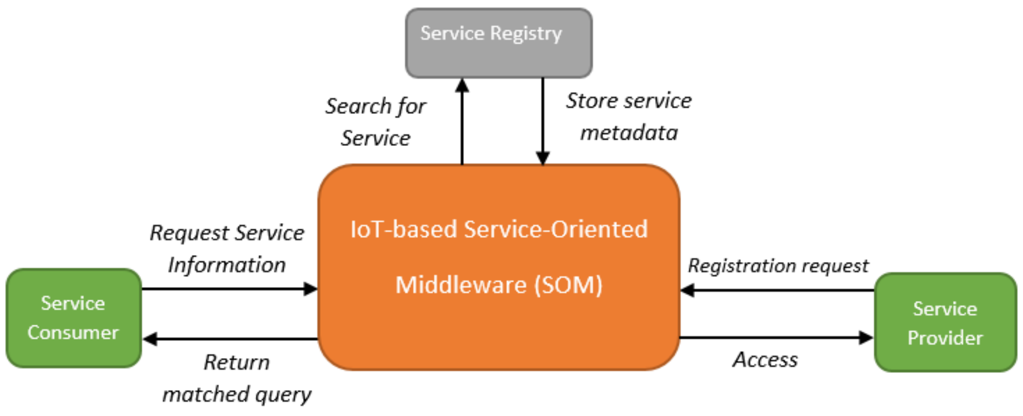


Figure 2.11: IoT Service-Oriented Architecture by Issarny et al. [52]

Similar to traditional SOA, the IoT-enabled SOA supports the complete IoT service lifecycle: *Service publication and discovery, Service negotiation and monitoring, Service composition and execution.* Each of these phases described in Section 2.1.4 is associated with a component in service-oriented middleware, each addressing a specific challenge that IoT brings to the service-oriented paradigm.

- **Service Publication and Discovery**: The key challenge in implementing this phase is dealing with the large scale of devices, usually sensors, that provide data of interest. With millions of devices expected to publish the descriptions of their real-world measurements, it is common to see a lot of redundancy in the registry of a traditional SOA during service discovery. This is because similar devices will publish their service descriptions without restriction. Also, the spontaneous interaction with the registry, e.g. to update their service descriptions, will generate many events. These uncontrolled events may cause problems such as event congestion and reduced event-processing capability in the registry of a traditional SOA.

  To minimise these problems, the middleware of the remodelled SOA contains a registration component that allows or prevents physical devices from registering their service description. The elimination technique for data redundancy in IoT described in [53] can be used by the registration component to carry out the on-demand registration. With the service registration component, only a subset of IoT devices can publish their service description, thereby reducing redundancy and event congestion.

  Apart from the issue of redundancy, the large number of devices creates another problem in the traditional SOA. With traditional SOA, tasks are associated with some business logic that can be satisfied by one or a few services. The tasks in IoT is often associated with a query that deals with the sensing of a physical quantity or initiating actions that can change the state of physical objects. A query in IoT (e.g. "*what is the quality in Lancaster*") cannot be satisfied by one or few services but numerous services. It becomes a challenge for service consumers with limited computation and communication capabilities to interact with the several service providers to acquire their measurement readings which may be of different formats and units. In addressing this challenge, the middleware in the IoT-enabled SOA handles the intensive computation and aggregation logic, rendering only the requested measurements to the service consumers.

- **Service Negotiation and Monitoring**: The highly dynamic nature of IoT services introduces a challenge in implementing this phase in the traditional SOA. IoT services are found in highly dynamic environments as their hosting devices can move around to establish connections with neighbouring devices, change state due to power shortage, and change operational mode due to connectivity and resources constraints. Thus, IoT services constantly degrade, vanish, and possibly re-appear.

  However, the traditional SOA infrastructure is considered static and long-lived as it comprises stationary data-centres with immense computing and networking capabilities hosting enterprise web services. This makes enterprise services relatively stable and unaffected by the changes that might occur in the real world . The negotiation process of enterprise web service typically involves the exchange of templates built from the rigid WS-Policy document, which uses the XML language. This negotiation approach may not be suitable for IoT services.

  In traditional SOA, the offer made by a negotiating participant is essentially a pre-defined template. The offer does not take into consideration the changes that may have occurred with a service. As a result, most successful negotiations in a traditional SOA would likely result in service delivery failure in a dynamic IoT negotiation environment. In addition, it is relatively inefficient to process XML documents in a resourced constrained environment as XML documents contain tags that require extra storage and bandwidth. To address this issue, the middleware of the remodelled SOA supports the use of lightweight data format e.g JSON, in specifying and updating offers, and contains a negotiation framework such as *IoTQoSystem* described in [12] that manages the QoS contract between IoT service providers and consumers in an IoT dynamic environment.

- **Service Composition and Execution**: The traditional SOA supports the creation of composite services from atomic services through the specification of a service invocation workflow that represents complex business processes. Composite services can be executed using a centralised (service orchestration) or decentralised(service choreography) approach [156]. Both execution modes allow constituent services to exchange few discrete messages. Directly applying service orchestration or choreography to IoT where there is a constant stream of IoT data that needs to be collaboratively processed and consumed isn't a trivial process. While the cloud is exclusively used to address this issue, the high energy and communication costs cannot be ignored. To improve the service composition and execution phase in the traditional SOA, the middleware of the remodelled SOA contains an in-network component that allocates the task of executing composite services to service providers based on a set of properties. A distributed data streaming element such as the *Dioptase* [157] can be used to map tasks to connected resourced constrained devices based on the current characteristics of these devices and the properties of the tasks.

  In addition, the existence of deep heterogeneity in IoT introduces complications in implementing service composition and execution in a classical SOA. While standardisation measures using technologies such as REST and SOAP have been used to address service access, the heterogeneity in IoT is much bigger with diversity in protocols, interaction modes, hardware features and operating platforms. To support interoperability among the heterogeneous components, the middleware in the IoT-based SOA contains a service bus protocol such as the eVolution Service Bus(VSB) described in [92] that facilitates the interaction between IoT services by carrying out a runtime conversion through the semantic mapping of protocols with respect to data and operations.

Chapter 3

QoS NEGOTIATION IN THE INTERNET OF THINGS (IoT) ENVIRONMENT

The Internet of things (IoT) promises to merge the physical world of things with the virtual world of the Internet [1]. In effect, physical things are equipped with a technology stack that makes them capable of interacting with each other over the Internet, resulting in a range of applications where tasks can be executed without human interference. These IoT based applications are modelled and implemented using service-orientation concepts [47]. This approach enables physical objects to communicate with each other via the provision and consumption of *IoT services* over the Internet. Typically, in this interaction, the functionalities of IoT devices are provided as IoT services to a domain-domiciled application or an application associated with a different domain. Consequently, there arises a need for IoT services to be delivered in a way that provides value and satisfaction to its end-users. In order to achieve this requirement in the complex and dynamic IoT environment, it is necessary to provide an automated negotiation mechanism that can ensure an acceptable level of QoS for the different service consumers.

This chapter provides an in-depth review of the current initiatives for supporting QoS negotiation in IoT environments. The chapter sets the context by introducing key models in IoT and identifying the QoS requirements for IoT middleware. The QoS requirements are used to formulate the assessment framework for establishing how well current negotiation initiatives address the QoS contentions in IoT environments.

3.1 IoT MODELS

3.1.1 IoT domain model

The IoT domain model provides a common vocabulary for defining abstractions, their responsibilities and relationships [48]. It contains components from the physical and digital world. Figure 3.1 illustrates the conceptual representation of the IoT domain model. The main concepts in the IoT domain model include the following:

- **User:** Given an IoT system, a user is an entity that uses the system by interacting with a physical entity to achieve its objective. Depending on the usage, a user can be a human or a digital artefact, e.g. software agent.
- **Physical Entity:** Physical Entity(PE) is the recognisable component in that physical world that is of interest to the user to attain a specific objective. It can be either human beings or physical objects. A set of attributes of the physical entity is used to represent the physical entity in the digital world. This representation is known as the Virtual Entity (VE) and has two basic characteristics. Virtual Entity is ideally synchronised with PE. This means that a change observed in PE is automatically reflected in VE. Similarly, a change in the state of any of the attributes of VE affects the corresponding property in PE. Virtual Entity is a Digital Artefact(DA). As a Digital Artefact, there is a possibility that one PE can be associated with more than one Virtual Entities. As a result, the concept of Augmented Entity(AE) is introduced. Augmented Entity is the combination of a PE together with one of its virtual counterparts(VE). AE can be seen

as the “things” in the context of IoT. Also, DA can either be an Active Digital Artefact(ADA), e.g. software applications and software agents, or Passive Digital Artefact(PDA), e.g. database entries and objects in an object-oriented programming language.

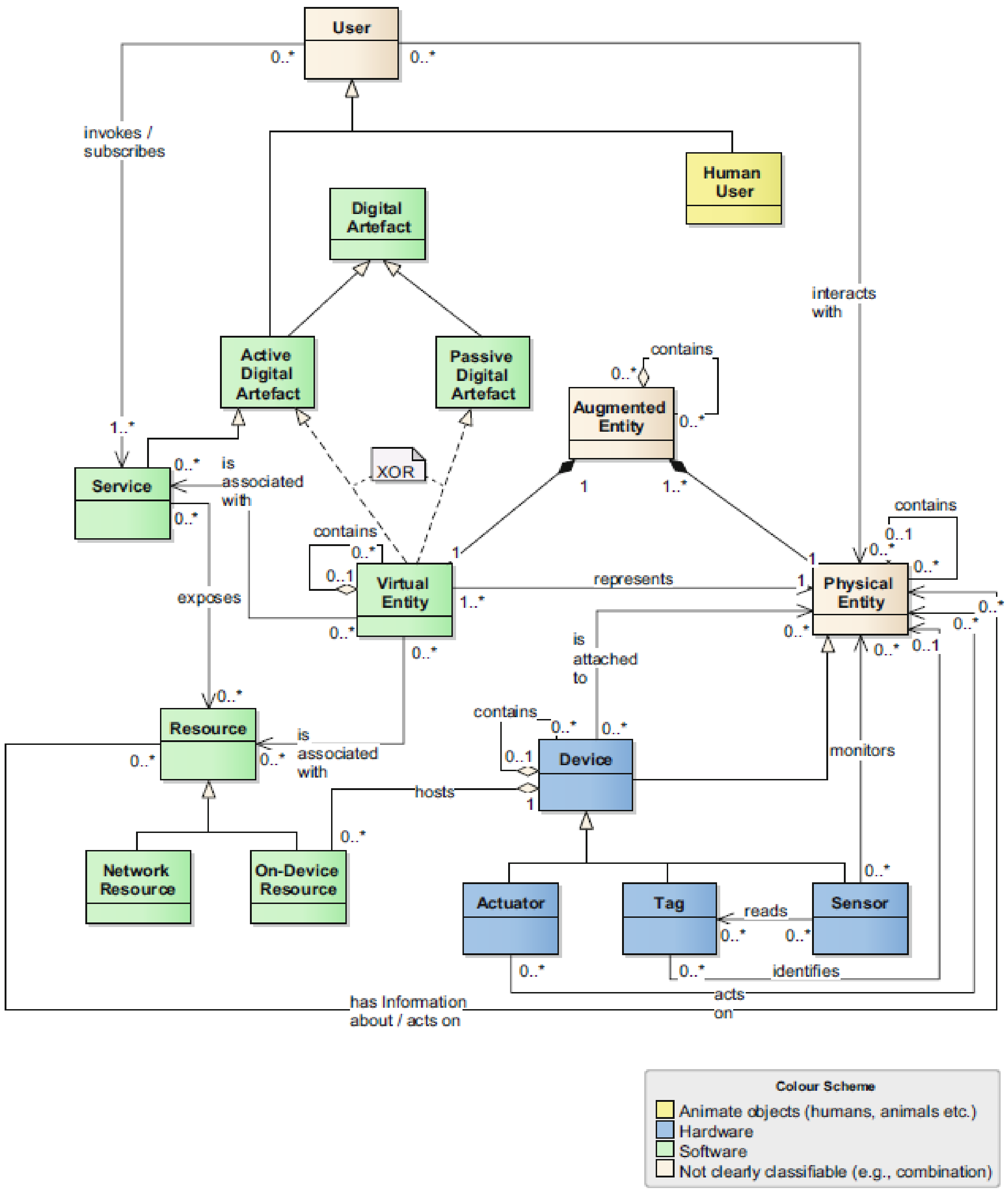


Figure 3.1: A conceptual representation of the IoT domain model by Haller et al. [48]

- **Device:** This is a technical artefact that establishes the connection between the physical world and the digital world by providing identification, sensing, actuation and computation capabilities. It is usually referred to as *“IoT device”* and can be physically attached to a PE or placed in the immediate surrounding of a PE. IoT devices provide the technological interface that mediates the interaction between the Physical Entity and the Virtual Entity, thus generating the Augmented Entity. In an IoT environment, three main types of IoT devices are of interest: tags, sensors and actuators. Tags uniquely identify a PE. Sensors monitor and provide information about a PE. Actuators change the state of a PE. An IoT device can be an aggregation of different IoT devices. For example, a node in a network comprising several sensors, actuators and data processing hardware can be viewed as an IoT device.

- **Resource:** Resources are software components that provide data about physical entities or facilitate the execution of actuation tasks on physical entities. They contain executable code that accesses, processes, and stores sensor data and includes code for controlling actuators. Resources usually have native interfaces. From a deployment perspective, Resources can be categorised as either on-device resources or network resources. While On-device resources are hosted locally on the devices, network resources are deployed somewhere in the network, e.g. resources hosted in a data-centre.
- **Service:** IoT service exposes the functionalities implemented by a resource. It hides the complexity of accessing a variety of heterogeneous resources by providing a well-defined and standardised interface. IoT service facilitates the interaction with the Physical Entity. IoT Services can be hierarchically structured in a way that a high-level service can invoke low-level services to provide high-level functionality. Also, the same type of IoT Service can be provided by different IoT devices attached to a PE or attached to a different PE.

3.1.2 IoT Service Model

The IoT service is a different type of service that facilitates interaction with the real world. Unlike a classic business service that revolves around a business logic to implement a real-world business rule, IoT service revolves around a thing-based query that identifies/senses/actuates*(hasType)* some real-world phenomenon [49]. In an environment consisting of heterogeneous and resource-constrained devices, IoT services expose the functionalities *(resources)* of these inter-connected devices to make them accessible to other parts of the IoT ecosystem through an interface *(hasInteface)* e.g. REST service interface. The functionality exposed through an IoT service can either be an output data *(hasOutput)* or input parameter*(hasInput)*. There are situations where an IoT service may not be available due to planned schedule maintenance or to save energy *(hasAvailability)*. An IoT service is usually associated with a working service area *(hasServiceArea)*. The observation area and the operation area defines the working area for sensing and actuating service IoT service respectively. For actuating IoT service, it is required to define the condition that needs to be fulfilled before controlling a Physical Entity *(hasCondition)*. Similarly, the effect of executing the service needs to be specified *(hasEffect)*. An important concept in the IoT service model is the role *(hasRole)* played by the users accessing IoT services. Figure 3.2 illustrates the IoT service model concepts and their relationships.

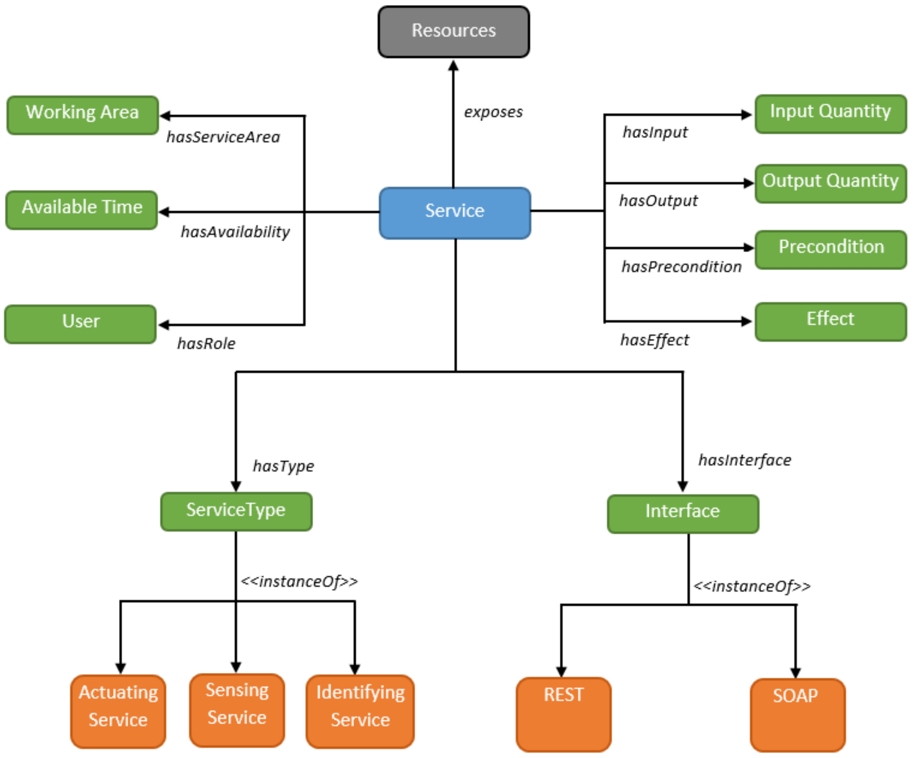


Figure 3.2: IoT Service Model by Bassi et al. [49]

3.1.3 IoT Information model

The IoT information model provides an abstract framework for modelling all the domain model concepts that are represented and controlled in the digital world [49]. It also models the relationship between these concepts. It provides a structure that defines the primary components of the information being managed in an IoT system at a conceptual level. This structure is used to define the functional interfaces of an IoT system as it is responsible for how information is being fetched, represented, gathered, processed and stored. The primary components of the IoT information model are Virtual Entity, Service Description and Association, as illustrated in Figure 3.3. Virtual Entity represents the Physical Entity in the digital world, and the Service Description details an IoT service. Association models the relationship between a Virtual Entity attributes and a Service Description. While static information or rarely updated information about an entity is included in a service description, dynamic information is externalised through Association.

For the automated discovery of IoT services and their associations, a Service Description Language(SDL) is required [50]. An ideal service description language for IoT services is required to have the following essential characteristics:

- **Physical Entity Centric**: The ability of an IoT service description language to define the concept of PE and model its relationships since IoT services are provided by PE or are required to control PE via IoT devices.
- **Service Area**: The ability of an IoT service description language to define the several perspectives of the location of IoT services. It should be able to specify the location of the service provider and service consumers. In addition, it should be able to capture the observation area and actuation area of sensors and actuators, respectively.

- **Service Schedule**: The ability of an IoT service description language to specify the availability of IoT services with respect to time and space as IoT services may or may not be available due to the scheduled times/designated locations of operations or for maintenance purpose.
- **Uncertainty of Information**: The ability of an IoT service description language to define the probability that the "quality of information" associated with IoT devices correctly represent the real-world property as evaluated by the information source at the time and context it was determined.
- **Extensible and Flexible**: The ability of an IoT service description language to be extended to accommodate new IoT service description parameters and can be used in different IoT contexts.

There is currently no specification standard language for describing IoT services, and since the field of IoT is still expanding, existing SDLs are usually adapted to suit the specific IoT context of interest.

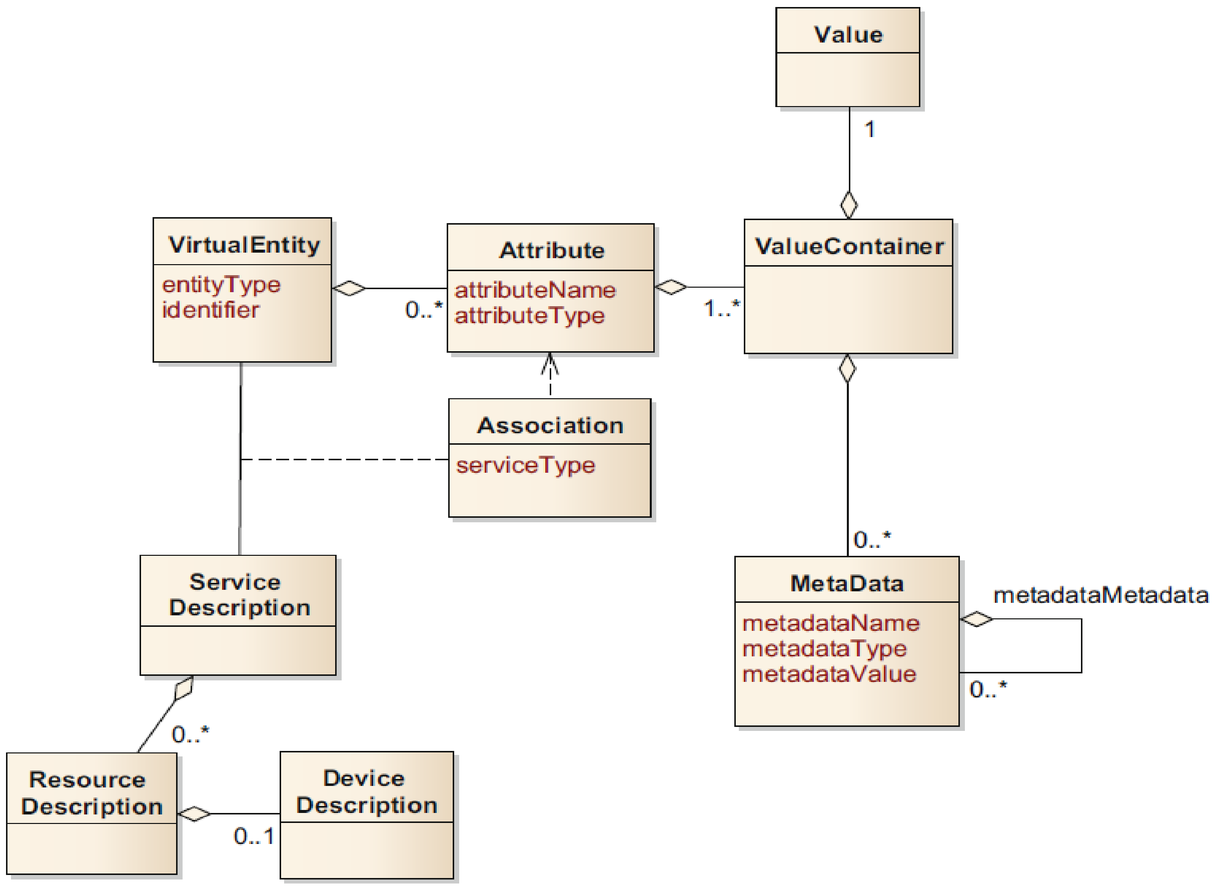


Figure 3.3: IoT Information Model by Bassi et al. [49]

3.2 QoS NEGOTIATION IN IoT MIDDLEWARE

QoS negotiation in IoT middleware is required to be automated and adaptive to cope with the uncertainties in the negotiation environment and dynamic preferences of the negotiation parties. This requirement is essential as it helps IoT-aware processes be autonomous and adaptive and better manage their activities and resources at runtime, with a reduced requirement for design time coordination. Also, it can help to discover overlooked solutions and maintain documented rationales for future references and reuse. This section discusses IoT middleware platforms and provides an in-depth review of the current QoS negotiation approaches in IoT middleware.

3.2.1 IoT Middleware Platform

An IoT platform is a multi-layer technology that enables the interaction, management, and automation of connected devices within the IoT ecosystem. It connects heterogeneous hardware devices via flexible connectivity options, enterprise-grade security mechanisms, and broad data processing powers. It also provides a set of ready-to-use features that substantially accelerate the development of IoT applications while also ensuring scalability and interoperability. IoT platform has a number of viewpoints. It is commonly referred to as the ***IoT middleware platform*** when the focal point is how it manages and enables the interaction between various devices and applications. It is also called the ***Cloud enablement platform*** to emphasise its primary business value, which empowers standard devices with cloud-based applications and services. When the focus is on the tools used by software developers to develop IoT applications, then it is called ***IoT application enablement platform*** [125]. In this thesis, an IoT platform will be simply be referred to as "IoT middleware".

An IoT platform as a middleware is the software infrastructure that enables the end-users to interact with smart objects [126]. IoT middleware is a key technology that provides the software system that serves as the mediator between connected IoT devices and consumer IoT applications[127]. It is the software interface between the layers of IoT devices and IoT communication networks on the one hand and the IoT application layer on the other. The IoT middleware simplifies the development process of IoT applications and the management of data, resources and QoS. The middleware aims to provide a common layer of abstraction and adaptation that integrates heterogeneous IoT devices and supports interoperability within the diverse IoT applications and services. To achieve this aim, the middleware is required to provide functional, non-functional support and architectural support, as illustrated in Table 3.1.

Table 3.1: IoT middleware requirements

Functional support	**Non-functional support**	**Architectural support**
Resource management	Security	Interoperability
Data management	Trust	Context-aware
Code management	Privacy	Distributed
Event management	Scalability	Adaptive
Service Process management	Reliability	Programming Abstraction
Device management	Availability	
Application development	QoS Management	

Several organisations have built IoT middleware to support one or more of the listed requirements. These middlewares differ in terms of features, functionalities and design approach. In terms of application development, they can be grouped into four broad categories[128] :

- **Publicly Traded IoT Middlewares**: These IoT middlewares are built and maintained by large publicly-traded organisations such as Amazon, Microsoft, Google and Oracle. Examples of this category of IoT middlewares include AWS IoT platform [129], Microsoft Azure IoT Hub [130], Google IoT Platform [131] and Oracle IoT Platform [132].
- **Open-source IoT Middlewares**: This set of IoT middlewares provide device management and data management services under open licenses. Popular examples of open-source middlewares include Kaa [133] and ThingSpeak [134].
- **Developer-Friendly IoT Middleware:** These IoT middlewares support the integration of IoT devices such as Arduino, Raspberry, etc., for the development of IoT applications. IoT Middlewares such as Carriots [135] and Temboo [136] falls under this category
- **End-to-End Connectivity IoT Middleware:** These are middlewares developed based on specific hardware and IoT devices. Particle Cloud [137] is a typical example of an end-to-end IoT middleware as it is designed to work with particle devices.

Razzaque et al [56]. surveyed existing IoT middlewares and grouped them into seven categories based on their design approaches:

- **Event-based IoT Middleware:** Event-based middlewares allow participants to interact through events. Each event is characterised by a type and carries a set of parameters that define the producers' state changes. Event-based middleware typically uses the publish/subscribe pattern to provide subscribers with access to a publisher's data streams. Hermes [138] and RUNES [139] are examples of event-based middleware developed for large scale distributed applications
- **Virtual Machine-oriented IoT Middleware:** Virtual Machine-oriented IoT middleware virtualises the distributed heterogeneous IoT infrastructure by allowing each node in the network to hold a Virtual Machine(VM). It organises an IoT application into separate modules and enables the VMs distributed across the network to interpret various software modules. The use of VMs provides a safe environment for the execution of IoT applications. Mate [140] and Melete[141] are VM-based middlewares for resource-constrained sensor modules.
- **Agent-based IoT Middleware:** Agent-based IoT middleware deals with executing IoT applications through software modularisation and mobile agents. IoT applications are broken down into software components, and software agents inject and distribute the program modules through the network. This approach enables IoT applications to be designed with high fault tolerance as agents are required to maintain their execution state as they migrate from one node to another. Ubiware [142] and Agilla [143] are IoT middleware solutions that adopt the agent-oriented approach in providing IoT middleware requirements such as resource and code management.
- **Tuple-spaces IoT middleware**: Tuple space IoT middleware adopts a design architecture based on a tuple-space structure. A tuple space is a data repository that supports the concurrent access of data. A group of tuple spaces form a federated tuple space, which resides on an IoT node. IoT application interacts by the writing and reading of tuples in a federated space. This approach allows IoT devices to easily and transitorily share data under acute network connectivity limitations. TeenyLIME [93]

and TS-Mid [94] are tuple space middleware designed for mobile devices to improve asynchronous communication in an IoT environment.

- **Database-oriented IoT middleware**: Database-oriented IoT middleware is a data-centric distributed middleware that views the IoT network as a virtual relational database system. Its architecture is designed to support the formulation of complex SQL queries and the mobility of querying nodes. TinyDB [144] and GSN [145] are examples of distributed query processing IoT middleware
- **Application-specific IoT middleware:** Application-specific IoT middleware implements an architecture that is specifically designed to fine-tune the network infrastructure for the efficient management of resources. AutoSec[146] and Adaptive Middleware [147] are some examples of software systems that adopt this IoT middleware design approach.
- **Service-oriented IoT Middleware:** Service-oriented middleware adopts the service-oriented approach in modelling the interaction between heterogeneous devices. Essentially, it abstracts the measurements of sensors, functionalities of actuators and properties of things as services. Service-oriented middleware provides the required interoperability and flexibility through loose coupling and reuse of software components. Hydra [54] and SOCRADES [55] are examples of IoT middlewares built on service-oriented paradigms.

The middleware is an essential component in the technology stack of IoT as it provides support for the interoperability of diverse applications and heterogeneous computing devices. This thesis focus is on developer-friendly service-oriented IoT middlewares.

3.2.2 Service-Oriented IoT Middleware

IoT service-oriented middlewares are IoT middlewares whose architectural design is based on service-oriented concepts. They abstract the functionalities of devices as services and facilitates the design and development of service-oriented IoT applications using service-oriented principles. They logically view the connected devices in IoT as a network of service providers and consumers and acts as an adapter in simplifying the interaction among them. They also provide a flexible programming model that allows developers to build applications belonging to different applications. [158]

Service-oriented middleware IoT solutions adopt the architecture discussed in Section 2.3.1 to promote service interoperability, reusability, composability and discoverability. Over the last few years, the middleware often proposed for IoT follows the SOA approach [164]. Alshinina et.al [159] argues that the service-oriented approach is the most suitable design architecture for developing applications that address challenges such as heterogeneity and QoS in IoT. Adopting service-oriented principles in IoT middleware offers several advantages in terms of device programmability and end-to-end integration. It reduces the need for gateway translations between software components thus, enabling the orchestration of services hosted on resource-constrained devices. Furthermore, it allows IoT middleware to support a network topology that is both unknown and dynamic through the standard WS-discovery or a RESTful discovery mechanism [55]

Several SOA based IoT middlewares have been developed to meet the IoT middleware requirements depicted in Table 3.1. However, IoT literature such as [56] and [57] that explores the current state of the art of IoT middleware indicates that none of the existing service-oriented middleware support all the requirements as requirements such as QoS management remain relatively unexplored. Each of the service-oriented middleware IoT solutions is designed to support a selected number of these requirements. For example, Fiware[160] and MiSense[161] supports data management and service process management, while SIRENA [162] and COSMOS [163] focuses on security.

3.2.3 QoS Negotiation Requirements in IoT Middleware

As discussed in Section 2.2.3, a QoS negotiation framework comprises three elements: negotiation object, negotiation protocol and negotiation strategy, with the negotiation context determining the relative importance and integration of these elements. The IoT-enabled service-oriented middleware is required to contain a negotiation framework that addresses the conflict in preferences between IoT service consumers and providers in a dynamic environment. To effectively resolve this QoS contention, the negotiation framework should support the following requirements:

- **Multi-parameter negotiation:** Most QoS negotiation involves participant negotiating over a set of QoS parameters such as availability, response time and throughput. These QoS parameters influence the negotiation strategy and the parties' preferences articulation that the negotiation framework must support. Depending on the specified negotiation context, negotiation parties can express their QoS parameters preferences in different ways. Consequently, there is a need for the negotiation framework to allow the negotiating parties to express multiple QoS parameter for the negotiation process(RQ1).
- **Dynamic user preferences:** Given the dynamic nature of the negotiation environment, the negotiation participants preferences can change since it is deeply influenced by the real world. During the negotiation process, there could be a change in the value of a QoS parameter as the negotiation participant learns a piece of new information about the physical world. Thus, it becomes necessary for the negotiation framework to support the continuous change of the negotiation participant preferences during the negotiation process. (RQ2).
- **Support for the intelligent selection of negotiation tactics**: Intelligent decision making is desirable in a dynamic negotiation environment to allow the selection of the right negotiation strategy at a given instance that will yield the most profitable agreement for all the participants. This involves providing the negotiation participants with the ability to change their negotiation tactic at runtime, based on the changes in the environment. The underlying strategy decision function should be robust to adapt to the different negotiation states by utilizing "peripheral knowledge" (RQ3).
- **Support for multilateral negotiations:** The IoT environment supports the execution of a task without human intervention. This task usually involves integrating services provided by different IoT devices to an actuator or an end-user application. With several service providers providing services that constitute a composite service, it becomes necessary for the negotiation framework to support a multilateral negotiation process that will yield the most profitable agreement between the various service providers and the service consumer. This characteristic of IoT underscores the need for carrying out an automated negotiation with multiple participants(RQ4).

- **Support for service monitoring and renegotiation:** After establishing an SLA, the need for the executed IoT service to be monitored and evaluated against the agreed SLA for any violation becomes essential due to the continuously changing IoT environment. If the QoS of the negotiated service is not monitored for failings at runtime, this could lead to negative consequences. In the event of a detected failing service, it is required for the negotiation framework to initiate a prompt renegotiation as both SLA monitoring and renegotiation are central to the reliability of negotiation frameworks(RQ5).
- **A balance between social welfare and success rate:** In generating a QoS agreement, there is usually a challenge of balancing the social welfare and the success rate.Existing literatures such as [58] show that in a competitive negotiation environment where QoS preferences are kept private, and the information is incomplete, the higher the probability of generating a QoS agreement with a high social welfare, the lower the probability of such negotiation being successful. Thus, the need to balance social welfare and success rate for QoS negotiation (RQ6).

These QoS requirements can be combined with other key negotiation features such as mode, technique and architecture to provide an effective mechanism for comparing and assessing QoS negotiation approaches in IoT. The following section reviews the state of QoS negotiation in the IoT environment.

3.2.4 State of the art in QoS negotiation approaches for IoT services

A negotiation framework is crucial for establishing the QoS agreement between service consumers and providers in an IoT environment. As illustrated in Figure 3.5, IoT service consumers can be IoT devices with actuation capabilities or end-user applications with an interface through which an IoT service can be accessed, while IoT service providers are typically IoT devices providing sensing capabilities. Compared to web services, QoS negotiation for IoT service is still in the early stage as few research works have been carried out to address the QoS contention between IoT service providers and service consumers.

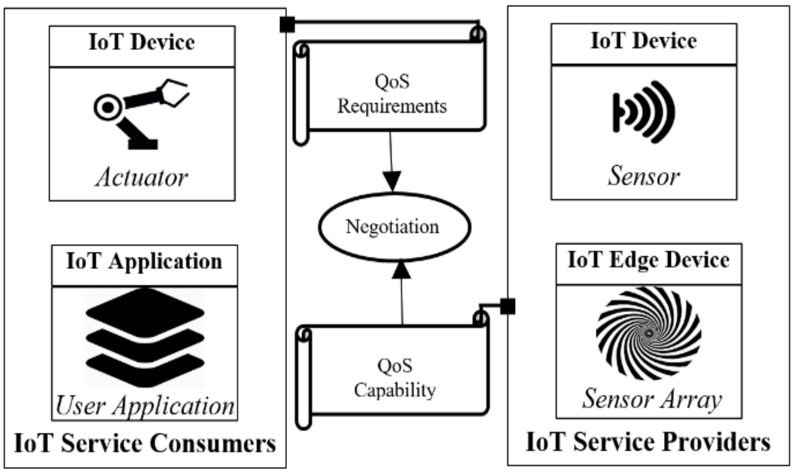


Figure 3.4: Participants involved in the QoS negotiation of IoT service

Mingozzi et al. [59] developed a framework that allows negotiation participants to negotiate the desired QoS using WS-Agreement-Negotiation standards. The negotiation framework is a component in *BETaaS* (Building the Environment for the Things as a Service) middleware platform [112]. The BETaaS negotiation framework is structured into layers, as seen in Figure 3.6, with each layer cooperating to provide QoS negotiation support for IoT applications. The application layer represents the IoT service consumers as they depend on the functionalities of physical things to carry out a set of given tasks. Physical things expose their functionalities through the TaaS layer(Things as a Service) as IoT services and the Service layer is responsible for providing these services to IoT applications after a negotiation procedure must have been carried out between the Service layer and the TaaS layer to determine the terms with which the functionalities of the physical things can be provided to IoT applications.

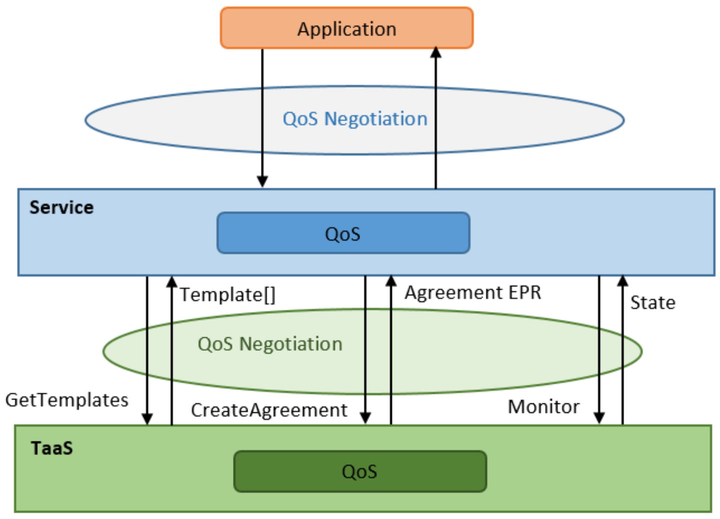


Figure 3.5: The QoS framework by Mingozzi et al. [59] with negotiation interactions

The WS-Agreement-Negotiation standard offers the negotiation framework the capability of generating a QoS agreement through the selection of a specific template that defines the QoS capabilities of physical things. The Service layer uses a template that closely matches with QoS requirements of the IoT application to create an offer. An agreed offer is created after the service provider (TaaS) responds with a confirmation message to the offer created by the service consumer(Service layer). Although the authors' work provides a simple model of service negotiation that attempts to satisfy the requirements of both participants via template selection, it only supports a one-shot negotiation, which mostly results in rejections, and if repeated, can be costly time-consuming. Also, the negotiation framework is tightly coupled to the BETaaS middleware platform, and as a result, it may be challenging to deploy the framework to other IoT middleware platforms.

Zheng et al. [60] combined game theory with a mixed negotiation strategy to resolve QoS contention between a cloud service provider and consumer in an IoT infrastructure. The Nash equilibrium of a negotiation game with two different negotiation tactics provides the theoretical foundation for resolving the difference in QoS preference between a cloud service provider and a cloud service consumer. The Nash equilibrium for the negotiation game is for the negotiation parties to choose between two different negotiation tactics for each negotiation round. To avoid a situation where a negotiation party is aware of the negotiation tactic selected by its counterpart, the selection of a negotiation tactic was left to chance. In other words, a negotiation party chooses a negotiation tactic based on a predefined probability. For each time step, an offer is generated with the negotiation tactic with the highest probability. The adopted negotiation approach is restricted to only bilateral negotiation scenarios and focuses on maintaining a balance between utility and success rate for an incomplete information negotiation game. This mixed negotiation strategy demonstrates a good balance between success rate and social welfare for multi-attribute bilateral negotiations. However, it ignores the changes that may occur in the negotiating environment as it only considers the negotiating participant's action.

Mišura and Žagar [61] developed an IoT mediator platform that contains a negotiation mechanism that facilitates the negotiation of services between IoT devices and IoT applications using the Contract Net Protocol(CNP) as the negotiation protocol and software agents to represent the negotiating participants. The mediator platform contains three main components: the HTTP interface, the negotiation module and the database, as shown in Figure 3.7. The HTTP module provides a REST [31] interface for devices and application to communicate with the platform. The negotiation module component uses its preselection module to query the database for the available devices based on the preselection conditions and uses the JADE container [85] to generate agents for the negotiation process. The database contains all the generated contract and information about each device and application.

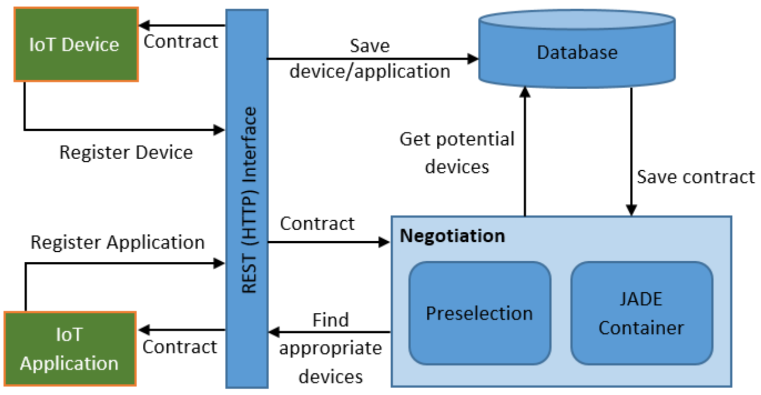


Figure 3.6: The QoS mediator platform architecture by Mišura and Žagar [61]

The generation of a QoS agreement begins with an application agent initiating a call for a proposal that describes the measurement needed to be carried out. The device agents examine the proposal and create an offer based on the requirements specified in the proposal. The offer is received by the application agent, and it can either accept the offer, rejects it or create a counteroffer. This exchange of offers continues until both negotiation agents agree on a specific offer and a request sent to the notary agent to validate the negotiation solution and the generation of a contract. The mediator platform, a web application, supports a web-based negotiation protocol and may be difficult to deploy on IoT nodes for real-time negotiations given its relatively large size.

Ghumman et al. [62] designed a flip-flop negotiation strategy based on a dynamic negotiation concession tactic and a time-dependent 3D utility function. This negotiation strategy aims to quickly reach an agreement between a cloud service provider and consumer through the polynomial extrapolation of the opponent's concession. The flip-flop negotiation strategy allows negotiating agents to make offers that gravitate towards their opponent's preferences using the polynomial interpolation method based on the opponent's concession pattern. The negotiation process using this strategy begins with the cloud service consumer generating a number of offers based on a predefined concession. The cloud service provider responds with a series of counter-offers, and the time the offers were made is recorded. The cloud service consumer estimates the utility of the final offer of the service provider using a polynomial extrapolation function and determines a new concession for the next offer. If the concession increases (flip), the change in the concession is pushed to the concession stack. Similarly, if the concession decreases (flop), the difference is pushed into the stack. When the cloud service provider adopts a greedy strategy, the flop step serves as a means to recover the loss made in the previous offer, thereby reaching an agreement quickly. The estimation of the opponent's final offer and the flip-flop process continues until a contract, or the negotiation deadline is reached. The proposed model is suitable for applications where time is a pivotal factor in a negotiation as the QoS contention is resolved quickly. However, this approach is characterised by a QoS agreement with a low social welfare because each negotiating participant is required to reduce its utility until an agreement is reached.

Alanezi and Mishra [63] implemented a privacy negotiation mechanism that resolves the difference in users' privacy requirements in an IoT environment. The negotiation scheme enables IoT applications and IoT deployment owners to express and enforce their privacy and preferences. In negotiating the privacy policies of the IoT application with the IoT deployment owners, the privacy negotiation model attempts to satisfy the privacy requirements of both parties. It uses XML to describe the privacy requirements of both users with the IoT application using the <data-in> tag to specify the type of data it wishes to acquire and the IoT deployment owners using the <data-out> tag to indicate its acceptable data collection practice. Essentially, the negotiation scheme matches the <data-in> and <data-out> tags defined in the IoT application privacy policy against the <data-out> and <data-in> tags defined in the IoT deployment owner privacy policy respectively.

The interaction between both negotiation parties using this privacy model begins with the IoT application requesting access to a specific type of sensor data from the IoT owner using the <data-in> tag. On receiving this request, the IoT owner computes the utility of the request. If the computed utility is higher than or equal to the utility computed from its <data-out> tag, it accepts the requests and relays the sensor data. However, if the IoT owner finds the request unacceptable, it uses the information in its <data-out> tag to create a proposal. On receiving the proposal, the IoT application checks the proposal utility against a second priority privacy policy. The negotiation fails if no second priority policy is defined in the XML file or the IoT application deems the proposal unacceptable using the first priority policy. However, if the IoT application finds the proposal acceptable, it informs the IoT owner and the sensor data is relayed to the IoT application. While this negotiation solution can satisfy the privacy requirements of the negotiating participants in simple negotiations scenario, its go/no-go scheme of privacy negotiation could lead to numerous negotiation failures in situations where both negotiation parties have stringent privacy requirements

Li and Clarke [64], [102] designed a QoS negotiation model that uses the different states of the WS-Agreement Negotiation (WSAG-Negotiation) specification as the decision rules that define the interaction between negotiating parties. WSAG supports the exchange of offers between a service provider and a service consumer and creates a QoS agreement from agreement templates. The template provides the blueprint for the creation of offers and the agreement as it contains default values of the negotiable QoS parameters. WSAG specifies the states with which an offer can take: Advisory, Solicited, Acceptable and Rejected. An advisory offer represents multiple back-and-forth interaction, and the actions negotiation parties can take include accepting an offer, rejecting an offer and generating a counteroffer. A solicited offer represents a single reply-response interaction, indicating that negotiating parties can only accept or refuse an offer. An acceptable offer and a rejected offer requires no negotiation as the former indicates that the offer is accepted and the latter indicates that the offer is rejected.

A negotiation session is usually between an IoT gateway and a service provider, and it begins with the IoT gateway sending a request specifying its QoS requirements. On receiving the request, the service provider creates a number of offers using the WSAG agreement template, each satisfying parts of the QoS requirements and their associated offer state. The service consumer evaluates each of the offers using a scoring function and selects the most preferred offer. The actions to be carried out on the selected offer depends on the state of the offer. The authors considered the domain-specific properties of IoT services in their approach. However, in a multi-attribute negotiation scenario, the adopted negotiation model can increase the overhead time as it involves a multi-offer two-stage decision process which may be redundant in some negotiation scenarios.

Table 3.2 summarizes how the current IoT service negotiation framework, and how they perform on the requirements outlined in Section 3.3.3.

Table 3.2: Summary of QoS negotiation frameworks for IoT service

Features / **QoS Negotiation Framework**	**Negotiation Mode**	**Negotiation Technique**	**Negotiation Architecture**	**QoS requirements**					
				RQ 1	**RQ 2**	**RQ 3**	**RQ 4**	**RQ 5**	**RQ 6**
Mingozzi et al. [59]	Bilateral	n/a	Broker-based	●	○	○	○	◐	n/a
Zheng et al. [60]	Bilateral	Mixed strategy	Agent-based	●	n/a	◐	○	○	●
K. Mišura and M. Žagar [61]	Bilateral	Imitative Tactics	Agent-based	●	○	○	○	n/a	n/a
Ghumman et al. [62]	Multilateral	Flip-Flop strategy	Agent-based	●	○	○	◐	○	○
Alanezi, and Mishra [63]	Bilateral	n/a	n/a	●	○	○	○	○	○
Li and Clarke [64] [102]	Bilateral	State-based strategy	Agent-based	●	○	◐	○	○	n/a

Legend	
●	Supported
◐	Weakly Supported
○	Not supported
n/a	No information available

This thesis also reviewed other relevant works that deal with the other aspects of QoS such as the selection, discovery, provisioning, modelling and placement of IoT services in the Internet of Things ecosystem.

White et al. [103] examined the current QoS approaches in the Internet of Things. A systematic mapping process was conducted to identify the research contributions made in the QoS of each layer in the IoT technology architecture and what quality factors were used for the evaluation of the QoS approaches. The authors provided visualisations that illustrates the QoS issues addressed in each layer and concludes with the areas that require further research.

Samann et al. [104] reviewed the QoS provisioning techniques for IoT systems. Each of the surveyed schemes was examined based on the specific problem they were designed to address and was evaluated using different QoS metrics and baselines. This study found out that most of the identified techniques selected fog computing as their network model to address QoS provisioning issues such as scalability, latency and network utilization. The reviewed techniques made use of QoS metrics that best fit the various QoS provisioning solutions. As a result, this study concludes by arguing that it is difficult to compare the effectiveness of each of the QoS provisioning solutions as each technique considered different QoS metrics for their evaluation.

Awan et al. [105] investigated the traffic delay problem resulting from transmitting a large volume of data between IoT devices. They developed a service model that provides a suitable QoS level for the transmission of delay-sensitive information in IoT. The service model analyses a finite capacity queuing system with a push-out buffer management mechanism and a pre-emptive resume service priority. With this model, the performance of IoT devices can be predicted under various traffic conditions and IoT applications that produces and consumes delay-sensitive information can be modelled to ensure that the highest priority is given to the most delay-sensitive data.

Cao et al. [106] combined Relational Topic Model (RTM) and Factorization Machines(FM) to recommend IoT services for the creation of IoT mashup applications. In achieving this, RTM was used to model the relationship between services and IoT mashup and mine the latent topics that characterise the correlation between them. The derived latent topics are then combined with multiple dimension QoS information to predict the relationship among IoT mashups and services.

Badawy et al. [107] designed a dynamic QoS provisioning framework (QoPF) that uses a backtracking search optimisation algorithm (BSOA) to address real-time adaptive sensing issue and performance degradation of IoT composite service. The framework maximises the QoS in composite IoT services by providing an optimal service composition through a linear search and ensuring a balance between performance and service reliability. The framework was evaluated using metrics such as throughput, jitter and delay time. The results indicate that the BSOA approach adopted by the framework outperformed other benchmark algorithms such as Differential Evolution(DE) and Particle Swarm Optimisation(PSO).

Alrawahi et al. [108] provided an optimisation solution that manages the QoS when implementing resource allocation in the Cloud of Things (CoT). Resource allocation in CoT was viewed as an optimisation problem where a dynamic and generic QoS model was needed to allocate resources to emerging CoT applications efficiently. In addressing this problem, a

QoS model was implemented to optimise five different QoS objectives. The overall QoS objective is to minimize the cost, energy consumption and response time while maximizing the resource coverage and fault tolerance. The QoS model main goal was to optimally allocate a set of resources provided by multiple providers to multiple consumers while satisfying the required QoS level collectively. The QoS model uses three optimisation algorithms which include: the improved Strength Pareto Evolutionary Approach (SPEA2), the multiobjective evolutionary algorithm based on decomposition (MOEA/D), and multiobjective indicator-based evolutionary algorithm (IBEA), to achieve this goal. The simulations conducted to evaluate the proposed QoS model show that the model was able to generate at least one optimal solution for both single-objective and multi-objective resource allocation problems.

Li et al. [109] developed a QoS-aware service discovery framework that can effectively locate trustworthy services based on the QoS demands and the dynamic context requirements. The developed framework is decentralised as it is structured as a distributed peer-to-peer architecture. The framework is underpinned by two discovery schemes which allow the framework to be trust-assured, robust and scalable. The first discovery scheme uses social trust to provide a scalable trust-based discovery scheme, while the second discovery scheme uses a peer-to-peer network to provide a trust-assured locality-preserving discovery scheme. Simulations studies on the framework demonstrate the ease of locating trustworthy services and the improvements in security and integrity it brings to service discovery frameworks.

Khanouche et al. [110] presented an energy-centred and QoS-aware selection algorithm that addresses the challenge of efficiently selecting services for the optimal management of both QoS and energy in the IoT service composition process. The service selection problem was modelled as both a multi-attribute and lexicographic problem with which an optimal solution is to be found. The optimal solution is required to satisfy the QoS requirements of a composite service while minimizing the energy consumption of the composite service. Consequently, the proposed service selection algorithm aimed at simultaneously ensuring high availability of composite service via the minimization of energy consumption and satisfying the user’s QoS requirements. In achieving this goal, the service selection algorithm first preselects services that offer the minimum QoS level specified by the users based on a lexicography optimisation strategy. The preselected services are then further evaluated based on the three factors(energy profile, user preferences and QoS attributes) that influences a service utility. The services with the best utility are then finally selected for the IoT service composition process. Results from the simulation carried out shows the promising performance of the proposed algorithm in terms of optimality, energy efficiency and selection time.

Skarlat et al. [111] developed a model that optimises the placement of IoT services on fog resources based on the QoS requirements of the IoT application. The QoS-aware service placement model considered the mapping of an IoT application request to the computation resources of a fog colony as a decentralized optimisation problem where the goal is to optimise the utilization of the fog landscape while satisfying the QoS requirements of the application. The model uses a number of constraints such as the deployment time of the application, the resources of a fog node, and the application's response time in determining the optimal placement of IoT services. The model's performance was evaluated using the iFogSim simulator, and the results showed a reduction in the execution cost and a good use of the fog landscape.

Chapter 4

REINFORCEMENT LEARNING QOS NEGOTIATION MODEL

In the management of QoS in IoT middleware, QoS negotiation is an important task in ensuring the successful execution of actuation tasks. However, given the underlying uncertainties and the dynamism in the negotiation environment, it raises the question of how best to manage the Service Level Agreement (SLA) of IoT services. In this chapter, a QoS negotiation strategy based on Reinforcement Learning (RF) is proposed to model the negotiation process with incomplete information in a way that increases its success rate and generates an SLA with a high utility. Reinforcement Learning is a Machine Learning (ML) paradigm where software agents learn by interacting (i.e., taking actions) within a dynamic environment, compared to other machine learning paradigms such as supervised learning and unsupervised learning [65]. While both supervised and unsupervised learning are data-driven (i.e. they learn from the input data), reinforcement learning focuses on goal-directed learning from interaction with the environment. Reinforcement learning formalises the agents' interaction using Markov Decision Process(MDP) [13]. The concept of RF can maximise the chances of generating a high utility SLA in a dynamic environment. By modelling the negotiation process as an MDP, negotiating parties can optimally make decisions based on the current state of the IoT environment that will lead to the generation of acceptable offers within the specified deadline.

4.1 QoS NEGOTIATION ENVIRONMENT

The IoT middleware provides the negotiation environment that facilitates the negotiation process that attempts to resolve Quality of Service (QoS) contentions between heterogeneous devices with different preferences. These IoT devices are usually represented by *software agents* in the negotiation environment. The IoT middleware is typically designed to be adaptive so that the changes in the physical world is reflected in the negotiation environment. The variations in network connectivity, changes in the application-level or contexts, declining battery level of IoT devices and variations in the workload of the CPU of the IoT edge node triggers the IoT middleware to evolve. To ensure users' satisfaction and the effectiveness of IoT middleware, the IoT middleware dynamically adapts itself to fit into these variations. Consequently, the QoS parameters, negotiation deadline and negotiation resources are dynamic, and the software agents interact with each other under these uncertain conditions to reach an agreement, as seen in Figure 4.1.

4.1.1 Software Agents

A software agent is a software program situated in some environment and capable of flexible, autonomous action in that environment to meet its design objectives [66]. It can autonomously perform a specific task for a user and profitably interact with its environment. A software agent can operate without the direct intervention of humans and has control over its action and internal state. A software agent is capable of perceiving its environment and reacting appropriately according to the changes observed in the environment. This means that a software agent is required to have its own internal model of its environment from which it can respond to the changes that occur in the environment. Apart from being reactive to changes in the environment, a software agent can exhibit opportunistic, goal-directed behaviour by taking actions that equally change the state of its environment [67].

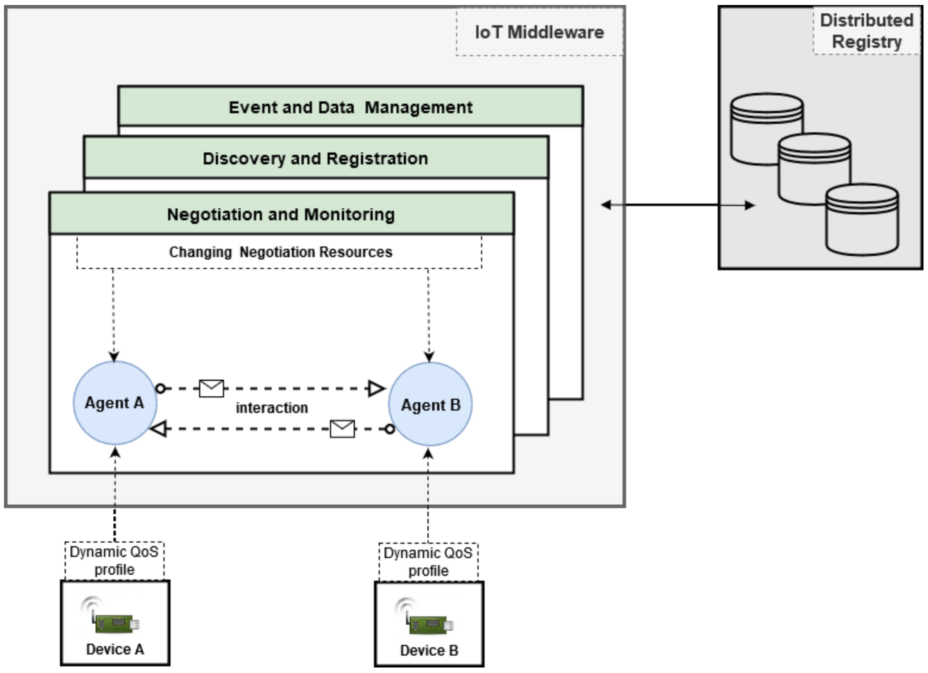


Figure 4.1: A schematic representation of agents interacting in a dynamic environment

The QoS negotiation model uses software agents to represents IoT service providers and consumers. It assumes the existence of a QoS preference gap in any given negotiation scenario, as shown in Figure 4.2. These agents negotiate over a set of negotiable QoS parameters in the IoT middleware environment characterized by several uncertainties. The QoS preference gap contains an agreement zone that can change during the negotiation process, and the negotiation agents are unaware of its location or presence. The negotiating agents do not have any prior knowledge of the environment as the complete information about the state transitions in the environment is unknown. Also, the negotiating agent's agents are self-centred and secretive. This means that they are interested in maximizing their preferences, and they do not disclose their exact preferences to other negotiating agents. As a result, they do not know other agents' preferences but can only observe actions taken previously by other agents.

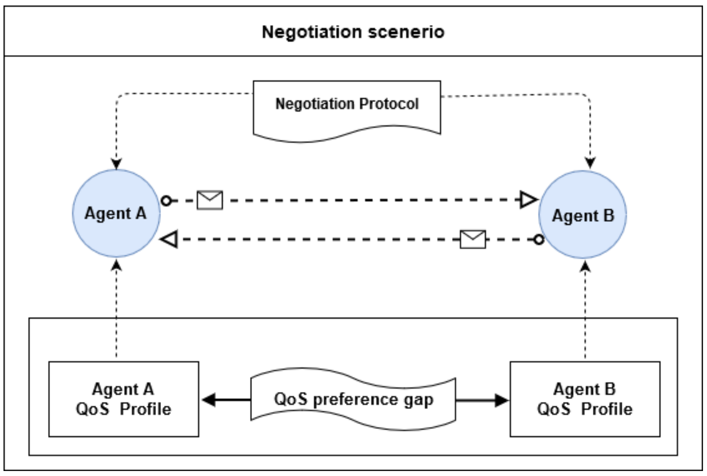


Figure 4.2: A schematic representation of a negotiation scenario with a dynamic QoS preference gap

These uncertainties demand that the negotiating agents learn about their environment to take actions to attain their objectives. Using reinforcement learning, the proposed QoS negotiation strategy allows agents to learn from experience by choosing the most suitable action at each stage of the negotiation process based on the changes observed in the negotiation environment that will achieve its long-term objective. The goal of the negotiating agents is to reach a high utility agreement before the designated deadline. Given the proactive behaviour of software agents and the well-defined goal of the negotiation process, the negotiating agents are capable of taking initiatives under specific circumstances to influence their environment to achieve this goal.

4.1.2 Utility Function

The negotiating agents are utility-based as they use the utility function from macroeconomics to map each offer to a utility value that represents the degree of preference [68]. The utility function maps offer to a real number [0,1], where 0 is the minimum value and 1 is the maximum utility. The proposed QoS negotiation strategy allows negotiating agents to use the utility function to generate offers and evaluate counter-offers. Agents take turns in a making offer in each round in the set {r = 0,1…$r_{deadline}$}.An offer contains an n number of negotiable QoS parameters, and each QoS parameter can take a value of (q_n) within its range of permissible values ($q_n^{preferred}$…. $q_n^{reserved}$) in the QoS profile. To model the non-linear changes associated with the QoS parameters of IoT services, the negotiating agents use the general exponential utility function to map each QoS parameter value to a utility value. For a QoS parameter whose $q_n^{preferred}$ value is less than the q_n^{reserved} value for a negotiating agent, the utility value is computed as:

$$U_1(q_n)\frac{e}{e-1} \times (e^{-q_n} - e^{-1}) \qquad (4.1)$$

For a QoS parameter whose $q_n^{preferred}$ value is greater than the q_n^{reserved} value for a negotiating agent, the utility value is computed as:

$$U_2(q_n)\frac{1}{e-1} \times (e^{-q_n} - 1) \qquad (4.2)$$

With the utility value of each parameter defined, the proposed QoS model assumes that the utility of each of the QoS parameter is linearly addictive. This means that the utility value of an offer can be computed as the weighted sum of the individual QoS parameter's utility and is defined as:

$$\text{U(f)} = \sum_{n=1}^{n} w_n \times U_i\,(q_n) \qquad (4.3)$$

where U(f) is a real number (0≤ $U(f) \leq 1$)

where i=1, when the $q_n^{preferred}$ value is less than the $q_n^{reserved}$ value

where i=2, when the $q_n^{preferred}$ value is greater than the $q_n^{reserved}$ value

where w_n = normalized weight for each QoS parameter. The sum of the normalised weights is expressed as:

$$\sum_{n=1}^{n} w_n = 1 \qquad (4.4)$$

To illustrate the utility space of an IoT service provider with an example, consider a utility-based negotiating agent representing the service provider whose QoS parameters values are in the range[0.1, 1.0] and is linearly addictive. During a negotiation scenario, the negotiating agent is characterized with the following QoS preferred parameters values and weights for response time(0.86,20%) , availability(0.74, 35%) and throughput (0.77, 45%) as seen in Figure 4.3. Using equation 4.1, the utility value for response time, availability and throughput is computed as 0.087, 0.173 and 0.150, respectively. The resulting utility value of the offer made by the negotiating agent is then calculated using equation 4.3 and is given as 0.145.

Given that a negotiating agent is willing to substitute the utility value of a QoS parameter for another and the permissible range prevents them from going out of bounds, this justifies why the QoS negotiation model uses a weighted sum function to compute the utility value of an offer containing multiple QoS parameter.

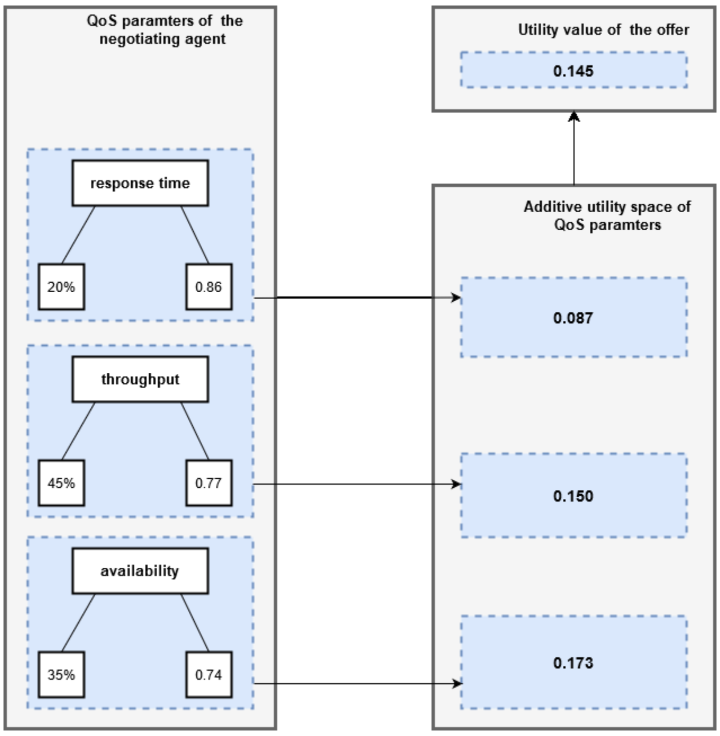

QoS paramters of the negotiating agent
response time
20%
0.86
throughput
45%
0.77
availability
35%
0.74
Utility value of the offer
0.145
Additive utility space of QoS paramters
0.087
0.150
0.173

Figure 4.3: An example of a linearly additive utility space of a negotiating agent

4.2 QoS NEGOTIATION MODEL COMPONENTS

The QoS negotiation model comprises three elements: QoS *Profile, Negotiation Protocol and Negotiation Strategy*, and considers the peculiarities of IoT services. The QoS profile component takes into account the attributes of IoT services associated with the physical world as indicated in the IoT service model described in Section 3.1.2. The negotiation protocol recognises the need for multilateral negotiation in an IoT environment, and the negotiation strategy component models the dynamic behaviour of IoT services.

4.2.1 QoS Profile

The QoS profile defines the non-functional attributes of IoT services, and it is expressed as the QoS constraints for a service provider and the QoS requirements for a service consumer. It specifies both the domain-independent attributes such as availability, response time and throughput,[33] and the domain-dependent attributes, which include the service coverage and available time [69]. The QoS parameters (i.e. the domain-independent attributes) are the non-functional attributes of an IoT service over which agents negotiate over their values. The negotiation space of a QoS parameter for a negotiating agent can be expressed as :

$$\Omega_{\mathrm{q}}^{\mathrm{na}}= \{q^{preferred} , q^{reserved}\} \qquad (4.5)$$

When $q^{preferred}$ is the maximum value, and $q^{reserved}$ is the minimum value, it means that the higher the QoS parameter value, the better it is for the negotiation agent. Similarly, when $q^{preferred}$ is the minimum value and is the maximum value, it means that the lower the QoS parameter value, the better it is for the negotiation agent. Consequently, the negotiation space of an agent's offer can be expressed as :

$$\Omega_{\mathrm{p}}^{\mathrm{na}}= \{\mathrm{U}(\mathrm{f}_{\mathrm{pd}}) , \mathrm{U}(\mathrm{f}_{\mathrm{rd}})\} \qquad (4.6)$$

where $U(f_{pd})$ represents the utility value of the preferred offer

where $U(f_{rd})$ represents the utility value of the reserved offer

With the negotiation space defined, negotiating agents can express their preferences using equation 4.1, 4.2 and 4.3. The service coverage attribute specified in the QoS profile represents the spatial features of IoT services, and it is model as a circle. This attribute indicates the observation area of the service provider. The negotiation space of the service coverage attribute for the service provider is expressed as:

$$\Omega_{\mathrm{loc}}^{\mathrm{p}}= \{\mathrm{loc}_1^{\mathrm{p}}, \mathrm{loc}_2^{\mathrm{p}}, \mathrm{loc}_3^{\mathrm{p}} \ldots\mathrm{loc}_{\mathrm{n}}^{\mathrm{p}}\} \quad (\mathrm{n}\geq 1) \qquad (4.7)$$

where loc_n^p represents the service area [loc_{centre}, loc_{radius}].

The service consumer uses the service coverage attribute to specify its actuation area or requested area for operation. The negotiation space of the service coverage attribute for the service consumer is expressed as:

$$\Omega_{\mathrm{loc}}^{\mathrm{c}}= \{\mathrm{loc}^{\mathrm{c}}, \mathrm{d}\} \qquad (4.8)$$

where loc^c represents the requested area and d represents the threshold distance.

The evaluation of service coverage is indicated by the distance between the requested area of service consumer i and the service area of service provider j and is computed as:

$$\mathrm{U}(\mathrm{sc}_{\mathrm{ij}})=\begin{cases} 1-\frac{dist(i,j)}{r}, & if\ dist(i,j) < r \\ 0, & if\ dist(i,j) \geq r \end{cases} \qquad (4.9)$$

where dis(i,j) is the geographical distance between the requested area and the centre of the service area and r is the radius of the service area.

Equation 4.9 shows that a service provider with a service area closer to the requested area has a higher service coverage preference. The available time attribute in the QoS profile represents the temporality of IoT services. For the service provider, it indicates the period its service is available for consumption. For the service consumer, it denotes the time and duration for the consumption of the requested service. The negotiation space of the available time attribute for the service provider is expressed as:

$$\Omega^{p}_{AT} = \{AT^{p}_{1}, AT^{p}_{2}, AT^{p}_{3} \ldots AT^{p}_{n}\} \quad (n \geq 1) \tag{4.10}$$

where AT^{p}_{n} represents the range of available times [at_{start} , at_{end}].

The temporality negotiation space specifies the list of available times from which the requested time of the service consumer is matched with. The evaluation of the available time is indicated by the degree at which the requested time of service consumer i matches with the available time of service provider j. The matching degree can be based on the time-dependent matching function or the duration-dependent matching. The time-dependent matching function is given as:

$$U(AT_{ij}) = \begin{cases} 1, & \text{if } R_{time} \subset AT_n \\ \frac{|AT_n|}{|R_{time}|}, & \text{if } R_{time} \cap AT_n \\ 0, & \text{if } R_{time} \cap AT_n = \emptyset \end{cases} \tag{4.11}$$

where R_{time} denote the requested time [rt_{start} , rt_{end}].

The duration-dependent matching function is given as :

$$U(AT_{ij}) = \begin{cases} 1, & \text{if } R_{dur} > AT_{ndur} \\ \frac{AT_{ndur}}{R_{dur}}, & \text{otherwise} \end{cases} \tag{4.12}$$

where R_{dur} and AT_{ndur} represents the request duration and Available time duration respectively

As illustrated in equation 4.11 and 4.12, there is a complete match between an available time and the requested time, when AT_{ij} equals to1. It is noteworthy to state that the domain-dependent attributes specified in the QoS profile are not negotiable IoT service attributes, hence they are used for the service selection process.

4.2.2 Negotiation Protocol

The QoS negotiation model adopts the turn-taking negotiation protocol called the Stacked Alternating Offer Protocol (SAOP) that allows negotiating agents to evaluate offers and take the desired action [70]. This protocol was chosen because of its support for bilateral and multilateral negotiations and its low communication cost. SAOP uses *rounds* to organise the negotiation process as each negotiating agents is allocated a turn in each round to take action. The negotiation process begins with a negotiating agent (i.e. the negotiating agent representing the service consumer) making an offer which is observed by the other negotiating agents. The negotiating agent assigned to take action can either accept the offer, reject the offer and provide a counteroffer or terminates the negotiation. The turn-taking process sequence is repeated until a termination condition is met. A negotiation is terminated when the specified deadline is reached or an agreement is found or a negotiating agent withdraws from the negotiation. However, in the proposed QoS model, agents are not allowed to withdraw from the negotiation process.

In the QoS negotiation model, SAOP is formally defined by a tuple { AG, AT, RS }, where *AG* represents the set negotiating agents, *AT* represents the set of possible actions that can be taken by the negotiating agents, and *RS* defines the rules that characterise the interaction between agents and are as follows:

Rule 1: Each negotiating agent is assigned to precisely one turn per round. This guarantees fairness as it ensures that none of the negotiating agents gets more than one turn in a turn-taking sequence. The turn-taking sequence is denoted by:

$\text{TurnSeq} = \text{Agt}^{|\text{Agt}|}$ is the sequence of agents such that

- $\forall s \in \text{TurnSeq} \;\; \forall a \in \text{Agt}, \;\; \exists i \in N^+, i \leq |s|$ such that $s_i = a$ and
- $\forall s \in \text{TurnSeq} \;\; \forall i, j \leq |s| : \; s_i = s_j \rightarrow i=j$

Rule 2: Negotiation agents can only take action that is permitted at that moment in their turn. The function *action* indicates the actions agent take and is expressed as :

$$\text{action} : \text{Agt} \times R \rightarrow \text{Act}.$$

where action(a; r) represents the action agent $a \in Agt$ took in round $r \in R$.

Similarly, the function *allowedAction* indicates the actions agents are permitted to take per turn *t* at a given round *r* and is given as :

$$\text{allowed Act}(r,t) = \begin{cases} offer \cup \{deadline\}, & if\ cont(r,t)\ \ t=1\ \ r_1 = 1 \\ offer \cup \{accept, deadline\}, & if\ cont(r,t)\ \ (t \neq 1\ \ r_1 \neq 1) \\ \emptyset, \quad otherwise & \end{cases} \quad (4.13)$$

Rule 3: The predicate *cont* determines whether to continue the negotiation after the current round *r* and turn *t*. Its value is defined by:

$$\forall r \in R \;\; \forall t \in N^+ : \text{cont}(r,t) \leftrightarrow \neg d(r,t) \wedge \neg \text{agr}(r,t) \quad (4.14)$$

where d(r,t) represents whether or not the negotiation deadline has elapsed and is given by:

$$\text{time-based deadline} : d(r,t) \leftrightarrow \text{currenttime} - \text{negostarttime} \geq \text{maxnegotime} \quad (4.15)$$

$$\text{round-based deadline}: d(r,t) \leftrightarrow \text{currentround} \geq \text{maxnegoround} \quad (4.16)$$

where agr(r,t) represents whether or not an agreement has been found and is given by:

$$d(r,t) \leftrightarrow \text{action}(s_{prev_2^{|Agt|-1}(r,t)}, prev_1^{|Agt|-1}(r,t)) \in \text{offer} \wedge \forall 0 \leq i \leq |\text{Agt}| -2:$$

$$\text{action}(s_{prev_2^i(r,t)}, prev_1^i(r,t)) = \text{accept} \quad (4.17)$$

Rule 4: The function outcome determines the outcome of a negotiation and is expressed as:

$$\text{Outcome}(r,t) = \begin{cases} undefined, & cont(r,t) \\ fail, & \neg cont(r,t) \wedge \neg agr(r,t) \\ success, & t > 0 \wedge \neg cont(r,t) \wedge agrOffer(b,r,t) \end{cases} \quad (4.18)$$

where agrOffer(b, r,t) represents the offer that was accepted and is given by:

$$\text{agrOffer}(\text{action}(s_{prev_2^{|Agt|-1}(r,t)}, prev_1^{|Agt|-1}(r,t))r,t) \leftrightarrow cont(r,t) \wedge ag(r,t) \quad (4.19)$$

The above rules form the fundamental framework for the multilateral turn-taking negotiation as It does not make use of a mediator. This approach allows negotiating agents to keep their preference private rather than exposing them to a third party.

4.2.3 Negotiation Strategy

The proposed negotiation strategy is designed to allow negotiating agents to use any negotiation tactic that adopts the utility function in the generation of offers. The common utility function-based negotiation tactics are the *concession* and *tradeoff* negotiation tactics [58]. The concession tactics involve a negotiating party making an offer with a lower utility value for every new offer it receives. The received offer typically has a high utility value for the negotiating agent that made the offer. If the utility value of the offer a negotiation party receives is higher than the utility value of the offer it intends to make, then it accepts the offer. Otherwise, it rejects the offer by making a counter-offer as shown in the equation below:

$$P^{a}(t`, q_{a\leftarrow}^{t}) = \begin{cases} accept \quad if\ U(q_{a\leftarrow}^{t}) \geq U(q_{a\rightarrow}^{t`}) \\ \quad otherwise \\ \quad offer \ \ q_{a\rightarrow}^{t`} \end{cases} \tag{4.20}$$

where,

t`= time agent a should send the next offer

t= time agent a received an offer for evaluation

$U(q_{a\leftarrow}^{t})$ *= the utility value of the offer agent a received for evaluation at time t*

$U(q_{a\rightarrow}^{t})$ *= the utility value of the offer agent a should make at time t`*

The concession tactic is based on a time-dependent function that allows negotiating agent to concede a certain amount of utility as the negotiation deadline approaches. The value of a QoS parameter q_n to be uttered by a negotiating agent a is modelled as an offer at time t with $0 \leq t \leq t_{max}^{a}$, by a function α^{a}(t), which is expressed mathematically as :

$$q_{a\rightarrow}^{t}[n] = \begin{cases} q_{a[n]}^{min} + \alpha^{a}(\text{t})(q_{a[n]}^{max} - q_{a[n]}^{min}), & if \ \ U_{[n]}(q_n) \ \ is\ decreasing \\ q_{a[n]}^{min} + (1 - \alpha^{a}(\text{t}))(q_{a[n]}^{max} - q_{a[n]}^{min}), & if \ \ U_{[n]}(q_n) \ \ is\ increasing \end{cases} \tag{4.21}$$

The exponential function detailed in [71] is used to compute the value of α^{a}(t) as shown in the equation below:

$$\alpha^{a}(\text{t}) = e^{(1-\frac{\min(t,t_{max})}{t_{max}})^{\beta}} \ln k \tag{4.22}$$

The time-dependent function is parameterized by the value of β, which determines how quickly an agent concedes up to the reserved value of the QoS parameter. Figure 4.4 shows a simple graphical representation of the concession negotiation tactic. Without any loss of generality, the two-dimensional space x,y is assumed to represent two QoS parameter, and the utility function is nonlinear and additive as described in Section 4.1.2. The curves l_1 and l_2 represent the indifference curve of an agent most preferred offer and its counteroffer, respectively, and curve l_3 represents the most preferred offer by the negotiating counterpart. If point A is the agent's initial offer and point B is its counter-offer, then the agent concedes by reducing its utility by a value of δ as it moves from point A to point B since it is closer to l_3, i.e. |BB'| < |AA'|

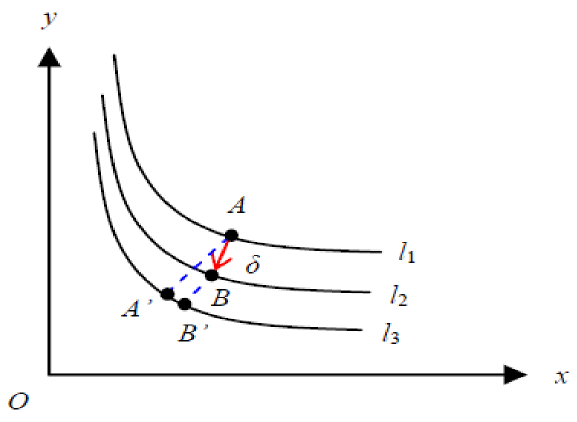


Figure 4.4: The concession negotiation tactic

In the trade-off negotiation tactic, a negotiating agent attempts to keep the utility of its offer stable while generating an offer that will be acceptable for its negotiating opponent. The negotiating agent increases the values of some QoS parameters while decreasing the values of other QoS parameters to maintain the desired utility (aspiration level). Usually, a negotiation party has no information about its negotiating counterpart's utility function and preferences, and the trade-off tactics allow negotiating agents to approximates the opponent preferences using the fuzzy similarity techniques to increase the opponent's utility [72]. Consequently, a negotiating agent can simultaneously maintain its aspiration level while maximizing the probability of an offer been accepted.

Offers with the desired utility value are mapped to an iso-curve to represent the aspiration level of an agent, as shown in the equation below:

$$\mathrm{iso}_a(\theta) = \{q^t_{a\to} \mid U_{[n]}(q^t_{a\to}) = \theta\} \tag{4.23}$$

An agent is now required to select the most similar offer from the set of offers using a similarity function based on a certain criteria. The similarity function between two values: *x, y,* based on criteria *h* is expressed as:

$$\mathrm{Sim}_h(x, y) = 1 - |h(x) - h(y)| \tag{4.24}$$

For multiple criteria, a weighted mean's method is used to aggregate the individual similarities. Thus the similarity between two values: x_j, y_j over a set of *m* criteria is given as follows:

$$\mathrm{Sim}_h(x_j, y_j) = \sum_{1\le i\le m} w_i \times (1 - |h_i(x_j) - h_i(y_j)|) \tag{4.25}$$

Formalizing the trade-off tactic, given an offer *x* made by agent *a* and a subsequent offer *y* received from another agent, agent *a* formulates the trade-off tactic the following way:

$$\text{trade-off}_a(x, y) = \arg\max_{z\,\in\,\mathbf{iso_a}(\emptyset)} \{\mathrm{Sim}(z, y)\} \tag{4.26}$$

Figure 4.5 depicts a simple graphical representation of the tradeoff negotiation tactic. Similar to the concession negotiation tactics, the two-dimensional space:x, y is assumed to represent two QoS parameters, and the utility function is nonlinear and addictive. The curve l_1 and l_2 represent the indifference curve of an agent's offer that indicates its aspiration level and that of its counterpart, respectively. Point A and B correspond to the agent's initial offer and counteroffer. When the agent makes a tradeoff, the utility of the offers stays the same as it moves from point A to point B along curve l_1 but is closer to curve l_2 since |AA'| > | BB'|. This approach ensures that the agent progresses towards the preferences of its opponent with no reduction in the utility of its generated offer.

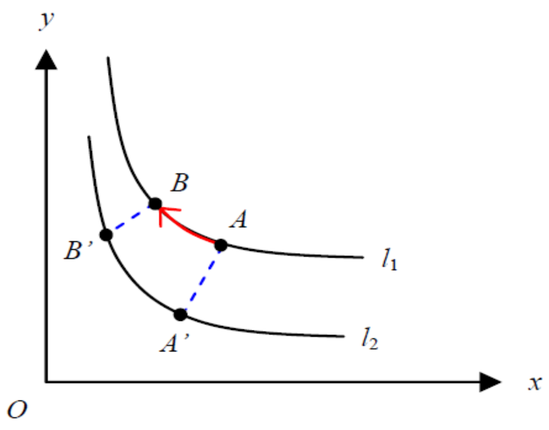


Figure 4.5: The tradeoff negotiation tactic

With the two established negotiation tactics, a negotiation strategy is developed to capitalize on the strength of both negotiation tactics that will balance the success rate and social welfare. The proposed negotiation strategy considers the dynamic behaviour of IoT services and the uncertainties in the negotiating environment, thus improving the general performance of the negotiation process. The negotiation strategy is based on model-based reinforcement learning, and it enables negotiating agents to decide which negotiation tactic to use for each step in the negotiation process that will maximize their utility and reach an agreement before the deadline elapses.

4.3 THE REINFORCEMENT LEARNING APPROACH

The machine learning-based negotiation strategy aims to enable negotiating agents to determine the best course of action, which will result in an agreement that maximises the agents' utility function. In practical terms, this equates to the negotiating agents strategically choosing either the concession tactic or the trade-off tactic based on the current negotiation state for the generation of offers that maximises the chances of reaching an agreement with high social welfare within the specific deadline. As stated in Section 4.2.3, the concession tactic enables an agent to generate offers that are of lower utility value to the offer received. This approach makes negotiating agents reach an agreement quicker but at a lower social welfare. Selecting the trade-off tactic allows an agent to generate attractive offers to other agents while maintaining its desired utility value. However, this negotiation tactic often leads to many negotiation failures as its approximations of the opponents' preferences can be less appealing. Consequently, we proposed the *Reinforcement Learning Negotiation Strategy(RLNS)* that combined both negotiation tactics in a way that improves the social welfare and success rate of a negotiation outcome. RLNS uses the context negotiation information such as the agent's current negotiation state and deadline criterion to decide whether to use the trade-off tactic or the concession tactic in the generation of a new offer at a particular instant.

4.3.1 Modelling the QoS Negotiation

The dynamism that characterises the negotiation environment necessitated the modelling of the QoS negotiation as a Markov Decision Process (MDP) as agents are required to make decisions under these conditions. MDP presents a standard formalism to describe multistage decision making in a dynamic environment [13]. It is a discrete-time stochastic control process that provides a mathematical framework for modelling decision-making in an environment that changes state randomly in response to action choices made by a decision-maker, commonly referred to as an *agent*. A finite number of states and actions are assumed in a Markov Decision Process. Each time the agent observes a state and takes action, it incurs intermediate costs that must be kept to a minimum (or, in the inverse scenario, rewards to be maximized). The cost and successor state are solely determined by the current state and the action taken. Based on the uncertainty of the environment where the interaction takes place, the state transition is typically probabilistic [123].

MDP extends the ***Markov Chain*** **(MC)** since a sequence of actions and maximizing rewards defines its control process. A Markov Chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event [124]. It is characterised by a set of states and transition probabilities. The combination of actions and rewards distinguishes Markov Decision Processes from Markov Chains. An MDP reduces to an MC if each state has only one action and all the rewards are the same.

In MDP problems, an *agent* interacts with the environment by selecting actions to which the environment responds by presenting a *reward* and a new *state* to the agent. The goal of the agent is to maximise the cumulative rewards through the series of actions it takes. Consider a mobile robot that decides whether to look for more trash in an office building or begin to find its way back to a battery recharging station. It makes this decision based on the current charge level of its battery and how quickly and easily it can find its recharging station. Given that the robot does know where the trashes are, each trash collected by the robot counts as a positive reward. However, the robot receives a negative reward when its battery has been depleted, and it needs to be manually taken to a recharging station. The goal of the robot becomes how to gather as much trash as possible without having to be rescued. In order to effectively achieve this goal, the robot will have to monitor its environment (i.e. the current charge of its battery and its location for the nearest charging station) and take actions (i.e. continuing searching for trash or find its way to the nearest charging station) appropriately. Since the effect of the robot actions cannot be fully predicted (searching for a trash can either deplete its battery or enable it to collect more trash), it is necessary for the robot to monitor its environment for it to make the correct choice of actions.

The above example illustrates an active decision-making agent interacting with its environment as it seeks to achieve a goal despite the uncertainty about its environment. Figure 4.6 shows the agent-environment interaction in a Markov Decision Process.

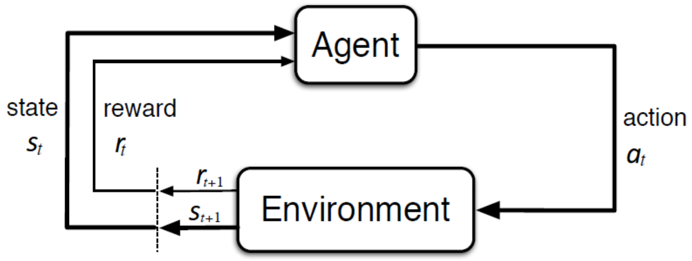


Figure 4.6: The The agent–environment interaction in a Markov decision process

Specifically, the interaction between the agent and the environment occurs in a sequence of discrete- n time steps, t=0,1,2,3…. T, where T is the final time step. For each time step t, the agent is presented with the environment's state $s_t \in S$, from which the agent is required to choose an action, $a_t \in A$. Due to the action selected, the agent receives a reward $r_{t+1} \in R$, and it's being presented with a new state $s_{t+1} \in S$. This process results in a sequence as seen below:

$$s_o, a_o, r_1, s_1, a_1, r_2, s_2, a_2, r_3 \ldots\ldots\ldots\ldots s_T \qquad (4.27)$$

Sutton and Barto [65] formally define a finite MDP as the tuple (S, A, P, R, γ) where:

- S is the state- space which contains a set of finite states with an initial state of s_0 and a final state of s_T.
- A is the action-space which, contains a set of finite actions.
- P is the dynamic function that defines the dynamics of MDP, where $P(s'|s, a)$ is the probability that action a∈A executed in state s∈S will transition to the next state s′∈S. The state-transition probabilities can be expressed as:

$$P(s'|s,a) \doteq \Pr\{s_t = s' \mid s_{t-1} = s, a_{t-1} = a\} = \sum_{r \in \mathrm{R}} p(s', r \mid s, a) \qquad (4.28)$$

- R is the reward function, where $R(s'|s,a)$ is the immediate reward an agent receives by executing action a∈A in state s∈S and is transitioned to s'∈S. It is expressed as :

$$R(s'|s,a) \doteq E\,[r_t \mid s_{t-1} = s, a_{t-1} = a, s_t = s'] = \sum_{r \in \mathrm{R}} r \frac{p(s', r \mid s,a)}{p(s' \mid s,a)} \qquad (4.29)$$

- γ is the discount rate that balances the trade-off between immediate rewards and future rewards and is used in generating the discounted reward. The sum of the discounted reward is given by:

$$G_t \doteq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{k=T} \gamma^k r_{t+k+1} \qquad (4.30)$$

The objective in a standard MDP is to find the optimum policy (π*) that yields the maximum sum of discounted rewards over a given period. A policy(π) is the mapping of states to the probability of selecting each possible action.

Based on these key concepts of MDP, we model the QoS negotiation as a set of n MDPs. We have n processes with each agent having its own view of the dynamics of the negotiating environment. The MDP inspired negotiation process is characterized by the following:

- **Discrete state-space:** The negotiation state-space is defined by the availability of resources for the negotiation process, negotiation deadline and the reserved offer. To represent the fact that each of the elements of the negotiation state could change due to the dynamics of the IoT environment, they are modelled as discrete finite sets. Each element can take two values resulting in eight different states as seen in the negotiation state set below:

$$S=\{\{r_h,r_l\},\{d_l,d_s\}\ \{u_f,u_c\}\} \qquad (4.31)$$

where,

r_h indicates that the available resources are high

r_l indicates that the available resources are low

d_l indicates that the available negotiation time is large

d_s indicates that the available negotiation time is small

u_f indicates that the offer received is far from the agent's reserved utility

u_c indicates that the offer received is close to the agent's reserved utility

- **Discrete action-space:** This is defined by the set of negotiation tactics agents can choose from in generating an offer. It is modelled as a discrete set and is expressed as :

$$A= \{c,t\} \tag{4.32}$$

where,

c is the concession negotiation tactic

t is the trade-off negotiation tactic

- **Transition function:** The transition function models the uncertainty in the negotiation environment. Given that the changes in the dynamics of the environment (i.e. the specific probabilities of the state transition) are exactly not known, they are estimated by the transition function. The transition function specifies the estimated probability distribution of the negotiation state transitions and is defined as :

$$P(s'|s,a): \rightarrow [0,1] \tag{4.33}$$

The required numbers to accommodate all the probability distribution is given by:$|S|^2 x|A|$. The eight different states and the two supported actions in each of these states give rise to a total of 128 state transitions, as seen in Appendix A.3.

- **Reward function:** Agents are rewarded based on the negotiation tactic chosen at a given state. An agent is highly rewarded if it selects the trade-off strategy when there is sufficient time and resource for the negotiation process and the offer received is far from the agent's reserved utility. Similarly, an agent is highly rewarded for choosing the concession strategy if the time and resources for the negotiation process are running out and the offer received is close to the agent's reserved utility. This is illustrated in the reward scheme as seen in Appendix A.3, with $r_1 > r_2 > r_3$

- **Fixed discount rate:** To ensure that the selection of the appropriate negotiation tactic is not based on the immediate reward that the agent receives but that the agent considers all possible future rewards, the discount rate value is fixed and is governed by the following expression:

$$\gamma \in [0,1] \tag{4.34}$$

Table 4.1 illustrates a snippet of the complete dynamism of the negotiation process with the transition probabilities and expected rewards as captured in Appendix A.3. Each row represents a possible combination of the current state, action and the next state. Each state transition has a probability of occurrence with a specific reward. The transition probabilities of a specific negotiation state with a particular action always sum to 1.

Table 4.1: Dynamics of the negotiation process as a finite Markov Decision Process

Current state (s)	Negotiation tactic (a)	Next state (s')	Transition scheme P(s'|a,s)	Reward scheme R(s'|s,a)
(r_l,d_l,u_c)	trade-off	(r_l,d_l,u_f)	α_1	$r_1 + r_3 + r_2$
(r_l,d_l,u_c)	concession	(r_l,d_l,u_f)	β_1	$2r_1 + r_3$
(r_l,d_l,u_f)	trade-off	(r_l,d_l,u_c)	α_3	$2r_1 + r_3$
(r_l,d_l,u_f)	concession	(r_l,d_l,u_c)	β_2	$r_1 + r_3 + r_2$
(r_l,d_s,u_c)	trade-off	(r_h,d_s,u_f)	α_2	$2r_2 + r_3$
(r_l,d_s,u_c)	concession	(r_h,d_s,u_f)	β_2	$3r_1$
(r_h,d_s,u_f)	trade-off	(r_l,d_s,u_c)	α_4	$2r_1 + r_3$

4.3.2 Reinforcement Learning Negotiation Strategy.

During QoS negotiation, agents have no information about other agents' utility function, the negotiation tactic used in generating offer and the exact probability of the state transitions. These uncertainties necessitated the development of a negotiation strategy that uses Reinforcement Learning (RL), as it is known as the model choice for making an optimal decision under uncertainties [73]. Reinforcement Learning is a Machine Learning (ML) paradigm that uses the formal framework of Markov decision processes (MDP) to define the interaction between an agent and its environment.

The reinforcement learning negotiation strategy aims at enabling negotiating agents to strategically choose either the concession negotiation tactic or the trade-off tactic for the generation of offers that maximises the chances of reaching an agreement with high social welfare within the specific deadline. In making this decision, an agent first considers the factors affecting the negotiation state space, the received offer, the immediate rewards and all possible future rewards before choosing the most suitable negotiation tactic. Given the uncertainties in the IoT environment, selecting the appropriate negotiation tactic at each step of the negotiation process requires the discovery of the optimal policy (π^*). The optimal policy is the sequence of negotiation tactics that yields the maximum sum of discounted rewards over a given period. To compute the optimal policy, the *value iteration* method is used in estimating the optimum policy [74]. The value iteration method was selected because it is not computationally expensive and uses less time to compute the optimal policy.

The first step in determining the optimal policy is by formalising the *state value-function,* $v_\pi(s)$ and the *action value-function* $q_\pi(s, a)$. The state value-function defines the expected cumulative reward for an agent beginning at a particular state s and under a specific policy π [75]. Formally, the state value-function of a state s under a policy π at the nth time-step during the negotiation process is defined as:

$$v_\pi(s) \doteq E_\pi[G_t \mid s_t = s] = E_\pi\left[\sum_{k=0}^{k=T} \gamma^k r_{t+k+1} \mid s_t = s\right], \text{ for all } s \in S \qquad (4.35)$$

where,

E_π is the expected reward value given that the agent follows policy π.

t is any time step with the value of the terminal state equating to zero

Similarly, the action value-function defines the expected cumulative reward for an agent that takes action a in state s under policy π and is expressed as:

$$q_\pi(s, a) \doteq E_\pi[G_t \mid s_t = s, a_t = a] = E_\pi\left[\sum_{k=0}^{k=T} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \qquad (4.36)$$

Amidst all the possible state value functions, it has been proven that there is always at least one policy(optimal policy) whose expected cumulative reward is higher than all the policies that exist in s ∈ S. The state value-function of the optimal policy is given by :

$$v_*(s) \doteq \max v_\pi(s) \qquad (4.37)$$

The optimal policy also shares the same optimal action value-function and is defined as:

$$q_*(s,a) \doteq \max q_\pi(s,a)\,, \quad \text{for all } s \in S \text{ and for all } a \in A \qquad (4.38)$$

The action-value function can be written in terms of the state-value function, resulting in the state-action pair(s,a). A state-action pair that follows an optimal policy is defined as

$$q_*(s,a) = E[r_{t+1} + \gamma v_*(s_{t+1}) \mid s_t = s, a_t = a] \qquad (4.39)$$

In computing the optimal policy, the value function is organised and structured to search for the optimal policy. The search for the optimal policy begins by computing the state-value function of an arbitrary policy, π and this given by:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} P(s' \mid s, a)\,[R(s'|s,a) + \gamma v_\pi(s')] \qquad (4.40)$$

where,

$\pi(a|s)$ is the probability of the action a taken in state s under policy π

After determining the state-value function $v_\pi(s)$ for an arbitrary policy, an evaluation is made to check whether it is better to select action a in state s (where $a \neq \pi(s)$), resulting in a different policy (π') or to follow the existing policy (π) in the next time-step. The action value-function of selecting action a is computed as:

$$q_\pi(s,a) = \sum_{s',r} P(s' \mid s, a)\,[R(s'|s,a) + \gamma v_\pi(s')] \qquad (4.41)$$

If $q_\pi(s, \pi'(s)) > v_\pi(s)$, it means that the policy π' has a higher expected cumulative reward than the policy π, and $v_{\pi'}(s) > v_\pi(s)$. Given a policy and its value function, a change in policy can be initiated and evaluated at a single state for a specific action. By extension, all states and all possible actions can be considered for changes in selecting an action in each state that seems to be the best based on $q_\pi(s,a)$. This consideration of a new policy π' is computed as:

$$\pi'(s) = \text{argmax}_a \sum_{s',r} P(s' \mid s, a)\,[R(s'|s,a) + \gamma v_\pi(s')] \qquad (4.42)$$

With a policy π, improved to π' using v_π, π' can equally be improved to π'' using $v_{\pi'}$. This process leads to a sequence of improving the policy and the value function until it converges to the optimal policy and optimal value function as shown below:

$$\pi_0 \rightarrow v_{\pi_0} \rightarrow \pi_1 \rightarrow v_{\pi_1} \rightarrow \pi_2 \rightarrow \ldots \pi_* \rightarrow v_*$$

This sequence leading to the discovery of the optimal policy can be computed using the value iteration algorithm shown below:

Algorithm 4.1 Value iteration for estimating the optimal policy

```
Input:  -Transition probability matrix, $P(s' \mid s, a)$
        -Reward scheme matrix, $R(s'|s, a)$
        -The discount rate, $\gamma$
Begin:
  Initialise v(s), for all s ϵ S
  loop:
    $\Delta \leftarrow 0$
    loop for each state s ϵ S
        $v \leftarrow v(s)$
        $v(s) \leftarrow \max_a \sum_{s',r} P(s' \mid s, a)\,[R(s'|s, a) + \gamma v_\pi(s')]$
        $\Delta \leftarrow \max(\Delta, |v - v(s)|$
  until $\Delta < \phi$
End
Output:  The deterministic optimal policy $\pi_*$, such that
        $\pi_*(s) = \text{argmax}_a \sum_{s',r} P(s'|\, s, a)\,[R(s'|s, a) + \gamma v(s')]$
```

Based on the negotiation environment state-space and the agent's opponent's offer, an agent is required to either accept the opponent offer or decide which negotiation tactic (concession or trade-off) to adopt to generate a counteroffer. An agent accepts an offer if the utility value of the received offer is far greater than the utility value of its reserved offer otherwise, a counteroffer is made.

Having defined the negotiation tactics for generating offers and the reinforcement learning method for computing the optimum policy, the reinforcement learning negotiation strategy is described as shown in algorithm 4.2. This strategy enables negotiating agents to appropriately map a negotiation tactic to a negotiation state resulting in the timely discovery of a QoS solution with high utility for all the negotiating participants.

Algorithm 4.2: Reinforcement Learning Negotiation Strategy

```
Input :   -The negotiating opponent offer ($Y_i$)
          -The deadline criterion
          - Array B with the best and worst values for n QoS parameters
          - Array C with the weights of n QoS parameter
```

- Array D with flags of n QoS parameter; A flag indicates if a QoS parameter preferred value is greater than its reserved value.

- The reward function R(s'|s, a)

- Parameter λ_1 and λ_2 ($0 < \lambda_1, \lambda_2$), indicating the degree of concession and trade-off

- The estimation function for the state transition, P(s'|a, s)

- The discount rate, γ

```
Begin:
 1.  Offer $Y_i$ is presented
 2.  while $Y_i$ is not accepted
 3.  a=value_iteration (R(s'| s, a), P(s'|a, s), $\gamma$,)
 4.  if a==concession then
 5.     $k_1 \leftarrow k_1+1$
 6.     $Y_{i+1} \leftarrow$ concession(B,C,D,$k_1$,$\lambda_1$)
 7.  else
 8.     $k_2 \leftarrow k_2+1$
 9.     $Y_{i+1} \leftarrow$ trade-off (B,C, D, $k_1$,$\lambda_2$)
 10. $k \leftarrow k_1+k_2$
 11. if $Y_{i+1}$ is out of bounds or deadline criterion is reached then
 12.    return FALSE
 13. else
 14.    offer $Y_{i+1}$ is presented
 15. return TRUE
End
```

Output: true if it is a success, otherwise false.

The proposed strategy algorithm begins in line 1 by allowing an agent to observe the negotiating participant's offer, which is usually an offer with a high utility for the agent's opponent. Given that the condition in line 2 is true, it proceeds to create a counteroffer in the while loop of lines 2-15 and return true in line 16 if the creation of the counteroffer is successful. In line 3, the negotiation strategy uses the value-iteration function to return an action (i.e. concession tactic or trade-off tactic) that maximizes the expected reward in the current negotiation state. To achieve this, it iteratively computes the state-value function for all the states to find the optimal policy for the current negotiation state. The concession function is invoked in line 6 when the action returned by that value iteration function in line 3 equates to the concession negotiation tactic. Similarly, the trade-off function is invoked in line 9 when the action returned by that value iteration function in line 3 equates to the trade-off negotiation tactic. The concession and trade-off function implement the concession and trade-off negotiation tactic of a QoS multi-parameter negotiation, respectively. The algorithmic descriptions of both the concession and trade-off strategy are presented in [58]. These tactics were chosen because the rate at which they converge for the generation of offers can be controlled. The variables k_1 and k_2 are used to count the number of times the concession and trade-off negotiation tactic function is invoked. In line 10, the variable k is used to count the total number of negotiation rounds.

Chapter 5

IoTQoSYSTEM DESIGN AND IMPLEMENTATION

This chapter presents the architectural components of the QoS negotiation framework, which includes the *IoTQoSystem* client and *IoTQoSystem* service. This chapter begins by providing an overview of the QoS negotiation framework. The overview discusses how the IoTQoSystem framework satisfies the objectives highlighted in Section 1.3, the decisions taken while developing the IoTQoSystem framework, and the justification of these decisions. This chapter describes the technology dependencies of the *IoTQoSystem* framework and discusses the framework's primary process of establishing the QoS agreement and proactively managing QoS violation. This chapter concludes by reviewing the design and implementation of the framework.

5.1 IoTQoSYSTEM OVERVIEW

The QoS negotiation framework, *IoTQoSystem,* is implemented to address the issues and challenges outlined in Section 1.2. The framework is designed to provide an automated negotiation of QoS parameters at runtime for the invocation of IoT services. *IoTQoSystem's* architectural design is based on the principle of microservices as the components of the negotiation framework can be deployed and tested independently. The developed framework is designed to be pluggable and extensible. Its architecture comprises two major components: a client and a service collaborating to resolve and manage the QoS contentions among IoT devices in an IoT environment.

5.1.1 Goal and Objectives

The main goal of the reinforcement learning QoS framework is to effectively establish a QoS contract and proactively manage QoS violations. Essentially, the framework is responsible for managing the QoS agreement reached between service providers and consumers in an IoT dynamic environment. The management of the QoS agreement comprises five primary processes: negotiation of the agreement, service provisioning, service monitoring, service renegotiation and service termination, as depicted in Figure 5.1.The QoS agreement is first established by a negotiation process aimed at maximizing the parties' utility while minimizing negotiation failures. Based on the established QoS agreement, the IoT service is delivered to the consumer and is monitored for violation in the QoS agreement. In the event of a failing service, a prompt renegotiation is automatically initiated with another service provider. This process continues until the IoT service lifetime expires.

The management of the QoS agreement is not a simple task, especially when the environment negotiation variables are frequently changing. This task is handled by a set of components that adapt their operational strategies to changes in the environment. The description of the framework within the context of the objectives of this thesis is as follows:

- ***Provide a reinforcement learning negotiation strategy for the generation and evaluation offers***. IoTQoSystem uses the proposed reinforcement learning negotiation strategy to guarantee a good success rate and utility to address the issue of poor utility and negotiation failures. It achieves this by using the context negotiation information such as the current negotiation state and the deadline criterion to decide the appropriate negotiation tactic to be utilized in the generation of offers that maximises the chances of reaching an agreement with high social welfare within the specified deadline.

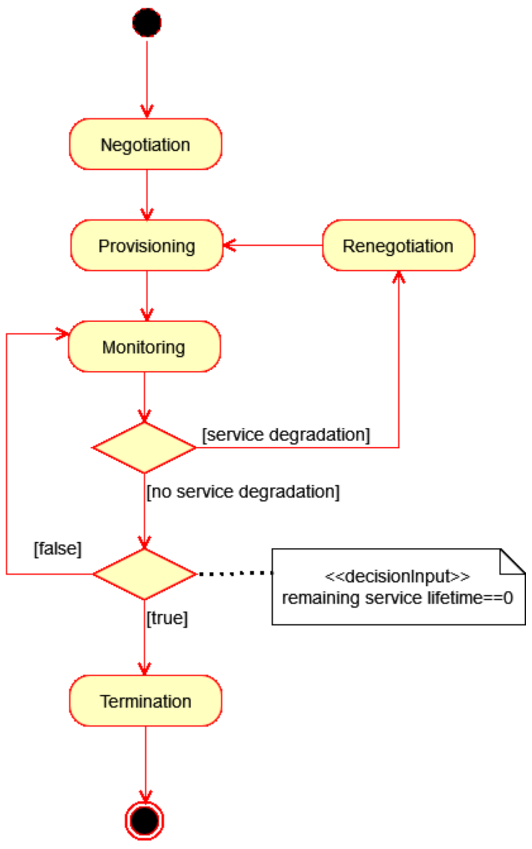


Figure 5.1: The QoS agreement management life-cycle

- ***Provide proactive support for QoS violations through monitoring and renegotiation.*** The QoS monitor component of the framework monitors the delivered IoT service by measuring and evaluating the value of QoS parameters of the service against the agreed terms in the QoS agreement. It monitors the changes in the quality of the negotiated service and automatically initiates an early renegotiation for degrading IoT service. During the provisioning of the negotiated service, changes in the measured QoS parameters data is used in forecasting the possibility of service failure. This minimizes the chances of service consumers experiencing service failure during service delivery.
- ***Provide flexible support for the expression of QoS preferences***. *IoTQoSystem* supports the updates of the QoS constraints in the QoS profile to reflect the changes observed in the physical world. It periodically monitors the resource status of IoT devices and uses this information to make the necessary changes needed in the QoS profile. For example, if the battery of an IoT device is running low, the preferred and reserved preferences of the QoS parameters are updated to reflect this change. This QoS management strategy potentially enables IoT devices to provide optimum IoT services and help prevent service failures in an IoT system.
- ***Provide a solution that ensures that the framework can cope at scale.*** The *IoTQoSystem* framework establishes the QoS agreement based on the current needs and constraints of all the users involved in the IoT-aware tasks. It uses a combination of

design and programming techniques and a negotiation protocol that ensures that it can scale with more devices, multiple negotiation sessions and can cope with a large infrastructure that exhibits distributed processing. The adoption of Java concurrency and multithreading enables the framework to conduct multiple negotiations in parallel, and the use of the SAOP negotiation protocol allows it to support multilateral negotiation. Having the framework's architecture based on microservices means that support can be distributed across an IoT Infrastructure.

5.1.2 Design Decisions and Justification

As part of this thesis, several important decisions were taken in the design and implementation of the *IoTQoSystem* framework. In taking this decision, a number of factors were considered to ensure that the framework achieves its aim and objectives. This section discusses the design and implementation decisions while developing the framework and provides justification for these decisions.

- **Programming Language:** The Java programming language was selected as the software language for the implementation of the framework. Java is a general-purpose computer programming language that is based on Object-oriented paradigm. It is considered a platform-independent language as its source code is first compiled into a binary *bytecode*. The bytecode can be run on any machine having the Java Runtime Environment (JRE) irrespective of the machine hardware or software configuration [76]. The decision of selecting Java was influenced by a number of factors such as its support for service-oriented technologies such as RESTful services, a large suite of libraries and the dynamic binding of these libraries, vibrant online community, support of the runtime environment on different IoT devices, interoperability with other software languages and my considerable knowledge and experience with Java. Recent survey data indicates that Java is one of the preferred programming languages for IoT application development. It tops the list of the programming language used in IoT gateway and edge nodes applications as well as for the IoT cloud applications, according to the IoT Developers Eclipse Survey 2019 report [77]. The selection of Java for IoT solutions by developers is primarily due to its versatility and flexibility. The results from the survey combined with the benefits of the Java programming language made Java the most suitable option in developing the *IoTQoSystem* framework.

- **QoS agreement Model:** In selecting a QoS agreement model, one of the factors considered was the expression of the agreement in a mutually understandable terms and in a format that maximizes syntactic and semantic interoperability. This means that the adopted QoS agreement model should provide a shared meaning of schema and content given the heterogeneity of devices involved in the negotiation process. Another factor taken into consideration was for the adopted model to be open to extension. This is important as it enables the agreement to be customized based on the needs and scenarios in which the framework is deployed.

  There exists a number of technologies and description languages that facilitates the modelling of the QoS agreement. The popular choice amongst them is the Web Services Description Language (WSDL) [28] and Web Application Description Language (WADL) [78]. Both technologies are language-independent and provide a shared meaning of schema. However, they fail to address the shared meaning of contents as they are basically a technical description. In addition, both description languages are not easily extensible as they do not elegantly support the linking of service operations to the physical world. Attempts have been made to address this issue, however, there is still no efficient link between these and their counterpart in the real world as it is still purely a technical specification.

Considering the needs of the QoS agreement model of the framework, the most promising description language is the Linked Unified Service Description Language (LUSDL) [79]. It is a platform-independent service description language that can be used in specifying QoS agreement properties. It is an extended version of USDL that builds upon the Linked Data principles. In providing a shared meaning of QoS agreement, Linked USDL uses formal ontology representation languages to manage the syntactic and semantic heterogeneity of QoS agreement. In addition, it uses linked data principles [80] to maximize interoperability and reuse and provide an elegant mechanism that foster the creation of extensions that increases the capabilities of the QoS agreement model when the need arises. Based on these attributes, it was decided that the Linked USDL would be the most suitable choice for the QoS agreement model for the framework.

- **Negotiation Model:** The selection of a negotiation model for *IoTQoSystem* is based on the QoS negotiation requirements stated in Section 3.2.3. The components of the negotiation model address the negotiation concerns in an IoT-service based system. The need to execute actuator tasks involving more than one service provider necessitated the adoption of the SAOP as the negotiation protocol described in Section 4.2.2. Apart from supporting both bilateral and multilateral negotiation modes, its low communication costs make it the most suitable negotiation protocol for resource-constrained environments. The uncertainty and dynamism of the negotiation environment prompted the decision to propose and utilise a negotiation technique based on reinforcement learning. The proposed negotiation technique described in Section 4.3 improves the success rate and social welfare of the negotiation process managed by the framework. For the negotiation architecture, it was decided that rather than using a broker to conduct the negotiation, the use of agents will be the most suitable choice as agents can independently take decisions that will yield a better utility based on the changes observed in the negotiation environment. Also, the adoption of agents by the framework eliminates the privacy concerns of negotiating parties, revealing their preferences to a third party.
- **Software Architecture:** Software architecture represents the highest decomposition of a software system. The software architecture defines the constraints on the implementation of a software system as both the structural elements (i.e. components unit of functionality) and non-functional properties (e.g. performance) are influenced by the selected architecture. It was an essential factor that was taken into consideration during the implementation of *IoTQoSystem* as it consists of the earliest set of design decisions that have the most far-reaching effect [81]. The heterogeneity and distributed nature of IoT played an essential role in selecting the software architecture for *IoTQoSystem*.

The heterogeneity characteristic of IoT pushes software design towards two architectural directions: Service Oriented Architecture (SOA) and Microservice. Both architectures suggest decomposing system functionalities into services across heterogeneous platforms, thus promoting interoperability. However, they differ in the granularity of the service size as the size of services in Microservice is comparably smaller and lighter compared to SOA. In addition, they differ in how they support heterogeneous interoperability. SOA supports the protocol-agnostic heterogeneous interoperability as it promotes the use of different messaging protocols such as REST, AMQP and RMI through its messaging middleware. Microservice supports protocol-aware interoperability as it simplifies the architectural pattern and corresponding implementation by restricting the choices of service integration [23]. With protocol-aware interoperability, the messaging protocol for invoking a service must be the same(e.g. REST) as it doesn't contain a messaging middleware. However, the implementation of the messaging protocol can be different. It was decided that using that Microservice will be the most suitable choice for implementing the negotiation framework as it allows the components of the framework to be tested and deployed independently. Also, it makes the framework lightweight, flexible and easy to update the framework functionalities in situations where the requirements cannot be completely anticipated in advance [82].

The selection of an architecture based on the distributed nature of IoT predominantly falls into two broad architectural models: client-server architecture and peer to peer architecture. The client-server architecture is a centralized distributed model where each node plays one of the two roles: the server role or the client role. The clients request services or functionalities from the server, and the server processes the requests and returns the result as a response to the client [17]. The client-server architectural model can either have a *thick client* or a *thin client.* In a client-server architecture with a thick client, the clients can process and execute their requests, and the server maintains the state and data of the system. With a thin client, the server performs all the processing, and the client only handles the presentation specifics. In contrast with the client-server architecture, the peer-to-peer (P2P) architecture is a decentralized distributed model, where every node has the same responsibility and can act as a server or a client. The requesting of services and the processing of requests can be performed by any of the nodes in the system [17].

It was decided that adopting the client-server architecture will be the most appropriate architecture for the design and implementation of the negotiation framework as it allows the framework to be more scalable and stable. The client component of *IoTQoSystem* can be described as being a thin client as it is primarily responsible for providing the most appropriate QoS profile to initiate a negotiation process. Since the client component is to deployed on IoT devices, which are generally resourced constrained, it becomes vital for the *IoTQoSystem* client to be lightweight.

- **Machine Learning models:** The nature of the problem addressed in this thesis necessitated the adoption of a machine learning model that will offer an optimum solution to the QoS contention among IoT devices in a dynamic environment. Specifically, software agents representing IoT devices are required to learn how to carry out the task of selecting a negotiation tactic that would yield the most utility within a specified deadline. Software agents learn by interacting with the changing negotiation environment by observing the current state of the negotiation environment and selecting the most appropriate negotiation tactic that maximizes their utility. Consequently, the QoS contention between IoT devices can be regarded as an interactive problem. Machine learning models are typically applied to situations where the task changes with time or across different participants [121]. Machine learning paradigms can be broadly divided into supervised, unsupervised, semi-supervised, and reinforcement learning [122].

Supervised learning is learning from a set of well-labelled training data. The training data comprises inputs paired with the correct outputs. The process of learning involves searching for the relationship between target outputs and the input features. The objective of a supervised learning model is to predict the correct output for new inputs using the relationship learnt from the training data. Supervised learning is an important type of learning; however, it is insufficient for learning from the environment interaction. It is impractical to obtain well-labelled training data that represent all the situations in which the software agents have to act. In the absence of a labelled dataset, software agents must rely on their own experience to learn. As a result, supervised learning is ineffective for dealing with interactive problems.

Unsupervised learning is learning from a set of unlabelled datasets (i.e datasets with only input features). An unsupervised learning algorithm is required to find clusters or groups with similar items within the collections of the unlabelled data. Unsupervised learning aims to discover the underlying structure or distribution in the dataset and group the data based on similarities. While uncovering structure in an agent's experience can be beneficial in interactive problems, it does not solve the challenge of maximising a reward signal on its own, thus making unsupervised learning unsuitable.

Semi-supervised learning is learning from a dataset using both supervised and unsupervised learning approaches. The dataset typically contains many unlabelled data and few labelled data. The learning process begins with discovering the hidden structure within the unlabelled data and uses the few labelled data within each identified cluster to label the other input features in the same group. Semi-supervised learning suffers from the same challenges as the supervised and unsupervised learning approaches. As a result, it is inappropriate in addressing the essential issues facing a software agent interacting over time with its environment.

Reinforcement Learning is learning how to map situations to actions to maximize a numerical reward signal [65]. Reinforcement Learning aims at taking actions based on the observations gained through interactions with the environment. Reinforcement Learning is characterised by software agents learning through the results of their actions rather than being explicitly taught. Software agents choose their actions based on previous experiences (exploitation) as well as new alternatives (exploration). Another distinguishing feature of reinforcement learning is that it explicitly considers the entire problem of a goal-directed agent interacting with an unknown environment. This differs from the other machine learning models, which focus on subproblems without considering how they might integrate into a wider picture. Based on these characteristics, it was decided that Reinforcement Learning would be the most suitable machine learning model for providing an optimum solution to the QoS contention among IoT devices in a dynamic environment.

- **Application Programming Interface(API):**

  The development of IoTQoSystem requires a service technology that will provide an interface through which the negotiation framework can consume services provided by other software systems (e.g. IoT middlewares) and transfer data over the internet using API calls. Two API technology styles based on service orientation has emerged; Simple Object Access Protocol (SOAP) and REpresentational State Transfer (REST) [153].

  SOAP is a messaging protocol that facilitates data exchange between service providers and consumers using HTTP and XML. It focuses on the transmission of XML-encoded messages over HTTP. SOAP is a stateless, one-way interaction between applications and devices and relies on service interfaces defined in a WSDL file to expose the functionalities of software components to client applications. On the other hand, REST is an architectural style that depends on a stateless communication protocol typically HTTP, to exchange data between clients and servers. REST uses the concept of resources in defining the client's requests and server responses. Every resource is associated with a unique Uniform Resource Identifier that captures the state of the resource. It uses HTTP to retrieve the representations of the different states of a resource.

  The REST web API was selected as the service technology interface for IoTQoSystem for the following reasons. Firstly, REST requires less bandwidth and computing power since its payload is lightweight. This small footprint makes it suitable for the resourced-constrained IoT devices on which the components of IoTQoSystem is deployed. In addition, since IoTQoSystem uses a client-server architecture paradigm and its components are loosely coupled, (i.e. its components are not coupled to a specific service API), utilising a REST service API becomes the best option. This is because it promotes and supports a loose coupling API design as changing services in the REST service provisioning does not require any change in the client code, unlike SOAP, where changing services often require a complicated change in the client code.

5.1.3 Technology Dependencies

There are many software libraries and hardware components that the current implementation of the *IoTQoSystem* framework depends on for it to be used. Table 5.1 lists the major software and hardware dependencies of the *IoTQoSystem* framework and describes their purpose in the system.

Table 5.1: The major technology dependencies of the IoTQoSystem framework

Dependencies	**Functions**
Java Runtime Environment	It is used to execute the I*oTQoSystem*'s Java programs. It is installed in the hardware platforms where the framework components are to be deployed

(JRE)	
IoT devices (Raspberry Pi)	It serves as the hardware deployment platform for the *IoTQoSystem* framework.
CHOReOS[1]	It serves as the service-oriented IoT middleware for the registration of QoS profile of IoT devices and the discovery of IoT devices
Linked USDL[2]	It serves as the description language for modelling the QoS preferences of IoT devices and the QoS agreement that defines the contract between IoT devices.

These technology dependencies are required for the execution and deployment of the framework. The software dependencies primarily make use of Java technology. The JRE contains the Java Virtual Machine(JVM), the execution engine that runs the Java programs from which the framework was built. The IoT devices are WIFi-enabled Single on Chip (SoC) boards for deploying the two components of the framework. The IoTQoSytem client component is deployed on a hardware platform that essentially has a battery for powering the hardware device and contains a fuel gauge for monitoring the battery level and sensors for detecting and measuring certain physical properties of the environment or the Physical Entity(PE) associated with the IoT device. The IoTQoSytem service component is deployed on a hardware platform that serves as the IoT gateway device. It is important to note that the IoT devices accommodating both components of the framework should be visible to each other over the network.

As indicated in Table 5.1, the *IoTQoSystem* framework uses the CHOReOS, a Java-based IoT service middleware, to enable the registration and discovery of IoT services involving the sensors and actuators. CHOReOS [83] adopts the IoT service-oriented architecture described in Section 3.2.1, and Table 5.1 illustrates the main middleware components and describes the requirements they help fulfil. The middleware fundamentally addresses the challenge of scale related to the discovery and registration of IoT services provided by IoT devices and the challenge of heterogeneity related to the composition and execution of IoT services. However, as observed in Table 5.2, it does not address the unknown dynamic nature of the environment associated with the negotiation of IoT services. The integration of *IoTQoSystem* enables the middleware to support QoS-aware and context-based dynamic negotiation.

Table 5.2:The description of the main components of CHOReOS

Components	**Description**
eXecutable Service Composition (XSC)	It enables the composition of heterogeneous IoT services

[1] https://github.com/choreos/choreos_middleware
[2] https://github.com/linked-usdl

eXtensible Service Access (XSA)	It facilitates the interconnection of heterogeneous IoT services.
eXtensible Service Discovery (XSD)	It facilitates the organization discovery of IoT services
Grid as a Service	It is responsible for managing large-scale choreographies that comprises hundreds to thousands of IoT services

CHOReOS provides a uniform interface through which an existing IoT gateway device can interact with it by providing REST APIs to be consumed. It should be noted that even though the framework architecture uses CHOReOS, it is not bound to it.

The framework uses the Linked USDL to specifying the QoS constraints of IoT devices and the QoS agreement properties. The Linked USDL has been designed to support a modular and extensible family of ontologies that provides the modelling, processing and sharing of service descriptions [84]. Table 5.2 shows the description of the five modules that comprise the Linked USDL Family. Specifically, the framework uses the *usdl-agreement* module for defining the QoS constraints, QoS requirements and the QoS agreement. The RDF approach of Linked USDL makes it easy to utilize existing vocabularies and add domain-specific elements. This allows the QoS profile to be extended and used in contexts with new and unforeseen requirements.

Table 5.3:The modules of LinkedUSDL

Components	**Description**
usdl-core	It defines the concepts central to the description of services.
usdl-agreement	It captures the information on the quality of the service provided.
usdl-sec	It describes the primary security properties of services.
usdl- ipr	It specifies the usage rights of services that are associated with the concept of copyright.
usdl- ipr	It defines the concepts that are required to describe the price structures in the service industry.

The other dependencies not listed in Table 5.1 includes the underlying operating system of the hardware platform, the software library, Java Agent Development Environment (JADE)[3], for generating software agents representing the IoT service providers and consumers in the *IoTQoSystem* framework

[3]https://jade.tilab.com/developers/source-repository/

5.2 IoTQoSYSTEM ARCHITECTURE

The architecture of the QoS negotiation framework primarily comprises two components: client and service, designed as a collection of independent microservices. They are both deployed on IoT devices and collaborate on the task of resolving the conflicting preferences between IoT service users. The client is responsible for managing the QoS profile that triggers the negotiation process, while the service is responsible for managing the QoS agreement generated from the negotiation process.

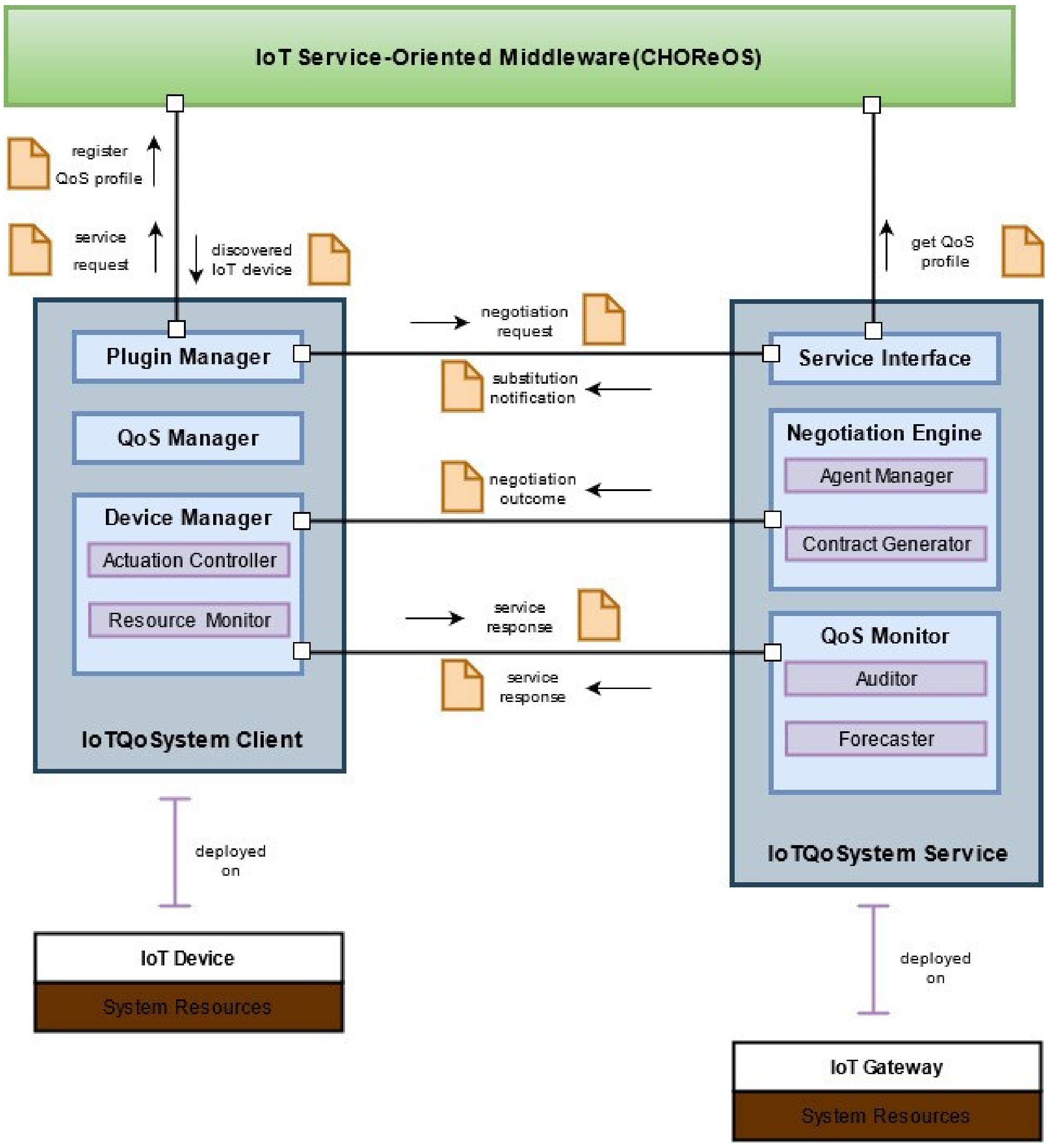


Figure 5.2: The high-level architecture of the IoTQoSystem framework

Having the framework's components implemented as microservices allows the framework to adopt a standard way of acquiring and consuming services. This enables the framework to seamlessly "plug into" any existing system. Figure 5.2 shows a high-level architectural view of the *IoTQoSystem* framework.

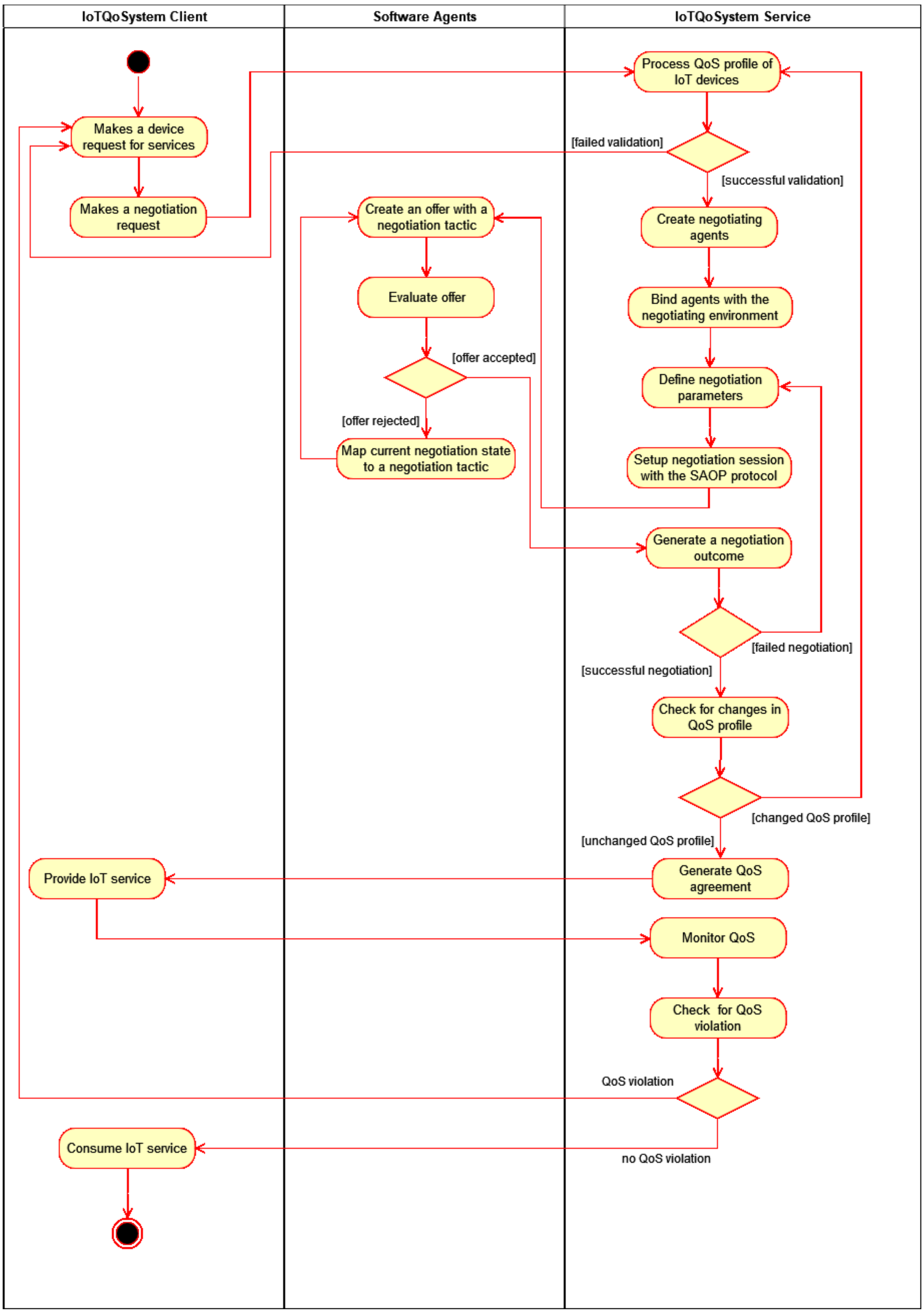

IoTQoSystem Client
Software Agents
IoTQoSystem Service
Process QoS profile of IoT devices
[failed validation]
[successful validation]
Makes a device request for services
Makes a negotiation request
Create an offer with a negotiation tactic
Create negotiating agents
Evaluate offer
Bind agents with the negotiating environment
[offer accepted]
Define negotiation parameters
[offer rejected]
Map current negotiation state to a negotiation tactic
Setup negotiation session with the SAOP protocol
Generate a negotiation outcome
[failed negotiation]
[successful negotiation]
Check for changes in QoS profile
[changed QoS profile]
[unchanged QoS profile]
Provide IoT service
Generate QoS agreement
Monitor QoS
Check for QoS violation
QoS violation
Consume IoT service
no QoS violation

Figure 5.3: The QoS management process implemented by the framework.

The framework resolves the QoS contention among IoT devices(i.e. IoT service consumer and providers) through a negotiation process modelled as a Markov Decision Process (MDP). This allows the framework to adopt a machine learning-based negotiation strategy that takes into account the uncertainties in the negotiation environment in generating a QoS agreement. The QoS agreement is monitored for QoS violations, and in the event of a failing service, an early renegotiation is automatically initiated. Figure 5.3 depicts the framework's primary process of establishing the QoS agreement and proactively managing QoS violations.

As shown in Figure 5.2, the client component is hosted on IoT devices that represent the service providers and consumers, while the service component is hosted on an IoT gateway. Using the API provided by the IoT middleware, the client component enables IoT devices to register their QoS profiles and request a service, while the service component allows the IoT gateway to retrieve the QoS profiles of intending negotiation IoT devices. After querying the IoT middleware for an IoT device capable of providing an IoT service (e.g. sensor data), the service consumer sent a negotiation request to the service component of IoTQoSystem. The service component fetches the QoS profiles of devices specified in the negotiation request from the middleware.

The service component begins the negotiation process by first validating the received QoS profiles. If the validation is successful, it proceeds to generate negotiation agents based on the information defined in the QoS profiles. The agents are bound with a model of the negotiating environment, and the negotiation session is initialised using the SAOP protocol for agents to exchange offers. If the negotiation agents agree on a particular offer and there is no change in the QoS profile of the devices involved, a QoS contract is generated and sent to them. Otherwise, the negotiation session is reinitialised with a different set of negotiation parameters. On receiving the QoS contract, the IoT device playing the role of a service provider begins the transmission of data based on the agreed terms specified in the contract to the IoT device, playing the part of the service consumer. The IoT gateway intercepts the data for QoS monitoring and prediction purposes. The detection of a QoS violation by the IoT gateway sets in motion the replacement of the current service provider, and the whole process is repeated. The following sections describe the framework components and their processes.

5.2.1 IoTQoSystem Client

The *IoTQoSystem* client is deployed on the devices serving as the IoT service provider and consumer. It manages the QoS profiles of IoT devices on which it is deployed. It uses the methods provided by the API interface of the middleware to register the QoS profile of the IoT device and discover IoT devices to fulfil its service requests. It interacts with the components of the *IoTQoSystem* service using the methods provided by them. An IoT device as a service provider allows the *IoTQoSystem* client to use the monitored resources to make changes to the QoS profile. As a service consumer, the QoS profile of an IoT device is dependent on the requirements of the actuation task, which is usually fixed. Since IoT devices are generally resourced-constrained, the *IoTQoSystem* client is designed to be lightweight, as reflected in the architecture and technologies adopted by the client component of the framework. Figure 5.4 shows the high-level architecture of the *IoTQoSystem* client.

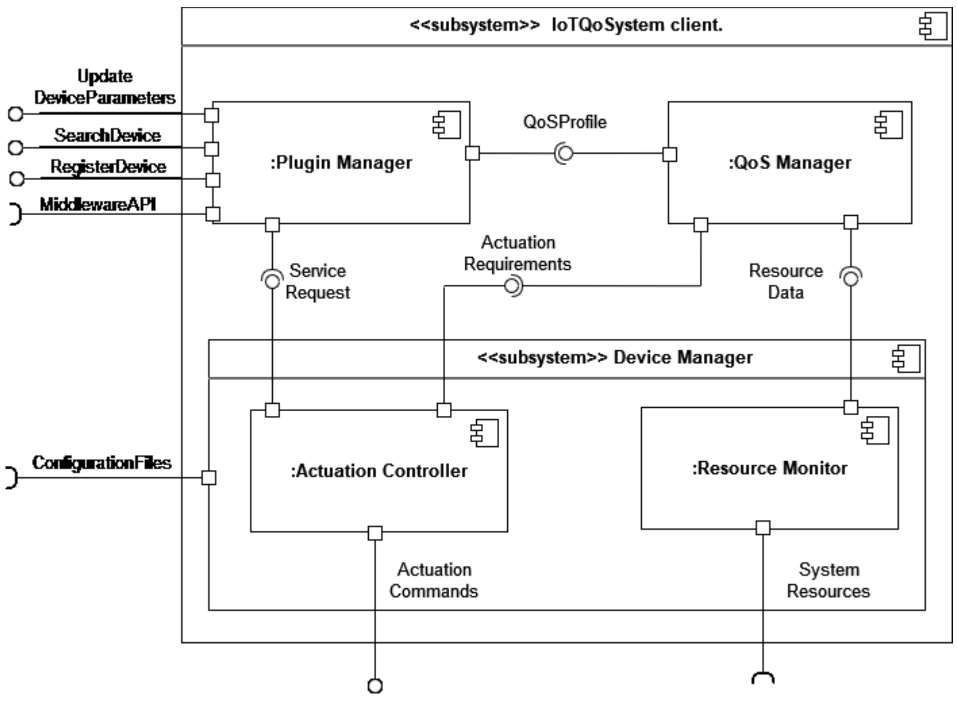


Figure 5.4: The architecture of the IoTQoSystem client

- **Plugin Manager:** The plugin manager is responsible for acquiring the necessary interfaces required for the *IoTQoSystem* client to be seamlessly "plug in" to a service-oriented IoT middleware. To achieve this, It implements the CHOReOS middleware components that facilitate the registration of the QoS profiles of IoT devices and the discovery of IoT devices for negotiation. These components include the RegistrationManger interface API that allows IoT devices to register and update their QoS profile and the QueryManager API that enables IoT devices to query the middleware for an IoT device(s) that can satisfy its request. The plugin manager makes it possible for the framework to be integrated with other service-oriented middleware, provided the supplied interface can be wrapped in a Plugin instance. The plugin manager also communicates with the *IoTQoSystem* service component as it is subscribed to the methods of the service interface, which allows it to send a negotiation request and receive a service substitution notification.
- **QoS Manager:** The QoS manager is responsible for managing the QoS profile, which is to be registered in the middleware via the plugin manager. It checks the validity of the QoS profile submitted by the device manager. As a service provider, the QoS profile of an IoT device specifies the QoS constraints, and the QoS manager ensures that these constraints are updated to reflect the current capability of the IoT device. To achieve this, it periodically checks if the QoS profile needs to be updated using the data read by the resource monitor of the device manager. The periodic check for updates is configurable and, by default, occurs once every 3 minutes. When the IoT device operates as a service consumer, the QoS profile specifies the QoS requirements needed for an actuation task. Given that these requirements are usually fixed and known at design time, the QoS manager allows the QoS profile to be submitted to the middleware through the plugin manager. The adoption of Linked USDL for capturing the QoS constraints and requirements enables the QoS profile to use Terse RDF Triple Language

(Turtle[4]) as its file format. Appendix A.1. shows the usage of Linked USDL for describing the QoS configurations of an IoT device.

- **Device Manager:** The device manager contains the decision logic responsible for the operation of an IoT device. Specifically, it loads the configuration files, which determines the role to be played by the IoT device either as a service provider or as a service consumer, thus indicating which of its components: ***actuation manager*** or ***resource monitor*** is to be used for the operation of the IoT device. Amongst the configuration files is the QoS profile created using the Linked USDL editor[5] and the XML configuration setting file that contains the information related to the device manager's components. The device manager is also subscribed to methods provided by the *IoTQoSystem* service, which enables it to communicates with the *IoTQoSystem* service modules.

  When the IoT device is a service provider, the resource monitor uses the information in the configuration setting file to monitor the external resources of the IoT device. The external resources are essential to the service providers as they affect the operation of the resourced-constrained device and, thus, the quality of service being provided. Specifically, the resource monitor monitors the battery level and the speed of the network connectivity. It contains a number of readers that are polled periodically to read the measurements of these set of resources. These measurements are then used to trigger the required changes in the QoS profile. By default, these resources are polled every 3 minutes. However, this periodic update can be set in the resource-monitoring configuration file. Using the methods provided by the *IoTQoSystem* service, the device manager can send sensor readings to the QoS monitor of the *IoTQoSystem* service based on the QoS agreement received. Similarly, when an IoT device is a service consumer, the actuation manager uses the information in the configuration setting file to send an IoT service request to the middleware through the plugin manager. The response from the service provider(s) through the *IoTQoSystem* service is used to initiate an actuation command.

5.2.2 IoTQoSystem Service

The *IoTQoSystem* service is deployed on an IoT gateway to manage the negotiation process. It generates the QoS agreement and monitors the QoS as the service is being consumed. It is also responsible for coordinating the negotiation agents and provides a negotiation interface through which IoT devices operating as service consumers can submit a negotiation request. Each module of the *IoTQoSystem* service provides a standard method of access to the components of the *IoTQoSystem* client that are subscribed to it. The description of the general behaviour and the functions of each of the modules in this component is discussed below. Figure 5.5 shows the high-level architecture of the *IoTQoSystem* service.

[4] https://www.w3.org/TR/turtle/

[5] https://github.com/linked-usdl/usdl-editor

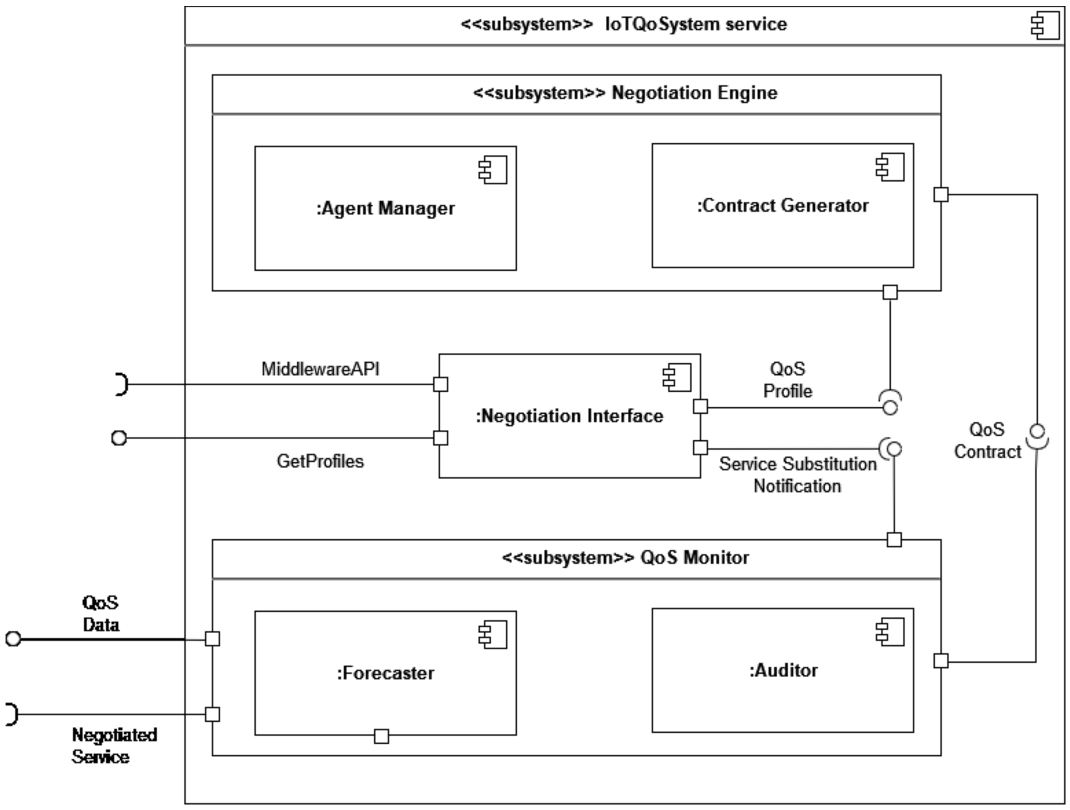


Figure 5.5: The architecture of the IoTQoSystem service

- **Negotiation Interface:** This module serves as the first entry point to the *IoTQoSystem* service component. It provides an interface that facilitates the interaction with the *IoTQoSystem* client component and implements the QueryManager API of the CHOReOS middleware with which it can retrieve the QoS profile of IoT devices. On receiving the negotiation request from an IoT device functioning as a service consumer, it retrieves the QoS profile of the specified IoT devices in the request from the CHOReOS middleware and delivers this information to the negotiation engine. The negotiation interface also carries out some internal processing like checking the acquired QoS profile of IoT devices for a gap in the QoS preferences that contains the agreement zone.
- **Negotiation Engine:** The negotiation engine is responsible for executing the entire negotiation process and the generation of the QoS agreement negotiation using the concept of "containers". A container is an instance of the negotiation engine that provides the environment to execute a negotiation session. With a set of containers, the framework can conduct multiple concurrent negotiations [99] as each container is a Java process that provides the required services needed to generate a QoS agreement between IoT devices. The negotiation engine serves as a bootstrap point with which its architectural components are launched. When the negotiation engine launches a container, the container is registered with it, and the agent manager and contract generator are initialized. Figure 5.6 shows a conceptual UML diagram that schematises the relationship between the components of the negotiation engine.

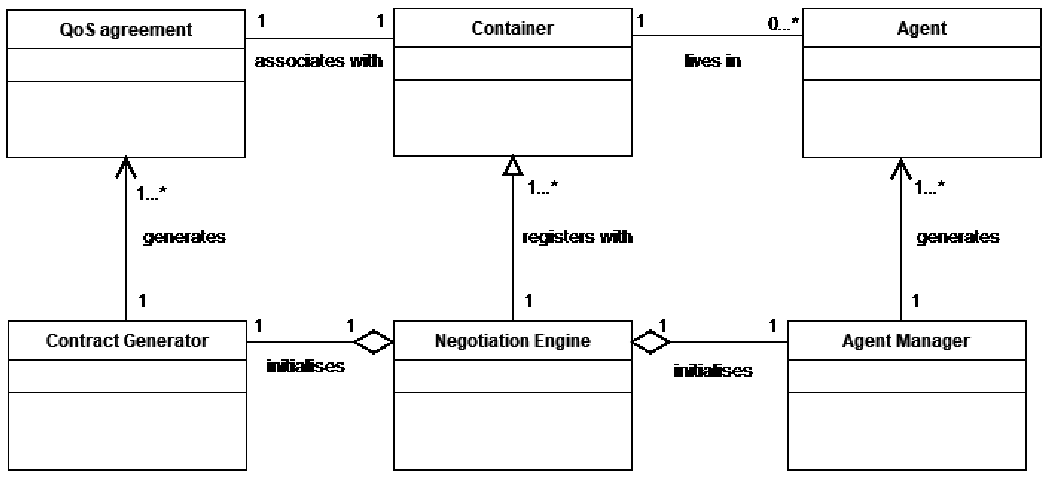


Figure 5.6: Relationship between the components of the Negotiation Engine

The ***agent manager*** provides the services required for an agent life cycle while the ***contract generator*** creates the formal QoS agreement from the discovered negotiation solution. The agent manager uses the Java Agent Development Framework (JADE) [85] to generate a negotiating agent for each IoT device and allows the agents to communicate with each other using the SAOP communication protocol. In the most basic negotiation scenario, the agent manager spawns one agent for a service consumer and one agent for the service provider in the container. In a multilateral negotiation scenario, the agent manager generates one agent for the service consumer and one agent for each service providers in the container. The choice for JADE is due to its support for the execution of multiple parallel tasks within the same Java thread and its low runtime's memory footprint of around 100KB [100]. The negotiation engine implements the utility function described in Section 4.1.2 and the reinforcement learning negotiation strategy model described in Section 4.3.2. These implementations, combined with the QoS profile received from the negotiation interface, are used to create an internal model for each agent from which it can take actions and respond to the changes that occur in the negotiation environment. Once a potential negotiation solution is identified, and there isn't any change in the QoS profiles of the IoT devices involved in the negotiation session, the contract generator proceeds to translate the resulting solution into a binding agreement that constitutes the contract between the IoT devices using the Apache Jena library[6]. Appendix A.2. shows an instantiation of a QoS agreement in the Turtle notation.

- **QoS Monitor:** This module is responsible for monitoring the IoT service being consumed by the service consumer for QoS violations prediction. The IoT service attributes are continuously evaluated against the QoS agreement, and an early renegotiation is automatically initiated in the event of a predicted failing service. Figure 5.7 illustrates the primary structural components of the QoS monitor.

[6] https://jena.apache.org/

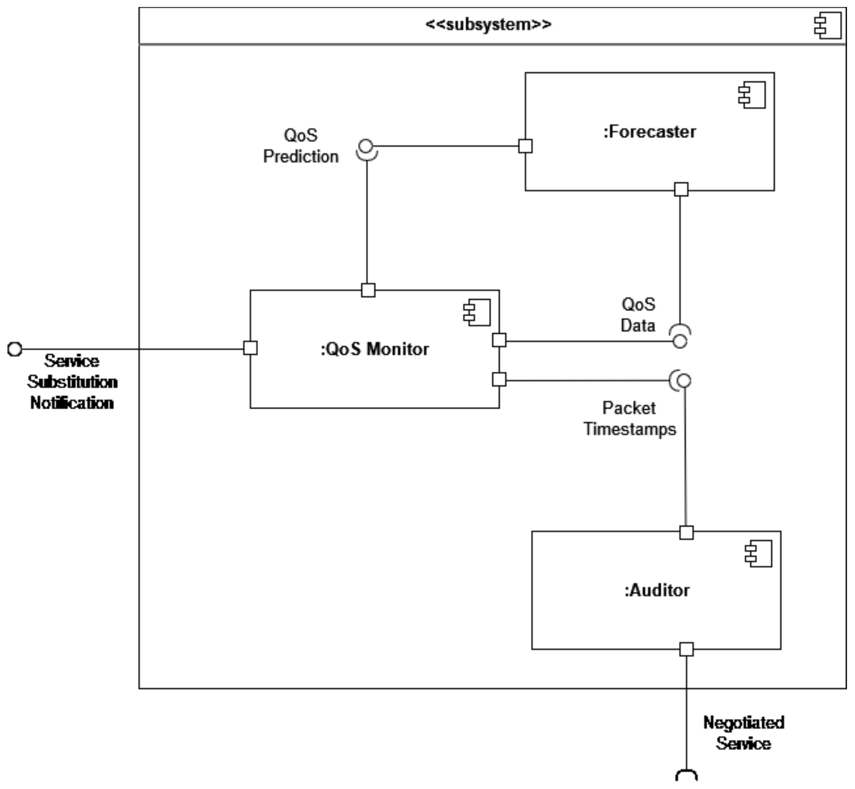


Figure 5.7: The architecture of the QoS monitoring components.

The QoS monitor uses an auditor, a passive monitoring dynamic proxies, that places no additional load on the service provider to transparently intercept the IoT service being provided to the service consumer using the packet capturing library Jpcap[7] library. The QoS monitor computes the values of the QoS parameters of the negotiated service using the network packets' timestamps collected by the auditor and stores these data in a MySQL database. The forecaster uses the QoS data of the measured service to predict a degrading service. The prediction made by the forecaster is based on the dynamic tendency prediction strategy described in [86]. The dynamic tendency prediction strategy is a one-step-ahead time series prediction strategy that uses the current measured value and the mean of the historical measured value of QoS parameters to predicts future QoS parameter values. Essentially, the next predicted value is derived by adding or subtracting an independent variable called the *variator*, according to the tendency of the value change. The dynamic tendency prediction strategy is formally expressed in Algorithm 5.1.

Algorithm 5.1 Dynamic tendency Prediction Strategy

```
Input:  Current actual value ($v_t$)
        Previous actual value ($v_{t-1}$)
Begin:
  if (($v_t$ - $v_{t-1}$)<0)
     tendency="decrease"
```

[7] https://github.com/jpcap/jpcap

```
        $p_t = v_t - variator$
    else
        tendency="increase"
        $p_t = v_t + variator$
 End
Output:  The next predicted value ($p_t$)
```

The value of the variator is based on the tendency of a change in direction with the mean history data as the threshold value. The variator adaptation process is illustrated in Algorithm 5.2

Algorithm 5.1 Variator adaptation process

```
Input:  Adaptation degree(ad)
        Previous actual value ($v_{t-1}$)
        Current actual value ($v_t$)
        Variator factor(vf)
Begin:
    mh= mean of the historical values
    $\Delta v = abs(v_t - v_{t-1})$
    if ($v_t$ <mh)
        variator= vf +( $\Delta v$ – vf) * ad
    else
        ph= percentage of the historical values greater than $v_t$
        variator= abs( vf * ph)
 End
Output: variator
```

Figure 5.8 illustrates the QoS monitoring process. The QoS parameters of the negotiated service are continuously measured, and the predicted values are evaluated against the agreed values defined in the QoS agreement. If the predicted value of a QoS parameter differs from the agreed value by a certain threshold value for a given number of times, this indicates that there is a possibility of a QoS violation, and the forecaster signals the negotiation interface for an alternate service provider.

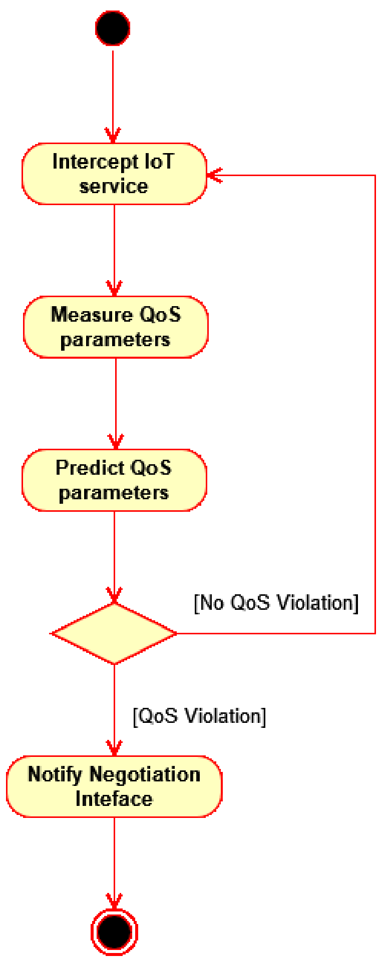


Figure 5.8: Overview of the QoS monitoring process

5.3 IoTQoSYSTEM REVIEW

With heterogeneous devices dynamically interacting with each other to perform actuation tasks, there arises the question of how best to resolve the QoS contentions between these devices with conflicting preferences to guarantee the execution of these tasks without failures. This thesis presents the design and implementation of a QoS negotiation framework, *IoTQoSystem,* that effectively establish QoS contracts and proactively manage QoS violations. Essentially, *IoTQoSystem* manages the QoS contract between IoT service providers and consumers in an IoT dynamic environment. Its architecture primarily consists of two components, designed as a collection of independent microservices, deployed on IoT devices and collaborating to provide automated negotiation of QoS parameters before IoT service provisioning. The first component manages the QoS profiles of IoT devices, while the second component manages the QoS agreements generated from the negotiation process. The framework was designed to be scalable, reliable and high performing.

Furthermore, *IoTQoSystem* was developed to satisfy the six QoS negotiation requirements (RQs) described in Section 3.2.3. In achieving this, it uses a combination of software development models such as a Machine Learning (ML) paradigm, QoS agreement model and a negotiation model. It uses Reinforcement Learning (RL), an ML paradigm, to design the framework's negotiation strategy. The ML-based negotiation strategy aims to enable negotiating agents to determine the best course of action, which will result in an agreement that maximises the agents' utility function, thus satisfying RQ3 and RQ6. It uses Linked USDL, a platform-independent, semantically enabled, flexible and technology agnostic service description language to specify the QoS agreements and QoS profile parameters. This adopted QoS agreement model allows the framework to achieve RQ1 and RQ2. In realising RQ4, it uses the SOAP negotiation protocol to carry out both bilateral and multilateral negotiations. The dynamic tendency prediction strategy was used in monitoring and detecting a failing service, which satisfies RQ5.

Chapter 6

EVALUATION

This chapter presents an evaluation of the *IoTQoSystem* negotiation framework using a set of experiments that test the research questions outlined in Section 1.2. The evaluation combines a simulation module and a small-size vertical-farming case study. The simulation module simulates the dynamic characteristics of IoT using a real-world IoT dataset and assesses the scalability of the framework. The vertical farming case study comprises an IoT gateway node and four plant nodes. The chapter concludes with a discussion of the results gathered during the evaluation.

6.1 EVALUATION DESIGN

The goal of software evaluation is to appraise the results of an action or a process in order to improve the quality of the actions or to select the best action alternative [87]. The goal of evaluating the *IoTQoSystem* is to determine the extent to which the research objectives have been satisfied and compare the evaluation results with the results from similar QoS negotiation approaches. This evaluation approach used in assessing *IoTQoSystem* is the *summative evaluation* [88], which is concerned with the global aspects of a software system. The justification for this is provided in Section 6.1.2.

6.1.1 Evaluation Techniques

Software evaluation techniques are the activities of the evaluators that can be defined in behavioural and organizational terms [87]. Several software evaluation techniques can be used to evaluate a software system, and they can be broadly classified into two groups [88]: the descriptive evaluation techniques and the predictive evaluation techniques.

- **Descriptive evaluation techniques:** These evaluation techniques objectively and reliably describe the status and the actual problems of software systems. They require a software prototype and at least a user and can be subdivided into three approaches :
  - Behaviour-based method: This method record user's actions and behaviours as the system is being used. Data are collected through observation techniques such as ethnography and user descriptive methods such as the "thinking aloud" protocol
  - Opinion-based method: This method elicits user opinions through various mediums such as interviews, questionnaires, and surveys. Data garnered through these methods are usually subjective.
  - Usability testing: This method combines both behaviour and opinion-based methods with some amount of experimental control, usually chosen by an expert to evaluate a software system.
- **Predictive evaluation techniques:** These techniques elicit recommendations and future requirements for the development of a software system. Unlike the descriptive evaluation techniques where the software system is used or observed by end-users, these techniques focus on predicting certain aspects of the software system and often involve experts. These techniques are associated with problem-solving methods that are used for software requirement analysis. Predictive evaluation techniques include

expert reviews seeking to anticipate usage problems that will arise and inspection methods such as inspecting the interaction between a user and the system.

6.1.2 Evaluation Justification

Given the different methods of evaluating software systems, it is vital to select an evaluation method that suits the research and most appropriately assess how well the outcome of the research addresses its objectives. The primary aim of the research described in this thesis was to investigate whether an approach based on reinforcement learning can provide effective QoS negotiation support for an IoT middleware. The research identified four key objectives, which motivated the development of a QoS negotiation framework, *IoTQoSystem*. Consequently, the evaluation of the processes of the *IoTQoSystem* framework will be based on these research objectives.

The first objective seeks to simultaneously improve the success rate of the negotiation process and the social welfare of the generated QoS agreement using the negotiation strategy proposed in this thesis. The evaluative experiment of this objective is to show the benefits of using the context negotiation information such as the current negotiation state and the deadline criterion to decide the appropriate negotiation tactic. A comparison between the experimental results using the proposed negotiation strategy and the experimental results from a similar negotiation strategy is used to determine whether this objective has been achieved.

The second objective seeks to apply a QoS evaluation mechanism that monitors the service being delivered and conducts QoS violation predictions to detect a possible IoT service failure. Experiments for objective 2 consider how the dynamic tendency prediction strategy can anticipate future imminent QoS violations. These experiments are designed to ascertain whether the QoS negotiation framework can provide the desired reliability expected in an IoT system.

The third objective seeks to address poor support for dynamic QoS preferences. The experiments for evaluating this objective details how *IoTQoSystem* can support changes in the QoS profile of IoT devices. The evaluation of this objective is heavily influenced by the assumption that the battery level of IoT devices plays an essential role in the longevity of IoT devices and the strength of the network connectivity affects the capability of IoT devices to provide IoT services, thus affects the QoS parameter constraints of service providers. The goal of the evaluative experiments associated with this objective is to determine whether a change in QoS preferences can prevent service failure in the vertical farming system.

The fourth objective extends objective1 by considering how multilateral and multiparty negotiation scenarios are supported by the negotiation framework. The experiments for objective1 demonstrate the bilateral SLA negotiation of IoT services, but objective four seeks to effectively resolve the QoS contentions between more than two IoT devices and concurrently conduct multiple negotiations. The experiment for objective four is designed to check whether the QoS negotiation framework can scale both horizontally and vertically without any significant drop in performance.

Given the nature of the experiments to be conducted, it can be argued that these objectives can be satisfied with the involvement of an expert rather than an end-user. As a result, this thesis supports the use of predictive evaluation techniques for the evaluation of the framework. The choice of predictive evaluation is further supported by the selected case study and the use of a simulation module. The method of operation of the case study using simulation data requires a software developer's expertise to provide an environment similar to an uncontrolled environment in the "wild" in which the framework is intended to operate in.

6.1.3 Overview of Case Study

The case study is a vertical farming system, one of the several applications of IoT in the agricultural sector. The vertical farming system is a growing application domain in IoT, and it is predicted that the installation of IoT devices in the agricultural sector will increase from 30 million in 2015 to 75 million by 2020 [89]. Vertical farming is a technology-based agricultural system where crops are grown in a contained and controlled environment [90]. The main idea of IoT-based vertical farming is to use technology to control environmental factors such as water and light to improve the quality of the farm produce, minimize risk, and maximize profits [91]. The case study adopts the two key processes of an IoT-enabled vertical farming system which ensures that the key requirements of photosynthesis, i.e. water and lights, are always available for the plants. The processes include watering plants when they need to watered and switching on the grow light when darkness falls.

The case study consists of four plant nodes and a gateway node. Each plant node has a set of actuators that are controlled by an associated set of sensors that helps in encouraging photosynthesis and germination of the plant grown in the plant pots, as seen in Figure 6.1. The actuators include a USB-powered 44GPH, 3.5V water pump, and a 5V grow light. The sensors include an AM2302 humiture sensor, a moisture sensor, and a light-dependent resistor (LDR) that is connected to a battery-powered Raspberry Pi 3B module. The water pump supplies water to the soil in the pots on which the plants grows. The grow light contains multiple LEDs that provide infra-red lights to the plant pots. The humiture sensor measures the environmental temperature and humidity of the plant pot, and the moisture sensor measures the moisture level of the soil in the plant pot. The readings from both sensors are used to control the water pump. The LDR measures the environmental light intensity, and its measurements are used to control the grow lights. The sensor data is made available as IoT services using Flask[8] , with JMeter[9] used in varying the workload of each IoT service. In addition to the sensors, a fuel gauge is connected to the Raspberry Pi to monitor the battery level. The client component of the QoS negotiation framework is deployed on the Raspberry Pi of each plant node, and the service component is deployed on a Raspberry Pi 3 serving as the gateway node. The CHOReOS middleware [83] is deployed on a Windows-powered laptop serving as a remote server through which each plant node can register and update their QoS profiles that specify the QoS requirements and constraints of its associated actuators and sensors, respectively. The sensing and actuation configurations are expressed as the QoS parameter values. These values are built from the dataset of a real-world vertical farming system, and they differ from plant node to plant node. This difference captures the conflicting QoS preferences between the plant nodes.

The motivation behind the case study is to optimise the processes that enable photosynthesis when one of the sensors associated with a plant node fails or degrades in its operation, and as a result, the plant node plays the role of the service consumer, as seen in Figure 6.1.

[8] https://flask-restful.readthedocs.io/en/latest/
[9] https://jmeter.apache.org/

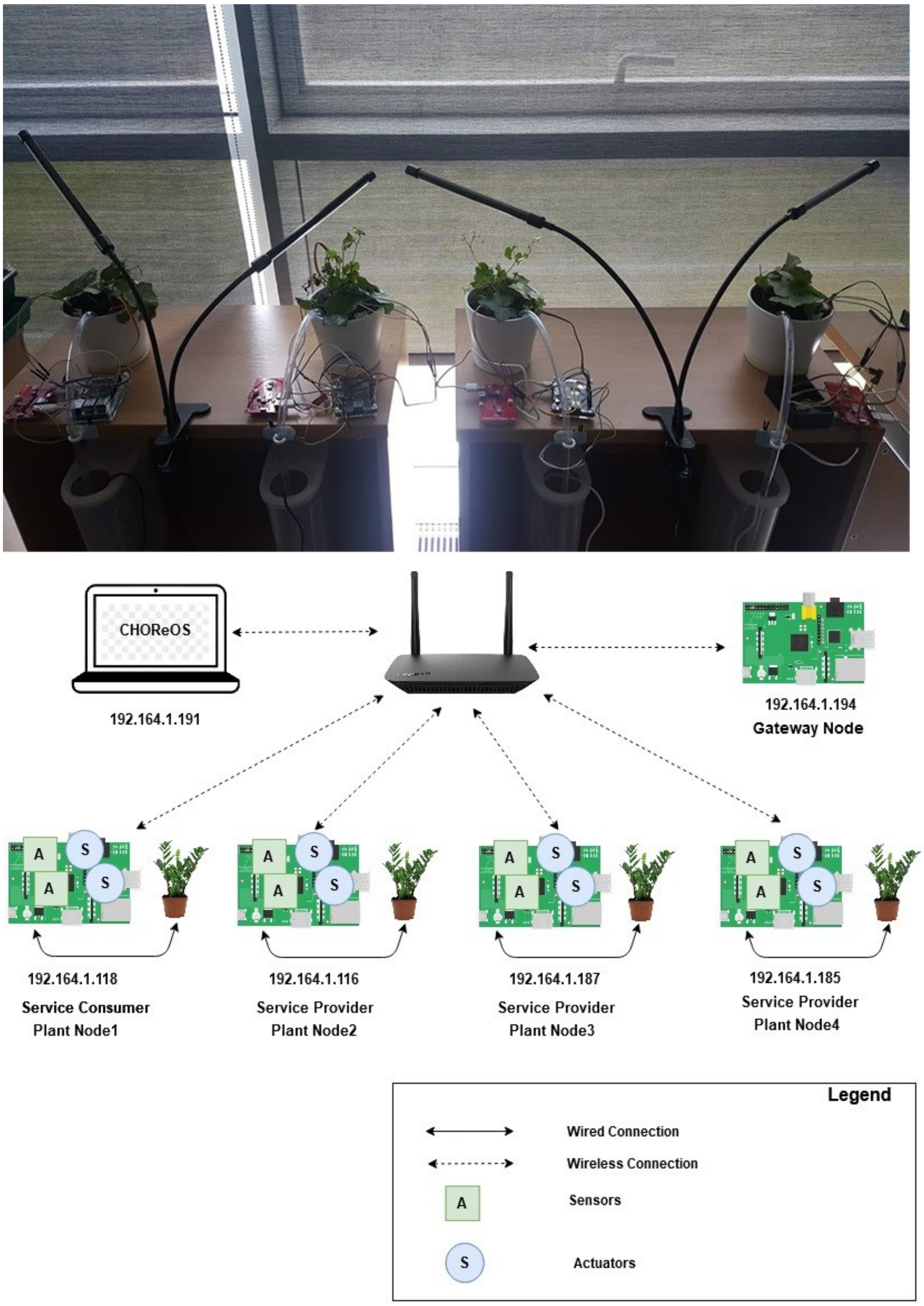
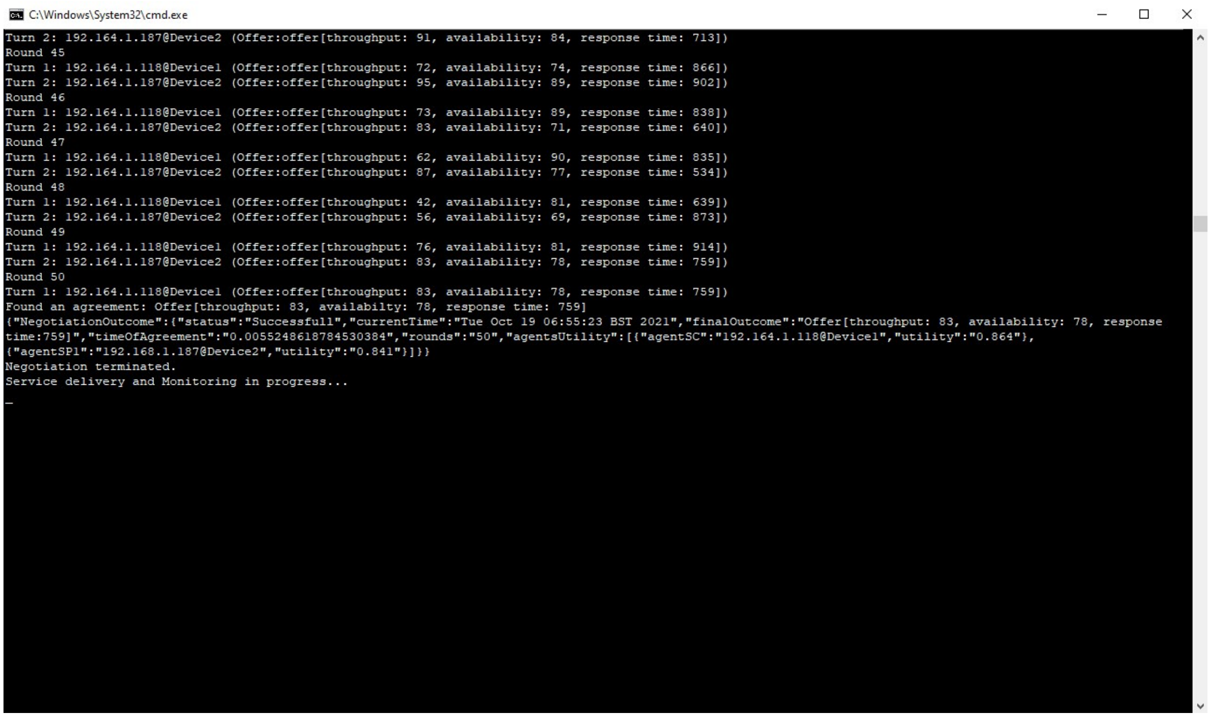


Figure 6.1:Top: The medium size vertical farming system; Bottom: The schematic experimental setup

The service consumer requests for sensor data which is to be used in controlling its actuator. It does this by first querying the CHOReOS middleware for an IoT device that best matches its request. In fulfilling this request, the CHOReOS middleware is required to select an IoT service provider based on its current QoS profile and existing service workload. CHOReOS does this by searching for a suitable service provider from amongst the three other plant nodes. The service component of the QoS negotiation framework generates the QoS agreement between the service consumer and provider and monitors the delivery of the IoT service. Figure 6.2 shows a screenshot of the service component completing a negotiation session and reaching an agreement between two plant nodes.



```
C:\Windows\System32\cmd.exe
Turn 2: 192.164.1.187@Device2 (Offer:offer[throughput: 91, availability: 84, response time: 713])
Round 45
Turn 1: 192.164.1.118@Device1 (Offer:offer[throughput: 72, availability: 74, response time: 866])
Turn 2: 192.164.1.187@Device2 (Offer:offer[throughput: 95, availability: 89, response time: 902])
Round 46
Turn 1: 192.164.1.118@Device1 (Offer:offer[throughput: 73, availability: 89, response time: 838])
Turn 2: 192.164.1.187@Device2 (Offer:offer[throughput: 83, availability: 71, response time: 640])
Round 47
Turn 1: 192.164.1.118@Device1 (Offer:offer[throughput: 62, availability: 90, response time: 835])
Turn 2: 192.164.1.187@Device2 (Offer:offer[throughput: 87, availability: 77, response time: 534])
Round 48
Turn 1: 192.164.1.118@Device1 (Offer:offer[throughput: 42, availability: 81, response time: 639])
Turn 2: 192.164.1.187@Device2 (Offer:offer[throughput: 56, availability: 69, response time: 873])
Round 49
Turn 1: 192.164.1.118@Device1 (Offer:offer[throughput: 76, availability: 81, response time: 914])
Turn 2: 192.164.1.187@Device2 (Offer:offer[throughput: 83, availability: 78, response time: 759])
Round 50
Turn 1: 192.164.1.118@Device1 (Offer:offer[throughput: 83, availability: 78, response time: 759])
Found an agreement: Offer[throughput: 83, availabilty: 78, response time: 759]
{"NegotiationOutcome":{"status":"Successfull","currentTime":"Tue Oct 19 06:55:23 BST 2021","finalOutcome":"Offer[throughput: 83, availability: 78, response
time:759]","timeOfAgreement":"0.0055248618784530384","rounds":"50","agentsUtility":[{"agentSC":"192.164.1.118@Device1","utility":"0.864"},
{"agentSP1":"192.168.1.187@Device2","utility":"0.841"}]}}
Negotiation terminated.
Service delivery and Monitoring in progress...
_
```

Figure 6.2: A screenshot of a QoS agreement reached between two plant nodes.

The evaluation experiments are aimed at assessing that the main qualities of the proposed negotiation framework. This includes (1) Performance: comparing the performance of the proposed negotiation strategy model in terms of social welfare and success rate with a state of the art negotiation strategy model; (2) Reliability: evaluating the framework ability to detect a failing service and initiating a renegotiation; (3) Adaptability: evaluating the framework ability to update the QoS profile of IoT devices as their external resources change; (4) Scalability: evaluating the framework ability to scale from small-scale to large-scale negotiation scenarios. The first three experiments were primarily carried out using the case study, while the fourth experiment was exclusively conducted using the simulation module.

6.1.4 Simulation Module

The simulation module is responsible for simulating the inherently dynamic nature of IoT, initializing the QoS parameter values of the plant nodes and assisting in evaluating the framework's scalability. The simulation module comprises four primary components; the NetworkSpeedSimulator simulates the unstable network connectivity; the CPUWorkLoadSimulator simulates the variation of resources in terms of the gateway node's CPU time and memory allocation; the VirtualNodeLauncher generates virtual node devices; the QoSMatrixLauncher generates a matrix of QoS parameter values. It also provides an interface with which each of its components can be accessed by the QoS negotiation framework.

- **NetworkSpeedSimulator:** It simulates the network fluctuations that could be experienced by IoT devices deployed in the "wild" or a hostile environment [56]. It uses the network script[10] to simulate a range of network speed in a wireless LAN. The

[10] https://gist.github.com/obscurerichard/3740206

script allows the network speed for each Raspberry Pi to be varied continuously. Preliminary results indicate that when the network script simulates a low bandwidth (100kbs) and high-latency (350ms) network connection, the data transmission becomes unreliable and unstable as the service consumer receives only a fraction of the data packets sent from the service provider. As a result, 100kbs was benchmarked as the critical network speed. Figure 6.3 illustrates a simulated network speed for a plant node set at different network speeds within 30 minutes with 100kbps as the critical network speed.

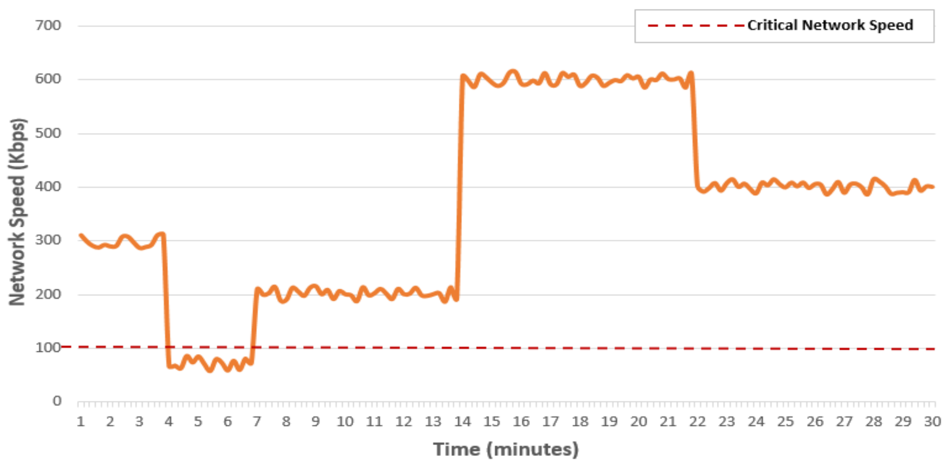


Figure 6.3: A simulation of the network speed of an IoT device

- **CPUWorkloadSimulator:** It simulates the resource variation in the gateway node's CPU time and memory allocation. To evaluate the validity of the proposed framework, it is crucial to consider the availability of negotiation resources as a primary factor that contributes to the uncertainties in the negotiation environment. Towards this purpose, the dataset provided by the Grid Workload Archive[11] (GWA-T-12 Bitbrains) was used as the basis to simulate the CPU workload of the gateway node.
  The dataset is based on the performance metrics of the CPU of a data node that carry out business computation for enterprises in a distributed network. The CPU usage of the data node in terms of percentage was used as indicated in Figure 6.3. When the CPU workload is high (i.e. above the CPU workload threshold), this indicates that the available CPU resources for negotiation are low. Similarly, when the CPU workload is low, it means that there are sufficient CPU resources for the negotiation. The proposed negotiation model uses the simulated data in determining the current negotiation state for each negotiation round. Figure 6.4 shows the simulated CPU workload of the gateway node with a value of 60% as the CPU workload threshold.

[11] http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains

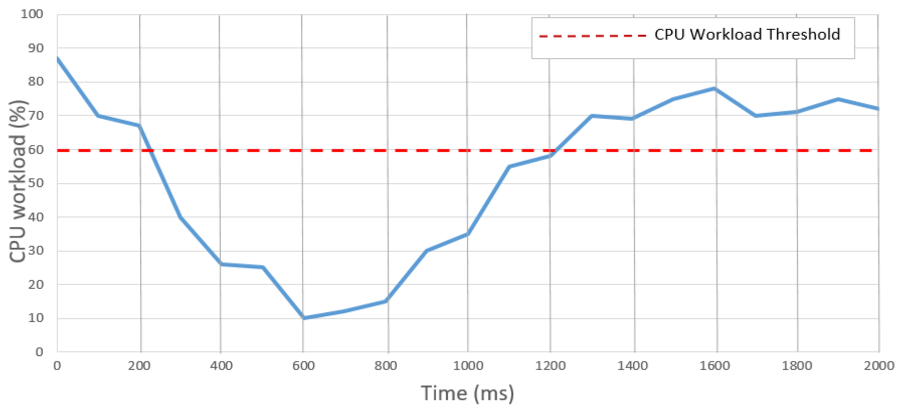


Figure 6.4:A simulation of the gateway node CPU workload

- **VirtualNodeLauncher:** It generates virtual node devices. Given that tens of plant nodes will be required to evaluate the scalability of *IoTQoSystem*, it was decided that virtual plant nodes will be used for both the multilateral and multiparty negotiation scenarios as a case study with scores of real plant nodes may be very expensive to set up. With the VirtulaNodeLaucher, multiple virtual plant nodes that emulate the actual plant nodes can be generated. Each generated virtual plant node is an independent Java thread characterised by a battery profile and a QoS profile Java instance. The battery profile simulates the LiPo battery used in the case study with its discharge profile illustrated in Fig 6.5. The LiPo battery used in powering the raspberry has a nominal voltage of 3.7v, a battery capacity of 3800mAh and a critical voltage of 3.3v. The critical voltage (3.3v), as shown in the figure, is the voltage below with which the charge controller (i.e. voltage regulator) disconnects the battery's load (i.e. raspberry pi). It is used as the threshold voltage for making changes in the QoS profile of IoT devices. The QoS profile for each virtual plant node is generated using the QoSMatrixLauncher.

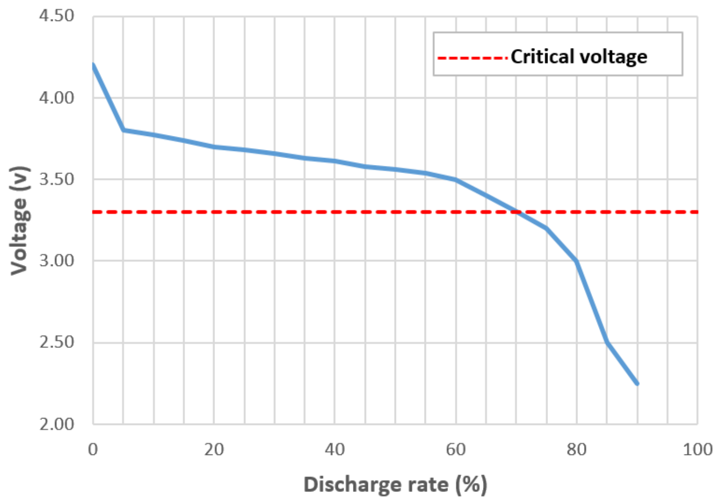


Figure 6.5:Battery discharge profile of an IoT device

- **QoSMatrixLauncher:** This is responsible for generating the permissible QoS parameter values representing plant node sensing and actuation configurations. Each plant node's sensing and actuation configurations are defined by the reserved value, preferred value, and weights of three QoS parameters: response time, availability, and throughput. These values are based on the *Donald Danforth Plant Science Center*[12] dataset and without loss of generality; their maximum and minimum values are 1.00 and 0.10, respectively. A plant node sensor configuration represents its QoS preference as a service provider, and its actuation configuration represents its QoS preference as a service consumer. The QoS preference allows the plant nodes to use the utility function equations described in Section 4.1.2 to map offers to utility values during the negotiation process. Table 6.1 shows an example of the initial QoS preferences of plant node1 and plant node2. It also captures the conflicting preferences between both plant nodes when one plays the role of a service provider, and the other plays the role of a service consumer and vice-versa. The definitions of the three QoS parameters are depicted in Table 6.2

Table 6.1: A simulation of the initial QoS preference of two plant nodes

QoS Parameters	**Plant Node1**						**Plant Node2**					
	Sensors Configuration			Actuators Configuration			Sensors Configuration			Actuators Configuration		
	RSV	**PFV**	**WGT**	**RSV**	**PFV**	**WGT**	**RSV**	**PFV**	**WGT**	**RSV**	**PFV**	**WGT**
Response time	0.85	0.42	0.39	0.67	0.92	0.33	0.81	0.34	0.25	0.62	0.85	0.41
Availability	0.94	0.77	0.40	0.76	0.95	0.28	0.93	0.81	0.47	0.70	0.89	0.32
Throughput	0.99	0.84	0.21	0.83	0.91	0.39	0.85	0.76	0.28	0.85	0.97	0.27

Legend	
PFV	Preferred value
RSV	Reserved value
WGT	Weight

Table 6.2: List of QoS parameters

QoS parameters	**Description**	**Maximum value**	**Minimum value**
Response time (milliseconds)	represents the duration of the IoT service invocation from 9:00 A.M to 4:00 P.M.	1000ms	100ms
Availability (percent)	represents the probability that the IoT service is ready for use from 9:00 A.M .to 4:00 P.M.	100%	10%
Throughput (service invocations per minute)	represents the number of IoT service invocations per minute from 9:00 A.M .to 4:00 P.M.	100spm	10spm

[12] https://plantcv.danforthcenter.org/pages/data.html

6.2 EVALUATION EXPERIMENTS

In conducting these experiments, the negotiation framework requires specific input parameters for its adopted negotiation strategy and prediction strategy, as described in Sections 4.3.2 and Section 5.2.2. The values of these input parameters were determined heuristically through a set of experiments to search for the space of optimum values and to understand the impact of the values of the different parameters on the negotiation results. The optimum values of these input parameters that generated the best results are shown in Appendix A.4. The case study for the experiments was set up with the service providers (i.e. plant nodes 2-4) having different QoS preferences, with plant node 2 having a QoS profile that best satisfies the IoT service request from the service consumer (plant node 1) and plant node 4 having the least desirable QoS profile for plant node 1 when their battery voltage level and network connectivity speed is above their threshold values.

6.2.1 Experiment 1: Reinforcement learning Negotiation Strategy Performance

This set of experiments evaluate the performance of the reinforcement learning negotiation strategy (Section 4.3.2) in bilateral negotiation scenarios. In these experiments, the negotiation framework uses the proposed negotiation strategy to establish a QoS agreement between a service consumer(plant node1) and a service provider. Plant node1 makes an IoT service request for a temperature and humidity sensor reading, which is to be used in controlling its water pump actuator. The CHOReOS middleware selects plant node 2, and the framework proceeds with the negotiation process between plant node1 and plant node 2 in a negotiation environment with incomplete information about the negotiation state transitions. With the establishment of a QoS agreement, the service consumer starts receiving sensor data from the service provider based on the details of the QoS agreement.

To demonstrate the performance of the negotiation strategy proposed, I compared the reinforcement learning negotiation model with the mixed strategy negotiation model described in [64]. The mixed negotiation model uses a random probabilistic model for the selection of a strategy during the negotiation process. This strategy was chosen for the comparison because it inspired the reinforcement learning negotiation strategy and provides a good balance between the success rate and social welfare. The common performance metrics: success rate and social welfare, as highlighted in [98], were used for the comparison. Fig. 6.5 shows the results of this experiment for both negotiation models with a varying number of negotiation rounds, with each round consisting of 10 negotiation runs.

The result shows that the proposed negotiation strategy outperforms the mixed strategy negotiation model regardless of the negotiation rounds. In terms of success rate, the reinforcement learning strategy model achieved an excellent score of 96.41%, as most of the negotiation were successful. The mixed strategy negotiation model had several negotiation failures when the negotiation deadline was short, as its overall success rate was 77.36%. This is reflected in the poor average social welfare for negotiation rounds that are not large. However, as the negotiation rounds increase, the mixed strategy negotiation model begins to find a solution for the conflicting QoS preferences, resulting in more successful negotiations.

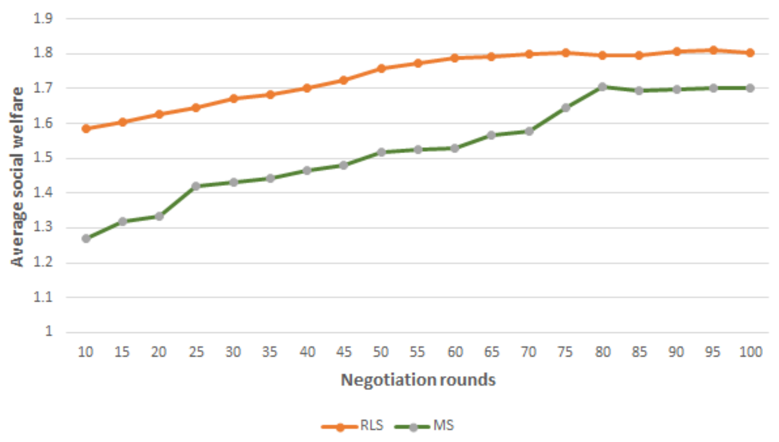


Figure 6.6:Performance results for the reinforcement learning model and the mixed strategy model

Furthermore, as shown in Figure 6.6, it takes the mixed strategy negotiation model significantly longer to reach its optimum social welfare compared to the proposed negotiation model. The reinforcement learning strategy was able to consistently attained its optimum social welfare with a minimum of 60 rounds, while it took a minimum of 80 rounds for the mixed strategy to reach its optimum social welfare. This performance improvement results from the reinforcement learning strategy's support for the intelligent selection of negotiation tactics, a QoS negotiation requirement (i.e RQ 3) poorly supported by the mixed negotiation strategy as highlighted in Table 3.2.

There was an average increase of 19.14% in the utility gained by the reinforcement negotiation strategy compared to the mixed strategy model. The utility gain is primarily a consequence of the reward scheme that maintained a balance between reaching an agreement before the deadline and a high social welfare for all negotiation durations. This allowed agents to still generate a high utility QoS agreement at short negotiation durations, unlike the mixed strategy that struggles to maintain this balance at short durations. As a result, most of the gains in utility comes from the negotiation sessions with smaller negotiation rounds, as indicated in Figure 6.6.

6.2.2 Experiment 2: QoS Violation Prediction

In evaluating the reliability of the framework, a set of experiments were conducted to demonstrate the ability of the framework to predict a degrading IoT service and initiate a service substitution before the service eventually fails. These experiments aim to predict a QoS violation in failing service scenarios in the case study. Essentially, the QoS monitor predicts the degradation of an IoT service being provided by the plant node representing the service provider in the following scenarios: (i) when its associated Raspberry Pi suffers from low power due to a significant drop in voltage below its critical voltage (ii) when its suffers from a reduced data transmission rate due to poor network connectivity.

- **Deteriorating power scenario**

  In this experiment, a degrading IoT service is set in motion when the battery's voltage of plant node 2 significantly depletes below its critical voltage as it sends the humiture sensor readings to plant node1 after a QoS agreement has been established. Using the QoS monitor of the framework, the response time defined in the QoS agreement as

520ms was monitored for a possible violation. A violation is predicted when the difference between the value of the predicted response time(r_p) and the value of the response time defined in the QoS agreement (r_a) for three consecutive periods goes beyond the threshold value of 10ms. These values were experimentally determined as they provide the best result for detecting QoS violations in the case study.Figure 6.7 shows the actual and predicted response values as the readings from the humiture sensor from plant node 2 is sent to plant node1.

The response time lags and wanes as the voltage of plant node 2 rapidly deteriorate, resulting in a prediction of a QoS violation when the response time was 530ms, a deviation from response time specified in QoS agreement (520ms). This prompted the framework to initiate a new QoS agreement with plant node 3, providing the substitute service. This substitution strategy allowed plant node 1 to continue receiving the sensor data without interruption caused by a power outage.

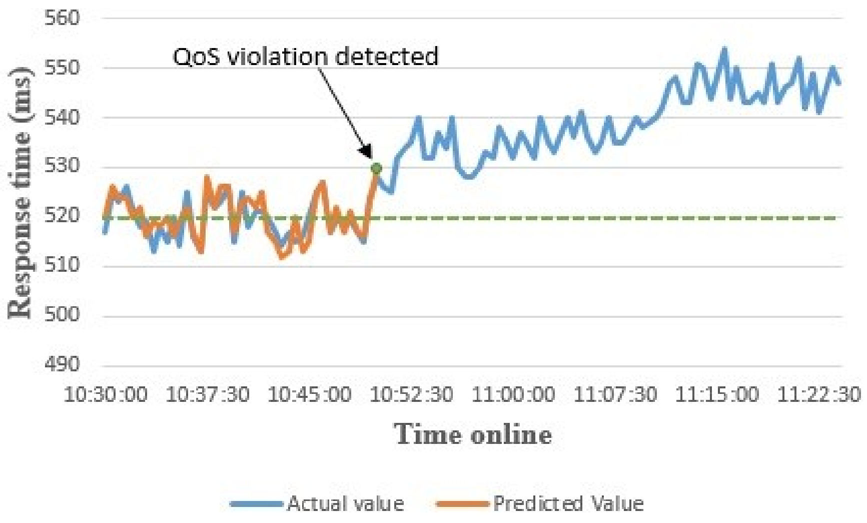


Figure 6.7:Predicting a response time violation using the dynamic tendency prediction strategy

- **Data transmission degradation scenario**

  In this experiment, a service failure was predicted and a renegotiation initiated when the data transmission rate of the initial service provider begins to decrease due to poor network connectivity. A QoS agreement was first established between plant node1 and plant node2 with the throughput defined as 25spm. As plant node1 begins to receive the sensor data from plant node 2, the network speed of plant node 2 was attenuated to trigger a QoS violation during the service provisioning. Similar to the low power scenario, A violation is predicted when the difference between the predicted throughput (tp) and the throughput defined in the QoS agreement (t_a) for three consecutive periods goes below the threshold value of 5.

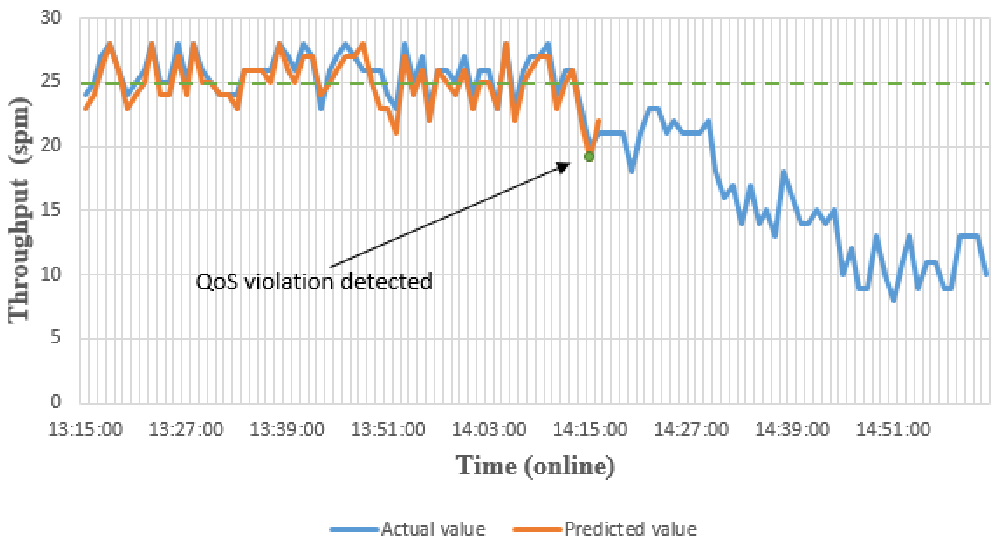


Figure 6.8:Predicting a throughput violation using the dynamic tendency prediction strategy.

Figure 6.8 shows the data trend of the predicted and actual throughput parameter as the network strength deteriorates. By monitoring the throughput values, the framework detected the degrading service at 14:15 when the predicted throughput value was 19spm. Between 14:13 and 14:15, the cumulative difference between t_p and t_a was above the permissible threshold, and as a result, a renegotiation led to the establishment of another QoS agreement with plant node3 to provide the senor readings to plant node1.

The results shown in Figure 6.7 and 6.8 indicates that the dynamic tendency prediction strategy provides a good accuracy in predicting a QoS violation during service provisioning. This stems from the fact that it tries to minimize the “change in direction” error by reducing the variation at possible turning points, as described in Section 5.2.2.

Another interesting observation from both experiments is that the prediction strategy had a Mean Absolute Percentage Error (MAPE) of 12.37% for the throughput time series measured every 60 seconds and a value of 8.62% for the response-time time series measured every 30 seconds. This difference suggests that the frequency of the data collection can affect the accuracy of the dynamic tendency prediction strategy. This observation confirms the claim made by the authors in [86] that there is a direct relationship between the accuracy of the dynamic tendency prediction strategy and the data collection frequency.

6.2.3 Experiment 3: QoS Profile Adaptability

These experiments evaluate the changes made in the QoS profile of plant nodes representing the service providers in the vertical farming system. The critical level of the plant nodes’ battery voltage(3.3V) and network connectivity speed (100kpbs) is used as the threshold value for making these changes. Specifically, the framework makes changes to the preferred and reserved values of each QoS parameter when the battery voltage or the network connectivity of the service providers falls below or rises above their respective critical levels. The changes are made based on a parameter variance $\delta(0.01 < \delta < 0.25)$, resulting in the new value for the negotiation space for each QoS parameter. Figure 6.9 shows an example of the changes made in the negotiation space for the throughput parameter across the three service providers. When the service providers' battery voltage or network connectivity drop below their respective critical levels, the preferred and reserved values of each QoS parameters reduce by δ. Similarly, when the battery voltage and network connectivity rise above their respective critical levels, each QoS parameter's preferred and reserved values increases by δ.

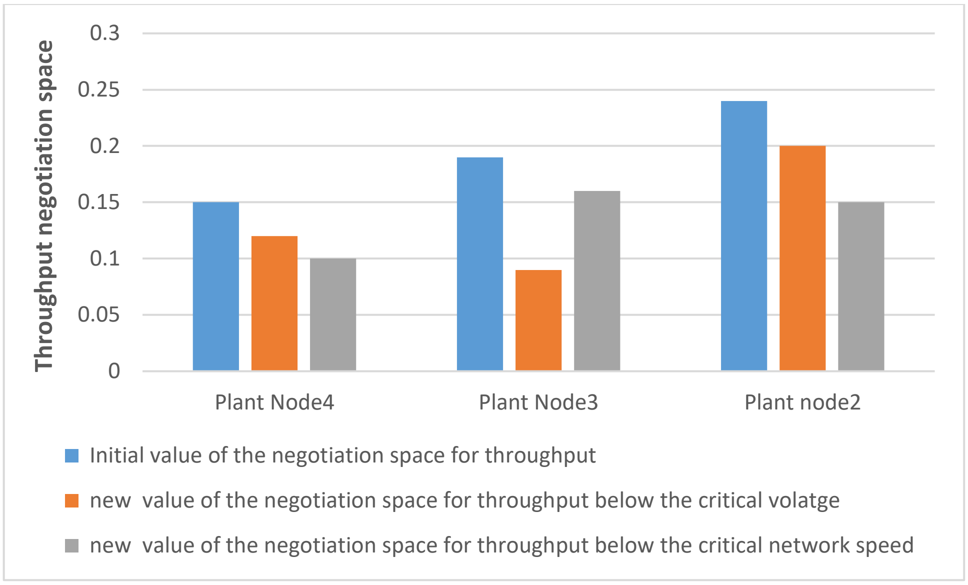


Figure 6.9:Plant node comparison of the throughput changes.

To illustrate the importance of a flexible QoS profile in this case study, a comparison is made to a situation where the QoS profile of a service provider is fixed before and after a QoS violation has been predicted

- **Pre-QoS violation detection scenario**
  This experiment demonstrates how a service failure can be avoided in the vertical farming system before the detection of a possible QoS violation. In achieving this, the experiment begins with a fixed QoS profile for plant node 2 with its preferred and reserved values of the QoS parameters, initially indicating that its battery level is above its respective threshold value. As the battery of plant node 2 deteriorates below 3.3V, plant node 1 queries the CHOReOS middleware for an IoT device(plant node) that best matches its request for temperature and humidity sensor values and plant node 2 was selected. The framework initiates a negotiation between plant node1 and plant node 2 and a QoS agreement was generated. However, plant node1 could not receive the humiture sensor readings, even though an agreement has been made between both plant nodes. This was due to the battery depletion of plant node 2 as the voltage had deteriorated below its cut-off voltage of 3V (Figure 6.10a), leading to a service failure.

  For comparison, the same experimental set-up was repeated but with all the service providers having a flexible QoS profile with plant node 2 having a QoS profile that reflects that its battery has deteriorated below 3.3V and plant nodes 3 and 4 having a QoS profile that its battery is above the critical voltage (Figure 6.10b). When plant node 1 queries the CHOReOS middleware for an IoT service, rather than plant node 2 being selected, plant node 3 was selected for the negotiation process with plant node 1. This was because the QoS profile of plant node 2 fell short of the service requirement since there is now a reduction in the negotiation space for its QoS parameters. With the establishment of the QoS agreement with plant node 3, plant node 1 successfully received the humiture readings from plant node 3.

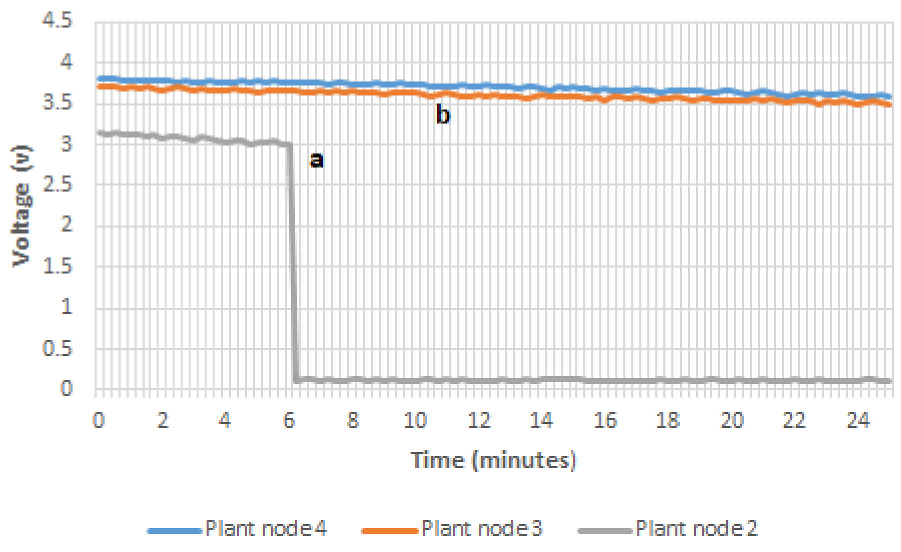


Figure 6.10:Voltage data used in changing the QoS profile of the service providers.

- **Post-QoS violation detection scenario**

This experiment demonstrates how a service failure can be avoided using an adaptable QoS profile as the framework provides a service replacement after the early detection of a failing service.

Firstly in this experiment, all the QoS profiles of the service providers were made to change based on the changes observed in their respective network speed except for the QoS profile of plant node 3, which was fixed. The current status of plant node 2 enabled it to be selected to provide sensor readings to plant node 1 after plant node 1 initiated an IoT request for a humiture sensor data to the middleware. As plant node 1 begins to receive the sensor data from plant node 2, the network speed of plant nodes 2 and 3 were attenuated, and the framework detected a QoS violation. This allowed plant node 3 to be selected as the replacement for plant node 2 as plant node 3 provides the next best QoS profile to fulfil the requirements of plant node 1. However, plant node 1 could not receive any sensor readings from plant node 3. This was because the QoS profile of plant node 3 did not change as its network data transmission deteriorated below its critical value (Figure 6.11a), and as a result, this led to a service failure.

A comparison was made with the same experimental set-up but with an adaptable QoS profile for all the service providers. It was observed that plant node 4 was selected instead of plant node 3 to replace plant node 2, which led to the continuous delivery of the humiture sensor values without service interruption or failure. This was because plant node 3 QoS preferences have changed to reflects its current declining network speed, and as a result, plant node 4 was selected as its QoS profile indicated that its network speed was above the network critical speed(Figure 6.11b).

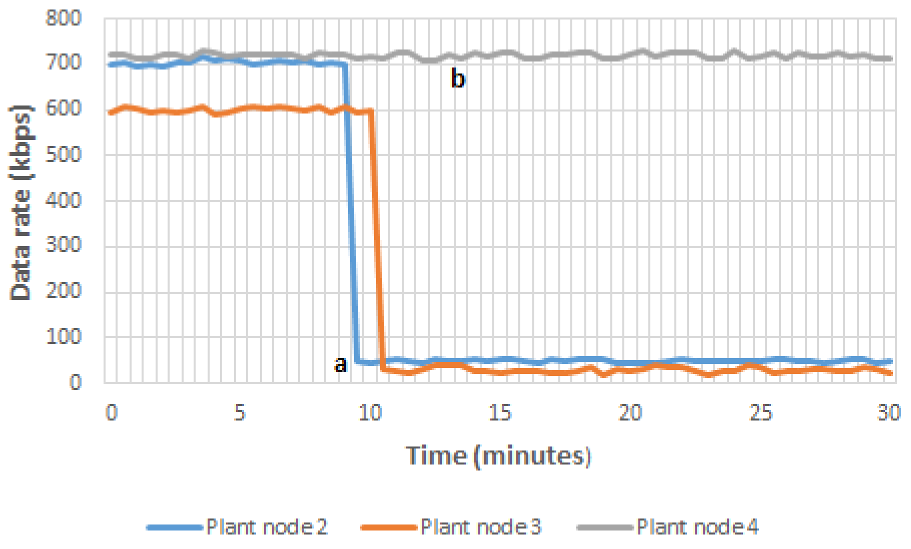


Figure 6.11: Network data rate values used in changing the QoS profile of the service providers.

The results from both experiments show how the external resources of IoT devices vary with time. These variations can affect the operations of IoT devices, and as a result, it becomes crucial for the QoS preferences of IoT devices to reflect their current constraints. As observed in both experiments, a fixed QoS preferences led to a service failure, while a dynamic QoS preference prevented the vertical farming system from failing during the service selection and provisioning process.

6.2.4 Experiment 4: Negotiation Model Scalability

This set of experiments evaluates the framework's ability to scale as the number of devices and negotiation increases. In this set of experiments, a relatively high utility QoS agreement was generated by the framework at varying scale and complexity. The scalability of the framework was evaluated using two negotiation approaches (i) multilateral negotiation, a negotiation where QoS agreements are reached by varying the number of devices participating in a negotiation (ii) multiparty negotiation, negotiations where QoS agreements are reached by varying the number of negotiation sessions the framework can handle concurrently.

- **Multilateral Negotiation**
  This experiment evaluates the scalability of the negotiation model in a series of multilateral negotiation scenarios as it seeks to generate a QoS agreement among several negotiation parties within a specific deadline. It involves three separate multilateral negotiation scenarios that consist of 4, 8 and 16 virtual nodes, respectively. The simulation module generated the set of virtual nodes in each negotiation scenario, and they consist of one service consumer and multiple service providers with conflicting QoS preferences. For each negotiation scenario, three sets of deadlines, 40 rounds, 50 rounds and 60 rounds, were used to understand the relationship between the social welfare and the negotiation duration. In this experiment, a comparison was also made in terms of the average sum of utilities(average social welfare) gained by all the negotiating participants in each negotiation scenario. Figure 6.12 illustrates the experimental results of each of the multilateral negotiations

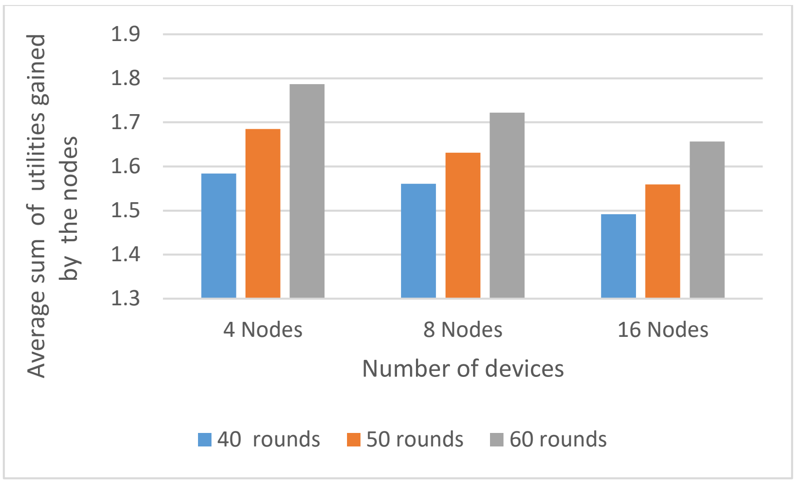


Figure 6.12: Average sum of utilities of the set of nodes over varying deadlines

Figure 6.12 shows the average sum of the utilities gained by the virtual nodes over 10 negotiation sessions for each deadline. The results suggest that the negotiation duration has a big impact on the success rate of multilateral negotiations, and there is a direct relationship between the negotiation duration and the success rate. Table 6.3 shows the breakdown of the percentage negotiation failure for each round across the three multilateral negotiation scenarios. Most of the failures were due to one or more of the nodes adopting a trade-off negotiation strategy in an attempt to increase their utility during the negotiation process.

Table 6.3:Percentage of negotiation failures for each multilateral negotiation scenarios

Number of rounds	Percentage of negotiation failures
40	13.66%
50	10.29%
60	6.37 %

In terms of the number of nodes for a specific deadline, The results from the experiments show the proximity of the social welfare across the three negotiation scenarios. The marginal decrease in the social welfare as the number of negotiating devices increases suggests the ability of the framework to scale without significant performance overhead.

- **Multiparty Negotiation**

This experiment investigates the scenario where the framework concurrently conducts multiple negotiations at various scales. In achieving this, the concurrent negotiation scenario was compared to the scenario where these multiple negotiations were carried out sequentially by the framework. This experiment involves generating QoS agreements for virtual nodes participating in a series of multi bilateral negotiations. The simulation module created the virtual nodes, and the multi bilateral negotiations were scaled from 2 bilateral negotiations to 12 bilateral negotiations. Figure 6.13 illustrates the percentage of time saved by conducting bilateral negotiations concurrently compared with sequentially.

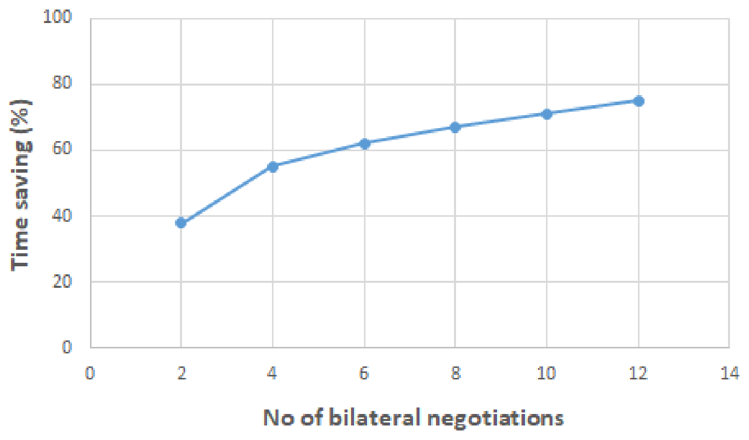


Figure 6.13: Percentage of time saved negotiating concurrently

As seen in Figure 6.13, the amount of time saved is proportional to the number of bilateral negotiations performed concurrently. The primary reason for this is that by negotiating concurrently, the time consumed by all the negotiation sessions is not more than the time consumed by the negotiation session with the largest deadline. Each negotiation session is only permitted to continue until its deadline is reached, and as a result, the longest time a negotiation session s, is allowed to continue is t^{s}_{max} . Consequently, the framework will stop all negotiation sessions at the longest period t = max (t^{1}_{max}, t^{2}_{max} …… t^{n}_{max}). In contrast to when the framework conducts each negotiation session sequentially, all the negotiation could be completed at t= (t^{1}_{max} + t^{2}_{max} + …… t^{n}_{max}) in the worst case.

6.3 EVALUATION SUMMARY

The developed negotiation framework, *IoTQoSystem*, has been evaluated using four main experiments to validate the thesis objectives outlined in Section 1.3. The first set of experiments described in Section 6.2.1 assessed the performance of the framework's negotiation strategy in a series of bilateral negotiation scenarios. The assessment involved comparing the performance of the proposed reinforcement learning with the mixed negotiation strategy. The results showed how the reinforcement learning strategy outperformed the mixed negotiation strategy in terms social welfare and success rate.

The second set of experiments described in Section 6.2.2 focused on how the framework proactively managed QoS violations in two service failure scenarios: battery voltage deterioration and data transmission degradation. Essentially, these experiments demonstrated how the framework detected a degrading service and initiated an early service replacement before the service eventually failed. The experimental results showed that the *IoTQoSystem* framework provided a good accuracy for the prediction of QoS violation in both scenarios

The third set of experiments described in Section 6.2.3 evaluated the flexibility of the framework in expressing the QoS preferences of IoT devices. These experiments demonstrated how the framework leveraged the QoS profile of IoT devices in averting service failures in the vertical farming system before and after a QoS violation was detected. The experimental results showed that updating the QoS profile of IoT devices in response to the changes in the device external resources increases an IoT system resilience.

Finally, the fourth set of experiments described in Section 6.2.4 evaluated the framework's ability to scale horizontally and vertically. These experiments demonstrated how the framework can generate QoS agreements among varying number of IoT devices and can perform multiple negotiations concurrently. The experimental results indicated that the framework can successfully perform both multilateral and concurrent negotiations.

Chapter 7

CONCLUSION

This chapter begins by providing a review of the research objectives described in Section 1.3. The achievement of these objectives is demonstrated through the implementation and evaluation of the proposed QoS negotiation framework discussed in chapter 5 and 6, respectively. The chapter then highlights the limitations of the work and discusses the lessons learnt. Finally, the chapter concludes by summarising the thesis findings and discussing the future directions that the development of *IoTQoSystem* may take

7.1 OBJECTIVES REVISITED

In this section, each of the research objectives is revisited, and a discussion is provided of how the reinforcement learning QoS negotiation framework satisfies these objectives. The objectives are based on the QoS negotiation requirements in IoT middleware and the limitation of current QoS negotiation initiatives for IoT systems. Each research objective is compared against the result from the evaluation experiments, showing how the key findings and contributions achieve their related objective.

- **Provide a reinforcement learning negotiation strategy for the generation and evaluation offers.** One of the major limitations of current QoS negotiation approaches is their inability to maintain a good balance between the total utility gained by each negotiating participants and the rate of successful negotiation. In a competitive negotiation environment where QoS preferences are kept private and the dynamics of the negotiation changes unpredictably, the higher the probability of generating a QoS agreement with a high social welfare, the lower the probability of such negotiation being successful. The *IoTQoSystem* framework described in this thesis solves this problem by using the context negotiation information such as the current negotiation state and the deadline criterion to decide the appropriate negotiation tactic to be utilized in the generation of offers that maximises the chances of reaching an agreement with high social welfare within the specified deadline. The experiments described in Section 6.2.1 shows how the reinforcement learning negotiation model outperformed the mixed negotiation model in a series of bilateral negotiation scenarios.
- **Provide proactive support for QoS violations through monitoring and renegotiation.** In addition to generating high utility QoS agreement, the *IoTQoSystem* framework provides a mechanism for monitoring the changes in the quality of the negotiated service and automatically initiating an early renegotiation for degrading IoT service. As demonstrated in Section 6.2.2, a failing service due to poor network connectivity and battery voltage deterioration was detected and using the measured data trend, the framework was able to predict a QoS violation. This resulted in an automatic renegotiation with another service provider, leading to an early service replacement.
- **Provide flexible support for the expression of QoS preferences.** The *IoTQoSystem* framework allows for the expression of multiple QoS constraints, which are defined in the QoS profile. It supports the updates of the QoS profile to reflect the current capability and needs of the associated IoT nodes. The framework periodically monitors

the underlying resources of IoT devices and uses this information to make the necessary changes needed in the QoS profile. The experiments in Section 6.2.3 illustrates how the QoS profile changes with the new information received about the IoT device external resources. Given that the framework uses Linked USDL in modelling the QoS profile, the QoS profile is capable of being extended to accommodate new IoT domain QoS parameters and, as a result, be used in different IoT contexts. This demonstrates the flexibility of the *IoTQoSystem* framework for expressing QoS preferences.

- **Provide a runtime solution that can scale.** The framework provides support for both multilateral and concurrent negotiations. *IoTQoSystem* uses a low-cost communication negotiation protocol, SOAP, to resolve the conflicting QoS preference between many IoT nodes and leverages on the programming concept of multithreading to perform several bilateral negotiations concurrently. The experiments in Section 6.2.4 shows that an increase in the number of IoT nodes and concurrent negotiations did not result in a proportional drop in the framework's performance. This suggests the framework ability to scale both horizontally and vertically without significant overhead.

7.2 REFLECTION

A number of design and implementation decisions were made during this research. These decisions may have impacted the implementation and evaluation of the *IoTQoSystem* framework described in Section 6.2. This section discusses these issues and provides a list of the lessons learnt.

7.2.1 Limitations

Although this thesis provides a solution towards resolving the issue of QoS contention between IoT devices, there are a number of limitations associated with the approach followed in providing this solution. These limitations are as follows:

- **Design Approach.** The *IoTQoSystem* architectural design is based on the principle of service-oriented computing. The advantage of this architectural design enables its components to be loosely coupled and easily extensible. However, this design approach limits its pluggability to only IoT middlewares that follow the same approach. Consequently, *IoTQoSystem* may not "pluggable" into IoT middlewares that use other design approaches such as TeenyLIME [93] and TS-Mid [94] that uses the tuple-space design approach. This factor limits the range of IoT middleware that the framework can provide QoS- aware and context-based dynamic negotiation to.

- **Evaluation Environment.** The evaluation environment introduces a limitation that arises from the maximum number of available TCP ports at an IP address. The host machine on which the simulation module was deployed has limited memory resources, and the Operating System is not configured to support unlimited processes during a multilateral negotiation scenario as each virtual plant node generated needs to be bound to a TCP port for the registration of its QoS profile. Consequently, the maximum number of TCP ports available at the host machine imposes a limit on the number of virtual nodes that can be generated and run concurrently.

- **Simulated Data.** The method of operation of the case study requires the use of a simulation module. The simulation module uses a suitable body of test data to simulate the dynamic nature of IoT and initialize the QoS configurations of the sensors and actuators. This technique provides a convenient way of evaluating the framework as it mitigates the difficulty of engineering the QoS requirements and constraints that accurately depict each plant node's utility function and eliminates the overhead cost of a live deployment with a large number of plant nodes. However, the use of simulated data may introduce a threat to the internal validity of the research results. To reduce the impact of this threat, the simulation was based on real-world datasets, and non-parametric tests were performed to analyse the results with no constraints imposed on the distribution of the dataset.

- **Renegotiation via service substitution**. Modifying an existing QoS agreement through renegotiation with the same service provider increases the trustworthiness of the service being provided and reduces the overhead of service substitution [97]. However, the *IoTQoSystem* framework uses a negotiation protocol that does not support this form of renegotiation. Specifically, the SAOP negotiation protocol does not allow agents to initiate a renegotiation within the same negotiation session as agents can only make an offer(indicating that it has rejected the offer presented by its negotiating counterpart) and accept an offer as shown in equation 4.13. As a result, renegotiation is only possible with the execution of another negotiation session with a different service provider. However, renegotiation via service substitution has added benefits as there is a higher chance that renegotiation with the same service provider could lead to another QoS violation.

7.2.1 Lesson Learned

The development of the framework began with all the data objects and actions being managed by a single tightly coupled codebase. All the classes and methods resided on one software instance during the early stage of the framework development. As the codebase grew, it became obvious that there needs to be a change in the software design as it was increasingly difficult to update parts of the software system without having a ripple effect of changes in other parts of the framework. As a result of the maintainability issues associated with the initial software design, the microservice software development pattern was adopted to reduce the programmatic development risk and improve the synchronization of the various modules of the framework. This design decision to transform logical components of the framework into services made the framework easier to manage and resilient.

In implementing a prediction strategy for the framework, the popular classical statistical model for time series, Autoregressive Integrated Moving Average (ARIMA) model [101], was initially adopted. This model produced a good prediction, however, it was slow and time-consuming as each time a new QoS parameter value occurs, the model needs to be retrained to forecast the next value. This made the ARIMA model less effective for the framework, and as a result, the dynamic tendency prediction strategy [86] was adopted as it provides a simple and fast approach for real-time series forecasts

7.3 FUTURE WORK

This section explores the ways in which the development of the QoS Negotiation framework may take in future revisions. This includes possible improvements that can be made to its design and features that could provide a better approach for the management of QoS agreement in a dynamic IoT environment. The discussion for future directions is as follows:

- **Provide a non-linear time series forecasting model.** The *IoTQoSystem* framework can monitor negotiated IoT service and predict violations of QoS parameters. The adoption of a service degradation prediction strategy enhances proactive QoS management by avoiding possible service failures. The QoS violation prediction is based on the dynamic tendency forecasting model. This model provides a good and less expensive technique to model the dynamic features of QoS parameters and forecast future values. However, the prediction strategy assumes that the measured QoS data is serially dependent and normally distributed. This assumption makes it difficult for the prediction model to accurately predict a QoS violation when the time-varying variation of the QoS data is steeply non-linear. If there is a sharp change in the QoS data trend due to a network or hardware glitch, the prediction strategy could struggle to predict the next QoS value accurately. As such, any implementation of forecasting within future revisions of this work must provide a mechanism in dealing with such QoS data outliers.
- **Provide support for multiple utility functions.** A potential improvement to the *IoTQoSystem* framework is to support a variety of utility functions, including utility functions that are custom-built for a specific IoT domain case study. Currently, the framework only supports a general utility function that may not be suitable for bespoke IoT systems. Future iterations of the framework should support multiple utility functions so as increase the framework robustness.
- **Provide support for different QoS agreement RDF data formats.** An important design decision was for the QoS agreement to be expressed in a mutually understandable format that maximizes syntactic and semantic interoperability (Section 5.12). This led to the adoption of linked USDL for modelling the QoS agreement. The QoS agreement is specified using the Terse RDF Triple Language, Turtle(.ttl), as it provides a format that is easily readable by humans. However, there are several RDF formats with which the QoS agreement can be expressed in such as N-Triples(.nt), JSON-LD(.json) and RDF/XML(.rdf). One way of improving the existing interoperability of the framework is to allow it to support these different RDF formats in future revisions.

7.4 FINAL REMARKS

The Internet of Things (IoT) marks a radical technology revolution as it promises to connect existing and future physical objects to the internet. With IoT, physical world objects can be embedded with identification, sensing, networking and computing capabilities that will allow them to communicate with one another over the Internet to accomplish some objectives [96]. As these devices communicate with each other, there have been concerns about the Quality of Service (QoS) associated with the services used in exposing the functionalities of these devices that enables the successful execution of an actuation task. These services are provided by devices with QoS configurations different from the devices requesting them and, as a result, creating a QoS contention between the service consumer and provider. In an environment where the interests of both service provider and consumer differs, the adoption of a negotiation mechanism becomes necessary for the realization of an actuation task.

To this end, this thesis presents a reinforcement learning negotiation strategy that effectively resolves the QoS contention between service providers and consumers. This work has presented a review of the current approaches used in managing the QoS negotiation and discusses how they fall short of the QoS negotiation requirements in IoT middleware. To address the limitations identified with the existing QoS negotiation research initiatives for IoT services, this thesis also presented a framework that uses a machine learning paradigm in establishing the QoS agreement and proactively managing the QoS violation in a dynamic IoT environment. The framework assumes the availability of a service-oriented IoT middleware that provides device discovery and QoS profile registration.

The evaluation of the developed framework demonstrates how a high utility agreement can be achieved by allowing devices represented by software agents to dynamically adapt their negotiation strategy using a model-based reinforcement learning as their QoS preferences evolves due to changes in the physical world. Indeed, in addition to establishing the QoS agreement with high social welfare, the evaluation illustrates the capability of the *IoTQoSystem* framework to proactively manage QoS violation through service failure forecasting and renegotiation.

Although this thesis has satisfactorily achieved all its objectives by developing the *IoTQoSystem* framework, possible future work should investigate integrating a forecasting model that can accurately predict QoS violation for QoS data that are heterogeneously time-varying without any overhead. The investigation should also include how the framework can support more RDF data format for it to be richly interoperable.

References

[1] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli , H. Sundmaeker , A. Bassi , IS. Jubert , M. Mazura ,M. Harrison , M. Eisenhauer , P. Doody. "Internet of things strategic research roadmap," in *Internet of Things: Global Technological and Societal Trends*, vol. 1, pp. 9–52, Jun. 2011.(cit. on p.1, 19, 20).

[2] Recommendation IT. Next Generation Networks-Frame-works and functional architecture models. ITU-T, Dec. 2004. (cit. on p.1).

[3] O. Vermesan, and F. Peter, eds. *Internet of things-from research and innovation to market deployment.* vol. 29, Aalborg: River publishers, 2014, ISBN: 9788793102941 (cit. on p. 1, 4)

[4] R. Minerva , A. Biru , D. Rotondi . Towards a definition of the Internet of Things (IoT). IEEE Internet Initiative. no. 1, pp. 1- 86. May 2015. (cit. on p.1).

[5] F. K. Shaikh, S. Zeadally and E. Exposito, "Enabling Technologies for Green Internet of Things," in *IEEE Systems Journal*, vol. 11, no. 2, pp. 983-994, June 2017. (cit. on p.2).

[6] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015. (cit. on p.2).

[7] G. Choudhary and A. K. Jain, "Internet of Things: A survey on architecture, technologies, protocols and challenges,"*2016 International Conference on Recent Advances and Innovations in Engineering*, Jaipur, pp. 1-8, 2016.(cit. on p.2).

[8] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang and W. Zhao,"A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," in *IEEE Internet of Things Journal*, vol.4, no.5, pp. 1125-1142, Oct. 2017. (cit. on p.2).

[9] I.S. Udoh and G. Kotonya, "Developing IoT applications: challenges and frameworks" in *IET Cyber-Physical Systems: Theory & Applications,* vol. 3, no. 2, pp. 65 -72, Jul. 2018. (cit. on p. 4, 7).

[10] IDC, " The Growth in Connected IoT Devices Is Expected to Generate 79.4ZB of Data in 2025, According to a New IDC Forecast", 2019. Accessed: 2020-03-30. [Online] Available: https://www.idc.com/getdoc.jsp?containerId=prUS45213219 (cit. on p. 4).

[11] CISCO, "Global Mobile Networks Will Support More Than 12 Billion Mobile Devices and IoT Connections by 2022", 2019, Accessed: 2020-03-30. [Online] Available: https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1967 403 (cit. on p. 4).

[12] I. Udoh and G. Kotonya, "A Dynamic QoS Negotiation Framework for IoT Services" in IEEE Global Conference on Internet of Things (GCIoT), pp. 1-7 Dec. 2019 (cit. on p. 7, 21, 24).

[13] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, New Jersey: Wiley and Sons, 2014, ISBN: 9781118625873. (cit. on p. 7, 41, 50).

[14] I.S Udoh, and G. Kotonya, “A Reinforcement Learning QoS Negotiation Model for IoT Middleware” in *International Conference on Internet of Things, Big Data and Security (IoTBDS)*,pp. 205-212, May 2020, (cit. on p.7).

[15] J. Koehler and G. Alonso 2007, “Service-Oriented Computing ”, 2007, Accessed: 2020-03-21. [Online] Available: https://ercim-news.ercim.eu/images/stories/EN70/ EN70-web.pdf (cit. on p. 9).

[16] T. Earl, *Service-Oriented Architecture: Analysis and Design for Services and Microservices*,Pearson Education, 2016, ISBN: 9780133858709 (cit. on p. 9).

[17] I. Sommerville, *Software Engineering*, Addison-Wesley, 2015, ISBN-13: 978-0-13-703515-1 (cit. on p. 9, 10, 63).

[18] M. Turner, D. Budgen and P. Brereton, "Turning software into a service," in *Computer*, vol. 36, no. 10, pp. 38-44, Oct. 2003 (cit. on p. 10).

[19] A. Barros, and D. Oberle, *Handbook of service description,* New York: Springer-Verlag, 2012, ISBN: 9781461418634 (cit. on p. 10).

[20] R. Wen, Y. Ma and X. Chen, "ESB Infrastructure's Autonomous Mechanism of SOA," in *International Symposium on Intelligent Ubiquitous Computing and Education*, pp. 13-17, 2009(cit. on p.11).

[21] I. Nadareishvili, R. Mitra, M. McLarty and M. Amundsen, *Microservice architecture: Aligning principles, practices, and culture,* O'Reilly Media, 2016, ISBN: 9781491956250 (cit. on p.11).

[22] T. Cerny ,M. J. Donahoo , J. Pechanec, “Disambiguation and comparison of SOA, Microservices and self-contained systems” in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pp. 228-235, Sep. 2017 (cit. on p.11).

[23] M. Richards, *Microservices vs. service-oriented architecture*, 2016, Accessed: 2020-05-04. [Online] Available: https://www.oreilly.com/radar/microservices-vs-service-oriented-architecture/ (cit. on p. 11, 62).

[24] P. Richardo, E. Thomas , and M. Zaigham, *Cloud Computing: Concepts, Technology and Architecture,* Pearson, 2013, ISBN: 9780133387568 (cit. on p. 12).

[25] K. Divya and S. Jeyalatha, "Key technologies in cloud computing," in *International Conference on Cloud Computing Technologies, Applications and Management* (ICCCTAM), pp. 196-199, 2012 (cit. on p. 12).

[26] B. David, H. Hugo ,M. Francis ,N. Eric , C. Michael, F. Chris, O. David ,*Web Services Architecture*, 2004, Accessed: 2020-07-18, [Online] Available: https://www.w3.org/ TR/ws-arch/#whatis (cit. on p. 12).

[27] G. Martin, H. Marc, M. Noah, M. Jean-Jacques, F.N. Henrik, K. Anish and L. Yves, *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, 2007, Accessed: 2020-05-11, [Online] Available: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ (cit. on p. 12).

[28] D. Booth, and K. Liu, *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, 2007, Accessed: 2020-05-11, [Online] Available: http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626 (cit. on p.12, 61).

[29] C. Luc Clement, H. Andrew, R. Claus and R. Tony Rogers, *UDDI Specification Technical Committee Draft*, 2004, Accessed: 2020-05-13, [Online] Available: http://www.w3.org /TR/2007/REC-wsdl20-primer-20070626 (cit. on p. 12).

[30] A. Alexandre, A. Assaf, A. Sid, B. Charlton, B. Ben, C. Francisco, F. Mark,G. Yaron, G. Alejandro, K. Neelakantan, KL. Canyang, K. Rania, K. Dieter, M. Mike, M. Vinkesh, T. Satish, R. Danny, TY. Prasad and Y. Alex, *Web Services Business Process Execution Language (WS-BPEL)* 2007, Accessed: 2020-05-13, [Online] Available: http://docs.oasis-open.org/wsbpel/2.0/plnktype (cit. on p. 12).

[31] R.T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures*, vol. 7. Irvine: University of California, 2000. (cit. on p. 12)

[32] S. Benbernou, I. Brandic, C. Cappiello, M. Carro, M. Comuzzi, A. Kertész, K. Kritikos, M. Parkin, B. Pernici and P. Plebani, "Modeling and negotiating service quality" in *Service research challenges and solutions for the future internet*, Berlin: Springer, pp. 157-208, 2010 (cit. on p. 14)

[33] K. Eunju, L. Yongkon, K. Yeongho, P. Hyungkeun, K. Jongwoo, M. Byoungsun, Y. Junghee and K. Guil *Web Services Quality Factors Version 1.0*, 2012, Accessed: 2020-03-22, [Online] Available: http://docs.oasisopen.org/wsqm/wsqf/v1.0/WS-Quality-Factors.pdf (cit. on p. 14, 45).

[34] O. Marc, M. Jordi and F. Xavier, "Quality models for web services: "A systematic mapping", in *Information and Software Technology*, vol. 56, no. 10, pp. 1167-1182, Oct. 2014 (cit. on p. 14).

[35] K. Kyriakos, P. Barbar, P.Pierluigi, C. Cinzia, C. Marco, B. Salima, B. Ivona, K. Attila, P. Michael and C. Manuel, "A survey on service quality description" in *ACM Computing Surveys.* vol. 46, no. 1, pp. 1-58, Jul. 2013 (cit. on p. 15)

[36] M. Moghaddam, and J. Davis, "Service Selection in Web Service Composition: A Comparative Review of Existing Approaches" in *Web Services Foundations*, New York; Springer, pp.321-346, 2014 (cit. on p. 15)

[37] N. R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. "Automated Negotiation: Prospects, Methods and Challenges." in *International Journal of Group Decision and Negotiation*, vol.10 no.2, pp.199-215, 2001 (cit. on p. 15)

[38] F. Curbera F, P. Hallam-Baker, V. M. Hondo, A. Nadalin, N. Nagaratnam and C. Sharp, *Web services policy framework (WS-Policy)*, 2006, Accessed: 2020-03-22, [Online] Available: http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf (cit. on p. 16).

[39] M. Comuzzi and B. Pernici, "An Architecture for Flexible Web Service QoS Negotiation", in *Proc. IEEE Int'l Enterprise Distributed Object Computing (EDOC) Conference*, pp. 70-82, Sep. 2005 (cit. on p. 6,16, 17, 19).

[40] F. Zulkernine and P. Martin, "An adaptive and intelligent SLA negotiation system for Web services", in *IEEE Transactions on Services Computing* vol. 4, no. 1, pp. 31–43, 2011(cit. on p. 17, 19).

[41] K. Hashmi, A. Alhosban, Z. Malik, B. Medjahed, and Benbernou, S. "Automated negotiation among web services" in *Web Services Foundations* , New York: Springer, pp. 451-482, 2014(cit. on p.18, 19).

[42] A. Abdelatey, M. Elkawkagy, A. El-Sisi, A. Keshk, "A Multilateral Agent-Based Service Level Agreement Negotiation Framework", in *International Conference on Advanced Intelligent Systems and Informatics*, NewYork: Springer, vol 533, pp. 576-586, Oct. 24 2017(cit. on p. 6, 18, 19).

[43] H. Chen, X. Liu, H. Xu and C. Wang , " A cloud service broker based on dynamic game theory for bilateral SLA negotiation in cloud environment" in *International Journal of Grid and Distributed Computing*, vol. 9, no. 9, pp. 251-268, Jan. 2016 (cit. on p.18, 19).

[44] S. Anithakumari and K. Chandrasekaran, "Negotiation and monitoring of service level agreements in cloud computing services ", in *Proceedings of the International Conference on Data Engineering and Communication Technology*, Singapore: Springer, pp. 651-659, 2017 (cit. on p. 18, 19).

[45] T. Edu-yaw and E. Kuada, "Service Level Agreement Negotiation and Monitoring System in Cloud Computing ", *IEEE 7th International Conference on Adaptive Science & Technology (ICAST)*, pp. 1-8, 2018 (cit. on p. 19, 19).

[46] D. Khellaf , H. Kenatef and O. Hioual, "Towards a SaaS Contracts Negotiation Model Based on a Multi Agent System" in *International Conference on Advanced Intelligent Systems for Sustainable Development* , vol. 1105, pp. 483-495, Jul. 2019, (cit. on p. 19, 19)

[47] S. K. Mohalik, M. B. Jayaraman, B. Ramamurthy, and A. Vulgarakis, "SOA-PE : A Service-Oriented Architecture for Planning and Execution in Cyber-Physical Systems," in *Proceedings International Conference on Smart Sensors and Systems (IC-SSS-2015)*, pp. 1-6, Dec. 2015. (cit. on p. 25)

[48] S. Haller, A. Serbanati, M. Bauer and F. Carrez, "A domain model for the internet of things", in *IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing* , pp. 411-417, Aug. 2013, (cit. on p. 25)

[49] A. Bassi , M. Bauer, M. Fiedler, R. van Kranenburg , S. Lange, S. Meissner and T. Kramp, *Enabling things to talk.* Springer Nature, 2013, ISBN: 9784431543947 (cit. on p. 27, 28)

[50] S. Meyer, K. Sperner, C. Magerkurth and J. Pasquier, " Towards modeling real-world aware business processes", in *Proceedings of the Second International Workshop on Web of Things* pp. 1-6, Jun 12, 2011(cit. on p. 28)

[51] S. K. Mohalik, N.C. Narendra, R. Badrinath and Le DH, "Adaptive service-oriented architectures for cyber physical systems", in *IEEE symposium on service-oriented system engineering (SOSE)*, pp. 57-62, Apr. 2017, (cit. on p. 22)

[52] V. Issarny, G. Bouloukakis, N. Georgantas and B. Billet, “Revisiting service-oriented architecture for the IoT: a middleware perspective” in *International conference on service-oriented computing,* Springer, pp. 3-17, Oct. 2016 (cit. on p. 20, 22).

[53] S. Xie and Z. Chen, “Anomaly detection and redundancy elimination of big sensor data in internet of things”, *arXiv preprint, arXiv:1703.03225*, Mar. 9 2017 (cit. on p. 23).

[54] M. Eisenhauer, P. Rosengren, and P. Antolin, “Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems,” in *The Internet of Things*. New York : Springer, pp. 367–373, 2010. (cit. on p. 32).

[55] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, “Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of web services,” *IEEE Transaction Services Computing*, vol. 3, no. 3, pp. 223–235, Jul. 2010 (cit. on p. 32).

[56] M. A. Razzaque, Milojevic-Jevric, M., Palade, A. and S. Clarke, “Middleware for internet of things: a survey”,in *IEEE Internet of things Journal*, vol. 3, no 1, pp.70-95, Nov. 2015 (cit. on p. 20, 31,32).

[57] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, “A survey of middleware for internet of things” in *Recent trends in wireless and mobile networks* , Berlin: Springer, pp. 288-296, Jun. 2011 (cit. on p. 32)

[58] X. Zheng, and P. Martin, and K. Brohman, “Cloud service negotiation: Concession vs. tradeoff approaches, in *Proceeding 12th IEEE/ACM International Symposium Cluster, Cloud Grid Computing*, pp. 515–522, May 2012 (cit. on p. 34, 48, 57).

[59] E. Mingozzi, G. Tanganelli, and C. Vallati, “A framework for qos negotiation in things-as-a-service oriented architectures,” in *Wireless Communications,Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE)*, pp. 1–5, May 2014 (cit. on p. 6, 34).

[60] X. Zheng, and P. Martin, K. Brohman and Da Xu, L., “Cloud Service Negotiation in Internet of Things Environment: A Mixed Approach” in *IEEE Transactions on Industrial Informatics*, vol. 10 no. 2, pp. 1506-1515, 2014. (cit. on p. 34).

[61] K. Mišura and M. Žagar, Negotiation in internet of things. in *Automatika: časopis za automatiku, mjerenje, elektroniku, računarstvo i komunikacije*; vol. 57 no.2, pp. 304-18, 2016 (cit. on p.6, 36).

[62] W. A. Ghumman, A. Schill, A and J. Lässig, “The Flip-Flop SLA Negotiation Strategy Using Concession Extrapolation and 3D Utility Function”, in *IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pp. 159-168, Nov.2016 (cit. on p. 6, 36).

[63] K. Alanezi, and S. Mishra, “A privacy negotiation mechanism for IoT” in *IEEE 16th International Conference on Dependable, Autonomic and Secure Computing,* pp. 512-519, Jan. 2019 (cit. on p. 6, 37)

[64] F. Li, and S. Clarke, “A Context-Based Strategy for SLA Negotiation in the IoT Environment”, *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)* pp. 208-213, Mar.2019 (cit. on p. 37).

[65] R. S.Sutton and A.G Barto, *Reinforcement learning: An introduction*. MIT Press, Oct. 2018, ISBN: 9780262039246 (cit. on p. 41, 52, 64).

[66] N. R. Jennings, K. Sycara, and M. Wooldridge, “A roadmap of agent research and development” in *Autonomous Agents and Multi-Agent Systems, vol. 1,* no.1, pp. 7-38, Mar. 1998 (cit. on p 41).

[67] W. Brenner,R. Zarnekow, and H. Wittig. *Intelligent software agents: foundations and applications*. Springer Science and Business Media, Dec.2012. ISBN:9783642804847, (cit. on p. 41).

[68] D. Besanko and R. R. Braeutigam, *Microeconomics*, Wiley, Nov 2013, ISBN: 9781118572276 (cit. on p. 43).

[69] X. Jin, S. Chun, J. Jung and K. Lee, "IoT Service Selection Based on Physical Service Model and Absolute Dominance Relationship," in *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pp. 65-72, Nov. 2014, (cit. on p. 45).

[70] R. Aydoğan, D. Festen, K. V. Hindriks and C. M. Jonker. “Alternating Offers Protocols for Multilateral Negotiation”. in *Modern Approaches to Agent-based Complex Automated Negotiation. Studies in Computational Intelligence,* Springer, pp. 153-167, 2017, (cit. on p. 15 , 46).

[71] P. Faratin, C. Sierra, and N. Jennings, “Negotiation Decision Functions for Autonomous Agents,” in *International Journal of Robotics and Autonomous Systems*, vol. 24, nos. 3/4, pp. 159-182, Sep. 1998, (cit. on p. 48)

[72] P. Faratin, C. Sierra, and N. Jennings, “Using similarity criteria to make issue trade-offs in automated negotiations” in *Artificial Intelligence*, Elsevier, vol. 142, no. 2, pp. 205-237, Dec. 2002, (cit. on p. 49)

[73] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach,* California: Pearson, 2014, ISBN: 9780136042594, (cit. on p. 54)

[74] H. M. Schwartz, H. M. *Multi-Agent Machine Learning: A Reinforcement Approach*, New Jersey: Wiley and Sons, Sep. 2014, ISBN: 9781118362082, (cit. on p. 54)

[75] E. Alpaydin, *Introduction to Machine Learning,* Cambridge, MA: MIT Press., USA, Mar. 2020, ISBN: 9780262043793, (cit. on p. 54).

[76] M. Campione, and K. Walrath. *The Java Tutorial: Object-Oriented Programming for the Internet (Book/CD)*, Boston, MA: Addison-Wesley Longman Publishing Co., Inc., Mar. 1998, ISBN:9780201310078, (cit. on p. 61)

[77] Eclipse Foundation, *IoT Developer Survey 2019 Results*, 2019, Accessed: 2020-05-12, [Online] Available: https://outreach.eclipse.foundation/download-the-eclipse-iot-developer-survey-results (cit. on p. 61)

[78] M. J. Hadley, *Web Application Description Language*, Aug. 2009, Accessed: 2020-05-11, [Online] Available: http://www.w3.org/Submission/wadl/ (cit. on p. 61).

[79] J. M. García, P. Fernández, C. Pedrinaci, M. Resinas, J. Cardoso, and A. Ruiz-Cortés, "Modeling service level agreements with linked USDL agreement" in *IEEE Transactions on Services Computing*, vol. 10, no. 1, pp. 52-65, Jul. 2016, (cit. on p. 61).

[80] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data - TheStory So Far," in *Semantic services, interoperability and web applications: emerging concepts*, pp. 205–227, 2011, (cit. on p. 61).

[81] L. Bass, C. Paul and R. Kazman. *Software architecture in practice*. Addison-Wesley Professional, Aug. 2004, ISBN: 0321154959, (cit. on p. 62).

[82] J. Thönes, "Microservices," in *IEEE Software*, vol. 32, no. 1, pp. 116-116, Jan.-Feb. 2015, (cit. on p. 62).

[83] A. B Hamida, F. Kon, N. Lago, A. Zarras, D. Athanasopoulos, D. Pilios, P. Vassiliadis, N. Georganta, V. Issarny, G. Mathioudakis and G. Bouloukakis, *Integrated CHOReOS middleware-Enabling large-scale, QoS-aware adaptive choreographies,* Dec. 2013, Accessed: 2020-07-21, [Online] Available: https://hal.inria.fr/hal-00912882/ document (cit. on p. 66).

[84] J. Cardoso, and P. Carlos "Evolution and overview of linked USDL", in *International Conference on Exploring Services Science*, Springer, pp. 50-64, Feb.2015, (cit. on p. 67).

[85] F. Bellifemine, F. Bergenti, G. Caire and A. Poggi, "A Java Agent Development Framework", in *Multi-Agent Programming. Multiagent Systems*, Boston, MA: Springer, pp. 125-147, 2005, (cit. on p. 73).

[86] L. Yang, I. Foster and J. M. Schopf, "Homeostatic and tendency-based CPU load predictions," in *Proceedings International Parallel and Distributed Processing Symposium*, pp. 9- pp., Apr. 2003. (cit. on p. 75).

[87] A. Azarian, and A. Siadat, "Synthesis of Software Evaluation Methodologies and the Proposal of a New Practical Approach" in *JSW*, vol. *6*, no. 11, pp. 2271-2281, Nov. 2011, (cit. on p. 79)

[88] G. Gediga, K. C. Hamborg and I. Düntsch, "Evaluation of software systems", *Encyclopedia of computer science and technology*, vol. 45, no.30 ,pp. 127–53, 2002, (cit. on p. 79).

[89] O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow and M. N. Hindia, "An Overview of Internet of Things (IoT) and Data Analytics in Agriculture: Benefits and Challenges," in IEEE Internet of Things Journal, vol. 5, no. 5, pp. 3758-3773, Oct. 2018, (cit. on p. 81).

[90] M. I. H. bin Ismail and N. M. Thamrin, "IoT implementation for indoor vertical farming watering system," in *International Conference on Electrical, Electronics and System Engineering (ICEESE)*, 2017, pp. 89-94, Nov. 2017, (cit. on p. 81).

[91] J.C. Zhao, J.F. Zhang, Y. Feng and J. X. Guo, "The study and application of the IoT technology in agriculture", in *International Conference Computer Science and Information Technolology*, vol. 2, pp. 462-465, Jul. 2010, (cit. on p. 81).

[92] G. Bouloukakis, N. Georgantas, S. Dutta, V. Issarny, "Integration of heterogeneous services and things into choreographies", in *International Conference on Service-Oriented Computing* Springer, pp. 184-188, Oct. 2016 (cit. on p. 24)

[93] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco, "TeenyLIME: Transiently shared tuple space middleware for wireless sensor networks," in *Proceedings of the international workshop on Middleware for sensor networks*, pp. 43-48, Nov. 2006. (cit. on p. 98, 31)

[94] R. de Cassia Acioli Lima, N. S. Rosa, and I. R. L. Marques, "TS-Mid: Middleware for wireless sensor networks based on tuple space," in *International Conference on Advanced Information Networking and Applications-Workshops*, pp. 886-891, Mar. 2008. (cit. on p. 32, 98)

[95] K. K. Patel and S. M. Patel, "Internet of Things - IoT: Definition, characteristics, architecture, enabling technologies, application & future challenges" in *International Journal of Engineering Science and Computing*, vol. 6, no. 5, pp. 6122-6131, May 2016. (cit. on p. 2)

[96] A. Whitmore, A. Agarwal, and L. Da Xu, "The Internet of Things-A survey of topics and trends," in *Information Systems Frontiers.*, vol. 17, no. 2, pp. 261– 274, Mar. 2015. (cit. on p. 100)

[97] V. Spoorthy and C. Sreedhar, "Multi-level SLAs with dynamic negotiations for remote sensing data as a service" in *International Journal of Scientific and Research Publications,* vol. 2, no. 10, pp. 1–5, 2012. (cit. on p. 99)

[98] B. Shojaiemehr , A. M. Rahmani and N. N Qader, "Cloud computing service negotiation: a systematic review" in *Computer Standards & Interfaces*, vol. 55, pp.196-206, Jan. 2018. (cit. on p. 87)

[99] B. Goetz, T. Peierls, D. Lea, J. Bloch, J. Bowbeer, and D. Holmes. *Java concurrency in practice*. Pearson Education, May 2006, ISBN: 0321349601, (cit. on p. 73).

[100] A. Poggi and M Tomaiuolo. "Integrating Peer-to-Peer and Multi-agent Technologies for the Realization of Content Sharing Applications" in *Information Retrieval and Mining in Distributed Environments*. Springer, Berlin, pp. 93-107, 2010. (cit. on p. 109)

[101] G. E. P. Box, G. M. Jenkins and G. C. Reinsel, *Time Series Analysis Forecasting and Control,* NJ, Englewood Cliffs: Prentice-Hall, 1994. (cit. on p. 99)

[102] F. Li, A. Palade and S.Clarke, "A Model for Distributed Service Level Agreement Negotiation in Internet of Things" in *International Conference on Service-Oriented Computing*, Springer, Cham, pp. 71-85, Oct. 2019. (cit. on p. 37).

[103] G. White, V. Nallur and S. Clarke, "Quality of service approaches in IoT: A systematic mapping", in *Journal of Systems and Software*, vol. 132, pp. 186-203, Oct. 2017. (cit. on p. 3, 39).

[104] F. E. Samann, S. R. Zeebaree, and S. Askar, "IoT Provisioning QoS based on Cloud and Fog Computing," in *Journal of Applied Science and Technology Trends*, vol.2, no.1, pp. 29-40, Mar. 2021. (cit. on p. 39).

[105] I. Awan, M. Younas and W. Naveed, "Modelling QoS in IoT Applications," in *International Conference on Network-Based Information Systems*, IEEE, pp. 99-105, Sep. 2014 (cit. on p. 39).

[106] B. Cao, J. Liu, Y. Wen, H. Li, Q. Xiao, and Chen, “QoS-aware service recommendation based on relational topic model and factorization machines for IoT Mashup applications”, in *Journal of Parallel and Distributed Computing*, vol. 132, pp. 177-189, Oct. 2019. (cit. on p. 39).

[107] M. M. Badawy, Z. H. Al and A. Hesham. “QoS provisioning framework for service-oriented Internet of things (IoT)”, in *Cluster Computing*, pp. 1-17, Jun. 2019. (cit. on p. 39).

[108] A. S. Alrawahi, K. Lee and A. Lotfi, “A Multiobjective QoS Model for Trading Cloud of Things Resources,” in *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 9447-9463, Dec. 2019. (cit. on p. 39).

[109] J. Li, Y. Bai, N. Zaman and V. C. M. Leung, "A Decentralized Trustworthy Context and QoS-Aware Service Discovery Framework for the Internet of Things," in *IEEE Access*, vol. 5, pp. 19154-19166, Sep. 2017. (cit. on p. 40).

[110] M. E. Khanouche, Y. Amirat, A. Chibani, M. Kerkar and A. Yachir, "Energy-Centered and QoS-Aware Services Selection for Internet of Things," in *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 3, pp. 1256-1269, Jul. 2016. (cit. on p. 40).

[111] O. Skarlat, M. Nardelli, S. Schulte and S. Dustdar, "Towards QoS-Aware Fog Service Placement," in *IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pp. 89-96, May 2017. (cit. on p. 40).

[112] E. Mingozzi, G. Tanganelli, C. Vallati, and V. Di Gregorio, “An open framework for accessing Things as a service,” in *Wireless Personal Multimedia Communications WPMC,* IEEE, pp.1-5, Jun. 2013. (cit. on p. 34)

[113] M. Aziez, S. Benharzallah and H. Bennoui, "Service discovery for the Internet of Things: Comparison study of the approaches," in *4th International Conference on Control, Decision and Information Technologies (CoDIT)*, IEEE, pp. 0599 - 0604, Apr. 2017. (cit. on p. 4).

[114] M. P. Papazoglou, and D. Georgakopoulos,”Introduction: Service-oriented computing” in *Communications of the ACM,* vol. 46, no. 13, pp.24-28, Oct. 2003. (cit on p.9).

[115] M. van Sinderen, and M. Spies. "Towards model-driven service-oriented enterprise computing " in *Enterprise Information Systems*, vol. 3, no. 3, pp. 211-217, July 2009. (cit on p.9).

[116] Y. Wei and M. B. Blake, "Service-Oriented Computing and Cloud Computing: Challenges and Opportunities," in *IEEE Internet Computing,* vol. 14, no. 6, pp. 72-75, Dec. 2010. (cit on p.9).

[117] B. H. Li , X. Chai , Y. Di Y, H. Yu, Z. Du and X. Peng, “ Research on service oriented simulation grid” in *Proceedings Autonomous Decentralized Systems*, IEEE, pp. 7-14, Apr. 2005. (cit on p.9).

[118] V. Issarny, Valerie, G. Bouloukakis, N. Georgantas, F. Sailhan, and G. Texier , “When service-oriented computing meets the IoT: A use case in the context of urban mobile crowdsensing.” in *European Conference on Service-Oriented and Cloud Computing*, Springer, pp. 1-16, 2018. (cit on p.9).

[119] M. P. Papazoglou, P. Traverso, S. Dustdar, S. and F. Leymann, "Service-oriented computing: a research roadmap " in *International Journal of Cooperative Information Systems*, vol. 17, no. 02, pp. 223-255, Jun. 2008 (cit on p.9).

[120] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, New York, 2004. (cit. on p. 10)

[121] J. Si; A. G. Barto, W. B. Powell and D. Wunsch, "Reinforcement Learning and Its Relationship to Supervised Learning," in *Handbook of Learning and Approximate Dynamic Programming* , IEEE, 2004, pp.45-63, (cit. on p.64)

[122] G. Rebala, A. Ravi , and S. Churiwala , "Machine Learning Definition and Basics" in *An Introduction to Machine Learning*, Springer, Cham, 2019 (cit. on p.64).

[123] M.L Littman, " Markov Decision Processes " in *International Encyclopedia of the Social and Behavioral Sciences*. Pergamon, 2001, pp. 9240-9242, (cit. on p.51)

[124] P. A. Gagniuc, *Markov Chains: From Theory to Implementation and Experimentation*. USA, NJ: John Wiley & Sons, 2017 (cit. on p.51)

[125] KaaIoT, *What is an IoT platform*, Jan. 2016, Accessed: 2021-09-11, [Online] Available: www.kaaiot.com/blog/what-is-iot-platform (cit. on p. 30).

[126] J. Mineraud, Julien, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms." in *Computer Communications*, vol. 89-90, no. pp. 5-16, Sept. 2016. (cit. on p. 30).

[127] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal and Q. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling Technologies," in *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1-20, Feb. 2017 (cit. on p. 30).

[128] P. Agarwal, Preeti and M. Alam, "Investigating IoT Middleware Platforms for Smart Application Development" in *Smart Cities-Opportunities and Challenges*, vol. 58, pp. 231-44, Springer, Apr. 2020. (cit. on p. 20, 30).

[129] Amazon Web Services, *Unlock your IoT data and accelerate business growth*, Oct. 2021. [Online], Accessed: 2021-10-13, Available at: https://aws.amazon.com/iot/ (cit. on p. 31).

[130] Azure IoT Hub, *Connect, monitor and manage billions of IoT assets*, Oct. 2021. [Online], Accessed: 2021-10-13, Available at: https://azure.microsoft.com/en-in/services/iot-hub/ (cit. on p. 31).

[131] Google Cloud, *Google Cloud IoT solutions,* Oct. 2021. [Online], Accessed: 2021-10-13, Available at: https://cloud.google.com/solutions/iot/(cit. on p. 31).

[132] Oracle Cloud, *Oracle cloud infrastructure*, Oct. 2021. [Online], Accessed: 2021-10-13, Available at: https://cloud.oracle.com/iot (cit. on p. 31).

[133] Kaa, *Elevate your IoT experience with Business-ready IoT Dashboards,* Oct. 2021. [Online], Accessed: 2021-10-13, Available at:https://www.kaaiot.com/ (cit. on p. 31).

[134] Thingspeak, *ThingSpeak for IoT Projects*, Oct. 2021. [Online], Accessed: 2021-10-13, Available at: https://thingspeak.com/ (cit. on p. 31).

[135] Altairs, *Altair SmartWorks*, Oct. 2021. [Online], Accessed: 2021-10-13, Available at: https://www.altairsmartworks.com/ (cit. on p. 31).

[136] Temboo, *Code the Internet of Everything*, Oct. 2021. [Online], Accessed: 2021-10-13, Available at: https://temboo.com/iot (cit. on p. 31).

[137] Particle, *Reprogram the world,* Oct. 2021. [Online], Accessed: 2021-10-13, Available at: https://www.particle.io/(cit. on p. 31).

[138] P. R. Pietzuch, *Hermes: A scalable event-based middleware,* Jun. 2004 [Online], Accessed: 2020-08-22, Available: http://www.cl.cam.ac.uk/techreports/UCAMCL-TR-590.pdf . (cit. on p.31).

[139] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G. P. Picco, T. Sivaharan, N. Weerasinghe and S. Zachariadis, "The runes middleware for networked embedded systems and its application in a disaster management scenario," in *IEEE International Conference in Pervasive Computing . Communication (PerCom'07)*, pp. 69–78, 2007 (cit. on p.31).

[140] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," *SIGARCH Computing Architecture News*, vol. 30, no. 5, Oct. 2002. (cit. on p.31).

[141] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting concurrent applications in wireless sensor networks," in *Proc. 4th International Conference on Embedded Network Sensor System.*, pp. 139–152, 2006. (cit. on p.31).

[142] N. Michal, K. Artem, K. Oleksiy, N. Sergiy, S. Michal, and T. Vagan, "Challenges of middleware for the Internet of Things," in *Automation Control—Theory and Practice*. InTech, 2009, (cit. on p.31).

[143] C.L. Fok, G.C. Roman, and C. Lu, "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks," in *ACM Transaction on Autonomous and Adaptive System (TAAS)*, vol. 4, no. 3, p. 16, Jul. 2009, (cit. on p.31).

[144] S. R. Madden, M. J. Franklin, J.M. Hellerstein, and W. Hong, "TinyDB: An acquisitional query processing system for sensor networks," *ACM Transactions Database System.*, vol. 30, no. 1, pp. 122–173, 2005, (cit. on p. 32).

[145] K. Aberer, M. Hauswirth, and A. Salehi, "A middleware for fast and flexible sensor network deployment," in *International Conference on Very Large Data Bases*, p. 1199, , 2006, (cit. on p. 32).

[146] Q. Han and N. Venkatasubramanian, "Autosec: An integrated middleware framework for dynamic service brokering," in *IEEE Distributed. System . Online*, vol. 2, no. 7, pp. 22–31, Oct. 2001, (cit. on p. 32).

[147] M. C. Huebscher and J. A. McCann, "Adaptive middleware for context aware applications in smart-homes," in *Proc. Middleware Pervasive Ad-Hoc Computing.*, pp. 111–116, 2004, (cit. on p. 32).

[148] A. Gerber and S. Kansal, *Making sense of IoT data*, Mar. 2020 [Online], Accessed: 2020-10-11, Available: https://developer.ibm.com/tutorials/iot-lp301-iot-manage-data/. (cit. on p. 20).

[149] M. Thoma, S. Meyer, K. Sperner, S. Meissner and T. Braun, "On IoT-services: Survey, Classification and Enterprise Integration," in *2012 IEEE International Conference on Green Computing and Communications*, pp. 257-260, 2012, (cit. on p. 20).

[150] A.A. Reineh, A.J. Paverd and A.P. Martin, "Trustworthy and Secure Service-Oriented Architecture for the Internet of Things," in *Cryptography and Security*, Jun. 2016, (cit. on p. 20).

[151] L. D. Xu, "Enterprise systems: State-of-the-art and future trends," in Industrial Informatics, IEEE Transactions on, Vol. 7, no. 4, pp. 630-640, Nov. 2011. (cit. on p. 20).

[152] D. Aksu and M. A. Aydin, "A Survey of IoT Architectural Reference Models," in *2019 16th International Multi-Conference on Systems, Signals & Devices (SSD)*, pp. 413-417, 2019, (cit. on p. 20, 22).

[153] G. Mulligan and D. Gračanin, "A comparison of SOAP and REST implementations of a service based interaction independence middleware framework," in *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pp. 1423-1432, 2009, (cit. on p. 21, 65).

[154] S. K. Mohalik, M. B. Jayaraman, B. Ramamurthy, and A. Vulgarakis, "SOA-PE : A Service-Oriented Architecture for Planning and Execution in Cyber-Physical Systems," in Proceedings International Conference on Smart Sensors and Systems (IC-SSS-2015). IEEE, 2015. (cit. on p. 22).

[155] Y. J. Dhas and P. Jeyanthi, "A Review on Internet of Things Protocol and Service-Oriented Middleware," in *2019 International Conference on Communication and Signal Processing (ICCSP)*, pp. 0104-0108, 2019, (cit. on p. 22).

[156] A. Barros, M. Dumas and P. Oaks, "Standards for Web Service Choreography and Orchestration: Status and Perspectives" in *Business Process Management Workshops* . Springer, pp. 61-74, 2005. (cit. on p. 24).

[157] B. Billet and V. Issarny, "Dioptase: a distributed data streaming middleware for the future web of things" in *Journal of Internet Services and Applications*, vol. 5, no.1, pp. 1-19, 2014. (cit. on p. 24).

[158] F.C. Delicato, P.F. Pires and A.Y. Zomaya, "Middleware platforms: state of the art, new issues, and future trends" in *The Art of Wireless Sensor Networks, Signals and Communication Technology*, Springer, pp. 645-674, 2014. (cit. on p. 32).

[159] R. Alshinina and K. Elleithy, "Performance and challenges of service-oriented architecture for wireless sensor networks" in *Sensors*, vol. 14, no. 3, pp. 536, 2017. (cit. on p. 32)

[160] J.A. López-Riquelme, N. Pavón-Pulido, H. Navarro-Hellín, F. Soto-Valles and R. Torres-Sánchez, "A software architecture based on FIWARE cloud for Precision Agriculture" in *Agricultural water management*, vol. 183, pp. 123-35, 2017. (cit. on p. 33)

[161] G.F. Anastasi, E. Bini, A. Romano and G. Lipari, "A service-oriented architecture for QoS configuration and management of Wireless Sensor Networks " in 15th Conference on Emerging Technologies Factory Automation (ETFA 2010), IEEE, pp. 1-8, 2010. (cit. on p. 33)

[162] H. Bohn, A. Bobek and F. Golatowski, “SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service-oriented framework for different domains,” in *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06),* pp. 43-43, 2006. (cit. on p. 33)

[163] M. Kim, J. W. Lee, Y. J. Lee and J.C. Ryou, “Cosmos: A middleware for integrated data processing over heterogeneous sensor networks” in *ETRI Journal*, vol. 30, no. 5, pp. 696-706, 2008, (cit. on p. 33)

[164] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey.”, *Computer networks* vol. 54, no.15, pp. 2787-2805, 2010, (cit. on p. 32).

[165] J. A. Stankovic, “Research Directions for the Internet of Things,” in *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3-9, Feb. 2014, (cit. on p. 1).

[166] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas, “Service Oriented Middleware for the Internet of Things: A Perspective”, in *European conference on a service-based Internet*, Springer, pp. 220-229, 2011, (cit. on p. 1).

[167] R. Duan, X Chen and T. Xing, “A QoS architecture for IoT”, in *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, 717-720. 2011,(cit. on p. 3).

[168] Z. Balfagih and M. F. Hassan, “Quality Model for Web Services from Multi-stakeholders' Perspective,” in *2009 International Conference on Information Management and Engineering*, pp. 287-291, 2009, (cit. on p. 4).

[169] W. Abramowicz, R. Hofman, W. Suryn and D. Zyskowski, “ SQuaRE based web services quality model.”, in *Proceedings of The International MultiConference of Engineers and Computer Scientists*, pp. 827-835, Mar. 2008, (cit. on p. 3).

[170] S. W. Choi, J. S. Her and S. D. Kim, “Modeling QoS Attributes and Metrics for Evaluating Services in SOA Considering Consumers' Perspective as the First Class Requirement,” in *The 2nd IEEE Asia-Pacific Service Computing Conference (APSCC 2007,* pp. 398-405, 2007, (cit. on p. 4).

Appendix

A.1 QOS CONFIGURATIONS OF PLANT NODE1 IN LINKED USDL

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix usdl: <http://www.linked-usdl.org/ns/usdl-core#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix gr: <http://purl.org/goodrelations/v1#> .


<http://linked-usdl.github.io/usdl-editor/>
  a usdl:ServiceDescription ;
  dcterms:title "QoS Profile of Plant node1";
  dcterms:description "This profile contains the QoS constraints and requirements of  Plant node1 sensors and actuators respectively";
  dcterms:creator _:b223 ;
  dcterms:created "2020-02-09T12:00"^^xsd:datetime ;

_:b223
  a foaf:Person ;
  foaf:name "Itoro Udoh" .

<http://linked-usdl.github.io/usdl-editor/#4TNHl5zNhnOej2TCX>
  usdl:hasRole _:b224 ;
  usdl:hasType _:b225 ;
  usdl:hasInterface _:b226 ;
  a usdl:Service ;
  dcterms:title "QoS Constraints of Sensors"@en ;
  dcterms:created "2020-02-09T12:00"^^xsd:datetime ;
  dcterms:description "<div>Attributes of the  sensing services</div>"@en ;
  <http://linked-usdl.github.io/usdl-editor/#62lOH9cAU6XZwHFuy> <http://linked-usdl.github.io/usdl-editor/#g8lpyJF5lre0A1HfS> ;
  <http://linked-usdl.github.io/usdl-editor/#1wI5Q4WFkQxbJ4PsX> <http://linked-usdl.github.io/usdl-editor/#lxjpH07SnwAzDRTg9> ;
  <http://linked-usdl.github.io/usdl-editor/#RPRKolWJTh65OhXPV> <http://linked-usdl.github.io/usdl-editor/#gbw6KUAeyLnPG1AKJ> ;
  <http://linked-usdl.github.io/usdl-editor/#fW6dbkISCSpDRFqoJ> <http://linked-usdl.github.io/usdl-editor/#5inUIxvVxyEvpIugu> ;
  <http://linked-usdl.github.io/usdl-editor/#8oXx1KKyc7qt0MO5E> <http://linked-usdl.github.io/usdl-editor/#wDpzqHHXhFpBJC0xG> ;

_:b224
  a skos:Concept ;
  rdfs:label "Service Provider" .

_:b225
  a skos:Concept ;
  rdfs:label "Sensing Service" .

_:b226
  a skos:Concept ;
  rdfs:label "REST" .

<http://linked-usdl.github.io/usdl-editor/#62lOH9cAU6XZwHFuy>
  a owl:Property ;
  rdfs:subPropertyOf gr:quantitativeProductOrServiceProperty ;
  rdfs:label "Response Time" ;
  rdfs:domain gr:ProductOrService ;
  rdfs:range gr:QuantitativeValue .

<http://linked-usdl.github.io/usdl-editor/#g8lpyJF5lre0A1HfS>
  a gr:QuantitativeValue ;
  gr:hasPrefValue "0.42" ;
  gr:hasResdValue "0.85" ;
  gr:hasWeight "0.39" ;
  gr:hasMinValue "100" ;
  gr:hasMaxValue "1000" ;
  gr:hasUnitOfMeasurement "milliseconds" .

<http://linked-usdl.github.io/usdl-editor/#1wI5Q4WFkQxbJ4PsX>
  a owl:Property ;
  rdfs:subPropertyOf gr:quantitativeProductOrServiceProperty ;
  rdfs:label "Availability" ;
  rdfs:domain gr:ProductOrService ;
  rdfs:range gr:QuantitativeValue .

<http://linked-usdl.github.io/usdl-editor/#lxjpH07SnwAzDRTg9>
  a gr:QuantitativeValue ;
  gr:hasPrefValue "0.77" ;
  gr:hasResdValue "0.94" ;
  gr:hasWeight "0.40" ;
  gr:hasMinValue "10" ;
  gr:hasMaxValue "100" ;
  gr:hasUnitOfMeasurement "percent" .

<http://linked-usdl.github.io/usdl-editor/#RPRKolWJTh65OhXPV>
  a owl:Property ;
  rdfs:subPropertyOf gr:quantitativeProductOrServiceProperty ;
  rdfs:label "Throughput" ;
  rdfs:domain gr:ProductOrService ;
  rdfs:range gr:QuantitativeValue .

<http://linked-usdl.github.io/usdl-editor/#gbw6KUAeyLnPG1AKJ>
  a gr:QuantitativeValue ;
  gr:hasPrefValue "0.84" ;
  gr:hasResdValue "0.99" ;
  gr:hasWeight "0.21" ;
  gr:hasMinValue "10" ;
  gr:hasMaxValue "100" ;
  gr:hasUnitOfMeasurement "service per minute" .

<http://linked-usdl.github.io/usdl-editor/#fW6dbkISCSpDRFqoJ>
  a owl:Property ;
  rdfs:subPropertyOf gr:quantitativeProductOrServiceProperty ;
  rdfs:label "Service Area" ;
  rdfs:domain gr:ProductOrService ;
  rdfs:range gr:QuantitativeValue .

<http://linked-usdl.github.io/usdl-editor/#5inUIxvVxyEvpIugu>
  a gr:QuantitativeValue ;
  gr:hasLocValue "loc_01".

<http://linked-usdl.github.io/usdl-editor/#8oXx1KKyc7qt0MO5E>
  a owl:Property ;
  rdfs:subPropertyOf gr:quantitativeProductOrServiceProperty ;
  rdfs:label "Available Time" ;
  rdfs:domain gr:ProductOrService ;
  rdfs:range gr:QuantitativeValue .

<http://linked-usdl.github.io/usdl-editor/#wDpzqHHXhFpBJC0xG>
  a gr:QuantitativeValue ;
  gr:hasStartTime "9:00".
  gr:hasEndTime "16:00".
  gr:hasUnitOfMeasurement "hrs"


<http://linked-usdl.github.io/usdl-editor/#6s6hQvy7AcVjwq0bv>
  usdl:hasRole _:b227 ;
  usdl:hasType _:b228 ;
  usdl:hasInterface _:b229 ;
  a usdl:Service ;
  dcterms:title "QoS Requirements of Actuators"@en ;
  dcterms:created "2020-02-09T12:00"^^xsd:datetime ;
  dcterms:description "<div>Attributes of actuators requirements</div>"@en ;
  <http://linked-usdl.github.io/usdl-editor/#rt673ER4bHc91VGik> <http://linked-usdl.github.io/usdl-editor/#ghVB3160GhiuzmNaP> ;
  <http://linked-usdl.github.io/usdl-editor/#knvdcsrt56CVytG89> <http://linked-usdl.github.io/usdl-editor/#gnCvb47jIonhfgQah> ;
  <http://linked-usdl.github.io/usdl-editor/#5vhCr731JbY904gbY> <http://linked-usdl.github.io/usdl-editor/#gbw6KUAeyLnPG1AKJ> ;
  <http://linked-usdl.github.io/usdl-editor/#jbFR784Vkiyral72C> <http://linked-usdl.github.io/usdl-editor/#5inUIxvVxyEvpIugu> ;
  <http://linked-usdl.github.io/usdl-editor/#Awg56Bk849CvbkYqw> <http://linked-usdl.github.io/usdl-editor/#jvf45DFe3hjnCroPV> ;

_:b227
  a skos:Concept ;
  rdfs:label "Service Consumer" .

_:b228
  a skos:Concept ;
  rdfs:label "Actuating Service" .

_:b229
  a skos:Concept ;
  rdfs:label "REST" .

<http://linked-usdl.github.io/usdl-editor/#rt673ER4bHc91VGik>
  a owl:Property ;
  rdfs:subPropertyOf gr:quantitativeProductOrServiceProperty ;
  rdfs:label "Response Time" ;
  rdfs:domain gr:ProductOrService ;
  rdfs:range gr:QuantitativeValue .

<http://linked-usdl.github.io/usdl-editor/#ghVB3160GhiuzmNaP>
  a gr:QuantitativeValue ;
  gr:hasPrefValue "0.92" ;
  gr:hasResdValue "0.67" ;
  gr:hasWeight "0.33" ;
  gr:hasMinValue "100" ;
  gr:hasMaxValue "1000" ;
  gr:hasUnitOfMeasurement "milliseconds" .

<http://linked-usdl.github.io/usdl-editor/#knvdcsrt56CVytG89>
  a owl:Property ;
  rdfs:subPropertyOf gr:quantitativeProductOrServiceProperty ;
  rdfs:label "Availability" ;
  rdfs:domain gr:ProductOrService ;
  rdfs:range gr:QuantitativeValue .

<http://linked-usdl.github.io/usdl-editor/#gnCvb47jIonhfgQah>
  a gr:QuantitativeValue ;
  gr:hasPrefValue "0.95" ;
  gr:hasResdValue "0.76" ;
```

```
gr:hasWeight "0.28" ;
gr:hasMinValue "10" ;
gr:hasMaxValue "100" ;
gr:hasUnitOfMeasurement "percent" .

<http://linked-usdl.github.io/usdl-editor/#5vhCr731JbY904gbY>
a owl:Property ;
rdfs:subPropertyOf gr:quantitativeProductOrServiceProperty ;
rdfs:label "Throughput" ;
rdfs:domain gr:ProductOrService ;
rdfs:range gr:QuantitativeValue .

<http://linked-usdl.github.io/usdl-editor/#gbw6KUAeyLnPG1AKJ>
a gr:QuantitativeValue ;
gr:hasPrefValue "0.91" ;
gr:hasResdValue "0.83" ;
gr:hasWeight "0.39" ;
gr:hasMinValue "10" ;
gr:hasMaxValue "100" ;
gr:hasUnitOfMeasurement "service per minute" .

<http://linked-usdl.github.io/usdl-editor/#jbFR784Vkiyral72C>
a owl:Property ;
rdfs:subPropertyOf gr:quantitativeProductOrServiceProperty ;
rdfs:label "Actuation Area" ;
rdfs:domain gr:ProductOrService ;
rdfs:range gr:QuantitativeValue .

<http://linked-usdl.github.io/usdl-editor/#5inUIxvVxyEvpIugu>
a gr:QuantitativeValue ;
gr:hasLocValue "loc_01".

<http://linked-usdl.github.io/usdl-editor/#Awg56Bk849CvbkYqw>
a owl:Property ;
rdfs:subPropertyOf gr:quantitativeProductOrServiceProperty ;
rdfs:label "Available Time" ;
rdfs:domain gr:ProductOrService ;
rdfs:range gr:QuantitativeValue .

<http://linked-usdl.github.io/usdl-editor/#jvf45DFe3hjnCroPV>
a gr:QuantitativeValue ;
gr:hasStartTime "9:00".
gr:hasEndTime "16:00".
gr:hasUnitOfMeasurement "hrs"
```

A.2 AN INSTANTIATION OF QOS AGREEMENT IN LINKED USDL

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix usdl: <http://www.linked-usdl.org/ns/usdl-core#> .
@prefix sla: <http://www.linked-usdl.org/ns/usdl-sla#> .

<ServiceLevelAgreement/>
  a usdl:ServiceOffering;
  dcterms:title "QoS Agreement";
  dcterms:description "This QoS agreement defines the attributes of the Service level profile of the Humiture
          Service ;
  dcterms:created "2020-02-10T13:00"^^xsd:datetime ;
  usdl:includes :SensorHumitureService;
  usdl:validFrom "2020-02-10T13:02"^^xsd:datetime;
  usdl:validThrough "2020-02-10T13:10"^^xsd:datetime;
  usdl:hasAgreementTerm : 1.836

<ServiceLevelProfile>
 a sla:ServiceLevelProfile ;
 dcterms:title "Standard Service Profile";
 sla:hasServiceLevel [
   a sla:GuaranteedState;
   dcterms:title "QoS parameters";
   sla:serviceLevelExpression [
     a sla:ServiceLevelExpression;
     sla:hasVariable :Response Time, :Availability, :Throughput];
     sla:obligatedParty usdl:Plant node1, Plant node2].


:Response Time
  a sla:Variable;
  rdfs:label "Response Time";
  sla:hasDefault [
    a gr:QuantitativeValue;
    gr:hasValue "546";
    gr:unitOfMeasurement "milliseconds" ] .

:Availability
  a sla:Variable;
  rdfs:label "Availability";
  sla:hasDefault [
    a gr:QuantitativeValue;
    gr:hasValue "87";
    gr:unitOfMeasurement "percent" ] .

:Throughput
  a sla:Variable;
  rdfs:label "Throughput";
  sla:hasDefault [
    a gr:QuantitativeValue;
    gr:hasValue "55";
    gr:unitOfMeasurement "spm" ] .
```

A.3 THE DYNAMICS OF THE NEGOTIATION PROCESS AS A FINITE MDP

Current state (s)	Negotiation tactic (a)	Next State (s')	Transition scheme P(s'|a,s)	Reward scheme $R(s'\|s,a)$
(r_h,d_l,u_f)	trade-off	(r_h,d_l,u_f)	$1- (\alpha_1+ \alpha_2+ \alpha_3+ \alpha_4+ \alpha_5+ \alpha_6+ \alpha_7)$	$3r_1$
(r_h,d_l,u_f)	trade-off	(r_l,d_l,u_f)	α_1	$3r_1$
(r_l,d_l,u_f)	trade-off	(r_h,d_l,u_f)	α_1	$2r_1 + r_2$
(r_h,d_l,u_f)	concession	(r_h,d_l,u_f)	$1- (\beta_1+ \beta_2+ \beta_3+ \beta_4+ \beta_5+ \beta_6+ \beta_7)$	$3r_3$
(r_h,d_l,u_f)	concession	(r_l,d_l,u_f)	β_1	$r_2 + 2r_3$
(r_l,d_l,u_f)	concession	(r_h,d_l,u_f)	β_7	$r_1 + 2r_3$
(r_h,d_l,u_f)	trade-off	(r_h,d_s,u_f)	α_2	$3r_1$
(r_h,d_s,u_f)	trade-off	(r_h,d_s,u_f)	$1- (\alpha_1+ \alpha_2+ \alpha_3+ \alpha_4+ \alpha_5+ \alpha_6+ \alpha_7)$	$2r_1 + r_3$
(r_h,d_s,u_f)	trade-off	(r_h,d_l,u_f)	α_1	$2r_1 + r_2$
(r_h,d_l,u_f)	concession	(r_h,d_s,u_f)	β_2	$r_2 + 2r_3$
(r_h,d_s,u_f)	concession	(r_h,d_s,u_f)	$1- (\beta_1+ \beta_2+ \beta_3+ \beta_4+ \beta_5+ \beta_6+ \beta_7)$	$r_1 + 2r_3$
(r_h,d_s,u_f)	concession	(r_h,d_l,u_f)	β_1	$r_1 + 2r_3$
(r_h,d_l,u_f)	trade-off	(r_h,d_l,u_c)	α_3	$3r_1$
(r_h,d_l,u_c)	trade-off	(r_h,d_l,u_c)	$1- (\alpha_1+ \alpha_2+ \alpha_3+ \alpha_4+ \alpha_5+ \alpha_6+ \alpha_7)$	$2r_1 + r_3$
(r_h,d_l,u_c)	trade-off	(r_h,d_l,u_f)	α_1	$2r_1 + r_2$
(r_h,d_l,u_f)	concession	(r_h,d_l,u_c)	β_3	$2r_3 + r_2$
(r_h,d_l,u_c)	concession	(r_h,d_l,u_c)	$1- (\beta_1+ \beta_2+ \beta_3+ \beta_4+ \beta_5+ \beta_6+ \beta_7)$	$r_1 + 2r_3$
(r_h,d_l,u_c)	concession	(r_h,d_l,u_f)	β_1	$r_1 + 2r_3$
(r_h,d_s,u_c)	trade-off	(r_h,d_s,u_c)	$1- (\alpha_1+ \alpha_2+ \alpha_3+ \alpha_4+ \alpha_5+ \alpha_6+ \alpha_7)$	$r_1 + 2r_3$
(r_h,d_s,u_c)	trade-off	(r_h,d_l,u_f)	α_1	$r_1 + 2r_2$
(r_h,d_l,u_f)	trade-off	(r_h,d_s,u_c)	α_4	$3r_1$
(r_h,d_s,u_c)	concession	(r_h,d_s,u_c)	$1- (\beta_1+ \beta_2+ \beta_3+ \beta_4+ \beta_5+ \beta_6+ \beta_7)$	$2r_1 + r_3$
(r_h,d_s,u_c)	concession	(r_h,d_l,u_f)	β_1	$2r_1 + r_3$
(r_h,d_l,u_f)	concession	(r_h,d_s,u_c)	β_4	$2r_2 + r_3$
(r_h,d_l,u_c)	trade-off	(r_h,d_s,u_f)	α_2	$2r_1 + r_2$
(r_h,d_l,u_c)	concession	(r_h,d_s,u_f)	β_2	$r_1 + r_2 + r_3$
(r_h,d_s,u_f)	trade-off	(r_h,d_l,u_c)	α_2	$2r_1 + r_2$
(r_h,d_s,u_f)	concession	(r_h,d_l,u_c)	β_2	$r_1 + r_2 r_3$
(r_h,d_l,u_c)	trade-off	(r_h,d_s,u_c)	α_7	$2r_1 + r_3$
(r_h,d_l,u_c)	concession	(r_h,d_s,u_c)	β_3	$r_1 + r_2 + r_3$
(r_h,d_s,u_c)	trade-off	(r_h,d_l,u_c)	α_2	$2r_1+ r_3$
(r_h,d_s,u_c)	concession	(r_h,d_l,u_c)	β_2	$2r_1 + r_3$
(r_h,d_s,u_f)	trade-off	(r_h,d_s,u_c)	α_3	$2r_1 + r_3$
(r_h,d_s,u_f)	concession	(r_h,d_s,u_c)	β_3	$r_1 + r_2 + r_3$
(r_h,d_s,u_c)	trade-off	(r_h,d_s,u_f)	α_3	$r_1 + r_2 + r_3$
(r_h,d_s,u_c)	concession	(r_h,d_s,u_f)	β_3	$2r_1 + r_3$
(r_l,d_l,u_f)	trade-off	(r_l,d_l,u_f)	$1- (\alpha_1+ \alpha_2+ \alpha_3+ \alpha_4+ \alpha_5+ \alpha_6+ \alpha_7)$	$2r_1 + r_3$
(r_l,d_l,u_f)	concession	(r_l,d_l,u_f)	$1- (\beta_1+ \beta_2+ \beta_3+ \beta_4+ \beta_5+ \beta_6+ \beta_7)$	$r_1 + 2r_3$
(r_l,d_s,u_f)	trade-off	(r_l,d_s,u_f)	$1- (\alpha_1+ \alpha_2+ \alpha_3+ \alpha_4+ \alpha_5+ \alpha_6+ \alpha_7)$	$r_1 + 2r_3$
(r_l,d_s,u_f)	concession	(r_l,d_s,u_f)	$1- (\beta_1+ \beta_2+ \beta_3+ \beta_4+ \beta_5+ \beta_6+ \beta_7)$	$2r_1 + r_3$
(r_l,d_s,u_f)	trade-off	(r_l,d_l,u_f)	α_1	$r_1 + r_2 + r_3$
(r_l,d_s,u_f)	concession	(r_l,d_l,u_f)	β_1	$2r_1 + r_3$
(r_l,d_l,u_f)	trade-off	(r_l,d_s,u_f)	α_2	$2r_1 + r_3$
(r_l,d_l,u_f)	concession	(r_l,d_s,u_f)	β_1	$r_1 + r_2 + r_3$
(r_l,d_s,u_c)	trade-off	(r_l,d_s,u_c)	$1- (\alpha_1+ \alpha_2+ \alpha_3+ \alpha_4+ \alpha_5+ \alpha_6+ \alpha_7)$	$3r_3$
(r_l,d_s,u_c)	concession	(r_l,d_s,u_c)	$1- (\beta_1+ \beta_2+ \beta_3+ \beta_4+ \beta_5+ \beta_6+ \beta_7)$	$3r_1$

(r_l,d_s,u_c)	trade-off	(r_l,d_s,u_f)	α_1	$2r_3 + r_2$
(r_l,d_s,u_c)	concession	(r_l,d_s,u_f)	β_1	$3r_1$
(r_l,d_s,u_f)	trade-off	(r_l,d_s,u_c)	α_2	$2r_3 + r_2$
(r_l,d_s,u_f)	concession	(r_l,d_s,u_c)	β_2	$2r_1 + r_2$
(r_l,d_l,u_c)	trade-off	(r_l,d_l,u_c)	$1- (\alpha_1+ \alpha_2+ \alpha_3+ \alpha_4+ \alpha_5+ \alpha_6+ \alpha_7)$	$r_1 + 2r_3$
(r_l,d_l,u_c)	concession	(r_l,d_l,u_c)	$1- (\beta_1+ \beta_2+ \beta_3+ \beta_4+ \beta_5+ \beta_6+ \beta_7)$	$2r_1+ r_3$
(r_l,d_l,u_c)	trade-off	(r_l,d_l,u_f)	α_1	$r_1+ r_3 + r_2$
(r_l,d_l,u_c)	concession	(r_l,d_l,u_f)	β_1	$2r_1 + r_3$
(r_l,d_l,u_f)	trade-off	(r_l,d_l,u_c)	α_3	$2r_1 + r_3$
(r_l,d_l,u_f)	concession	(r_l,d_l,u_c)	β_2	$r_1 + r_3 + r_2$
(r_l,d_s,u_c)	trade-off	(r_h,d_s,u_f)	α_2	$2r_2 + r_3$
(r_l,d_s,u_c)	concession	(r_h,d_s,u_f)	β_2	$3r_1$
(r_h,d_s,u_f)	trade-off	(r_l,d_s,u_c)	α_4	$2r_1 + r_3$
(r_h,d_s,u_f)	concession	(r_l,d_s,u_c)	β_4	$r_1 + 2r_2$
(r_l,d_l,u_c)	trade-off	(r_h,d_s,u_c)	α_2	$r_1 + r_2 + r_3$
(r_l,d_l,u_c)	concession	(r_h,d_s,u_c)	β_2	$2r_1 + r_2$
(r_h,d_s,u_c)	trade-off	(r_l,d_l,u_c)	α_4	$r_1 + r_2 + r_3$
(r_h,d_s,u_c)	concession	(r_l,d_l,u_c)	β_4	$2r_1 + r_2$
(r_l,d_s,u_f)	trade-off	(r_h,d_s,u_f)	α_3	$r_1 + r_2 + r_3$
(r_l,d_s,u_f)	concession	(r_h,d_s,u_f)	β_3	$2r_1 + r_3$
(r_h,d_s,u_f)	trade-off	(r_l,d_s,u_f)	α_5	$2r_1 + r_3$
(r_h,d_s,u_f)	concession	(r_l,d_s,u_f)	β_5	$r_1 + r_2 + r_3$
(r_h,d_s,u_f)	trade-off	(r_l,d_l,u_f)	α_6	$2r_1 + r_2$
(r_h,d_s,u_f)	concession	(r_l,d_l,u_f)	β_6	$r_1 + r_2 + r_3$
(r_l,d_l,u_f)	trade-off	(r_h,d_s,u_f)	α_4	$2r_1 + r_2$
(r_l,d_l,u_f)	concession	(r_h,d_s,u_f)	β_3	$r_1 + r_2 + r_3$
(r_l,d_l,u_c)	trade-off	(r_h,d_s,u_f)	α_3	$r_1 + 2r_2$
(r_l,d_l,u_c)	concession	(r_h,d_s,u_f)	β_3	$2r_1 + r_2$
(r_h,d_s,u_f)	trade-off	(r_l,d_l,u_c)	α_7	$2r_1 + r_2$
(r_h,d_s,u_f)	concession	(r_l,d_l,u_c)	β_7	$r_1 + 2r_2$
(r_l,d_s,u_c)	trade-off	(r_l,d_l,u_f)	α_3	$2r_2 + r_3$
(r_l,d_s,u_c)	concession	(r_l,d_l,u_f)	β_3	$3r_1$
(r_l,d_l,u_f)	trade-off	(r_l,d_s,u_c)	α_5	$2r_1 + r_3$
(r_l,d_l,u_f)	concession	(r_l,d_s,u_c)	β_4	$r_1 + 2r_2$
(r_l,d_s,u_c)	trade-off	(r_l,d_l,u_c)	α_4	$r_2 + 2r_3$
(r_l,d_s,u_c)	concession	(r_l,d_l,u_c)	β_4	$3r_1$
(r_l,d_l,u_c)	trade-off	(r_l,d_s,u_c)	α_4	$r_1 + 2r_3$
(r_l,d_l,u_c)	concession	(r_l,d_s,u_c)	β_4	$2r_1 + r_2$
(r_l,d_s,u_c)	trade-off	(r_h,d_s,u_c)	α_5	$r_2 + 2r_3$
(r_l,d_s,u_c)	concession	(r_h,d_s,u_c)	β_5	$3r_1$
(r_h,d_s,u_c)	trade-off	(r_l,d_s,u_c)	α_5	$r_1 + 2r_3$
(r_h,d_s,u_c)	concession	(r_l,d_s,u_c)	β_5	$2r_1 + r_2$
(r_l,d_s,u_c)	trade-off	(r_h,d_l,u_f)	α_6	$3r_2$
(r_l,d_s,u_c)	concession	(r_h,d_l,u_f)	β_6	$3r_1$
(r_h,d_l,u_f)	trade-off	(r_l,d_s,u_c)	α_5	$3r_1$
(r_h,d_l,u_f)	concession	(r_l,d_s,u_c)	β_5	$3r_2$
(r_l,d_s,u_c)	trade-off	(r_h,d_l,u_c)	α_7	$2r_2 + r_3$
(r_l,d_s,u_c)	concession	(r_h,d_l,u_c)	β_7	$3r_1$
(r_h,d_l,u_c)	trade-off	(r_l,d_s,u_c)	α_3	$2r_1 + r_3$
(r_h,d_l,u_c)	concession	(r_l,d_s,u_c)	β_4	$r_1 + 2r_2$
(r_l,d_s,u_f)	trade-off	(r_l,d_l,u_c)	α_4	$r_1 + r_2 + r_3$

(r_l,d_s,u_f)	concession	(r_l,d_l,u_c)	β_4	$2r_1 + r_2$
(r_l,d_l,u_c)	trade-off	(r_l,d_s,u_f)	α_5	$r_1 + r_2\ r_3$
(r_l,d_l,u_c)	concession	(r_l,d_s,u_f)	β_5	$2r_1 + r_2$
(r_l,d_s,u_f)	trade-off	(r_h,d_s,u_c)	α_5	$r_1 + r_2 + r_3$
(r_l,d_s,u_f)	concession	(r_h,d_s,u_c)	β_5	$2r_1 + r_2$
(r_h,d_s,u_c)	trade-off	(r_l,d_s,u_f)	α_6	$r_1 + r_2 + r_3$
(r_h,d_s,u_c)	concession	(r_l,d_s,u_f)	β_6	$2r_1 + r_2$
(r_l,d_s,u_f)	trade-off	(r_h,d_l,u_f)	α_6	$r_1 + 2r_2$
(r_l,d_s,u_f)	concession	(r_h,d_l,u_f)	β_6	$2r_1 + r_3$
(r_h,d_l,u_f)	trade-off	(r_l,d_s,u_f)	α_6	$3r_1$
(r_h,d_l,u_f)	concession	(r_l,d_s,u_f)	β_6	$2r_2 + r_3$
(r_l,d_s,u_f)	trade-off	(r_h,d_l,u_c)	α_7	$r_1 + 2r_2$
(r_l,d_s,u_f)	concession	(r_h,d_l,u_c)	β_7	$2r_1 + r_2$
(r_h,d_l,u_c)	trade-off	(r_l,d_s,u_f)	α_4	$2r_1 + r_2$
(r_h,d_l,u_c)	concession	(r_l,d_s,u_f)	β_5	$r_1 + 2r_2$
(r_l,d_l,u_f)	trade-off	(r_h,d_s,u_c)	α_6	$2r_1 + r_2$
(r_l,d_l,u_f)	concession	(r_h,d_s,u_c)	β_5	$r_1 + 2r_2$
(r_h,d_s,u_c)	trade-off	(r_l,d_l,u_f)	α_7	$r_1 + 2r_2$
(r_h,d_s,u_c)	concession	(r_l,d_l,u_f)	β_7	$2r_1 + r_2$
(r_l,d_l,u_f)	trade-off	(r_h,d_l,u_c)	α_7	$2r_1 + r_2$
(r_l,d_l,u_f)	concession	(r_h,d_l,u_c)	β_6	$r_1 + r_2 + r_3$
(r_h,d_l,u_c)	trade-off	(r_l,d_l,u_f)	α_5	$2r_1 + r_2$
(r_h,d_l,u_c)	concession	(r_l,d_l,u_f)	β_6	$r_1 + r_2 + r_3$
(r_h,d_l,u_f)	trade-off	(r_l,d_l,u_c)	α_7	$3r_1$
(r_h,d_l,u_f)	concession	(r_l,d_l,u_c)	β_7	$2r_2 + r_3$
(r_l,d_l,u_c)	trade-off	(r_h,d_l,u_f)	α_6	$r_1 + 2r_2$
(r_l,d_l,u_c)	concession	(r_h,d_l,u_f)	β_6	$2r_1 + r_3$
(r_h,d_l,u_c)	trade-off	(r_l,d_l,u_c)	α_6	$2r_1+ r_3$
(r_h,d_l,u_c)	concession	(r_l,d_l,u_c)	β_7	$r_1 + r_2 + r_3$
(r_l,d_l,u_c)	trade-off	(r_h,d_l,u_c)	α_7	$r_1 + r_2+ r_3$
(r_l,d_l,u_c)	concession	(r_h,d_l,u_c)	β_7	$2r_1 + r_3$

A.4 INPUT PARAMETERS OF THE REINFORCEMENT LEARNING NEGOTIATION STRATEGY AND PREDICTION STRATEGY

Negotiation Strategy		**Prediction Strategy**	
Parameters	value	Parameters	value
discount rate (γ)	0.85	adaptation degree (ad)	0.25
degree of concession	0.15	variator factor (vf)	0.05
degree of trade-off	0.15		
transition scheme	α_1=0.05 , α_2=0.1 , α_3 = 0.15, α_4=0.1, α_5=0.05, α_6=0.15, α_7 =0.20, β_1 =0.1β_2 =0.05, β_3=0.05, β_4=0.1, β_5=,0.15 β_6=0.05,β_7=0.05		
reward scheme	r_1 = 0.3 r_2 = 0.15 r_3 =0.1		

Code Listings

In this section are some of the Java code listings used in the IoTQoSystem implementation, as described in Chapter 5.

C.1 QoS PROFILE VALIDATION

Once the service component of the negotiation framework retrieves the QoS profile of devices, it carries out some processing, such as ensuring that the sum of all the QoS parameter weights equals 1 as depicted in Equation 4.4. Listing C.1 shows the Java implementation of how the QoS profile of devices is validated in the negotiation framework's service component.

Listing C1: The QoS profile validation implementation in Java.

```java
package com.service.negointerface;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Set;

import com.service.negointerface.request.Profile;

/**
 * This is class used for validating the Profiles retrieved from the Middleware
 * @author Udoh
 *
 */
public class StartValidation {

    private static HashMap<String, Profile> negotiatingDevices = new HashMap<>();
    private static boolean isFirst = true;
    private static List<String> negoDevicesIp;
    private static int counter = 0;
    private static PrepareData prepareData = new PrepareData();
    private static Profile firstProfile;
    public static com.custom.server.response.Session outcomeSession=null;

    /**
     * Reset the method After negotiation terminated
     */
    public static  void reset() {
        negotiatingDevices = new HashMap<>();
        isFirst = true;
        negoDevicesIp=new ArrayList<String>();
        counter = 0;
        prepareData = new PrepareData();
        firstProfile = null;
        outcomeSession = null;
    }

    /**
     * Perform the Profile validation
     * @param profile
     * @return
     * @throws Exception
     */
    public static  com.custom.server.response.Session processNegotiatonRequest(Profile
       profile) throws Exception {
        counter++;
        String tactic = profile.getTactic();
        String ipAddress = profile.getIpAddress();
        if (isFirst) {
            PrepareData.reset();
            System.out.println("Negotiation Session Started; Profile Received from " +
              profile.getDeviceDescr() + ": " + profile.getIpAddress());
            System.out.println("Validation of Profiles in Progress.");
            if (sumWeightParameter(profile) && validateMinMaxUtilityValue(profile)) {
                firstProfile = profile;
                isFirst = false;
                negoDevicesIp =
                Arrays.asList(profile.getIpOfNegotiatingDevices().split(","));
                System.out.println("--First "+negoDevicesIp.toString());
                negotiatingDevices.put(ipAddress, profile);
                prepareData.setData(tactic);
            } else {
                counter--;
                System.out.println("Profile Validation not successful,Negotiation Session
                Paused");
                throw new Exception("Validation failed");
            }

        } else {
            System.out.println("Profile Received from " + profile.getDeviceDescr() + ": " +
              profile.getIpAddress());
            System.out.println("Validation of Profiles in Progress.");

            if (negoDevicesIp.contains(ipAddress)) {
                if (firstProfile.getParameter1().equals(profile.getParameter1()) &&
                   firstProfile.getParameter2().equals(profile.getParameter2())
                   && firstProfile.getParameter3().equals(profile.getParameter3())) {
                       if (sumWeightParameter(profile) &&
                         validateMinMaxUtilityValue(profile)) {
                            negotiatingDevices.put(ipAddress, profile);
                            prepareData.setData(tactic);

                        // Check if all the profiles have been received.
                        if (counter == (negoDevicesIp.size()+1)) {
                            // Select the protocol
                            String saopProtocol = soapProtocol(negotiatingDevices);
                            outcomeSession = prepareData.startNegotiation(soapProtocol,
                            negotiatingDevices);
                        }

                    } else {
                        // send Error
                         counter--;
                        System.out.println("Profile Validation not successful,Negotiation
                        Session Paused");
                        throw new Exception("Validation failed");
                    }
              } else {
                     // Send Error
                      counter--;
                      System.out.println("Profile Validation not successful,Negotiation
                        Session Paused");
                       throw new Exception("Validation failed, Parameter name is
                          different");
                 }
             } else {
                // Send Error to RSP
                 counter--;
                System.out.println("Profile Validation not successful,Negotiation Session
                  Paused");
                throw new Exception("Validation failed, IP is not in Negotiating Devices IP
                   list");
            }
        }
        return outcomeSession;
    }
    /**
     * Check weight of parameter submitted in profile
     * @param profile
     * @return
     */
    public static  boolean sumWeightParameter(Profile profile) {
        boolean result = false;
        float sum = profile.getParameter1().getWeight() +
          profile.getParameter2().getWeight() + profile.getParameter3().getWeight();
        if (sum == 1) {
            result = true;
        }
        return result;
    }

    /**
     *  Validate MinMax Utility value
     * @param prof
     */
    public static  boolean validateMinMaxUtilityValue(Profile prof) {
        boolean validated = false;
        if ((prof.getParameter1().getMinValueUtility() >= 0 &&
        prof.getParameter1().getMinValueUtility() <= 1)
                && (prof.getParameter2().getMinValueUtility() >= 0
                &&prof.getParameter2().getMinValueUtility() <= 1)
                && (prof.getParameter3().getMinValueUtility() >= 0
                && prof.getParameter3().getMinValueUtility() <= 1)) {
            if ((prof.getParameter1().getMaxValueUtility() >= 0 &&
                    prof.getParameter1().getMaxValueUtility() <= 1)
                    && (prof.getParameter2().getMaxValueUtility() >= 0
                    && prof.getParameter2().getMaxValueUtility() <= 1)
                    && (prof.getParameter3().getMaxValueUtility() >= 0
                    && prof.getParameter3().getMaxValueUtility() <= 1)) {
                validated = true;
            }
        }

        return validated;
    }
}
```

C.2 OFFER STRATEGY OF A NEGOTIATING AGENT

Negotiating agents are required to generate and evaluate offers. To achieve this, an offer strategy needs to be bound with them during the negotiation process. Listing C.2 shows the Java implementation of how the offer strategy is bound to agents.

Listing C1: The Java implementation of the offer strategy of a negotiating agent.

```
package com.service.engine.strategy;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;

import com.Offer;
import com.offering.OfferDetails;
import com.service.NegotiationSession;
import com.service.NoModel;
import com.service.OMStrategy;
import com.service.OfferingStrategy;
import com.service.OpponentModel;
import com.service.SortedOutcomeSpace;
import com.service.opponentmodel.DefaultModel;
import com.service.sharedagentstate.anac2012.AgentSPSAS;
import com.issue.Issue;
import com.issue.IssueDiscrete;
import com.issue.IssueInteger;
import com.issue.IssueReal;
import com.issue.Value;
import com.issue.ValueDiscrete;
import com.issue.ValueInteger;
import com.issue.ValueReal;
import com.utility.AdditiveUtilitySpace;

/**
 * This is the decoupled offer strategy of AgentSP
 *
 * @author Udoh
 */
public class AgentSP_Offering extends OfferingStrategy {

     private boolean EQUIVALENCE_TEST = false;
     private Random random100;
     private ArrayList<Double> observationUtility = new ArrayList<Double>();
     private HashMap<Offer, Double> offerTables = new HashMap<Offer, Double>();
     private static boolean firstOffer;
     private static boolean forecastTime = true;
     private static boolean discountFactor;
     private static OfferDetails offereMaxOffer = null;
     private static double offereMaxUtility;
     private int currentOfferNumber = 0;
     private int lastOfferNumber = 1;
     private AdditiveUtilitySpace utilitySpace;
     private boolean alreadyDone = false;
     private SortedOutcomeSpace outcomeSpace;

     public AgentSP_Offering() {
     }

     public AgentSP_Offering(NegotiationSession negoSession, OpponentModel model,
     OMStrategy oms) throws Exception {
            init(negoSession, model, oms, null);
     }

     /**
      * Init required for the Decoupled negotiation framework.
      */
     @Override
     public void init(NegotiationSession negoSession, OpponentModel model, OMStrategy oms,
     Map<String, Double> parameters) throws Exception {
            super.init(negoSession, model, omStrategy, parameters);
            if (model instanceof DefaultModel) {
                model = new NoModel();
            }
            if (!(model instanceof NoModel)) {
                outcomeSpace = new SortedOutcomeSpace(negoSession.getUtilitySpace());
            }
            this.opponentModel = model;
            this.omStrategy = oms;

            helper = new AgentSPSAS(negotiationSession);
            firstOffer = true;
            try {
                utilitySpace = (AdditiveUtilitySpace) negoSession.getUtilitySpace();
                getDiscountFactor();
                getReservationFactor();
                Offer b = negoSession.getMaxOfferinDomain().getOffer();
                offerTables.put(b, getUtility(b));
                ((AgentSPSAS) helper).getOfferRunk().add(b);
                if (discountFactor) {
                    ((AgentSPSAS) helper).setSigmoidGain(-3.0);
                    ((AgentSPSAS) helper).setPercent(0.55);
                } else {
                    ((AgentSPSAS) helper).setSigmoidGain(-5.0);
                    ((AgentSPSAS) helper).setPercent(0.70);
                }
                if (EQUIVALENCE_TEST) {
                    random100 = new Random(100);
                } else {
                    random100 = new Random();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

    }

    @Override
    public OfferDetails determineOpeningOffer() {

            return determineNextOffer();
    }

    @Override
    public OfferDetails determineNextOffer() {
            if (negotiationSession.getOpponentOfferHistory().getHistory().isEmpty()) {
                if (!alreadyDone) {
                    ((AgentSPSAS) helper).updateMinimumOfferUtility(0);
                    alreadyDone = true;
                }
                return negotiationSession.getMaxOfferinDomain();

            }
            try {
                OfferDetails partnerOffer;
                if (firstOffer) {
                   partnerOffer =
                   negotiationSession.getOpponentOfferHistory().getHistory().get(0);
                } else {
                    partnerOffer = negotiationSession.getOpponentOfferHistory().
                    getLastOfferDetails();
                }
                double time = negotiationSession.getTime();
                double offeredutil;
                if (discountFactor) {
                    offeredutil = getUtility(partnerOffer.getOffer())* (1 /
                                  Math.pow(negotiationSession.getUtilitySpace().
                                  getDiscountFactor(), time));
                } else {
                    offeredutil = getUtility(partnerOffer.getOffer());

                }
                if (firstOffer) {
                    offereMaxOffer = partnerOffer;
                    offereMaxUtility = offeredutil;
                    ((AgentSPSAS) helper).setFirstOfferUtility(offeredutil);
                    observationUtility.add(offeredutil);
                    if (offeredutil > 0.5) {
                        ((AgentSPSAS) helper).setP(0.90);
                    } else {
                        ((AgentSPSAS) helper).setP(0.80);
                    }
```

```
                        firstOffer = !firstOffer;
                }
                ((AgentSPSAS) helper).updateMinimumOfferUtility(time);
                if (offeredutil > offereMaxUtility) {
                        offereMaxOffer = partnerOffer;
                        offereMaxUtility = offeredutil;
                        observationUtility.add(offeredutil);
                        if ((time > 0.5) && !discountFactor) {
                                newupdateSigmoidFunction();
                        }
                }
                if ((time > 0.5) && forecastTime) {
                        updateSigmoidFunction();
                        forecastTime = !forecastTime;
                }
                if (offereMaxUtility > ((AgentSPSAS) helper).getMinimumOfferUtility()){
                        nextOffer = offereMaxOffer;
                } else if (time > 0.985) {
                        if (offereMaxUtility > ((AgentSPSAS) helper).getReservation()) {
                                nextOffer = offereMaxOffer;
                        } else {
                                Offer nOffer = ((AgentSPSAS) helper).getOfferRunk()
                                .get(((AgentSPSAS)helper).getOfferRunk().size() -
                                lastOfferNumber);
                                nextOffer = new OfferDetails(nOffer,
                                negotiationSession.getUtilitySpace().getUtility(nOffer);
                                lastOfferNumber++;
                        }
                } else {
                     if (offeredutil > ((AgentSPSAS) helper).getMinimumOffereDutil()){
                                HashMap<Offer, Double> getOffers = getOfferTable(1);
                                if (getOffers.size() >= 1) {
                                        currentOfferNumber = 0;
                                        ((AgentSPSAS) helper).getOfferRunk().clear();
                                        offerTables = getOffers;
                                        sortOffer(getOffers);
                                } else {
                                        getOffers = getOfferTable(2);
                                        if (getOffers.size() >= 1) {
                                                sortOffer(getOffers);
                                                Offer maxOffer =
                                                getMaxOfferUtility(getOffers);
                                                currentOfferNumber =
                                                ((AgentSPSAS)helper).getOfferRunk()
                                                .indexOf(maxOffer);
                                        }
                                }
                                Offer nOffer = ((AgentSPSAS)
                                     helper).getOfferRunk().get(currentOfferNumber);
                                nextOffer = new OfferDetails(nOffer,
                                     negotiationSession.getUtilitySpace().getUtility
                                     (nOffer);
                                if (currentOfferNumber + 1 < ((AgentSPSAS)
                                     helper).getOfferRunk().size()) {
                                        currentOfferNumber++;
                                }
                        } else {
                                HashMap<Offer, Double> getOffers = getOfferTable(2);
                                if (getOffers.size() >= 1) {
                                        sortOffer(getOffers); // Sort OfferTable
                                        Offer maxOffer = getMaxOfferUtility(getOffers);
                                        currentOfferNumber = ((AgentSPSAS)
                                        helper).getOfferRunk().indexOf(maxOffer);
                                }
                                Offer nOffer = ((AgentSPSAS)
                                      helper).getOfferRunk().get(currentOfferNumber);
                                 nextOffer = new OfferDetails(nOffer,
                                      negotiationSession.getUtilitySpace().
                                      getUtility(nOffer);
                               if (currentOfferNumber + 1 < ((AgentSPSAS)
                                   helper).getOfferRunk().size()) {
                                        currentOfferNumber++;
                                } else {
                                        currentOfferNumber = 0;
                                }
                        }

                }
        } catch (Exception e) {
                e.printStackTrace();
        }
        if (!(opponentModel instanceof NoModel)) {
                try {
                    nextOffer = omStrategy.getOffer(outcomeSpace,
                    utilitySpace.getUtility(nextOffer.getOffer()));
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
        return nextOffer;

}

private void getReservationFactor() {
        if (utilitySpace.getReservationValue() != null) {
                ((AgentSPSAS)
                helper).setReservation(utilitySpace.getReservationValue());
        }
}

private void getDiscountFactor() {
        discountFactor = utilitySpace.isDiscounted();
}

private void newupdateSigmoidFunction() {
    double latestObservation = observationUtility.get(observationUtility.size() - 1);
    double concessionPercent = Math.abs(latestObservation - ((AgentSPSAS)
            helper).getFirstOffereUtility())
            / (1.0 - ((AgentSPSAS) helper).getFirstOffereUtility());
    double modPercent = Math
           .abs(((AgentSPSAS) helper).getMinimumOffereDutil()- ((AgentSPSAS)
            helper).getFirstOffereUtility())
           / (1.0 - ((AgentSPSAS) helper).getFirstOffereUtility());
   if (modPercent < concessionPercent) {
                ((AgentSPSAS) helper).setPercent(concessionPercent);
        }
}

private Offer getMaxOfferUtility(HashMap<Offer, Double> offerTable) {
        Double maxOfferUtility = 0.0;
        Offer maxOffer = null;
        for (Offer b : offerTable.keySet()) {
                if (getUtility(b) > maxOfferUtility) {
                        maxOfferUtility = getUtility(b);
                        maxOffer = b;
                }
        }
        return maxOffer;
}

/**
 * OfferTable
 *
 * @param offerTable
 */
private void sortOffer(final HashMap<Offer, Double> getOffers) {

        for (Offer offer : getOffers.keySet()) {
                offerTables.put(offer, getUtility(offer));
                ((AgentSPSAS) helper).getOfferRunk().add(offer); // Add offerRunk
        }

        if (!EQUIVALENCE_TEST) {
                Collections.sort(((AgentSPSAS) helper).getOfferRunk(), new
                   Comparator<Offer>() {
                        @Override
                        public int compare(Offer o1, Offer o2) {
                                return (int) Math.ceil(-(offerTables.get(o1) -
                                offerTables.get(o2)));
                        }
                });
        }
}

private Offer clone(Offer source) throws Exception {
        HashMap<Integer, Value> hash = new HashMap<Integer, Value>();
        for (Issue i : utilitySpace.getDomain().getIssues()) {
                hash.put(i.getNumber(), source.getValue(i.getNumber()));
        }
        return new Offer(utilitySpace.getDomain(), hash);
	}

	/**
	 * @param maxOffer
	 * @return
	 * @throws Exception
	 */
	private HashMap<Offer, Double> getOfferTable(int flag) throws Exception {
		HashMap<Offer, Double> getOffers = new HashMap<Offer, Double>();
		List<Issue> issues = utilitySpace.getDomain().getIssues();
		Offer standardOffer = null;
		for (Issue lIssue : issues) {
			switch (lIssue.getType()) {
			case DISCRETE:
				IssueDiscrete lIssueDiscrete = (IssueDiscrete) lIssue;
				for (ValueDiscrete value : lIssueDiscrete.getValues()) {
					if (flag == 0) {
						standardOffer =
						utilitySpace.getMaxUtilityOffer();
					} else if (flag == 1) {
						standardOffer =
						negotiationSession.getOpponentOfferHistory().
						getLastOffer();
					} else {
						standardOffer = ((AgentSPSAS)
						helper).getOfferRunk().get(currentOfferNumber);
					}
					standardOffer = clone(standardOffer);
					standardOffer =
					   standardOffer.putValue(lIssue.getNumber(), value);
					double utility = getUtility(standardOffer);
					if ((utility > ((AgentSPSAS)
					   helper).getMinimumOfferUtility())
					   && (!((AgentSPSAS) helper).getOfferRunk().
					   contains(standardOffer))){
					   getOffers.put(standardOffer, utility);
					}
				}
				break;
			case REAL:
				IssueReal lIssueReal = (IssueReal) lIssue;
				int optionInd =
				  random100.nextInt(lIssueReal.getNumberOfDiscretizationSteps()
				  1);
				Value pValue = new ValueReal(
				  lIssueReal.getLowerBound() + (lIssueReal.getUpperBound() -
				  lIssueReal.getLowerBound())
				  * (double) (optionInd) /
				  (double)(lIssueReal.getNumberOfDiscretizationSteps()));
				standardOffer =
				  standardOffer.putValue(lIssueReal.getNumber(),pValue);
				double utility = getUtility(standardOffer);
				getOffers.put(standardOffer, utility);
				break;
			case INTEGER:
				IssueInteger lIssueInteger = (IssueInteger) lIssue;
				  int optionIndex2 = lIssueInteger.getLowerBound()
				  + random100.nextInt(lIssueInteger.getUpperBound() -
				  lIssueInteger.getLowerBound());
				Value pValue2 = new ValueInteger(optionIndex2);
				standardOffer =
				  standardOffer.putValue(lIssueInteger.getNumber(), pValue2);
				double utility2 = getUtility(standardOffer);
				getOffers.put(standardOffer, utility2);
				break;
			default:
				throw new Exception("issue type " + lIssue.getType() + " not
				  supported by AgentSP");
			}
		}

		return getOffers;
	}

	public double getUtility(Offer offer) {
		return negotiationSession.getUtilitySpace().getUtilityWithDiscount(offer,
		negotiationSession.getTimeline());
	}

	private void updateSigmoidFunction() {
		int observationSize = observationUtility.size();
		double latestObservation = observationUtility.get(observationSize - 1);
		double concessionPercent = Math.abs(latestObservation - ((AgentSPSAS)
		helper).getFirstOffereUtility())
				/ (1.0 - ((AgentSPSAS) helper).getFirstOffereUtility());
		if (discountFactor) {
			if ((concessionPercent < 0.20) || (observationSize < 3)) {
				((AgentSPSAS) helper).setPercent(0.35);
				((AgentSPSAS) helper).setSigmoidGain(-2);
			} else {
				((AgentSPSAS) helper).setPercent(0.45);
			}
		} else {
			if ((concessionPercent < 0.20) || (observationSize < 3)) {
				((AgentSPSAS) helper).setPercent(0.50);
				((AgentSPSAS) helper).setSigmoidGain(-4);
			} else if (concessionPercent > 0.60) {
				((AgentSPSAS) helper).setPercent(0.80);
				((AgentSPSAS) helper).setSigmoidGain(-6);
			} else {
				((AgentSPSAS) helper).setPercent(0.60);
			}
		}
	}

	@Override
	public String getName() {
		return "AgentSP";
	}
```

C.3 AGENT OFFER OPERATIONS

During negotiation, agents carry out specific operations such as initialising offers, evaluating offers and computing the utility of an opponent's offer. Listing C.3 shows the Java implementation of the functions related to an agents offer.

Listing C3: The Java implementation of agent offer operations.

```java
package com.service.agent

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import com.Offer;
import com.Domain;
import com.offerding.OfferDetails;
import com.service.NegotiationSession;
import com.service.NoModel;
import com.service.OMStrategy;
import com.service.OpponentModel;
import com.service.SharedAgentState;
import com.service.SortedOutcomeSpace;
import com.issue.Issue;
import com.issue.Value;
import com.issue.ValueInteger;
import com.issue.ValueReal;
import com.utility.AdditiveUtilitySpace;
import com.utility.Evaluator;
import com.utility.EvaluatorDiscrete;
import com.utility.EvaluatorInteger;
import com.utility.EvaluatorReal;
import com.service.agent.OffersComparator;
import com.service.agent.OpponentOffers;

public class AgentOP extends SharedAgentState {
    private AdditiveUtilitySpace utilitySpace;
    private OpponentOffers opponentOffers;
    private ArrayList<Offer> allOffers = null;
    private Offer maxLastOpponentOffer;
    private int numPossibleOffers = 0;
    private int index = 0;
    private double lastTimeLeft = 0;
    private int minSize = 160000;
    private Offer myBestOffer = null;
    private OpponentModel opponentModel;
    private SortedOutcomeSpace outcomeSpace;
    private OMStrategy oms;

    public AgentOP(NegotiationSession negoSession,
                   OpponentOffers opponentOffers, OpponentModel opponentModel,
                   OMStrategy oms) {
            NAME = "AgentLR";
            this.oms = oms;
            this.utilitySpace = (AdditiveUtilitySpace) negoSession
                          .getUtilitySpace();
            this.opponentOffers = opponentOffers;
            this.opponentModel = opponentModel;
            if (!(opponentModel instanceof NoModel)) {
                   outcomeSpace = new SortedOutcomeSpace(utilitySpace);
            }
    }

    private void initOffers() {
            allOffers = getAllOffers();
            OffersComparator offersComparator = new OffersComparator(utilitySpace);
            // sort the offers in order of highest utility
            Collections.sort(allOffers, offersComparator);
    }

    /**
     * Calculate the next offer for the agent (from 1/4 most optimal offers)
     *
     */
    public OfferDetails getNextOffer(double time) {
            OfferDetails currentAction = null;
            try {
                Offer newOffer = allOffers.get(index);
                currentAction = new OfferDetails(newOffer,
                                utilitySpace.getUtility(newOffer));
                index++;
                if (index > numPossibleOffers) {
                        // the time is over compromising in a high rate
                        if (time >= 0.9) {
                                if (time - lastTimeLeft > 0.008) {
                                    double myBestUtility = utilitySpace
                                        .getUtility(myBestOffer);
                                    double oppBestUtility = utilitySpace
                                        .getUtility(opponentOffers.getOpponentsOffers()
                                        .get(0));
                                    double avg = (myBestUtility + oppBestUtility)/2;
                                     if (index >= allOffers.size())
                                            index = allOffers.size() - 1;
                                    else if (utilitySpace.getUtility(allOffers
                                            .get(index)) < avg) {
                                    else if (utilitySpace.getUtility(allOffers
                                            .get(index)) < avg) {
                                      index--;
                                      double maxUtilty = 0;
                                      int maxOfferIndex = numPossibleOffers;
                                      for(int i = numPossibleOffers; i <= index; i++){
                                              double utiliy = getUtilityOfOpponentsOffer(
                                                 utilitySpace.getDomain(),
                                                 allOffers.get(i));
                                              if (utiliy > maxUtilty) {
                                                maxUtilty = utiliy;
                                                maxOfferIndex = i;
                                              }
                                        }
                                        numPossibleOffers = maxOfferIndex;
                                        } else
                                            index--;
                                } else
                                   index = 0;
                        } else {
                           index = 0;
                           double discount = utilitySpace.getDiscountFactor();
                           if (time - lastTimeLeft > 0.05) {
                                if (utilitySpace.getUtility(opponentOffers
                                    .getMaxUtilityOfferForMe()) > utilitySpace
                                    .getUtility(maxLastOpponentOffer)
                                    || (discount < 1 && time - lastTimeLeft > 0.1)) {
                                    double maxUtilty = 0;
                                    for (int i = 0; i <= numPossibleOffers; i++) {
                                        double utiliy = getUtilityOfOpponentsOffer(
                                            utilitySpace.getDomain(),
                                            allOffers.get(i));
                                        if (utiliy > maxUtilty){
                                            maxUtilty = utiliy;
                                        }
                                        for (int i = numPossibleOffers + 1; i < allOffers
                                             .size(); i++) {
                                           double utiliy = getUtilityOfOpponentsOffer(
                                               utilitySpace.getDomain(),
                                               allOffers.get(i));
                                           if (utiliy >= maxUtilty) {
                                                numPossibleOffers = i;
                                                break;
                                           }
                                        }
                                        maxLastOpponentOffer = opponentOffers
                                           .getMaxUtilityOfferForMe();
                                        lastTimeLeft = time;
                                    }
                           }
                     }
                }
        } catch (Exception e) {
                e.printStackTrace();
        }
        if (!(opponentModel instanceof NoModel)) {
                try {
                        currentAction = oms.getOffer(outcomeSpace,
                             utilitySpace.getUtility(currentAction.getOffer()));
                } catch (Exception e) {
                        e.printStackTrace();
                }
```

```java
		}
		return currentAction;
	}

	/**
	 * Calculate the next optimal offer for the agent (from 1/4 most optimal offers)
	 *
	 */
	public OfferDetails getNextOptimicalOffer(double time) {
		OfferDetails currentAction = null;
		Offer newOffer = null;
		try {
			if (allOffers == null)
				initOffers();
			newOffer = allOffers.get(index);
			currentAction = new OfferDetails(newOffer,
					utilitySpace.getUtility(newOffer));
			index++;
			double myBestUtility = utilitySpace.getUtilityWithDiscount(
					myBestOffer, time);
			double oppBestUtility = utilitySpace.getUtilityWithDiscount(
					opponentOffers.getOpponentsOffers().get(0), time);
			double downBond = myBestUtility - (myBestUtility - oppBestUtility)
					/ 4;
			// check if time passes and compromise a little bit
			if (time - lastTimeLeft > 0.1
					&& numPossibleOffers < allOffers.size() - 1
					&& downBond <= utilitySpace.getUtilityWithDiscount(
						allOffers.get(numPossibleOffers + 1), time)) {
				double futureUtility = utilitySpace.getUtilityWithDiscount(
						allOffers.get(numPossibleOffers), time + 0.1);
				while (utilitySpace.getUtilityWithDiscount(
					allOffers.get(numPossibleOffers), time) >= futureUtility
						&& numPossibleOffers < allOffers.size() - 1)
					numPossibleOffers++;
				lastTimeLeft = time;
			}
			if (index > numPossibleOffers)
				index = 0;
		} catch (Exception e) {
			e.printStackTrace();
		}
		maxLastOpponentOffer = opponentOffers.getMaxUtilityOfferForMe();

		if (!(opponentModel instanceof NoModel)) {
			try {
				currentAction = oms.getOffer(outcomeSpace,
					utilitySpace.getUtility(currentAction.getOffer()));
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
		return currentAction;

	}

	/*
	 * returns the Evaluator of an issue
	 */
	public Evaluator getMyEvaluator(int issueID) {
		return utilitySpace.getEvaluator(issueID);
	}

	/*
	 * returns all offers
	 */
	private ArrayList<Offer> getAllOffers() {
		ArrayList<Offer> offers = new ArrayList<Offer>();
		List<Issue> issues = utilitySpace.getDomain().getIssues();

		HashMap<Integer, Value> issusesFirstValue = new HashMap<Integer, Value>();
		for (Issue issue : issues) {

			Value v = getIsuueValues(issue).get(0);
			issusesFirstValue.put(issue.getNumber(), v);
		}
		try {
			offers.add(new Offer(utilitySpace.getDomain(), issusesFirstValue));
		} catch (Exception e) {
			e.printStackTrace();
```

```
            }

            for (Issue issue : issues) {
               ArrayList<Offer> tempOffers = new ArrayList<Offer>();
               ArrayList<Value> issueValues = getIsuueValues(issue);
               for (Offer offer : offers) {
                 for (Value value : issueValues) {
                    HashMap<Integer, Value> lNewOfferValues = getOfferValues(offer);
                    lNewOfferValues.put(issue.getNumber(), value);
                    try {
                           Offer newOffer = new Offer(utilitySpace.getDomain(),
                                                         lNewOfferValues);
                                        tempOffers.add(newOffer);

                        } catch (Exception e) {
                                        e.printStackTrace();
                          }
                     }
                 }
                 offers = tempOffers;
            }

            // remove offers that are not good enough (the utility is less than 1/4 of
            // the difference between the negotiating agents)
            double myBestUtility = 1;
            double oppBestUtility = 0;
            try {
                    myBestOffer = utilitySpace.getMaxUtilityOffer();
                    myBestUtility = utilitySpace.getUtility(myBestOffer);
                    oppBestUtility = utilitySpace.getUtility(opponentOffers
                                .getOpponentsOffers().get(0));
            } catch (Exception e1) {
                    e1.printStackTrace();
            }
            return filterOffers(offers, myBestUtility, oppBestUtility, 0.75D);
    }

    private ArrayList<Offer> filterOffers(ArrayList<Offer> offers,
                    double myBestUtility, double oppBestUtility, double fraction) {
        double downBond = myBestUtility - (myBestUtility - oppBestUtility)
                           * fraction;
        ArrayList<Offer> filteredOffers = new ArrayList<Offer>();
        for (Offer offer : offers) {
            try {
            double reservation = utilitySpace.getReservationValue() != null ? utilitySpace
                                        .getReservationValue() : 0;
                           if (utilitySpace.getUtility(offer) < downBond
                                        || utilitySpace.getUtility(offer) < reservation)
                                continue;
                           else
                                filteredOffers.add(offer);

                    } catch (Exception e) {
                           e.printStackTrace();
                    }
            }
            if (filteredOffers.size() < minSize) {
                    return filteredOffers;
            }
            return filterOffers(filteredOffers, myBestUtility, oppBestUtility,
                           fraction * 0.85D);
    }

    /*
     * returns offer values
     */
    private HashMap<Integer, Value> getOfferValues(Offer offer) {
            HashMap<Integer, Value> offerValues = new HashMap<Integer, Value>();
            List<Issue> allIsuues = utilitySpace.getDomain().getIssues();
            for (Issue issue : allIsuues) {
                    try {
                           offerValues.put(issue.getNumber(),
                                        offer.getValue(issue.getNumber()));
                    } catch (Exception e) {
                           e.printStackTrace();
                    }

            }
            return offerValues;
    }
```

```java

    /*
     * returns issue values
     */
    public ArrayList<Value> getIsuueValues(Issue issue) {

            Evaluator e = getMyEvaluator(issue.getNumber());
            ArrayList<Value> retValues = new ArrayList<Value>();
            switch (e.getType()) {
            case DISCRETE:
                    EvaluatorDiscrete eD = ((EvaluatorDiscrete) e);
                    retValues.addAll(eD.getValues());
                    break;
            case REAL:
                    EvaluatorReal eR = ((EvaluatorReal) e);

                    double intervalReal = (eR.getUpperBound() - eR.getLowerBound()) / 10;
                    for (int i = 0; i <= 10; i++) {
                            retValues.add(new ValueReal(eR.getLowerBound() + i
                                            * intervalReal));
                    }
                    break;
            case INTEGER:
                    EvaluatorInteger eI = ((EvaluatorInteger) e);

                    int intervalInteger = (eI.getUpperBound() - eI.getLowerBound()) / 10;
                    for (int i = 0; i <= 10; i++) {
                            retValues.add(new ValueInteger(eI.getLowerBound() + i
                                            * intervalInteger));
                    }
                    break;
            }
            return retValues;
    }

    /*
     * returns the minimum utility of the offer that the agent voted
     */
    public double getMyOffersMinUtility(double time) {
            if (allOffers == null)
                    initOffers();
            return utilitySpace.getUtilityWithDiscount(
                            allOffers.get(numPossibleOffers), time);
    }

    /*
     * returns the offer with the minimum utility that the agent voted
     */
    public Offer getMyminOfferfromOffers() {
            if (allOffers == null)
                    initOffers();
            return allOffers.get(numPossibleOffers);
    }

    /*
     * returns the offer utility
     */
    public double getUtility(Offer offer) {
            try {
                    return utilitySpace.getUtility(offer);
            } catch (Exception e) {
                    e.printStackTrace();
            }
            return 0;
    }

    public double getUtilityOfOpponentsOffer(Domain domain, Offer offer) {
            double utility;
            if (opponentModel instanceof NoModel) {
                    utility = opponentOffers.getOpponentOfferUtility(
                                    utilitySpace.getDomain(), offer);
            } else {
                    utility = opponentModel.getOfferEvaluation(offer);
            }
            return utility;
    }
}
```