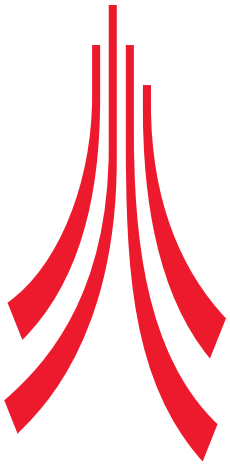
An Integrated Framework for Ensuring Runtime Quality in Service-Oriented Systems

Daniel Barnaby Robinson
Computer Science B.Sc. (Hons)



Computing Department
Lancaster University

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

June 2009

An Integrated Framework for Ensuring Runtime Quality in Service-Oriented Systems

Daniel Barnaby Robinson
Computer Science B.Sc. (Hons)

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

June 2009

As a relatively new software model, there remain many challenges in realising a true service-oriented vision. The service-oriented systems which underpin modern business processes must be able to react to constantly changing environments and business requirements. The quality of a service-oriented system depends not only on the quality of service (QoS) provided by services, but on the interdependencies between services, resource constraints imposed by the runtime environment, and events such as network outages. It is difficult to anticipate the impact that many of these emergent factors will have on the behaviour of the system. The third-party nature of software services also presents the service consumer with limited control over the quality of a system.

Existing quality assurance initiatives for service-oriented systems are currently limited in the service quality control they offer the consumer, provide poor support for expressing quality characteristics, provide poor support for quality assurance at runtime, provide poor support for resource-restricted systems, and offer limited scope for integration and customisation to provide an end-to-end quality assurance solution. To address these issues, this thesis presents the development of an integrated quality assurance framework, which combines quality assurance approaches from the service description and selection, service monitoring, service negotiation, and reputation system domains. The approach is illustrated with a series of service-oriented experiments, which evaluate the role of the framework in the system quality assurance process.

Declaration

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

Daniel Barnaby Robinson
8th June, 2009

Acknowledgements

I would first like to thank my supervisor, Gerald Kotonya, for the thoughtful guidance and encouragement he has shown me whilst undertaking this research journey. I am also grateful to Keith Bennett and Peter Sawyer, for sitting on my thesis defence panel. I would also like to thank Simon Lock and Peter Sawyer, for sitting on both my first and second year review panels. Finally, I would like to thank the Engineering and Physical Sciences Research Council (EPSRC), for funding my time as a research student at Lancaster University.

Related Publications

1. Robinson, D., & Kotonya, G. (2008a). A runtime quality architecture for service-oriented systems. In A. Bouguettaya, I. Krüger, & T. Margaria (Eds.), *International conference on service-oriented computing (ICSOC)* (Vol. 5364, p. 468-482).

2. Robinson, D., & Kotonya, G. (2008b). A self-managing brokerage model for quality assurance in service-oriented systems. In X. Li, C. S. Smidts, & J. Xu (Eds.), *High assurance systems engineering (HASE)* (p. 424-433). IEEE Computer Society.

Contents

Abstract **1**

Declaration **2**

Acknowledgements **3**

Related Publications **4**

Contents **5**

1 Introduction **13**

1.1 Problem Statement 13
1.2 Key Issues with Existing Work 14
1.3 Research Objectives 15
1.4 Research Contributions 15
1.5 Thesis Structure 17

2 Service-Oriented Systems **19**

2.1 Software as a Service (SaaS) 19
2.2 Service-Oriented Architecture (SOA) 21
2.3 Service Description 22
2.4 Service Discovery and Selection 23
2.5 Service-Level Agreement (SLA) 24
2.6 Service Composition 24
2.6.1 Orchestration 25
2.6.2 Choreography 26
2.7 System Domains 26
2.7.1 Business and Enterprise 27
2.7.2 Grid and Utility Computing 27

2.7.3 Embedded Systems 27
2.7.4 Ubiquitous and Mobile Computing 28
2.8 Summary 28

3 Service Quality Assurance **29**
3.1 Software Quality 30
3.2 Ensuring Software Quality 30
3.3 Service Quality Characteristics 31
3.4 Approaches for Service Quality Assurance 32
3.5 Service Description 33
3.5.1 Agreement Approaches 34
3.5.2 Semantic Approaches 35
3.6 Service Discovery and Selection 38
3.7 Service Reputation Systems 41
3.8 Service Negotiation 46
3.9 Service Monitoring 51
3.10 Overview and Integration Discussion 58
3.11 Summary 60

4 Quality Assurance Framework **61**
4.1 Framework Overview 61
4.2 Brokerage System 63
4.2.1 Broker Engine 64
4.2.2 Negotiation Protocol 66
4.2.3 Negotiation Model 68
4.3 Monitoring System 69
4.3.1 Service Renegotiation 72
4.3.2 Forecasting Future QoS 73
4.4 Reputation System 73
4.5 Service Ontology 74
4.5.1 Quality Schema 75
4.5.2 Service Schema 77
4.5.3 Strategy Schema 79
4.6 Service Acceptability 85
4.7 Customisable Components 87
4.7.1 Monitor Assembly 88

4.7.2 Extending Forecast Support . . . 88
4.8 Summary . . . 91

5 Evaluation **92**
5.1 Service Simulation . . . 92
5.2 Service Doping . . . 93
5.3 Service Strategy . . . 94
5.3.1 Consumer Strategy . . . 95
5.3.2 Provider Strategy . . . 96
5.4 Initial Service Provider Selection . . . 97
5.4.1 Initial Service Provider Reputation . . . 98
5.4.2 Initial Service Negotiation . . . 99
5.4.3 Initial Service Provider Selection Results . . . 102
5.5 Service Monitoring . . . 104
5.6 Regular Service Performance . . . 104
5.7 Recurring Service Failure . . . 107
5.8 Handling SLA Violations . . . 112
5.8.1 Severe Violations . . . 113
5.8.2 Moderate Violations . . . 113
5.9 Service Outage . . . 114
5.10 Off-Peak and Peak Service Performance . . . 116
5.11 Consumer Competition . . . 120
5.12 Framework Limitations . . . 120
5.12.1 No Acceptable Services . . . 122
5.12.2 Framework System Unavailability . . . 123
5.13 Summary . . . 124

6 Conclusions **126**
6.1 Objectives Revisited . . . 126
6.2 Future Research Directions . . . 129
6.3 Concluding Remarks . . . 132

A Ontology XML Examples **133**
A.1 Example Service Contract . . . 134
A.2 Example Service Strategy . . . 135

B System Visualisation Tool **138**
B.1 Initial State 138
B.2 Negotiation Visualisation 140
B.3 Service Performance Visualisation 145

References **147**

List of Figures

2.1 Service as a collection of capabilities 20
2.2 Typical service interaction 21
2.3 SOA interaction pattern 22
2.4 Service discovery and selection 23
2.5 Example service composition 25
2.6 Service orchestration overview 25
2.7 Service choreography overview 26

4.1 Quality assurance framework overview 62
4.2 Framework brokerage architecture 63
4.3 Broker engine overview 64
4.4 Proposal engine overview 65
4.5 Negotiation message overview 66
4.6 Consumer negotiation protocol states 67
4.7 Provider negotiation protocol states 67
4.8 Overview of the bargaining negotiation process 69
4.9 Primary framework monitoring architecture 70
4.10 Alternative service monitoring approaches 71
4.11 Service monitoring and brokerage system integration 72
4.12 Reputation system integration 74
4.13 Quality ontology elements overview 75
4.14 Quality element 75
4.15 Constraint element 76
4.16 Quality ontology XML example 76
4.17 Service ontology elements overview 77
4.18 Service element 78
4.19 ServiceContract element 78
4.20 OperationContract element 79

4.21 Service ontology XML example 80
4.22 Strategy ontology elements overview 80
4.23 Strategy element 81
4.24 ServiceStrategy element 82
4.25 OperationStrategy element 83
4.26 QualityStrategy element 84
4.27 ConstraintStrategy element 84
4.28 Overview of customisable framework components 88
4.29 Monitor assembly components overview 89
4.30 Forecaster component architecture 90

5.1 Service doping architecture 94
5.2 Initial map service provider reputation 98
5.3 Initial historical map service provider ratings 99
5.4 Initial negotiation with broker of map service provider 1 100
5.5 Initial accepted map service proposal 100
5.6 Initial negotiation sessions with other map service provider brokers 101
5.7 Initial map service proposal acceptability 102
5.8 Initial overall map service provider acceptability 103
5.9 Initial overall navigation composition acceptability 103
5.10 Regular service monitor events 105
5.11 Regular service invocations 106
5.12 Regular composition invocations 106
5.13 Recurring map service response time failure 108
5.14 Recurring map service failures and switching providers 109
5.15 Updated map service provider ratings 109
5.16 Updated map service provider reputation 110
5.17 Updated overall map service provider acceptability 110
5.18 Renegotiation with map service provider 1 111
5.19 Recurring map service failure impact on composition 112
5.20 Map service failure, followed by provider QoS stabilisation 114
5.21 Map service failure, followed by another failure after renegotiation . 115
5.22 Weather service outage and availability failure 116
5.23 Weather service outage and switching provider 117
5.24 Weather service outage impact on navigation composition 117
5.25 Location service response time failure during peak usage period . . 118

5.26 Location service failure and renegotiation 119
5.27 Peak service performance impact on navigation composition 119
5.28 Negotiation failure due to consumer competition 121
5.29 No acceptable map service proposals 122

B.1 System visualisation tool initialised state 139
B.2 Negotiation session list viewer 140
B.3 Negotiation session viewer 141
B.4 Negotiation message viewer 142
B.5 Service provider proposal viewer 142
B.6 Service provider reputation viewer 143
B.7 Service provider overall acceptability viewer 143
B.8 Composition proposal acceptability viewer 144
B.9 Composition overall acceptability viewer 144
B.10 Monitor event viewer 145
B.11 Service invocation acceptability viewer 146
B.12 Composition invocation acceptability viewer 146

List of Tables

3.1 Service description research initiatives 34
3.2 Service discovery and selection research initiatives 38
3.3 Service reputation system research initiatives 42
3.4 Service negotiation research initiatives 47
3.5 Service monitoring research initiatives 53
3.6 Summary of quality assurance domains 59

5.1 Size and complexity of simulation data 93
5.2 Strategy summary for the automobile navigation system 96
5.3 Map service strategy for the automobile navigation system 96
5.4 Summarised service strategies for the map service providers 97
5.5 Different framework configuration scenarios 124

Chapter 1

Introduction

1.1 Problem Statement

Constant change and innovation are characteristics of modern business, driven by quality-oriented organisations that are continuously adapting to shifts in the business environment (Truex et al., 1999). The service-oriented systems which underpin modern business processes must be able to react to constantly changing environments and business requirements, while maintaining satisfactory levels of system quality.

As a relatively new software model, there remain many challenges in realising a true service-oriented vision. The dynamic nature and complexity of systems composed from services poses a particular challenge for managing system quality (Woodside & Menascé, 2006). The quality of a service-oriented system depends not only on the quality of service (QoS) provided by service vendors, but on the interdependencies between services, resource constraints imposed by the runtime environment, and events such as network outages. It is difficult to anticipate the impact that many of these emergent factors will have on the behaviour of the system. The third-party nature of software services also presents the service consumer with limited control over the system quality. While the consumer may obtain a service-level agreement (SLA) from a service provider, which describes QoS guarantees for the non-functional attributes of a service, such agreements are difficult to enforce and do not themselves provide the means to ensure the quality of a system.

1.2 Key Issues with Existing Work

Current approaches to ensuring quality in service-oriented systems focus on discrete aspects of the quality assurance process, and are not intended to provide an end-to-end solution. These aspects include the enhancement of service characteristics with non-functional and semantic information, the negotiation of QoS characteristics, the monitoring of service performance, and systems which collate and report the reputation of service vendors. Current quality assurance initiatives for service-oriented systems have the following key issues:

- *Limited consumer control over service quality.* Existing quality assurance initiatives primarily focus on predicting system quality from static service properties (Grassi & Patella, 2006). These approaches provide the consumer with little control over service quality, outside of the static SLA it obtains from a service provider. SLAs are difficult to enforce, and require services to be monitored for compliance.

- *Poor support for expressing quality characteristics.* Existing service description languages primarily focus on the structural properties of a service, and provide poor support for expressing non-functional service characteristics. This makes it difficult for the consumer to accurately discover, compose and substitute services which match its non-functional requirements (O'Sullivan et al., 2002).

- *Poor support for runtime quality assurance.* Existing service monitoring initiatives are largely manual approaches which focus on static service properties. Static monitoring approaches provide poor support for detecting and responding to emergent runtime quality problems in the service execution environment. As such, existing service monitoring initiatives lack the ability to effectively recover from an SLA violation or service failure.

- *Poor support for resource-restricted systems.* Existing quality assurance initiatives require the service consumer to expend significant resources on activities such as service negotiation and monitoring. Ensuring quality is particularly problematic for service-oriented systems which operate in resource-restricted environments (Milanovic et al., 2004). Not only must a service provide an acceptable QoS, but it must be capable of being integrated within the resource constraints of the service consumer.

- *Limited scope for integration and customisation.* Existing quality assurance approaches for service-oriented systems focus on discrete aspects of the quality assurance process, and lack support for integrating with approaches from other domains. The multitude of techniques within each quality assurance domain requires the solution to support integration and customisation.

In addition, many existing quality assurance approaches for service-oriented systems have not been evaluated on real service-oriented systems. The lack of a significant evaluation means many initiatives are unable to adequately demonstrate their suitability for purpose. In some cases, it is possible to attribute evaluation shortcomings to the closed nature and complexity of real-world service-oriented systems.

1.3 Research Objectives

The principle aim of this research is to improve upon existing quality assurance approaches for service-oriented systems, by addressing the key issues identified in Section 1.2. The primary research objective is to develop and evaluate an integrated quality assurance solution for service-oriented systems. The specific research objectives of this work are to:

- provide the service consumer with increased control over service quality
- provide support for the expression of quality characteristics, as a means of supporting the quality assurance process
- provide a runtime solution that can detect and recover from SLA violations and service failures
- provide a solution that supports resource-restricted systems
- provide customisation support for integrating different quality assurance techniques and facilitating experimentation

1.4 Research Contributions

The first key research contribution of this thesis is an extensive literature review, documenting the state of the art in quality assurance solutions for service-oriented systems.

The second key research contribution is an integrated quality assurance framework, which improves upon existing approaches for ensuring quality in service-oriented systems. The approach integrates quality assurance techniques from the service description, discovery and selection, reputation, negotiation, and monitoring research domains, and provides a pluggable and extensible assurance solution.

Service-oriented architecture (SOA) is essentially a technology-independent concept. The framework has been implemented and evaluated using the Jini (Sun Microsystems, Inc., 2009b) service architecture. As there are a variety of different SOA implementations, it may be possible to apply the framework to other architectures, such as the Web Services Architecture (W3C Working Group, 2004). The framework has been evaluated with a simulated street navigation application composed of multiple Jini services, which is required by different simulated consumer devices, such as an automobile navigation system or mobile phone.

The key contributions provided by the integrated quality assurance framework for service-oriented systems are now listed, with a brief discussion of how they address the research objectives stated in Section 1.3:

- *Service brokerage system.* The framework provides a pluggable service brokerage system, which supplies an automated runtime method for securing SLAs that are closer to meeting consumer service requirements. The system also compensates service providers accordingly. The brokerage system provides the consumer with the means to handle and recover from SLA violations and service failures, by renegotiating and substituting problematic services. Service brokers conduct the negotiation process independently on behalf of service consumers and providers, making the brokerage approach well-suited for resource-restricted systems. The pluggability of the brokerage system enables the integration of different negotiation models, decision algorithms and service strategies.

- *Service monitoring system.* The framework provides a pluggable service monitoring system for measuring runtime service performance, auditing SLAs for compliance, and for forecasting trends in QoS. The monitoring system informs the service consumer of any SLA violations and service failures, and provides the consumer with the impetus for activating recovery techniques, such as service renegotiation and substitution. The monitoring system is capable of performing its quality assurance tasks independently of the ser-

vice consumer and provider, removing the monitoring burden from resource-restricted systems. The pluggability of the monitoring system enables the integration of different quality measurement components, SLA auditors and forecasting models.

- *Reputation system.* The framework provides a reputation system for facilitating trust between service consumers and providers. The reputation system collates service ratings provided by service consumers. These ratings reflect the collective consumer experience of particular services and service providers. The reputation system integrates with the framework's brokerage system, to provide the service consumer with additional criteria during the service negotiation and selection processes. This reputation criteria is particularly useful when the consumer has no prior experience of a service provider, and improves the control over its choice of service provider.

- *Quality ontology.* The framework provides a quality ontology for the expression of service quality characteristics, strategies for consumer and provider service requirements, and for the specification of SLAs between consumers and providers. The ontology provides a common set of quality attribute descriptions for service consumers and providers, with the aim of reducing ambiguity and misunderstanding. The quality ontology is integrated by the framework's service brokerage, monitoring and reputation systems, and supports the automation of the quality assurance processes they fulfill.

- *Service doping approach.* The development of the framework includes the design and implementation of a service doping approach, used to simulate different QoS scenarios in service-oriented systems. The approach enables experiments which would be difficult to perform otherwise, without a suitable service marketplace and co-operation from commercial service providers.

Some of the results of this research have been published in two conference papers (Robinson & Kotonya, 2008a, 2008b).

1.5 Thesis Structure

Chapter 2 continues the thesis with the background into service-oriented systems, and the principles of the *software as a service* development model. The chap-

ter then identifies system domains where service-oriented software development is being used, and the specific quality issues that impact systems in these domains.

Chapter 3 begins with an introduction to the subject of software quality assurance, and traditional approaches for ensuring software quality. The chapter then examines the particular quality characteristics of services, and provides an overview of the current quality assurance approaches for service-oriented systems. Chapter 3 continues with a detailed survey of existing research initiatives for tackling quality issues in service-oriented systems. These initiatives are categorised and discussed within the domains of service description, service discovery and selection, service reputation systems, service negotiation, and service monitoring. The chapter concludes by discussing the limitations of the existing quality assurance initiatives for service-oriented systems, and puts forward the potential benefits of an integrated solution comprised of techniques from each quality assurance domain.

Chapter 4 discusses the design and implementation of an integrated quality assurance framework for service-oriented systems, which is a key contribution of this research. The discussion begins with the three integrated systems provided by the framework for brokering, monitoring and rating services. The chapter then discusses a service quality ontology for the expression of service characteristics, strategies and SLAs. The chapter continues with an explanation of how strategies are specified for consumer and provider service requirements, and the utility-based formulas used to calculate service acceptability. The chapter concludes with a discussion and example of the customisable aspects of the quality assurance framework.

Chapter 5 provides a series of experiments based on a simulated service-oriented application, in order to demonstrate and evaluate the quality assurance framework against the research objectives stated in Section 1.3. The evaluation presents several experiments developed using a service doping approach, which provides the means to simulate different quality issues experienced by a service-oriented system. Within each experiment, the role of the quality assurance framework in resolving these issues is demonstrated. The evaluation also examines scenarios which highlight the limitations of the current quality assurance framework.

Chapter 6 provides an evaluation of the quality assurance framework, and assesses the extent to which the research objectives have been addressed. The chapter then discusses future research directions the work could take. The chapter concludes the thesis with a summary of the research and some final reflections.

Chapter 2

Service-Oriented Systems

Service-oriented architecture (SOA) is a software development framework that is seeing increased adoption within the IT industry (Erl, 2007). SOA supports the dynamic composition and reconfiguration of software systems from networked software services, which provide functionality to the service consumer on an *as needed* basis. The SOA development model provides significant benefits over traditional product-oriented software deployment. These benefits include the dynamic integration and rapid deployment of platform-independent systems, and reduced capital investment (Erl, 2005; Sommerville, 2006). This chapter provides an introduction to service-oriented systems and the SOA development model.

2.1 Software as a Service (SaaS)

Software as a service (Brereton et al., 1999; Bennett et al., 2000) is being promoted by software industry leaders as the basis for the next computing paradigm. Service-oriented architecture (SOA) is a conceptual structure for realising this vision, based on the design principle of service-orientation. Service-orientation is a design approach which specifies the implementation of processing or solution logic in the form of services.

Services are loosely-coupled and reusable software components, which encapsulate discrete functionality (Sommerville, 2006). Services are self-describing and should facilitate the rapid, low-cost development and deployment of distributed service-oriented applications (Brogi et al., 2008). A service can be described as a collection of capabilities, grouped together by the functional context provided by the service (Erl, 2007). The service contains the logic required to carry out

these capabilities, and provides a service contract that describes which of these capabilities is available for invocation. An example service is shown in Figure 2.1.

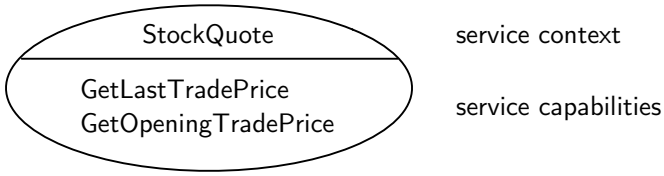


Figure 2.1: A service can be considered as a collection of capabilities.

Services are discoverable and dynamically-bound at runtime. Services are interoperable, yet can be implemented using different programming languages on different platforms. The major requirements of services are dependability, composability and reuse (Milanovic et al., 2003). Services are provided by service providers or vendors, and are requested and used by service consumers. It is possible to have many providers of the same service, with consumers free to choose between them. Services are typically accessed across a computer network.

Services are often *stateless*, in that they do not maintain state between requests made by different consumers of the service. This statelessness promotes the scalability of services. If state information is required, it should be passed to the service by the consumer with the service request that requires it. Services can be *stateful* when needed, for complex multi-stage operations. Stateful services have the benefit of reducing message overhead. With stateful services, consumers may pass an identification with any service requests made, in order to provide identifying information to the service. The service interaction of both stateless and stateful services can be described as connectionless, with short-lived interactions between the consumer and service. The typical service interaction process is shown in Figure 2.2.

Communication with a service is done through the service interface, making the service a black-box component. This enables service-oriented solutions to run across heterogeneous hardware platforms, and be implemented using different technologies and programming languages. It also facilitates the integration of legacy, native and proprietary systems into the architecture, through the use of a wrapper (Benatallah et al., 2002).

The software as a service (SaaS) model overcomes several limitations of traditional software development and use (Turner et al., 2003). Service-oriented applications can be composed at runtime, with services added, removed or replaced as

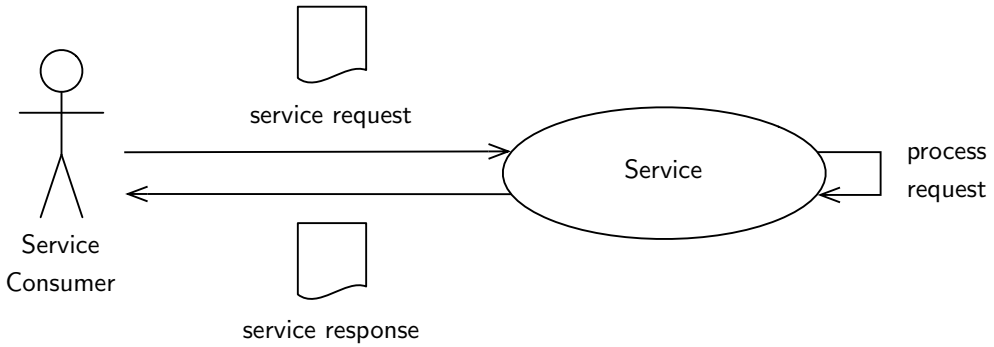


Figure 2.2: Typical service interaction.

needed. This is in contrast to the traditional approach of software as a *product*, where these types of activities are static and essentially frozen before the software product is delivered. Many of the principles of SOA can be automated, enabling flexible service-oriented applications which can rapidly respond to emergent properties, and changes in requirements and the environment.

2.2 Service-Oriented Architecture (SOA)

The principles of service-oriented architecture (SOA) are loose-coupling, the service contract, autonomy, abstraction, reusability, composability, statelessness, and discoverability (Erl, 2005). SOA provides a distinct approach to the software engineering principle of *separation of concerns*. This theory states that a large problem can be more effectively solved through decomposition into smaller problems or concerns.

SOA is a method of building distributed systems from loosely-coupled services. SOA is an implementation-independent software model, based on the principle of service-orientation, and can be realised using any suitable technology platform. There is currently an increasing interest in using the SOA architectural approach for the development of complex software systems, with the use of the Web Services Architecture (W3C Working Group, 2004) as an enabling technology (Saunders et al., 2006). However, SOA is fundamentally a technology-independent concept. Other examples of SOA implementations include Jini (Sun Microsystems, Inc., 2009b) and the OSGi Service Platform (OSGi Alliance, 2009).

The SOA interaction pattern, also referred to as the *publish-/find-bind-execute* model, is shown in Figure 2.3. The consumer typically initiates service requests.

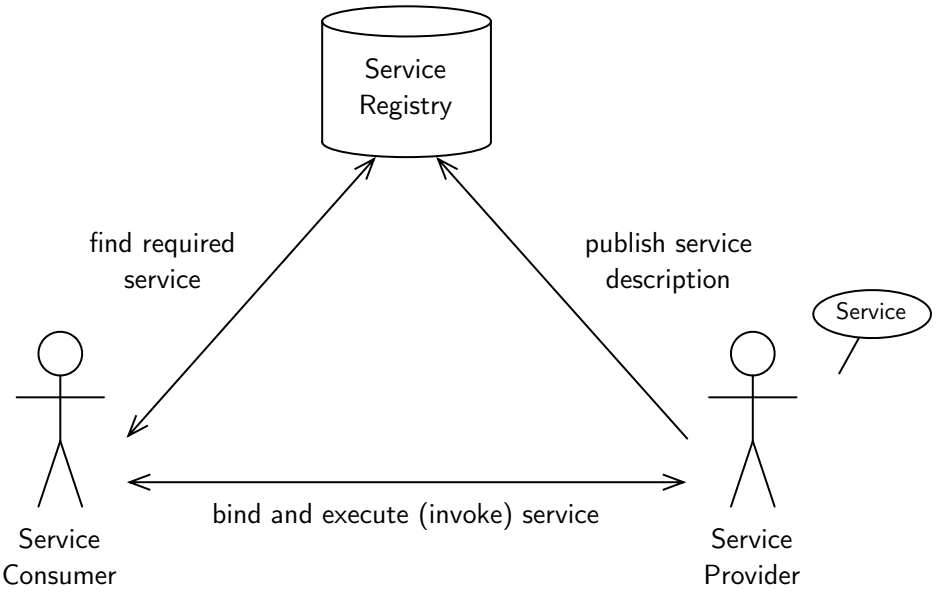


Figure 2.3: The SOA interaction pattern.

2.3 Service Description

Service descriptions are used by service consumers to describe the services they require, and by service providers to advertise the services they offer. In an ideal SOA, these descriptions would not only provide descriptions of the functional attributes of a service, but also non-functional attributes and rich semantic information. Services are not merely functions, but functions performed on behalf of a service consumer for a given cost. This cost is not just monetary, but involves a whole collection of limitations (O'Sullivan et al., 2002). It is therefore important that a service description also expresses the non-functional or qualitative constraints which accompany the functionality of the service. By including such information, service providers are able to differentiate themselves from other providers of functionally-alike services. Service consumers are then able to discover and select providers of services which best satisfy the non-functional consumer requirements.

Service descriptions primarily facilitate the advertisement, discovery and selection of services, but can also benefit other processes such as service negotiation, composition and monitoring. Specialist languages for describing services are in development, which have the aim of automating and improving service interoperability (Martin et al., 2004).

2.4 Service Discovery and Selection

The advertisement and discovery of services is a key principle of SOA, and an integral part of the SOA interaction pattern (shown earlier in Figure 2.3). In this abstract model, service providers publish descriptions to a registry that describe the services they provide. These service descriptions are then advertised by the registry for service consumers to discover, as shown in Figure 2.4. Once a suitable service is discovered, the registry provides the consumer with the location of the provider of the service. The consumer uses the location endpoint to bind to and invoke (use) the service.

Service discovery can be performed purely on the functional aspects of a service, which limits consumer choice to those services which provide a compatible interface (and also the service cost, if applicable). Service discovery and selection can also integrate non-functional and semantic criteria (Maximilien & Singh, 2004a; Oldham et al., 2006), as well as provider reputation (Xu et al., 2007). In the event that several functionally-compatible services exist, any additional criteria aids the service consumer in selecting the service which is most acceptable to the consumer's non-functional requirements.

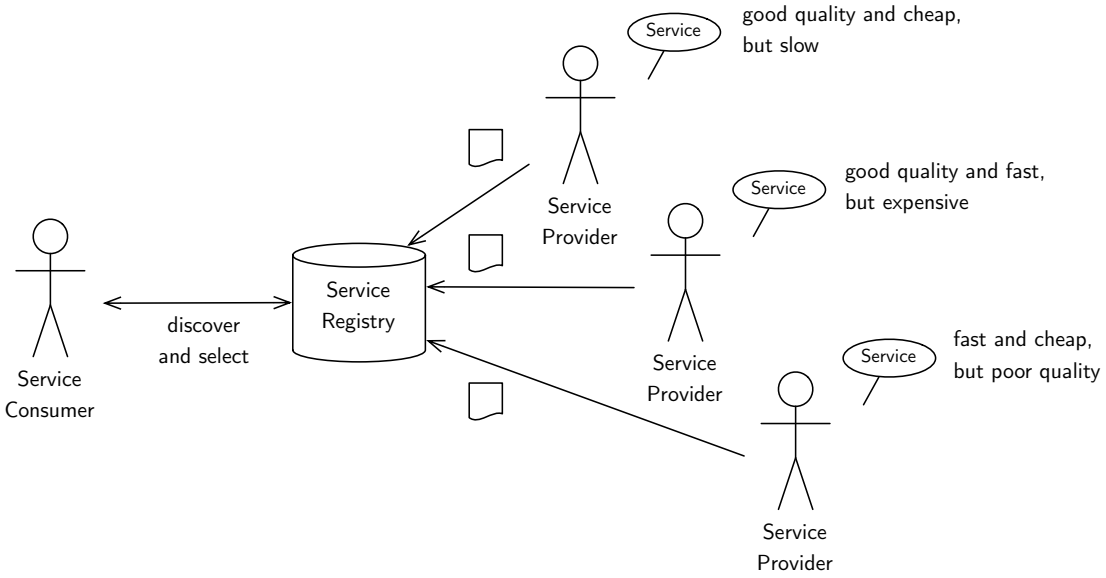


Figure 2.4: Service discovery and selection.

2.5 Service-Level Agreement (SLA)

The service-level agreement (SLA) is used to describe a formally-negotiated business contract between two parties, such as the consumer and provider of a service. An SLA concerns the terms and conditions of service provision and use, i.e. what a consumer can expect from a provider, and restrictions on what a consumer can demand from a provider. SLAs are composed from smaller service-level objectives (SLOs), which describe specific non-functional commitments over the use and provision of a service. An SLA usually associates a cost with a certain level of service, and may also specify penalties for non-compliance. As SLAs are merely contracts and do not enforce anything by themselves, they must be monitored or audited at runtime for violations by either party.

Service providers should be able to enter into agreements with an awareness of their resources, and be able to allocate and manage their resources, in order to meet runtime QoS guarantees and avoid SLA violations (Ludwig et al., 2004). Service usage conditions may also be placed on service consumers.

The service consumer may directly negotiate an SLA with a service provider, or may delegate this activity to a third-party service broker (Menascé & Dubey, 2007). Similarly, the service consumer can directly measure and audit service performance in order to detect any SLA violations, or may utilise a third-party SLA monitoring solution (Skene et al., 2007).

2.6 Service Composition

The ability to compose services together is a key principle of SOA. Service composition is the process of creating new services composed of smaller single services or other composed services. Complex business processes can be composed from multiple services, with each service providing some discrete piece of functionality. SOA promotes reusability by composing new services out of existing services, and composing new compositions from existing compositions. A simple hypothetical example is shown in Figure 2.5, where a road navigation service has been composed from a location service, a map service and a traffic report service.

Two other service-oriented concepts related to service composition are service orchestration and choreography, which are now discussed separately.

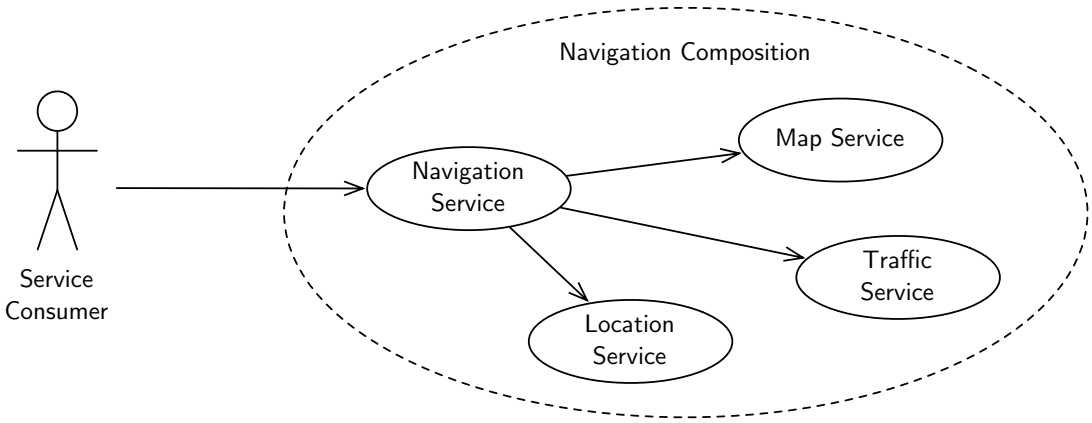


Figure 2.5: Example service composition.

2.6.1 Orchestration

An orchestration is an executable process, comprised of two or more services, which is centrally controlled. In the composition example given in Figure 2.5, the process controller is the navigation service. The navigation service is responsible for querying the location service for the coordinates of the consumer's current location, querying the map service for a map for the coordinates, and querying the traffic service for traffic data for the coordinates. The map and traffic data is combined by the navigation service, to provide a navigation aid to the consumer.

There are different models for service orchestration. The *hub and spoke* model, shown in Figure 2.6, is a common implementation of orchestration, which enables multiple services to interface with a central orchestration engine (Erl, 2005). Orchestrations are typically owned by a single organisation, and control almost every part of a complex process.

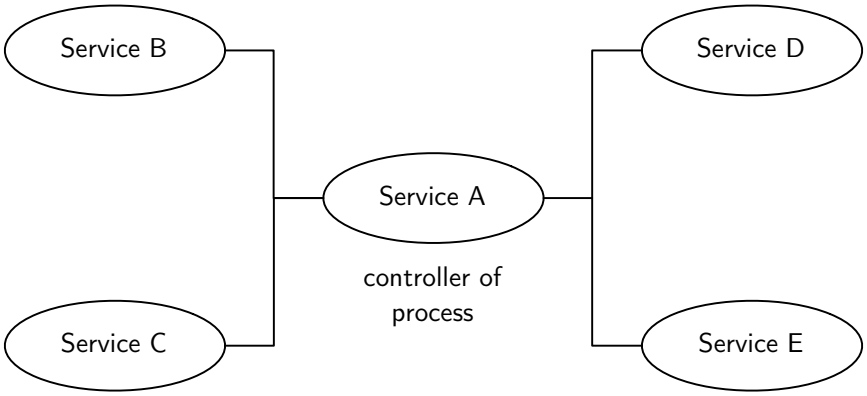


Figure 2.6: Service orchestration overview.

2.6.2 Choreography

Choreography enables collaboration between participants, when multiple services from different organisations must be composed together to achieve a common goal (Erl, 2005). Rather than compose existing services to form a new service which has central control over the whole process, choreography defines collaboration rules and policies which enable different services to collaborate with one another to form a process (Josuttis, 2007). Each service involved in the collaboration only contributes to a part of the process. An overview of choreography and its role in enabling cross-organisational collaboration is provided in Figure 2.7, adapted from an example in (Erl, 2005).

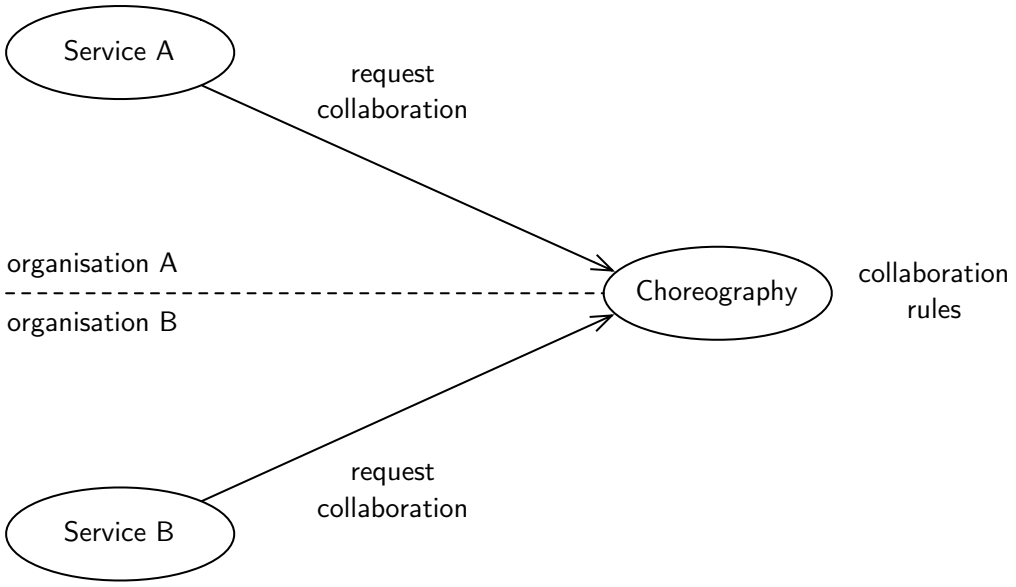


Figure 2.7: Service choreography overview.

2.7 System Domains

There are several system domains where service-oriented approaches to software development and use are being deployed. Applications in these domains are affected by both generic and domain-specific quality issues, and are often required to have automated responses to changes in requirements, the service execution environment, resource availability, and to general system faults.

2.7.1 Business and Enterprise

The service-oriented systems which underpin modern business processes must be able to react to constantly changing environments and business requirements. The adoption of service-oriented solution logic can introduce new issues to ensuring quality in business systems. Service qualities such as *availability*, *security*, *response time*, and *throughput* are particularly of concern. Guarantees for these qualities may be less important for services which are consumed once and then discarded, but critical for services which are important parts of longer running service compositions and bound by SLAs (Menascé, 2002). Ensuring quality is particularly complex when applications are composed of services provided by different organisations, which may be based in different countries across the world.

2.7.2 Grid and Utility Computing

The term *grid computing* is given to an approach for sharing computing resources across the Internet, with the aim of turning a global network of computers into a single powerful computational resource. The process of offering the computational resources of a grid as a metered service is known as *utility computing* (also referred to as *on-demand computing* and *cloud computing*). Grids are already in use for resource-intensive scientific applications, but could also become a future model for enterprise applications. *Resource-oriented* qualities are particularly of concern in grid computing, as well as qualities such as *availability*, *security*, *response time*, and *throughput*. Ensuring quality in grid systems is particularly challenging because of the complexity of grid systems consisting of thousands of components across multiple domains, where quality must be ensured at local and global levels (Menascé & Casalicchio, 2004).

2.7.3 Embedded Systems

Embedded systems are devices which feature programmable computers, but are themselves not intended to be general purpose computers (Wolf, 2001). The software for these systems is becoming increasingly complex, with a need to dynamically add and remove advanced functionality during the lifetime of a device (Tournier et al., 2005). Embedded systems have non-functional qualities which distinguish them from general purpose systems. These qualities are primarily *resource-oriented* and *performance-related*, and affect the overall quality of

a system. Examples include types of resource consumption, concerning issues such as *processor*, *memory*, *storage*, *power*, and *network bandwidth* requirements. Other qualities include *performance*, *reliability*, *availability*, and *security*. These qualities are often in competition with one another, e.g. an increase in system *security* levels can have an impact on the *performance* of a system. Similarly, increased system *performance* requirements can negatively impact the *power consumption* of a system. The competition between interdependent and competing qualities typically leads to system trade-offs being made.

2.7.4 Ubiquitous and Mobile Computing

Ubiquitous computing (Weiser, 1991), also referred to as *pervasive computing*, describes a shift away from the desktop interaction model of contemporary computing, to a network of small and inexpensive devices which are embedded in everyday objects distributed throughout the environment. Ubiquitous applications can benefit by dynamically discovering services as and when they are needed, as different sets of services may be exposed as contexts change (Baresi et al., 2004b). This is especially true with mobile applications which move between environments, and pervasive applications which are situated in changeable environments. These changes in context create issues with ensuring system quality for ubiquitous computing applications. For example, there are *trust* issues when services have to be accepted from providers which the consumer has no prior experience of. Mobile applications also require certain levels of qualities from the network, such as *latency*, *jitter* and *bandwidth*, and have to consider *resource-oriented* qualities, such as *power consumption*.

2.8 Summary

This chapter has provided the background to service-oriented systems and the key components of a service-oriented software architecture. The chapter discussed the nature of software composed from services, and the benefits of the service delivery model. The chapter concluded by identifying several system domains with particular quality assurance issues, which could benefit from a service-oriented approach to ensuring software quality. The following chapter provides a review of the state of the art in quality assurance solutions for service-oriented systems.

Chapter 3

Service Quality Assurance

This chapter begins with a discussion of software quality, and the background to the problem of ensuring quality in software. The chapter then continues with a discussion of software quality in the context of service characteristics, and presents the problem of ensuring quality in service-oriented systems.

The chapter then provides a detailed survey of current quality assurance initiatives for service-oriented systems. The survey begins with an examination of approaches that improve the characterisation of services with descriptive non-functional and semantic information. Improved service descriptions facilitate a variety of service-oriented processes, such as QoS advertisement and the creation of SLAs. There follows an evaluation of approaches for improving service discovery, which incorporate additional criteria, such as service quality attributes, to discover services which are closer to meeting a consumer's non-functional requirements. The chapter then reviews reputation system initiatives, which provide historical data on service provider performance. This reputation data provides the consumer with additional criteria for the service selection process. The discussion then assesses current service negotiation initiatives, which enable consumers to obtain agreements for services that are closer to meeting their requirements. The final service-oriented quality assurance domain discussed is service monitoring, which provides initiatives for detecting and recovering from SLA violations, service failures and errors.

The chapter concludes with a discussion of how initiatives from these different service-oriented quality assurance domains can be combined into an integrated solution for ensuring quality in service-oriented systems.

3.1 Software Quality

Quality can be defined as *the degree to which a set of inherent characteristics fulfils requirements* (ISO, 2000). Software quality characteristics may be inter-dependently related with one another, and frequently trade-offs must be made between competing and conflicting qualities (Chung et al., 1999).

Software quality is typically identified in software requirements, which specify the quality characteristics required from software, and guide the measurement methods and acceptance criteria for assessing these characteristics (Abran & Moore, 2004). Software requirements consist of functional and non-functional requirements. Functional software requirements specify what a system *does*, while non-functional quality requirements specify *how well* the system satisfies these functions (Gilb, 1988).

It is possible for software to function correctly, while not satisfying certain non-functional requirements. For example, a software function may perform a calculation correctly, but take an unacceptable amount of time to return the result of the calculation. Such non-functional qualities are frequently critical to the success of software, and an approach is required to ensure software qualities do not fall below acceptable levels.

3.2 Ensuring Software Quality

There are two general approaches for ensuring software quality. Firstly, quality issues can be addressed *statically* during the design and implementation of a system. Secondly, quality can be addressed *dynamically* through runtime negotiation of quality requirements. The static approach offers well-defined behaviour at the sacrifice of flexibility, while the dynamic approach offers increased flexibility at the sacrifice of well-defined behaviour (Tournier et al., 2005).

Software quality management approaches are summarised in (Heineman et al., 2004) as static analysis and predication of quality requirements, runtime enforcement of quality policies, and standards-based quality middleware extensions and frameworks. Static quality management approaches rely on predicting the properties of a system based on the properties of its constituent components (Lüders et al., 2005). With static approaches, emergent properties cannot be reliably predicted without extensive testing. Dynamic approaches can use prediction as a

starting point, but are then able to measure the *actual* system quality at runtime, and respond to quality issues and emergent properties. With increased agility in systems, quality assurance testing can be reduced through the introduction of runtime monitoring. Static prediction of resource usage can lead to systems which don't have the resources to function correctly at a certain level of system quality, or end up wasting resources because a large amount of redundancy has been built into them. Dynamic approaches allow for the allocation of resources as required.

Software specifications are not static, complete or homogeneous, but are subject to change because of quality characteristics which cannot be identified in advance (Shaw, 1996). Because of this, a hybrid of static and dynamic approaches can be desirable for ensuring software quality. For example, a static approach may be used for the safety-critical parts of a system, where well-defined behaviour is required. The safety-critical parts of the system can be implemented using product-oriented development approaches, such as component-based or model-driven development. The static approach can then be combined with a dynamic approach, used for the changeable or volatile parts of the system that require flexibility. These changeable or volatile parts can be implemented using a more loosely-coupled development approach, such as the service-oriented model.

3.3 Service Quality Characteristics

Qualities can be considered to be constraints over the functionality of a service (O'Sullivan et al., 2002). A characteristic of distributed systems is the volatility of service quality (Toma et al., 2006). It is therefore important that mechanisms are in place for maintaining the overall system quality (Menascé et al., 2007).

Traditionally, the term *quality of service* (QoS) has been associated with telephony and computer networking. QoS may be required for certain applications, such as *voice over IP* (VoIP), which have requirements on the data flowing across the network. For example, network applications may specify requirements that describe acceptable levels of *latency*, *jitter* and *dropped data packets*. However, quality cannot be ensured in service-oriented systems without considering application-level QoS. Currently, there is no industry-wide accepted technology for accomplishing this. Application-level QoS can be considered the *Achilles' heel* of internet services, with the complexity of the multitiered architecture and dynamic nature of service composition (Woodside & Menascé, 2006).

Service-oriented systems are distributed and composed from numerous services which can be discovered and replaced at runtime. It is possible for several different service providers to offer services with common functionality, but with different non-functional quality characteristics. These characteristics can be used by the service consumer to distinguish the acceptability of different service providers.

3.4 Approaches for Service Quality Assurance

Initiatives are underway to enhance services with non-functional QoS and semantic information. These initiatives enable providers to advertise the non-functional and semantic characteristics of the services they offer. Improved service descriptions aid the service consumer with the processes of service discovery, selection and composition. This in turn helps to increase the quality of service-oriented software, as software can be composed which better meets consumer requirements. The notable service description research initiatives are discussed in Section 3.5. Service discovery initiatives which incorporate enhanced service descriptions and selection criteria are evaluated in Section 3.6.

Reputation systems are designed to address issues of trust between parties who have not dealt with one another before. Common examples of reputation systems are the feedback mechanisms used by online auction sites and marketplaces. For the service consumer, reputation systems help to distinguish between low and high quality service providers (Jøsang et al., 2007). The inclusion of provider reputation as part of the service selection criteria, benefits the quality assurance process for the service consumer. Notable service reputation system research initiatives are reviewed in Section 3.7.

Service negotiation is an additional process that can bring software composed from services closer to meeting consumer requirements. Through negotiation with service providers, SLAs can be obtained for services which better meet consumer requirements. Service providers can also benefit from negotiation, by utilising their spare resources to provide a better QoS to those consumers who will pay for it. The notable service negotiation research initiatives are discussed in Section 3.8.

Service monitoring is a further process used to detect service failures and SLA violations at runtime. SLA violations detected through monitoring can prompt renegotiation with the provider of the problematic service, or replacement of the service through the selection of an alternate service provider. By being able to de-

tect and respond to problematic services, a method of maintaining system quality can be achieved. Notable service monitoring research is discussed in Section 3.9.

3.5 Service Description

Current industry standards for service description lack support for accurately describing the non-functional and behavioural aspects of services. The most prominent standard is the Web Services Description Language (WSDL) (Christensen et al., 2001), which provides the location of a service and a functional description of the service's input and output messages. With the emergence of a service marketplace where multiple providers supply functionally-equivalent services which implement a common service type, non-functional QoS properties will be the criteria which distinguish one provider from another (Tian et al., 2004). Service providers therefore require a standard method for describing the non-functional characteristics of the services they offer.

To reduce ambiguity, it is important that a standard method of describing non-functional attributes is shared between the participants in a service-oriented system (Dobson et al., 2005). A standard description method facilitates processes such as service advertisement, discovery, selection, composition, substitution, negotiation, and runtime service monitoring (O'Sullivan et al., 2002). A common description method, often called an *ontology*, helps to reduce ambiguity and contradictions in SLAs between service consumers and providers (Müller et al., 2008).

The significant service description research initiatives which feature support for QoS characterisation are summarised in Table 3.1. The attributes chosen for the evaluation criteria are common characteristics found between the discussed initiatives. The description initiatives either fully-support the identified evaluation criteria, provide partial consideration of the evaluation criteria, or do not support the evaluation criteria in any way.

The service description research initiatives listed in Table 3.1, have been selected based on their support for the following identified evaluation criteria: support for the expression of service quality attributes; the inclusion of a common service quality ontology to be shared by service consumers and providers; the provision of service quality criteria to support enhanced service discovery and selection; and the support to express SLAs over consumed and provided services.

Service Description Approach	Quality Attributes	Quality Ontology	Discovery Support	Selection Support	SLA Support
Ludwig et al. (2003)	✓	×	×	×	✓
Lamanna et al. (2003)	✓	×	×	×	✓
Martin et al. (2004)	✓	✓	✓	✓	×
Dobson et al. (2005)	✓	✓	✓	✓	✓
Toma et al. (2006)	✓	✓	✓	✓	✓
Andrieux et al. (2006)	✓	×	✓	✓	✓
Küster and König-Ries (2007)	✓	×	✓	×	✓

Key: (✓) full support, (∼) partial consideration, (×) no support

Table 3.1: Service description research initatives focus on improving the characterisation of non-functional service quality attributes.

3.5.1 Agreement Approaches

Some service description initiatives are primarily designed for formalising non-functional QoS attributes in agreements between service consumers and providers. One such example is the Web Service Level Agreement (WSLA) language specification, developed by Ludwig et al. The WSLA specification provides an approach for defining and monitoring flexible and individualised SLAs to service consumers in inter-domain environments.

An example where the WSLA specification has been used is given in (Dan et al., 2004). Dan et al. integrate the WSLA specification with a framework comprised of resource management, workload management and monitoring systems. The framework provides an SLA-driven approach to support the automated management of utility computing services, and is used to offer differentiated levels of service to service consumers. To demonstrate the approach, Dan et al. provide an example scenario based on a financial institution that offers a suite of web services for stock portfolio management. Rather than providing its own infrastructure for hosting these services, the financial institution seeks out a service provider with the resources required to host the services. The service provider is also required to supply management services, such as data backup, backup restoration and data security. An SLA between the financial institution and the service provider describes the characteristics of the provided infrastructure, with guarantees for quality attributes such as *network throughput* and *processor requirements*.

The WSLA language specification facilitates customisable and accurate QoS descriptions for service monitoring and control. However, as the customised QoS

metrics are defined within the SLA itself, WSLA cannot be used to describe services for the purposes of service advertisement, discovery and selection.

The SLAng language specification developed by Lamanna et al. provides a language for defining SLAs for end-to-end QoS agreements between network, storage and middleware services. The QoS agreements describe targets for quality attributes such as *performance*, *availability* and *reliability*. Like the WSLA specification, SLAng provides a formal and precise language for describing SLAs, together with a set of built-in QoS metrics. Lamanna et al. evaluate the SLAng approach using the Common Picture eXchange environment (CPXe), an architecture for integrating digital devices, internet storage and print services. SLAs are used within the architecture, to regulate the collaborations involved with activities such as using online print services from home, uploading photos from a retail kiosk, and ordering prints from a retail photo finisher.

The primary shortcoming of the SLAng initiative is that QoS metrics are specified within the language itself. This limits the flexibility of the approach, as it is difficult to extend its functionality with additional QoS metrics.

The Web Services Agreement Specification (WS-Agreement), developed by the Open Grid Forum (OGF) and led by Andrieux et al., is a web service protocol used in industry for establishing an agreement between a service provider and consumer. Agreements can concern qualities such as service *response time* and service *availability*, or may provide service resource assurances for *memory*, *processor* and *storage* attributes. The WS-Agreement specification can be used in a wide variety of domains, with agreement terms developed for each domain as required. For example, the WS-Agreement specification is used for establishing agreements concerning multimedia content and QoS negotiation in (Jouve et al., 2006).

The WS-Agreement approach allows for any language to be used for the expression of QoS attributes and constraints. This design delivers flexibility and enables WS-Agreement to be integrated with service discovery and selection processes. However, the flexibility of the design increases the chance of language incompatibilities between participants in the agreement.

3.5.2 Semantic Approaches

Other service description initiatives are included within the domain of *semantic web services* (McIlraith et al., 2001). These initiatives are designed to provide machine-readable descriptions of service semantics, and aid processes such as the

discovery, selection and composition of services. Semantic approaches assert that the incorporation of service semantics into a service description model is the key to self-describing, automated and dynamic service-oriented applications (Kritikos & Plexousakis, 2007). Semantic approaches also assert that ontologies should be used to formalise every term of the service description model.

The Ontology Web Language for Services (OWL-S) is a semantic service description initiative developed by Martin et al., designed to aid the automation of service discovery, selection, composition, substitution, and invocation, through the inclusion of rich semantic information. The approach makes use of the Ontology Web Language (OWL) (McGuinness & Harmelen, 2004), which is designed to facilitate machine-interpretability of information content, through the inclusion of additional vocabulary and formal semantics. Martin et al. maintain that WSDL's lack of support for semantic descriptions makes it impossible to develop software which can dynamically locate and use a service without human assistance. OWL-S provides an approach for describing non-functional service properties in a machine-readable manner, to support the automation of service discovery, selection and composition processes. OWL-S provides a service description framework which can be exploited by matchmaking algorithms for service selection. As an example application of OWL-S, the description initiative has been integrated with the UDDI service registry, to enable the semantic matching of service capabilities (Paolucci et al., 2002). The example application provides a simple case study that focuses on the *price* quality attribute of a car sales service.

OWL-S is able to support basic service contracts with the notion that the service interface itself is the service contract. However, OWL-S lacks support for the creation of SLAs which contain specific quality objectives described in terms of values, value ranges and metrics. OWL-S itself provides the general framework for the description of non-functional properties, rather than providing an ontology of its own. The support OWL-S provides for QoS specification is therefore limited, and insufficient for developing QoS metrics to support service monitoring.

The Quality of Service Ontology (QoSOnt) developed by Dobson et al. utilises the OWL approach to extend a service's specification with the expression of non-functional quality constraints. To demonstrate and evaluate the use of the QoSOnt approach, Dobson et al. developed the Service QoS Requirements Matcher (SQRM) tool. The SQRM tool supports the discovery, differentiation and selection of services, based on the QoS requirements of the user. The user's QoS requirements are

specified using the QoSOnt ontology. Dobson et al. provide examples of supported quality attributes including service *availability* and *mean time to complete*.

The QoSOnt initiative addresses QoS specification and includes some QoS metrics, which makes the approach suitable for supporting service monitoring, in addition to service discovery and selection activities. It also makes the QoSOnt approach suitable for supporting the description and creation of individual SLAs which specify QoS guarantees. However, QoSOnt lacks support for the expression of QoS relationships, which is a limitation given the typical interdependencies and competition between service qualities.

Toma et al. propose an approach for modelling QoS characteristics using the Web Service Modeling Ontology (WSMO) (Lausen et al., 2005). The approach is designed to describe quality attributes such as service *cost*, *response time* and *availability*. The approach focuses on the goals of supporting the automation and mediation of service provision and use. Toma et al. improve upon the basic QoS characteristics model provided by WSMO, through the inclusion of a QoS model for service selection. The improved QoS model is combined with a service selection algorithm in (Wang et al., 2006), which demonstrates the model's suitability for supporting service selection.

Toma et al. provide a QoS model that supports the description of SLAs, and provides limited QoS metrics to support service monitoring. However, as with the approach proposed by Dobson et al., the model lacks the ability to express QoS relationships between interdependent and competing service qualities.

The approach developed by Küster and König-Ries enhances service descriptions with dynamically-changing information, and provides the means to obtain this information during the service discovery process. The authors assert that the ability to automate the contracting process is the key to automatic service usage, as service descriptions are not static, but are dynamic and change over time.

Küster and König-Ries utilise the DIANE Service Description (DSD) (Klein et al., 2005) language, which supports a matchmaking algorithm for service selection. Küster and König-Ries integrate an automated contracting phase with the matchmaking algorithm, to support dynamically-changing service descriptions. To illustrate the matchmaking approach, the authors provide a motivating scenario. The scenario concerns a user that is seeking to purchase a notebook computer with particular quality attributes from three potential providers. These quality attributes include *screen size*, *processor*, *memory*, and *cost*.

The approach for service contracting developed by Küster and König-Ries is very general, and designed to be adapted to different domains. As such, it does not provide QoS characteristics that support the creation of SLAs, or QoS metrics to support the service monitoring process.

3.6 Service Discovery and Selection

The advertisement and discovery of services is a key concept of SOA (Erl, 2005). Service providers publish descriptions of the services they provide to a service registry. The registry then advertises these services for service consumers to discover. When initiatives for the expression of QoS characteristics are integrated with the service discovery and selection processes, service providers have the means to differentiate themselves from other providers of functionally-alike services, by advertising the non-functional QoS attributes of the services they provide. This additional QoS information enables service consumers to discover and select providers for services which best satisfy their non-functional requirements.

The significant service discovery and selection research initiatives are given in Table 3.2. In addition to supporting the expression of quality characteristics, the initiatives have been selected based on the following identified criteria: the provision of a service selection mechanism; the support for SLA creation; the provision of an automated solution. Some of these initiatives utilise *provider reputation* in the service selection process, and are discussed separately in Section 3.7.

Service Discovery Approach	Quality Attributes	Quality Ontology	Selection Mechanism	SLA Creation	Automated Solution
Paolucci et al. (2002)	×	×	✓	×	✓
ShaikhAli et al. (2003)	✓	×	✓	×	✓
Maximilien and Singh (2004a)	✓	✓	✓	×	✓
Wishart et al. (2005)*	×	×	∼	×	✓
Ali et al. (2006)*	×	×	✓	×	✓
Oldham et al. (2006)	✓	✓	✓	✓	✓
Xu et al. (2007)*	✓	✓	✓	×	✓

Key: (✓) full support, (∼) partial consideration, (×) no support

Table 3.2: Service discovery and selection research initiatives focus on improving the discovery and selection of suitable services using additional criteria such as QoS parameters and provider reputation.

* reputation-based discovery approach, discussed separately in Section 3.7

Paolucci et al. posit that web services should be able to locate other services which provide a solution their problems, and that services should be able to interoperate and form complex new services. The authors propose an approach for the location of services on the basis of the capabilities services provide. To support the location of services, the approach offers a service profile ontology which enables semantic descriptions of functional service capabilities. These semantic descriptions are processed by a matching engine, which compares service requests for similarity with service advertisements. To illustrate the approach, the authors provide a simple example based on an automobile sales service, which reports what automobiles can be purchased for a specified *price*.

Paolucci et al. integrate the OWL-S service ontology with the UDDI (Clement et al., 2004) service registry, to provide a method for the automated discovery, selection and interoperation of web services. However, the authors do not consider the non-functional quality attributes of services, such as *availability*, *reliability* and *response time*. As such, the approach proposed by Paolucci et al. provides no QoS metrics and is unable to support the creation of SLAs.

ShaikhAli et al. developed the UDDIe extension to the UDDI service registry. The extension enables the association of user-defined properties with a service's description. The extension then allows services to be discovered using these user-defined properties. Such properties can include quality attributes such as *cost*, *bandwidth* and *memory usage*. UDDIe also provides support for service leases, which are used to specify a limited time period during which services are registered with the service registry. The inclusion of a lease addresses the problem of missing or inconsistent service references that can plague a standard UDDI registry. As an example application, ShaikhAli et al. discuss the use of UDDIe in supporting QoS management in the context of grid computing, with services that provide scientific programs and mathematical routines.

The extension to UDDI proposed by ShaikhAli et al. enables the expression of service quality attributes for the purposes of service discovery. However, UDDIe does not itself provide a common quality ontology to be shared between users of the system. The lack of a common ontology means that service consumers and providers must agree upon terms for describing QoS characteristics, in order to share a common understanding of a service's description. This step makes it difficult to automate service interoperability between multiple providers. UDDIe provides an approach where the service interface acts as the contract between the

service consumer and provider, but does not support the creation of SLAs for specific QoS guarantees.

Maximilien and Singh provide the Web Services Agent Framework (WSAF) approach for the dynamic selection of web services. Autonomous agents perform the service selection processes on behalf of service consumers and providers, using the standard SOA interaction pattern (discussed in Section 2.2). The agent framework provides a QoS ontology, and a separate policy language that enables service consumers and providers to respectively express their QoS requirements and advertisements. The ontology facilitates not only the expression of quality attributes and QoS metrics, but supports the specification of relationships between interdependent service qualities.

Maximilien and Singh demonstrate their discovery approach with a case study that involves a consumer shopping for finance and insurance in order to purchase an automobile. To support the case study, the authors provide domain-specific ontologies for the finance and insurance domains, and a common middle ontology that provides quality attributes such as service *security* and *performance*.

The agents provided by the Maximilien and Singh approach do not support the selection of composite services, i.e. services which are composed from smaller interrelated services. The agents also do not support the negotiation of individual SLAs for specific QoS guarantees. The authors investigate the integration of provider reputation with the selection process in (Maximilien & Singh, 2004b), which is discussed separately in Section 3.7.

Oldham et al. discuss a tool for matching service consumers and providers, which operates on service agreements expressed using an extension of the WS-Agreement specification. This extension enables the description and inclusion of semantic QoS information within an agreement. A semantic QoS ontology enables the expression of SLAs which state the respective QoS requirements and capabilities of service consumers and providers. This SLA approach provides service consumers with assurances on specific service quality attributes, such as service *response time*, *availability* or *reliability*. The QoS information provides the service selection mechanism with additional criteria when selecting service providers on behalf of the consumer.

Oldham et al. demonstrate the matching tool with an example situated in the agriculture domain. The authors use WS-Agreement to specify farming contracts which contain guarantees and objectives, and conditions which must be satisfied

for the objectives to be met. In the given example, the service consumer is a merchant and farmers act in the role of service providers. The matching tool is used to narrow down the available farmers into a select group, that contains only those farmers that meet the merchant's requirements.

The semantic QoS ontology developed by Oldham et al. has the potential to support service negotiation, and to support the description of QoS metrics for runtime service monitoring. However, the implementation of the tool provided by the authors does not support these features.

3.7 Service Reputation Systems

In a service marketplace, it is common for transactions to occur between parties who haven't previously interacted. Reputation systems are collaborative mechanisms for addressing trust issues between such unfamiliar parties.

A comprehensive survey of trust and reputation systems for online service provision is given in (Jøsang et al., 2007). The survey discusses both centralised and distributed approaches to the dissemination of reputation information, and identifies the types of systems where either a centralised or distributed approach is better suited. The survey provides the reader with example applications for reputation systems, such as peer-to-peer file sharing networks, online auction feedback systems, and online shopping reviews. In peer-to-peer networks, nodes can share information with one another about their interactions with other nodes. This enables a node to limit its interaction with selfish nodes that contribute little bandwidth, or that host malicious software such as viruses. Online auction sites typically allow a buyer and seller to provide feedback on each other after a transaction, in the form of a numerical rating and text comments. This feedback is made available for prospective buyers and sellers to view, and provides them with an idea of how reputable a particular buyer or seller is before entering into a transaction. Online shopping sites often provide the ability for users to write reviews for the products they sell. These reviews are made available alongside a product's listing, and form a reputation for the product's quality. The consumers that submit product reviews also develop a reputation for the quality and reliability of the reviews they write, as other consumers are able to rate the helpfulness of the product reviews.

Jøsang et al. do not propose how such reputation systems could be integrated

or adapted for use with service-oriented systems. This section provides a discussion of significant research initiatives which integrate reputation systems with a view to improving some quality aspect of service-oriented systems. These reputation initiatives focus on improving the selection of services on behalf of the service consumer. As such, they support the general description of services and service quality attributes. The additional evaluation criteria identified are the support of integration with the service discovery and SLA creation processes. The notable service-oriented reputation system research efforts are summarised in Table 3.3.

Service Reputation Approach	Quality Attributes	Quality Ontology	Discovery Mechanism	Selection Criteria	SLA Creation
Maximilien and Singh (2004b)	✓	✓	×	✓	×
Wishart et al. (2005)	×	×	✓	✓	×
Ali et al. (2006)	✓	×	✓	✓	×
Jurca et al. (2007)	✓	✓	×	✓	✓
Xu et al. (2007)	✓	×	✓	✓	×

Key: (✓) full support, (∼) partial consideration, (×) no support

Table 3.3: Service reputation system research initiatives focus on improving consumer service selection via additional provider reputation criteria.

Maximilien and Singh provide a reputation-based approach to service selection, which combines service consumer preferences with the trustworthiness of service providers. In the approach, software agents act on behalf of service consumers and share QoS information with one another, based on their interactions with the services they are attached to. Initially, each service provider is assigned the same reputation. Over time, unreliable service providers develop a poor reputation which makes them less likely to be selected for use by the agents. The information shared by these agents provides the basis for service consumers to establish trust with service providers. As part of the work, the authors developed a substantial QoS ontology for the expression of service quality attributes, QoS metrics and relationships between interdependent qualities. The ontology provides a structured hierarchy of quality attributes from the *economic*, *performance*, *availability*, *reliability*, *stability*, *security*, *robustness*, and *integrity* domains. The framework also includes a semantic matchmaking algorithm that selects services based on consumer and provider policies.

Maximilien and Singh evaluate their approach with a series of service simulations. The service quality levels provided by service providers are artificially

increased or decreased, so as to simulate changes in service performance. The authors demonstrate the impact of these changes on the reputation of the service providers, and on the service selection decisions performed by consumer agents.

The approach developed by Maximilien and Singh improves upon standard functional service selection, through the inclusion of semantic and reputation matchmaking criteria. However, the approach lacks support for the creation of SLAs between service consumers and providers. It also provides no mechanism for service consumers and providers to negotiate agreements for services with specific QoS guarantees. The approach also does not include support for resolving problematic services, and does not specify how service quality data is collected.

Wishart et al. put forward a reputation-enhanced service discovery protocol, that collates customer testimonial ratings to create a global QoS score for each available service. The protocol enables service consumers to consider QoS issues when making service selection decisions, with fewer assumptions about the trustworthiness and reliability of service providers. Instead, the trustworthiness of a provider is based on the actual performance of the services it offers.

To demonstrate the operation of the discovery protocol, Wishart et al. provide a fictional scenario involving an online data storage facility. The facility advertises its services on a local service discovery network, which operates using the authors' discovery protocol. A client with online storage requirements uses the protocol resolver to retrieve a testimonial of the online data storage facility. The client decides to use the facility, but in practice the facility does not meet the client's expectations. The client submits feedback on the unsatisfactory experience to the protocol resolver, which worsens the testimonial rating of the storage facility.

The protocol proposed by Wishart et al. is centralised, which offers the advantage of offloading all management and computation of reputation data to the service repository. The protocol is also general and flexible enough to be integrated with existing discovery initiatives, rather than being limited to a specific discovery mechanism. However, the reputation data the system collates is very primitive; customer testimonial ratings are simply a value between 0.0 and 1.0, which reflects the overall performance of a service. The authors do not consider rating individual QoS attributes, such as *response time* and *availability*, which would allow consumers to make better informed decisions about potential service provider suitability. While the discovery protocol provides additional selection criteria, it leaves the actual selection decision to the consumer performing the search.

Ali et al. provide an automated reputation-enhanced service discovery and selection framework for semantic grid services. The framework features a dynamic composition algorithm which adapts to available services, and a reputation model that forms the basis for interaction decisions. The authors illustrate the approach with an example scenario from the automotive industry, where a consumer agent seeks out an automobile sales service with certain reputation metrics. The reputation metrics are expressed in terms of quality attributes, such as service *price*, *reliability* and *availability*.

The approach from Ali et al. enables service consumers to differentiate between multiple providers of the same service, through the inclusion of reputation information during the discovery and selection processes. However, the framework is limited to the discovery and selection of services based on provider reputation alone, and does not feature selection support based on the QoS attributes of services. The framework does not include support for service negotiation and SLA creation, and makes the assumption that service consumers and providers interact directly with one another to achieve these processes.

Jurca et al. discuss a QoS monitoring mechanism based on service ratings supplied by service consumers. The mechanism collates consumer ratings to develop the global reputation of service providers. Service consumers are then provided with a method for querying the reputation of specific providers. By supplying consumers with advance notice of a provider's reputation, the mechanism provides an incentive for service providers to supply services in accordance with the SLAs they have formed with consumers. The approach also features a consumer reward payment mechanism, to provide consumers with the incentive to report honest accounts of the services they use. The authors assert that existing SLA monitoring approaches are unsuitable for monitoring QoS. They argue that monitoring approaches based on intercepting service requests do not scale, that provider-side monitoring approaches are untrustworthy, and that third-party probing increases the load on service providers through the generation of additional service requests.

To illustrate their approach, Jurca et al. have implemented a prototype of the QoS monitoring mechanism on top of the Apache Axis (Apache Software Foundation, 2009) web service middleware. The prototype provides directory services for service advertisements, reputation mechanisms for collecting client feedback and imposing penalties on providers that do not respect the terms of SLAs, and a banking module for handling payments between clients and service providers. The

approach is illustrated using service *availability* and *correctness* quality attributes.

Jurca et al. propose that their incentive-based reputation system negates the need for an SLA monitoring system. However, their approach has in practice shifted the responsibility for SLA monitoring to the service consumer, making it unsuitable for consumers with limited resources. The approach also lacks a common QoS ontology for consumers to describe their experiences, which makes it difficult to automate the selection of service providers.

Xu et al. propose another reputation-enhanced discovery approach, which also recognises the need to include QoS attributes in the discovery and selection of services. Their proposal integrates service ratings with a QoS attribute selection process, as a means of validating the QoS promises made by service providers. The approach consists of a extended UDDI discovery model that incorporates QoS information, a reputation management system which collates and reports provider reputation, and a discovery agent responsible for co-ordinating the discovery process. Using a matching algorithm based on the algorithm proposed in (Maximilien & Singh, 2004b), services that match consumer requirements are ranked by both the provided QoS and reputation scores. The results of the matching process are returned to the consumer during the service discovery request.

In order to demonstrate the effectiveness of their algorithm, Xu et al. present two experiments. The experiments feature service consumers with the same functional service requirements, but with differing QoS and reputation requirements. The quality attributes used for the experiments are service *price*, *response time*, *availability*, and *throughput*. The first experiment demonstrates that a consumer is more likely to select a service that best meets its requirements, if the consumer specifies its QoS and reputation requirements in the discovery request. The second experiment demonstrates that services with a consistent QoS performance are more likely to be selected over services with an unstable QoS performance.

The initiative proposed by Xu et al. offers a significant improvement upon service discovery approaches that consider only functional and non-functional service attributes. However, the authors do not address the negotiation of QoS parameters or the creation of SLAs, which are processes supported by more advanced brokerage initiatives. Their approach also requires the service consumer to undertake QoS monitoring, which is impractical for consumers with limited resources.

3.8 Service Negotiation

In general, negotiation refers to communication processes that further coordination and cooperation (Kraus, 2001). In terms of the software as a service (SaaS) model, negotiation involves the interaction between a service consumer and one or more service providers, that are identified either through discovery, or who are already known to the service consumer (Turner et al., 2003).

Functional characteristics that describe what a service does can be assumed to be non-negotiable (Elfatatry & Layzell, 2005). However, services typically have a number of interrelated and competing non-functional quality attributes, which describe issues such as how *reliable* a service is. It is important for the consumer and provider of a service to mutually agree upon an SLA that specifies guarantees for the values of these quality attributes (Benatallah et al., 2002). This issue has so far not been well-addressed in existing research, particularly for the negotiation of service compositions, where QoS agreements must be reached between the service consumer and multiple service providers (Yan et al., 2007).

SOA does not specify service negotiation as a core principle, but research into extending the basic architecture with the negotiation of QoS and other terms of service usage is an active area of research (Kretzschmar, 2006). Service consumers and providers may directly negotiate terms of service with one another, or the SOA publish-/find-bind-execute interaction pattern may be extended with service brokers or agents that negotiate on behalf of service consumers and providers.

Current QoS negotiation research initiatives focus on the automated negotiation of QoS parameters for the formation of SLAs between service consumers and providers. The late-binding involved with software composed from services requires that non-functional software attributes can be automatically negotiated and resolved (Bennett et al., 2001). When negotiation items are numerical and their quantity fixed, a high degree of automation can be achieved (Jennings et al., 2000). To support complex QoS negotiation, service consumers and providers must share a common quality ontology (Elfatatry & Layzell, 2005). Current negotiation approaches assume such an ontology is already shared between the negotiation participants. Service consumers and providers also require a common negotiation model, which provides rules for the exchange of service proposals (Su et al., 2001).

The most notable QoS negotiation research initiatives are summarised in Table 3.4. The primary evaluation criteria identified are the support for the automated negotiation of QoS characteristics and SLA creation. Unlike initiatives

Service Negotiation Approach	QoS Negotiation	Quality Ontology	SLA Creation	SLA Enforcement	Automated Solution
Gimpel et al. (2003)	✓	×	✓	×	✓
Comuzzi and Pernici (2005)	✓	×	✓	∼	✓
Elfatatry and Layzell (2005)	∼	×	∼	×	∼
Menascé and Dubey (2007)	✓	×	✓	×	✓
Yan et al. (2007)	✓	×	✓	×	✓
Pouyllau and Haar (2007)	✓	×	✓	×	∼

Key: (✓) full support, (∼) partial consideration, (×) no support

Table 3.4: Service negotiation research initiatives focus on the automated negotiation of QoS parameters and the creation of SLAs.

from the service description, discovery and selection, and reputation research domains, the identified negotiation initiatives lack a quality ontology to support the negotiation process, and lack effective means of enforcing negotiated SLAs.

Gimpel et al. discuss a policy-driven automated negotiation decision-making framework, that provides a negotiation infrastructure for negotiating service contracts on the behalf of service consumers and providers. The authors developed a hybrid negotiation model, that integrates utility-based and rule-based decision making approaches. Service consumers and providers express their negotiation strategies in the form of negotiation policies. These negotiation policies are represented by a combination of utility functions and rules. The negotiation framework provides a negotiation protocol consisting of a set of negotiation message primitives, and a set of rules which define legal actions. Either party may begin a negotiation with a *request for negotiation* message. Follow-up messages are *accept*, *reject*, *offer*, *withdraw*, or *terminate*. The framework provides negotiation agents with decision-making components, which are responsible for the negotiation of service contracts. Service consumers and providers provide their agents with negotiation policies, which provide instructions for realising their negotiation goals. The negotiation agents exchange negotiation messages using the negotiation protocol defined by the framework.

To illustrate their approach, Gimpel et al. provide an example scenario involving an online stock quote service, which is able to negotiate certain service quality attributes for particular consumers. One such consumer is an online newspaper service, which must increase its capacity to supply stock quotes due to changes in the behaviour of its subscribers. The newspaper service looks to the stock quote

service to satisfy its increased capacity requirements, and negotiates a contract that includes guarantees for service quality attributes such as *throughput*, *response time*, *availability*, and *price*.

The framework developed by Gimpel et al. supports partially-automated service contract negotiation. However, the framework does not provide a service quality ontology for the negotiation participants. This requires the negotiation participants to agree upon the semantics of quality attributes before commencing negotiation. The framework also lacks support for the specification of relationships between interdependent qualities, e.g. how an increase in service *reliability* may affect the *cost* of a service. The framework provides resource management support for service providers, to prevent them from agreeing to contracts they cannot fulfil. However, the authors do not consider the monitoring or enforcement of negotiated service agreements on behalf of the service consumer.

Comuzzi and Pernici extend the publish-/find-bind-execute SOA interaction pattern with a centralised QoS negotiation broker. Service consumers and providers supply the negotiation broker with their preferences for QoS attributes and negotiation strategies. The broker is capable of fully-automated negotiation using the consumer and provider preferences, but also offers an approach where negotiation is only automated on behalf of the provider, leaving the consumer to manually interact with the broker. This second approach is provided for consumers unable to specify their preferences, or untrusting of the service provisioning platform.

Comuzzi and Pernici provide the motivation for their negotiation approach with an example stock quote service, which provides real-time stock quotes from stock exchanges across the world. The stock quote service has three key quality attributes, which are service *price*, *availability* and *data quality*. These quality attributes are themselves expressed as a combination of other quality attributes, such as *completeness*, *accuracy* and *timeliness*. The negotiation broker developed by the authors is used to negotiate these attributes, on behalf of the stock quote service provider and the service consumer during invocation of the service.

The approach Comuzzi and Pernici discuss alleviates consumer trust issues, by providing fully-automated negotiation with support for manual consumer intervention. The result of the negotiation is a service contract that is suitable for being monitored and managed. However, the approach does not actually tackle the issue of SLA monitoring and enforcement. The centralised approach mitigates the need for a negotiation messaging system, and greatly reduces the amount of

network traffic, and hence time, required to negotiate a contract. However, such a completely centralised approach is unlikely to scale with increasing numbers of service consumers and providers.

Elfatatry and Layzell provide a rule-based negotiation description language to facilitate the automation of service negotiation in a service-oriented context. The language consists of rules which drive the negotiation process, and artefacts which result from the negotiation process. The negotiation description language provides support for a service provider to create a service profile that describes the functional and non-functional characteristics for each service it offers. The service consumer also creates a service profile for each service required, which contains selection schemes for selecting combinations of non-functional service characteristics. The service profile also declares the negotiation protocols supported by its owner. The negotiation description language provides a strategy profile that contains a collection of tactics for the creation and evaluation of service proposals, and a set of rules for selecting between different tactics. The negotiation results are formalised using a contract template, which consists of a series of service and contract pre- and post-conditions. The language also features a meta-protocol profile, which supports the negotiation of the negotiation protocol itself.

Elfatatry and Layzell illustrate the negotiation description language, with an example scenario featuring a voice-enabled word-processor application composed from multiple software services. The word-processor application comprises a text editor service, a spell-check service and a voice-to-text service, each supplied by a different provider. In this example, the provider of the word-processor application is the consumer of these three different services. The authors demonstrate the role of the negotiation description language in the negotiation and contracting of the services which form the word-processor application.

The negotiation description language provided by Elfatatry and Layzell facilitates the automation of service negotiation between a service consumer and provider. However, the language is merely intended to support the process of negotiation, and does not provide this functionality itself. The authors also do not consider other areas such as service discovery and QoS monitoring.

Menascé and Dubey discuss a QoS broker architecture for service selection based on QoS attributes. The approach applies utility functions to assign values that indicate the usefulness of a service to the service consumer. This usefulness is the combined utility of multiple quality attributes, such as service *availability*,

response time and *throughput*. Service providers register with the QoS broker, and provide the broker with the resource demands and cost of the services they offer. The resource information is used by the QoS broker to ensure providers don't commit to SLAs they cannot fulfil due to resource depletion. Consumers register with the QoS broker, and provide the broker with utility functions for the QoS and cost of the services they require. The QoS broker uses this information to select services which have the highest utility to consumers.

To demonstrate the approach, Menascé and Dubey discuss a working prototype of the service broker, which is evaluated using an online travel agent scenario. The online travel agent makes use of airline reservation service providers, in order to provide a travel booking service to consumers. The authors demonstrate how the service broker maximises a utility function, to find the airline reservation service provider with the most utility to the travel agent for a given cost.

The centralised approach proposed by Menascé and Dubey performs the service selection process on behalf of service consumers, making it suitable for consumers with limited resources. However, the service selection decision is based on static QoS attributes, and does not enable service consumers and providers to negotiate individual SLAs for bespoke QoS guarantees. The approach also does not support the enforcement of the agreements created by the QoS broker.

Yan et al. assert that service-level QoS depends on the structure of the service composition, as well as the characteristics of the QoS attributes. The authors provide an agent-based approach for negotiating end-to-end quality constraints for service compositions. The approach makes use of a utility function-based decision-making model. The approach also provides an SLA management component, which maintains up-to-date service profiles for all services in a service composition. These profiles may change over the lifetime of the composition, due to the negotiation and renegotiation of QoS attributes.

Yan et al. demonstrate their approach with a prototype implementation and case study. The case study is situated in the tourism industry, and involves a tourist user operating a mobile device to request the route information from the user's current location to a particular tourist attraction. The route information is retrieved from a composite service, composed of a device location service, route calculation service and route description service. The user may specify certain QoS requirements of the route information service, such as *response time* and *cost*. The authors demonstrate how their negotiation agents support the user in negotiating

these QoS requirements with multiple providers of the route information service.

The approach proposed by Yan et al. provides a comprehensive framework for SLA negotiation and renegotiation. However, the approach does not support measures for the enforcement of negotiated SLAs. The authors briefly mention a component for monitoring and visualising service composition workflow, but do not propose how the component could measure the runtime QoS of the composition.

Pouyllau and Haar discuss the problem of securing QoS guarantees for a workflow process composed of multiple services. Each of the services in the process is responsible for providing some discrete functionality to the workflow, and typically several interdependencies between the services will exist. The authors propose a protocol for the negotiation of end-to-end QoS contracts between a chain of service providers across multiple domains, that takes into consideration QoS issues caused by service interdependencies.

Pouyllau and Haar provide a prototype negotiation platform, based on a web service implementation of their negotiation protocol. The authors use this platform to evaluate the performance of the protocol in computing multiple contract chains, and determining the optimal chain of service providers for the required QoS. When executed in a realistic context, the authors show the algorithm can successfully complete within three seconds.

Pouyllau and Haar tackle an important and difficult problem, but their proposal does not include support for monitoring the workflow process, enforcing SLA compliance, and recovering from service failures. The authors' approach is designed to address QoS issues between service providers, and as such does not consider the needs of the service consumer.

3.9 Service Monitoring

Service monitoring is an increasingly important research issue, as ever more numbers of companies conduct business over the Internet (Molina-Jimenez et al., 2004). Service monitors can be used to determine in practice, if services meet the terms and conditions agreed in the SLAs between service consumers and providers (Benjamim et al., 2004). An SLA contains specific guarantees for the QoS a consumer can expect a provider to supply. SLAs must be monitored and audited for service provider compliance, in order to provide real QoS guarantees to the consumer (Benjamim et al., 2004). The SLA may also contain conditions

of use imposed on the consumer by the service provider, and these may also be monitored to ensure compliance.

Emergent system qualities can result from the service composition process, and from changes in the runtime environment. These emergent qualities require a dynamic runtime quality assurance approach, which service monitoring can facilitate. However, current service monitoring initiatives are largely manual and static approaches. A system designer may annotate business processes with comments describing the monitoring to be performed, as described in (Baresi et al., 2004b). An aspect-oriented approach described in (Bianculli & Ghezzi, 2007), weaves monitoring aspects into a business process. These aspects intercept the business process at different points in its execution, and check it for conformance. A web service requirements monitoring approach described in (Mahbub & Spanoudakis, 2004), focuses on the formalisation of monitoring rules at design-time. These aforementioned monitoring approaches are primarily static in nature, which makes them unsuitable for assuring runtime and emergent system qualities. Consequently, these static initiatives are limited in their ability to handle problematic services, and do not support advanced recovery techniques such as service renegotiation.

Service consumers and providers may themselves take direct responsibility for QoS measurements and SLA auditing. However, this monitoring approach is susceptible to abuse from either party making false accusations concerning service provision or use. It is argued that the service consumer and provider should instead utilise the services of a mutually-trusted third-party monitoring system (Benjamim et al., 2004). One suggested approach is for the monitoring system to periodically *probe* the service provider, in order to assess its current performance. However, these probes generate additional service requests, which increase the load on the service provider. The provider may also be able to distinguish the monitoring system requests from those of the consumer, and provide a superior QoS to the monitor. An alternative is for the monitoring system to implement a *passive* monitoring mechanism. The passive approach intercepts and audits service requests between the consumer and provider, without generating additional requests itself.

Some service monitoring initiatives are designed to support service providers in avoiding SLA violations, rather than supporting the service consumer in detecting and responding to problematic QoS. Examples of these include the control loop approaches proposed in (Hoffman, 2005) and (Litoiu et al., 2008).

Other monitoring initiatives are consumer-focused, and are able to detect ser-

vice failures and SLA violations. However, few of the current consumer-focused approaches include support for recovering from such failures. Additional evaluation criteria identified between the monitoring initiatives include the support of QoS metrics, integration with SLAs, and the degree of automation provided by the monitoring approach. The notable service monitoring initiatives that provide the service consumer with QoS support are summarised in Table 3.5.

Service Monitoring Approach	QoS Metrics	SLA Integration	Failure Detection	Failure Recovery	Automated Solution
Molina-Jimenez et al. (2004)	✓	✓	✓	×	×
Ludwig et al. (2004)	✓	✓	✓	×	✓
Lazovik et al. (2004)	×	×	✓	✓	✓
Baresi and Guinea (2005)	×	×	✓	×	∼
Moser et al. (2008)	✓	×	✓	∼	✓
Herssens et al. (2008)	×	✓	✓	∼	✓

Key: (✓) full support, (∼) partial consideration, (×) no support

Table 3.5: Service monitoring research initiatives focus on detecting failures in service quality.

Molina-Jimenez et al. discuss an approach for monitoring an SLA on behalf of the service consumer. The approach evaluates whether the service performance provided by the service provider complies with the QoS guarantees described in the SLA. The authors assert that it is only practical to offer guaranteed QoS to consumers that share the same internet service provider (ISP) as the providers of the services they use. Service consumers who connect to the Internet with a different ISP can only be offered a best-effort QoS. The authors propose a third-party monitoring service that is mutually-acceptable to both the service consumer and provider. The third-party approach means that monitoring results can be trusted equally by both parties. The monitoring service periodically probes the service provider, measures the service performance, and compares the measured performance with the SLA. The monitoring service then notifies the service consumer of any SLA violations. The design of the monitoring approach is intended to support e-commerce applications, grid computing and web services.

The monitoring approach from Molina-Jimenez et al. notifies the service consumer of SLA violations, but does not provide any support for handling or resolving SLA violations and service failures. In addition, probing the provider increases the provider's load with additional service requests. The service provider may also

differentiate between the service consumer requests and the monitoring service probes, and provide each with a different QoS. As Molina-Jimenez et al. have not implemented or evaluated the monitoring approach they propose, it is difficult to properly ascertain its suitability for purpose.

Ludwig et al. propose the Cremona architecture for the creation and monitoring of service contracts expressed using the WS-Agreement specification. WS-Agreement itself provides a description format for service agreements, a basic protocol for establishing agreements, and an interface specification for monitoring agreements at runtime. Ludwig et al. argue that the traditional publish-/find-bind-execute service interaction model is not sufficient when services with customised quality guarantees are required. To solve this problem, the Cremona architecture extends the traditional service interaction model by providing the service consumer and provider with individual agreement management components, which interact with the service registry. This approach means that service consumers and providers negotiate directly, rather than via a third-party service. Once agreements are established, the architecture provides monitors that map the state of the service provider to the service guarantee status.

The role of the Cremona architecture in creating and managing agreements is illustrated with an example of workload-sharing across distributed data centres. In the example, agreements are used to describe guarantees for the average *response time* of a set of web-based transactions, that together form a financial service application. The agreements are used to shift transactional workload between the distributed data centres that host the application, and Ludwig et al. describe how the Cremona architecture supports the shifting process.

The Cremona architecture proposed by Ludwig et al. performs monitoring on behalf of the service provider. Consequently, it offers the service consumer little control over service quality. The approach also doesn't support relationships between service agreements, making it unsuitable for the negotiation and monitoring of service compositions. Lastly, the authors do not discuss how QoS measurements are collected and audited against the QoS guarantees specified in agreements.

Lazovik et al. discuss choreography languages which support the execution of business processes composed from services. The authors state that while choreography languages can guarantee the static properties of business processes, such as the consistency of service interfaces, message ordering and message invocations, choreography languages are unable to check the runtime properties of a business

process. To add runtime property support, the authors propose an assertion-based monitoring approach that associates assertions with business processes. The approach is based on a service request language, designed to create solutions for business problems through the retrieval and aggregation of services. The authors then extend the service request language with assertions for expressing definitions and classifications of business rules. The language is combined with a framework which automatically associates business rules with the processes involved in a user request. The framework then prepares and monitors the execution of the user request against the business process services. If an assertion is violated, the framework attempts to find an alternate execution path for the business process.

Lazovik et al. provide an example to illustrate their monitoring approach. The authors describe a travel planning business process, which integrates multiple services in order to book a trip on behalf of a user. The business process begins with the user requesting a trip to their desired destination, along with requirements for details such as *cost*, *hotel arrangements* and *modes of transport*. Satisfying the request requires the interaction of several autonomous service providers, including a travel agency, hotel booking company and airline. Assertions are applied to the travel planning process, to ensure consistency at each stage in the operation.

The monitoring approach proposed by Lazovik et al. is static in nature and not able to address runtime quality issues. The provided assertion language operates on the functional characteristics of the services involved in the process, and does not support non-functional QoS attributes. The framework is designed to find an alternate execution path for the business process, so it can complete successfully. If a service in the process fails, the framework attempts to secure the same service from an alternate provider with less strict assertions. If the framework cannot devise an alternate execution path, or a particular service becomes completely unavailable, the business process fails.

Baresi and Guinea provide a method of associating assertion-based monitoring rules with business processes described using the Web Services Business Process Execution Language (WS-BPEL) (Jordan & Evdemon, 2007). The approach is designed to monitor and reorganise service compositions in response to service faults. The types of service fault the approach addresses are given in (Baresi et al., 2004b). The first fault type is incorrect matching during the service selection process, which results in a service that is functionally, or semantically, incompatible with the service consumer's requirements. The second fault type is when a service

simply fails to respond to the consumer's service request. The third fault type is when the service request returns with an error. The final fault type the authors discuss, is when a service's behaviour does not match the contract imposed on it.

To illustrate their approach, Baresi and Guinea reuse an example pizza delivery business process previously developed in (Baresi et al., 2004b). The pizza delivery process integrates services for customer authentication, customer pizza preferences, pizza selection, credit card validation, looking up delivery addresses, retrieving GPS coordinates, and street maps for the delivery locations. In (Baresi & Guinea, 2005), the authors demonstrate how they associate a post-condition with the map service aspect of the pizza delivery business process.

The approach proposed by Baresi and Guinea includes support for the dynamic selection and execution of monitoring rules at runtime. The authors also provide a user-oriented language for the development of monitoring rules from data acquisition and analysis. The approach makes us of service monitors, which the authors discuss separately in (Baresi et al., 2004a). These monitors observe service compositions for timeouts, runtime errors and violations of functional contracts. When monitors detect a problem, they halt execution of the composition and signal that a problem has been detected. However, the approach does not implement any failure recovery mechanism, such as service negotiation or service substitution. The monitoring approach is limited to simple faults, such as service timeouts and the monitoring of functional contracts. It is not apparent that the approach is general enough to monitor arbitrary non-functional QoS attributes.

Moser et al. discuss a system for the monitoring and adaptation of WS-BPEL business processes in a non-intrusive manner. The approach monitors the QoS attributes of the services involved in a business process. Pluggable service replacement strategies guide the selection of syntactically or semantically equivalent services at runtime. The authors use an aspect-oriented technique to intercept service messages, in order to monitor service behaviour.

Moser et al. evaluate their approach with a case study based on a purchase order web service implemented as a BPEL process. The purchase order web service performs all activities required to place a purchase order, and is composed of five different smaller services. The first service is used to check the stock status of the item(s) required for the purchase order. The second service is used to calculate the total price of the purchase order, including shipping and taxes. The third service validates the consumer's credit card, which if successful is charged

by a fourth service. The fifth service is responsible for initiating the delivery of the purchase order to the consumer. The authors evaluate their approach by substituting alternative credit card services used during the process, and measuring the impact these service substitutions have on system performance.

The initiative proposed by Moser et al. supports the efficient replacement of problematic services at runtime. However, the approach to maintaining system quality is simplistic, and doesn't offer the flexibility of advanced recovery strategies and techniques such as service negotiation. The authors also do not explain how QoS attributes are selected for monitoring. As the approach does not integrate with a service contract or an SLA, it would be difficult to automatically derive the QoS attributes required for monitoring.

Herssens et al. discuss the problem of revising an SLA in response to changes in a service's context. The authors define service context to be a combination of several different non-functional properties. To address the problem, the authors propose a method to autonomously monitor a service's context at runtime, and adapt the SLA between the service consumer and provider as required. The system stores SLAs, and periodically receives context updates from service consumers and providers. The system compares the consumer and provider contexts with the QoS guarantees in the SLA. The system automatically adapts the SLA in response to any faults detected.

Herssens et al. demonstrate their approach using a case study based in the European Space Agency (ESA) program on Earth observation. The ESA provides a large set of web services to access data captured by the medium-resolution imaging spectrometer (MERIS) instrument installed on the Envisat satellite. The data concerns observations of ocean colour and biology, earth vegetation and atmosphere. The ESA services are subject to non-functional quality attributes, including service *latency*, *reliability* and *availability*. The authors devise example SLAs for these services, which include objectives such as *the reliability must be superior to 90%*. The authors then discuss how their SLA manager is able to manage and adapt SLAs, by acting as a third-party mediator between the service user and provider.

Rather than supporting QoS measurement and fault detection, the approach proposed by Herssens et al. adapts SLAs at runtime in response to service faults. Consequently, the approach does not provide a true fault recovery mechanism, and is therefore not capable of maintaining the quality levels in a service-oriented system. The approach also makes the assumption that the context information

provided to the system by consumers and providers is accurate and truthful, which may not be the case.

3.10 Overview and Integration Discussion

This chapter has provided a discussion of several different domains for quality assurance in service-oriented systems. Approaches from each domain offer some quality assurance benefit, but the support is limited due to the lack of integration with approaches from other domains. The most notable research initiatives from each service quality assurance domain are summarised in Table 3.6.

Approaches from the service description domain improve the characterisation of services with non-functional and semantic information. This additional information supports the processes of service advertisement, discovery, selection, composition, substitution, negotiation, and monitoring. However, service description initiatives are themselves not able to ensure quality in service-oriented systems, and instead serve as an enabling factor in other quality assurance approaches.

Approaches from the service discovery domain can be enhanced with non-functional and semantic service information, in order to facilitate QoS advertisement and service selection. However, these enhanced discovery initiatives require the service consumer to trust that service providers will in practice supply the QoS levels they advertise. It is also possible that the consumer will be unable to discover services which meet its non-functional QoS requirements.

Approaches from the reputation system domain can be used to facilitate trust through consumer collaboration. Reputation systems collate and supply additional provider reputation criteria for the service selection process. These systems therefore give service providers an incentive to provide services in accordance with the QoS they advertise. However, the service consumer may still be left unable to find a matching service for its non-functional QoS requirements.

Approaches from the service negotiation domain enable the consumer to secure SLAs for QoS guarantees that are closer to meeting the consumer's non-functional QoS requirements. Service providers can also benefit from negotiation, by maximising their resources between different consumers. However, negotiation initiatives do not give providers an incentive to provide services in accordance with negotiated SLAs, as current negotiation initiatives lack the ability to effectively monitor SLAs for compliance.

Quality Assurance Domain	Service Description	Non-Functional Properties	QoS Ontology	Service Discovery	Service Selection	Reputation Criteria	Service Negotiation	SLA Creation	SLA Enforcement	Service Monitoring	Failure Recovery
Service Description											
Martin et al., (2004)	✓	✓	✓	∼	∼	×	×	×	×	×	×
Dobson et al. (2005)	✓	✓	✓	∼	∼	×	×	∼	×	×	×
Toma et al. (2006)	✓	✓	✓	∼	∼	×	×	∼	×	×	×
Service Discovery and Selection											
Maximilien and Singh (2004a)	✓	✓	✓	✓	✓	×	×	×	×	×	×
Oldham et al. (2006)	✓	✓	✓	✓	✓	×	×	✓	×	×	×
Xu et al. (2007)	✓	✓	✓	✓	✓	×	×	×	×	×	×
Service Reputation System											
Wishart et al. (2005)	✓	×	×	✓	✓	✓	×	×	×	×	×
Ali et al. (2006)	✓	✓	×	✓	✓	✓	×	×	×	×	×
Jurca et al. (2007)	✓	✓	✓	×	✓	✓	×	✓	×	×	×
Service Negotiation											
Comuzzi and Pernici (2005)	✓	✓	×	×	✓	×	✓	✓	∼	∼	×
Menascé and Dubey (2007)	✓	✓	×	✓	✓	×	✓	✓	×	×	×
Yan et al. (2007)	✓	✓	×	✓	✓	×	✓	✓	×	∼	×
Service Monitoring											
Molina-Jimenez et al. (2004)	×	✓	×	×	×	×	×	×	∼	✓	×
Ludwig et al. (2004)	✓	✓	×	∼	✓	×	∼	✓	∼	✓	×
Moser et al. (2008)	∼	✓	×	×	✓	×	×	×	×	✓	∼
Integrated Assurance Solution											
Robinson and Kotonya (2008a, 2008b)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Key: (✓) full support, (∼) partial consideration, (×) no support

Table 3.6: Summary of quality assurance domains.

Approaches from the service monitoring domain can be used to determine if services adhere to SLAs. However, existing monitoring initiatives have poor support for handling SLA violations and service failures, and lack support for advanced recovery techniques such as service negotiation and renegotiation. In addition, current service monitoring initiatives generally focus on reporting past events, and lack support for estimating future QoS based on previous observations.

A software framework provides a set of components which address a problem within a specific domain, and a model which specifies how these components interact with each other. This thesis proposes an integrated service quality assurance framework, that combines approaches from different service quality assurance domains, to provide an improved method of ensuring quality in service-oriented systems. The framework is intended to provide the consumer with increased control over service quality, provide support for the expression of quality characteristics, provide a runtime solution for detecting and recovering from SLA violations and service failures, provide a solution that supports resource-restricted systems, and provide customisation support for the integration of different techniques from within each service quality assurance domain.

3.11 Summary

This chapter began with an introduction to software quality, and the problem of ensuring quality in software. The chapter then discussed software quality in the context of service characteristics, and introduced the problem of ensuring quality in service-oriented systems.

The chapter continued with a detailed survey of current quality assurance approaches for service-oriented systems. These approaches have been grouped into the following service quality assurance domains: service description, service discovery and selection, reputation systems, service negotiation, and service monitoring. The significant research initiatives from each service quality assurance domain have been examined, along with the benefits they bring to the quality assurance process. The limitations with initiatives from each quality assurance domain have also been discussed, and a new solution has been proposed in the form of an integrated quality assurance framework, that combines approaches from each quality assurance domain. The following chapter describes the design and implementation of an integrated quality assurance framework for service-oriented systems.

Chapter 4

Quality Assurance Framework

This chapter discusses the design and implementation of an integrated quality assurance framework for ensuring quality in service-oriented systems. The discussion begins with an overview of the framework, then examines each of the major quality assurance systems provided by the framework. The brokerage system supports the negotiation of SLAs on behalf of service consumers and providers. The monitoring system checks services at runtime for SLA compliance. The reputation system provides consumers with a method of sharing service experience with one another, and adds additional criteria to the service negotiation process.

The chapter then discusses the service ontology provided by the framework. The ontology provides a common set of terms for describing services, service constraints, and service strategies. The discussion then examines the specification of consumer and provider service strategies, and the approach used to calculate the acceptability of services. The chapter concludes with a discussion of the customisation support provided by the quality assurance framework.

4.1 Framework Overview

The quality assurance framework provides three systems for brokering, monitoring and rating services. An overview of these systems is shown in Figure 4.1.

The framework brokerage system creates service brokers for service consumers and providers on demand. The service brokers perform QoS negotiation, SLA creation and evaluation, and resource management activities on behalf of their clients. The framework reputation system collates the global reputation of service providers, from service ratings provided by service consumers. The reputation

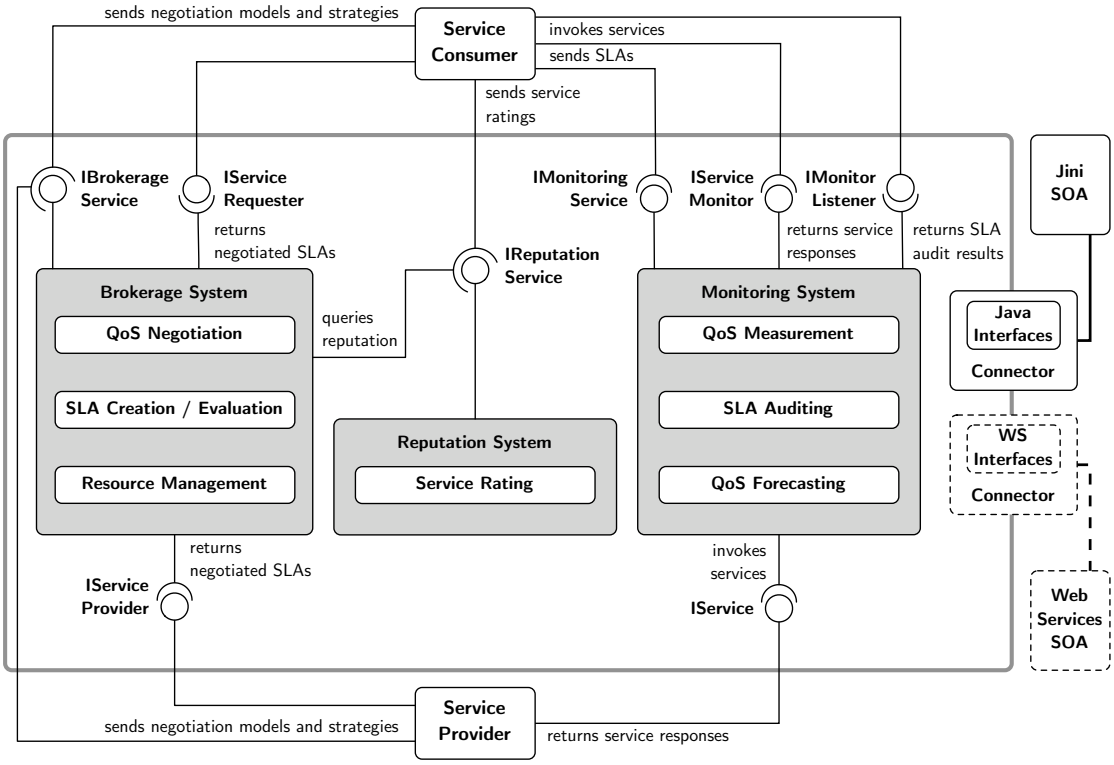


Figure 4.1: The quality assurance framework provides three primary systems for brokering, monitoring and rating services. The framework currently integrates with the Jini SOA.

system is queried by service brokers during service negotiation, and provides additional criteria to the service agreement decision process. Negotiated SLAs are supplied to the framework monitoring system, which measures runtime service performance, audits SLAs for compliance, and forecasts future QoS estimations.

The design of the framework is intended to be independent of any specific service technology, such as the Web Services Architecture (W3C Working Group, 2004), Jini (Sun Microsystems, Inc., 2009b), the Common Object Request Broker Architecture (CORBA) (Object Management Group, Inc., 2004), and the OSGi Service Platform (OSGi Alliance, 2009). However, the current framework implementation integrates with systems which utilise the Jini service platform.

The framework functionality is exposed as Java interfaces, which are published to the Jini service registry for service consumers and providers to discover and use. The Jini service platform was primarily chosen for its dynamic service discovery system, and the benefits provided by the Java (Sun Microsystems, Inc., 2009a) development platform on which it is based. The benefits of the Java platform

include the ubiquity of the Java virtual machine across a range of platforms, the network-centric nature of the platform, the sophisticated security model the platform provides, and Java language features, such as strong typing for guaranteeing the runtime behaviour of software, and exception mechanisms for error handling.

4.2 Brokerage System

The quality assurance framework provides a brokerage architecture for the automated negotiation of services and service agreements. The architecture features one or more brokerage systems, which create service brokers for consumers and providers on demand. These brokerage systems are registered with the service registry, for consumers and providers to discover. An overview of the service brokerage architecture is shown in Figure 4.2.

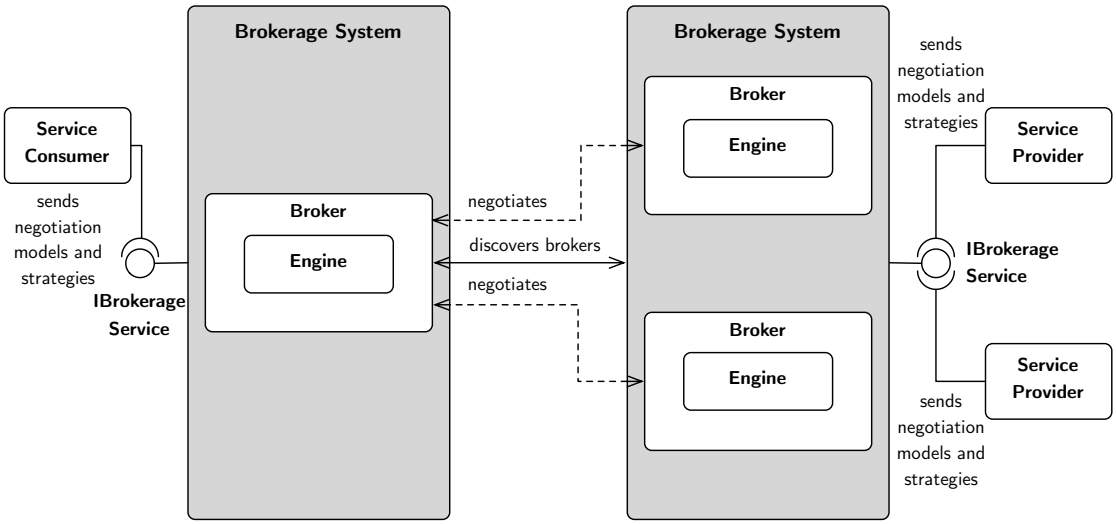


Figure 4.2: The framework brokerage architecture. Negotiation can be contained within a single brokerage system, or span two or more distributed brokerage systems (as shown).

The brokerage system is designed to support the integration of a variety of different negotiation models and decision support algorithms. The service consumer or provider supplies the brokerage system with templates that describe the negotiation models to use. These templates also contain the details of decision algorithms and service strategies, that are applied by the brokers to create and evaluate service proposals during negotiation.

For security, a private key is shared between a broker and its consumer or

provider client. Tokens generated using this key are required for certain restricted operations, to verify the identity of the client invoking the operation. Such operations include requesting the renegotiation or unleasing of a service. In addition, all communication is performed using the Secure Sockets Layer (SSL) network encryption protocol, to prevent eavesdropping, message tampering and forgery.

4.2.1 Broker Engine

The brokerage system features an engine builder component, which uses the consumer or provider templates to assemble a bespoke service broker engine for processing negotiation messages and service proposals. The broker engine can also perform resource management on behalf of a service provider, if required. The broker engine is shown in Figure 4.3.

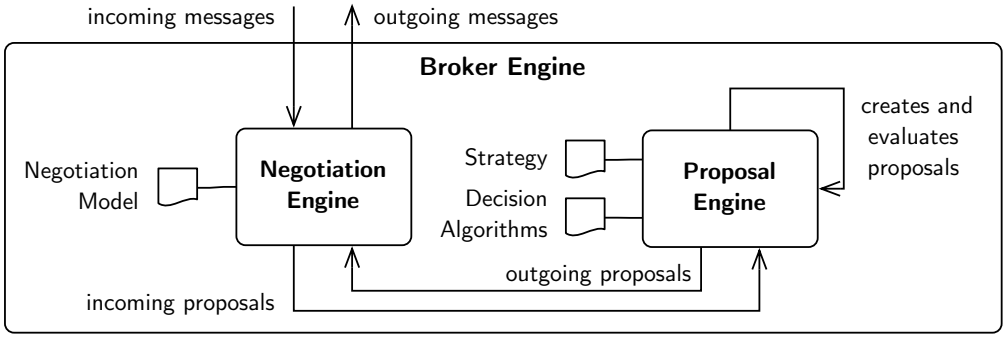


Figure 4.3: The broker engine is assembled by the brokerage system, with one negotiation engine per negotiation protocol. The proposal engine creates and evaluates service proposals.

The broker engine contains a separate negotiation engine for each negotiation protocol it supports. The negotiation engine concurrently negotiates with multiple parties, and maintains a separate negotiation session for each negotiation between itself and another party.

The negotiation engine maintains the integrity of an active negotiation session, by performing assertions on the session's state when processing incoming negotiation messages. If the negotiation engine receives an illegal message for the current state of a session, it signals an error to the sender of the message. For example, if the current negotiation session state is *proposal sent*, the legal incoming message types may be *accept proposal*, *reject proposal*, *propose proposal* and *terminate*. If the incoming message is legal for the session's current state, the negotiation engine forwards any proposal the message contains to the proposal engine.

The proposal engine contains two components for the evaluation and creation of service proposals, as shown in Figure 4.4. The proposal evaluator component examines each incoming proposal, and provides the negotiation engine with a negotiation instruction describing the next action to take. The proposal evaluator maintains a record of which qualities have been negotiated so far in the current session. If an incoming proposal is acceptable but there are still qualities remaining to be negotiated, the proposal evaluator instructs the negotiation engine to *propose proposal*. In this case, the negotiation engine requests a new proposal from the proposal creator component, which adjusts the next quality to negotiate. Once all qualities have been negotiated, the proposal evaluator issues either an *accept proposal* or a *reject proposal* instruction.

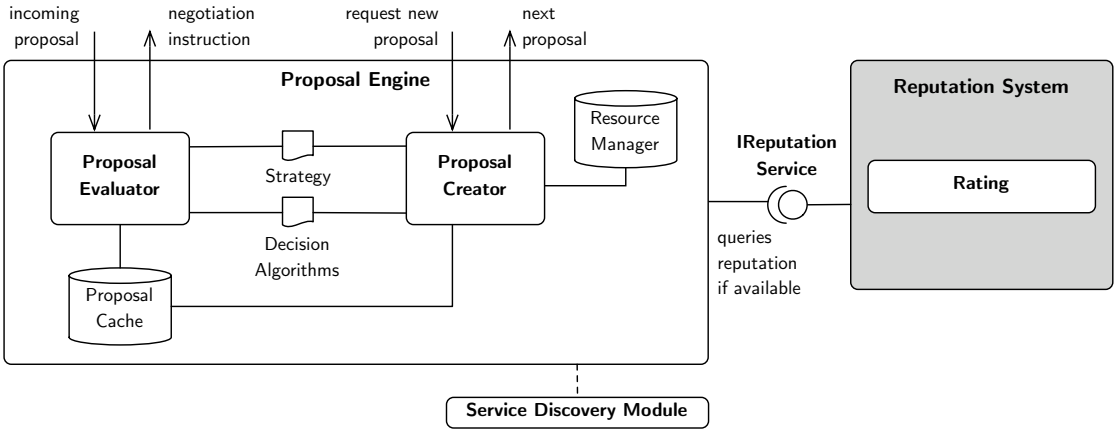


Figure 4.4: The proposal engine creates and evaluates service proposals.

The proposal engine is able to broker service proposals with multiple providers of the same service type. It is also able to broker service proposals for multiple service types, which together form a service composition. In these cases, the proposal engine waits until negotiation is either completed or terminated with each provider, before making the final decision to accept or reject a proposal.

The proposal engine queries the framework reputation system, if available, for the reputation of the parties from which it receives service proposals. Provider reputation is used by consumer proposal engines, to limit negotiation to the brokers of providers that have a certain level of reputation, as specified in the consumer strategy. During negotiation, the consumer proposal engine also combines the reputation of a provider with the service proposal from the provider's broker, in order to determine a proposal's overall acceptability to the consumer. Provider proposal engines use the reputation system to avoid the brokers of consumers who

have previously provided them with poor or unfair ratings.

The automated negotiation messaging process relies on universally-unique identifiers (UUIDs) for the purposes of identifying the participants in the negotiation. UUIDs are also used to identify individual negotiation messages and service proposals, negotiation sessions, and previous negotiation messages which are being responded to. Each message contains a negotiation primitive, such as *propose proposal*, and a timestamp used for ordering the messages in a negotiation session. If the message contains a service proposal, timestamps are provided which indicate when the proposal was created, and when the proposal expires. The proposal expiry timestamp encourages negotiation to be completed in a timely manner. The structure of the negotiation message is shown in Figure 4.5.

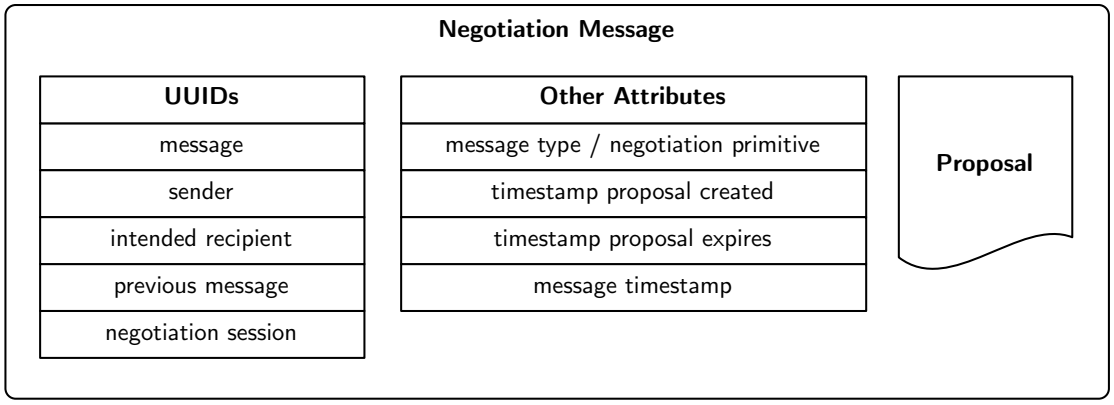


Figure 4.5: Service brokers use negotiation messages to exchange service proposals. Each negotiation message contains a series of UUIDs and attributes to support the automated negotiation process.

4.2.2 Negotiation Protocol

The negotiation protocol currently implemented for the framework, provides a subset of the negotiation message primitives and negotiation states from the protocol proposed in (Su et al., 2001). The negotiation states specify rules for the legal message primitives at each point in a negotiation. The protocol is sufficient to enable the automated negotiation of a service agreement between a consumer and provider broker. The protocol specifically supports models where negotiation is led by the consumer service broker. The consumer and provider negotiation protocols are respectively represented as state diagrams in Figure 4.6 and Figure 4.7.

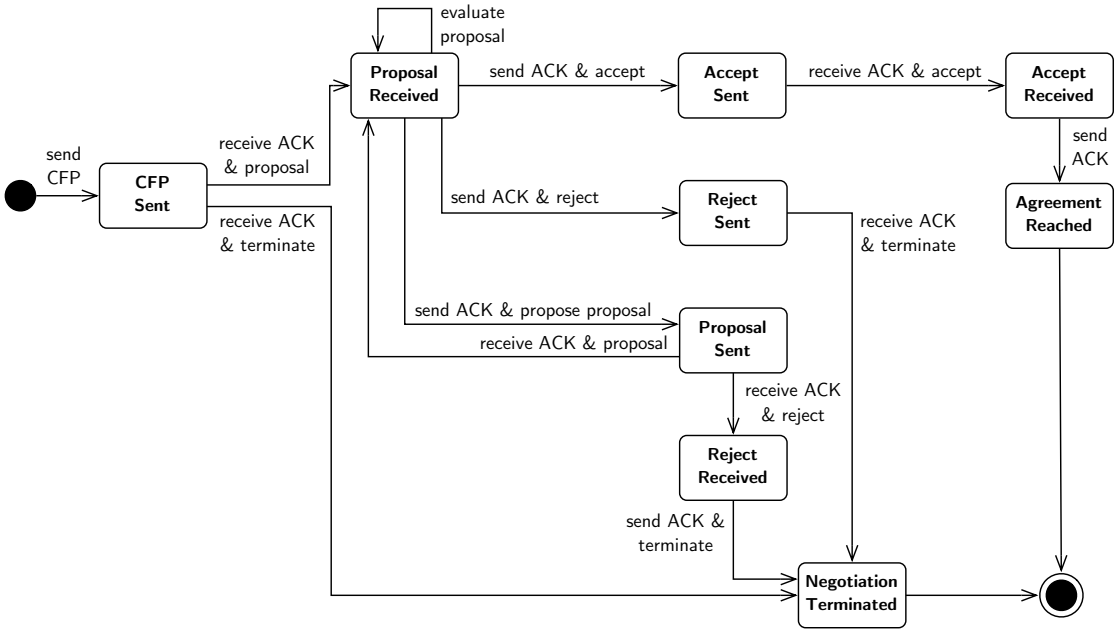


Figure 4.6: The consumer negotiation protocol states. A negotiation session is created when the consumer negotiation engine sends a *call for proposal* (CFP) message to a provider broker. The session ends with both parties accepting a proposal, or is unilaterally-terminated.

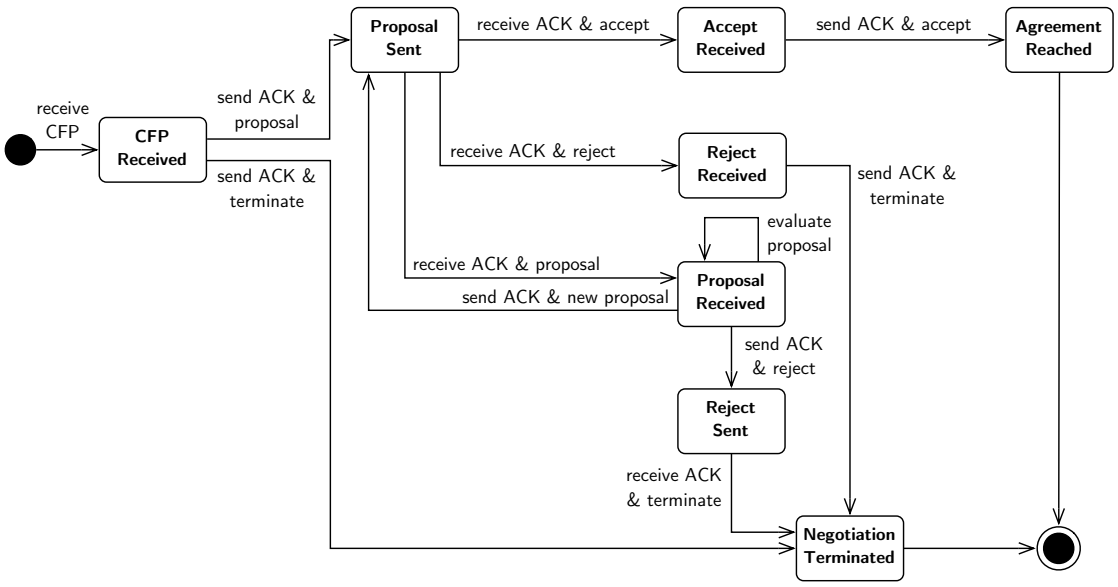


Figure 4.7: The provider negotiation protocol states. The provider negotiation engine is inactive until it receives a CFP message from a consumer broker.

With this negotiation protocol, the consumer broker actively seeks out brokers of providers for services that are functionally-compatible with the consumer's requirements. Provider brokers wait passively for negotiation requests from consumer brokers.

4.2.3 Negotiation Model

There are many different possible negotiation models. Some examples include fixed-price negotiation, auctions, reverse auctions, bargaining, and request for quote (RFQ) models. Two different negotiation models were implemented during the development of the quality assurance framework.

The first implemented negotiation model is a fixed-price negotiation model. The model uses a decision algorithm which will accept an advertised service proposal *as is*, as long as all proposed qualities are within an acceptable range. If multiple advertisements for the same service exist, the algorithm selects the advertisement with the most acceptability. The fixed-price model does not provide support for the negotiation of individual qualities. Instead, the model offers a simple service selection technique, similar to the notion of *catalogue shopping*.

The second implemented negotiation model is a bargaining model, based on static service strategies. With the bargaining model, consumer brokers negotiate service qualities one at a time. The first step is for the consumer broker to issue a *call for proposal* (CFP) message to the provider broker. After receiving a response to the CFP message, the consumer broker selects the first quality to negotiate and sends a revised proposal back to the provider broker. After reviewing the consumer broker's revised proposal, the provider broker counters with a proposal that contains not only its offer for the quality change proposed by the consumer, but revised offers for any other qualities which are related to that quality. For example, if the consumer broker begins by negotiating a service's *response time* quality, the provider broker may counter with a revised value for the *response time* quality, and also a revised value for the service's *cost* quality. To avoid deadlock during negotiation, the broker proposal engines determine which qualities have changed since the previous proposal, and refrain from negotiating them later in the session. An overview of this process is shown in Figure 4.8.

The consumer broker may negotiate with multiple provider brokers of the same service type, and with multiple provider brokers of different service types when negotiating a service composition. Once the consumer broker has finished nego-

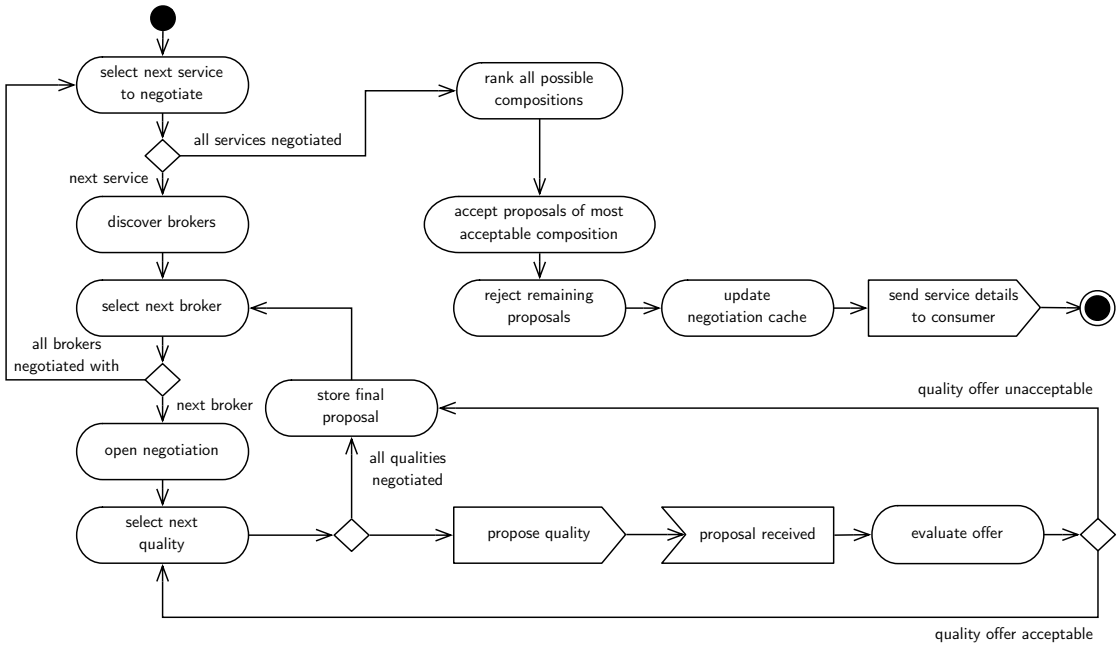


Figure 4.8: Overview of the bargaining negotiation process.

tiating with this set of provider brokers, the consumer broker ranks each service proposal and each possible service composition. The consumer broker then accepts the service proposals for the most acceptable composition. The consumer broker rejects the remaining service proposals, but stores them in ranked order in a proposal cache (shown in Figure 4.4). The proposal cache is used to optimise future negotiation decisions.

4.3 Monitoring System

The quality assurance framework provides a service monitoring system, which actively monitors the quality of negotiated services for SLA violations and service failures. The primary monitoring approach adopted by the framework is a *passive* model, which transparently intercepts service requests and responses between service consumers and providers. To support the passive monitoring approach, service monitors are implemented as dynamic service proxies using the decorator design pattern (Gamma et al., 1995). The passive service monitoring model is shown in Figure 4.9.

The quality assurance framework also supports two additional secondary monitoring models. The first additional model is an *audit-only* approach. With the

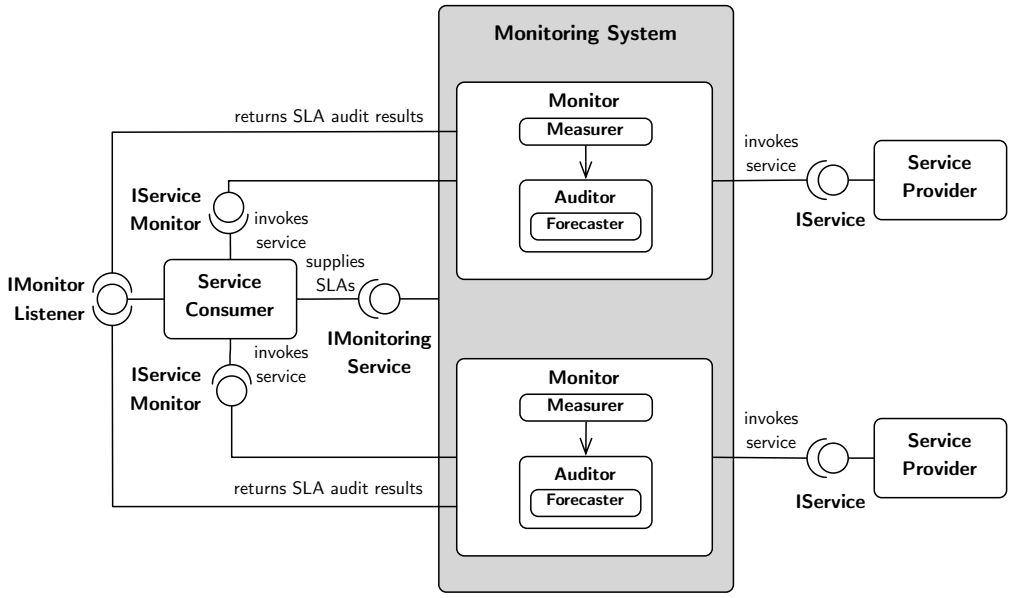
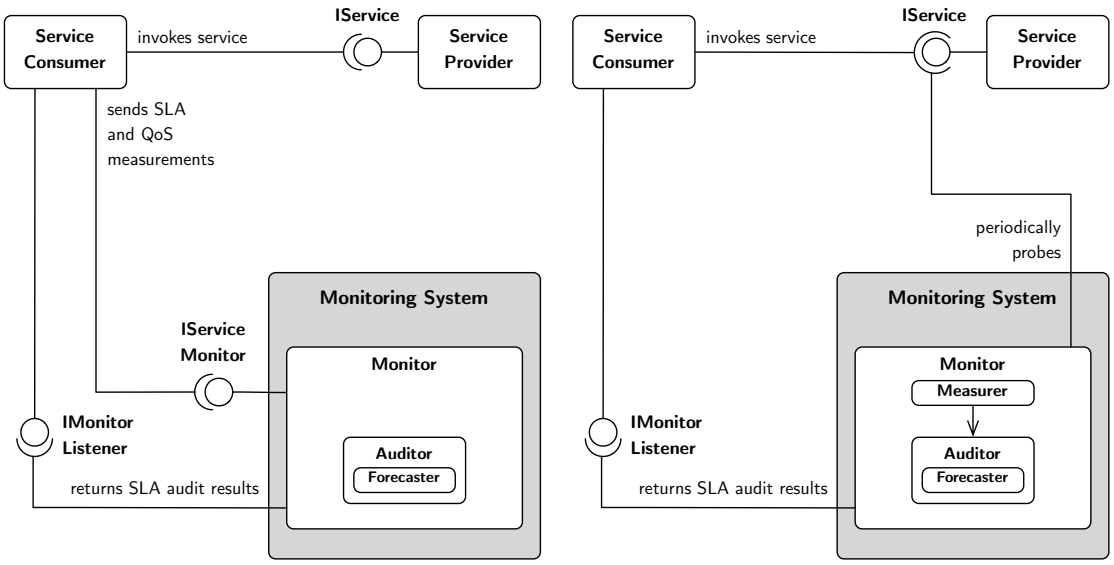


Figure 4.9: The primary framework monitoring architecture. This view shows the primary passive monitoring approach adopted by the framework.

audit-only approach, the service consumer is responsible for collecting QoS measurements itself. The consumer then sends the QoS measurements to the monitoring system, which audits the measurements against the SLA for compliance. The audit-only approach saves the consumer resources that would otherwise be spent auditing the SLA. However, the consumer must expend valuable resources in collecting QoS measurements, and then providing the QoS measurements to the monitoring system. The audit-only approach enables the consumer to trust in the measured QoS, but makes no provision for the monitoring system and service provider to verify the consumer's claims. As such, the audit-only approach is susceptible to abuse from the consumer. The audit-only monitoring approach is shown in Figure 4.10a.

The second additional monitoring model supported by the framework is an *independent probe* approach. The independent probe monitoring approach uses a monitor to periodically probe the service provider, independently of the service requests made by the consumer. The independent probe approach reduces the load on the consumer, but increases the load on the service provider through the generation of additional service requests from the monitoring system. In addition, the service provider is able to distinguish the additional monitor requests from the consumer requests. By distinguishing requests, the service provider has the ability to supply a better QoS level to the monitor, in order to avoid SLA violations. The



(a) Consumer collects QoS measurements. **(b)** Monitor independently probes provider.

Figure 4.10: Alternative service monitoring approaches.

independent probe monitoring approach is shown in Figure 4.10b.

The primary passive monitoring approach adopted by the framework has two significant advantages over the other secondary monitoring approaches. The first advantage is that no additional load is placed on either the service consumer or provider. Instead, the resources of the consumer and provider are left to handling service requests, and do not have to be used for QoS measurement and SLA auditing. The second advantage is that both consumers and providers can mutually trust the monitoring results. All QoS measurement and auditing is contained within the monitoring system, and the provider is unable to distinguish monitor requests from consumer requests.

The auditor component of the service monitor performs assertions on the measured service quality. These assertions compare the measured service performance with the QoS guarantees specified in the SLA. This process is performed both before and after a service is invoked. Pre-invocation audit assertions may be used to check certain pre-conditions before invoking a service. If a pre-invocation audit detects no violation, the service is invoked by the monitor. Post-invocation audit assertions may be used to check the characteristics of the service response and any post-conditions. If no violation is detected during the post-invocation audit, the monitor waits until the consumer next invokes the service.

4.3.1 Service Renegotiation

If a quality's measurement does not conform to its SLA objective, the auditor component signals an SLA violation to the service consumer. On receiving this notification, the consumer can elect to not invoke the service further, and instead instruct its service broker to renegotiate the SLA. If renegotiation is unsuccessful, the consumer broker will attempt to secure service from an alternate service provider, as shown in Figure 4.11.

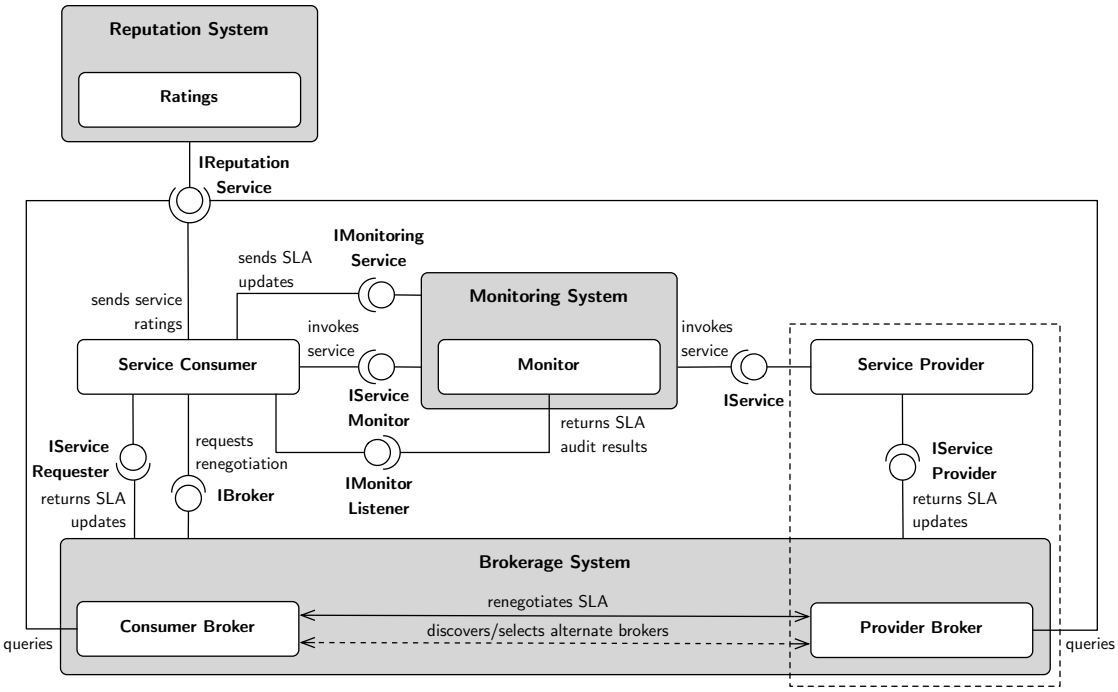


Figure 4.11: Service monitoring and brokerage system integration.

When renegotiating a quality that has violated its SLA objective, the consumer broker assumes that the provider cannot guarantee any level for that quality, better than the level which caused the SLA violation. Instead, the consumer broker expects an offer of improvement in other service qualities, so as to increase the overall acceptability of the QoS guarantee for the consumer. The consumer broker compares the renegotiated proposal, with any proposals it has received from other provider brokers of the same service type. If the renegotiated proposal still offers the most acceptability to the consumer, the broker accepts the renegotiated SLA and continues using the same provider. If the renegotiated proposal no longer offers the most acceptability, the consumer broker will attempt to switch the consumer to an alternate provider, and will reject the renegotiated proposal if successful.

4.3.2 Forecasting Future QoS

The monitors assembled by the monitoring system support the inclusion of a forecasting component to complement the auditing process. The forecasting component estimates future QoS based on previous QoS observations. When requesting a service monitor, the service consumer specifies which of the available forecasting methods to use, and any additional parameters. For example, the consumer may request a simple moving average (SMA) forecasting model, and supply a parameter that specifies the maximum number of data points to consider.

During the auditing process, the latest QoS observations are provided to the forecasting component, which updates its estimations of future QoS. The auditor compares the estimations from the forecaster component with the SLA values, giving it the potential to detect when a particular service quality is likely to fail. If the auditor detects that a quality is likely to fail, the auditor signals to the consumer that the quality is *failing*, rather than signalling an SLA violation. This signal serves to provide the consumer with an early-warning of possible quality issues. The forecasting component is discussed further in Section 4.7.2.

4.4 Reputation System

The reputation system provided by the quality assurance framework, supplies a collaborative mechanism for service consumers to rate the services and providers they have used. The role the reputation system plays in the quality assurance process is shown in Figure 4.12.

A single service instance is defined by the duration of the *lease* specified in the SLA. This service instance may be rated one time only by the service consumer. The consumer submits the service rating to the reputation system after unleasing the service. Services are unleased when the lease agreed between the consumer and provider expires, or after an SLA violation where renegotiation has been performed. The consumer rates the individual qualities experienced while using a service, and provides an overall rating of the service. By rating individual service qualities, consumers can use the reputation system to determine which providers are particularly reputable for the specific qualities they require.

The reputation system combines service ratings for each service provider by service type. This forms the global reputation for each provider's ability to supply a particular service type in accordance with an SLA. The reputation system then

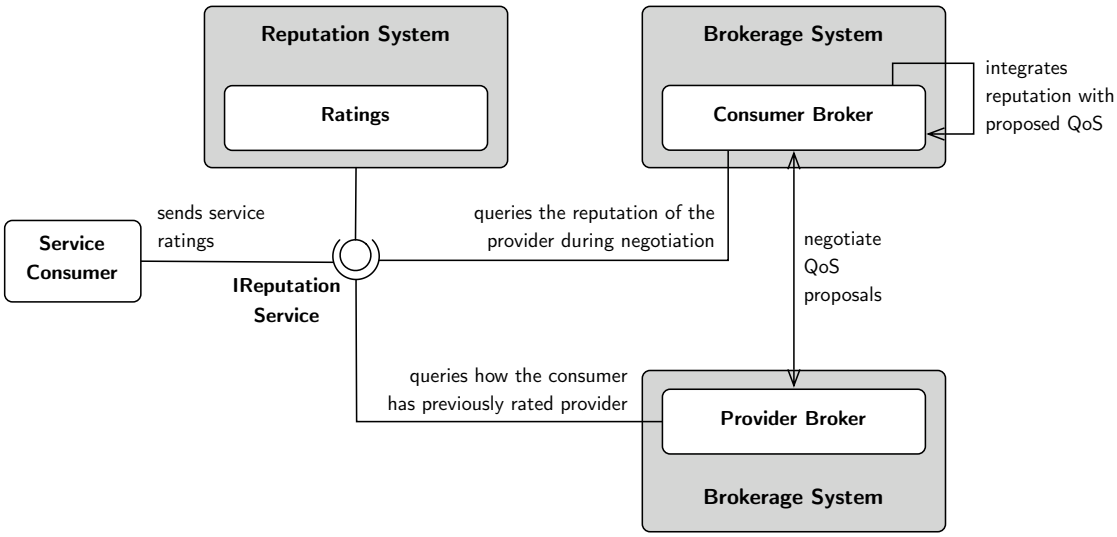


Figure 4.12: The reputation system provides a collaborative mechanism for service consumers to rate the services they have used. The system is queried by consumer and provider brokers during negotiation.

provides several query methods for accessing the reputation data. The system provides a query method that returns the overall reputation of a provider for a particular service type. This method is used by consumer brokers, to limit service negotiation to those providers which meet the minimum reputation threshold specified in the consumer strategy. The consumer broker also combines any available provider reputation data, with the QoS proposed by the provider broker during negotiation. Another query method provided by the reputation system, returns details of the ratings a consumer has given a particular provider. This mechanism is used by provider brokers, to limit negotiation to consumers who have given their clients fair ratings in the past, or who have not used their clients' services before.

4.5 Service Ontology

The quality assurance framework incorporates a service ontology, which provides service consumers and providers with a shared set of terms for describing services, service constraints, and service strategies. The ontology itself is not a key research contribution, but an important requirement for supporting the automation of framework processes such as negotiation and monitoring. For this reason, a simple service ontology to support the research goals was developed, rather than adapting an existing ontology, such as the one proposed in (Dobson et al., 2005).

The service ontology is implemented with the XML Schema (World Wide Web Consortium (W3C), 2001) language. The ontology consists of three distinct schemas, each providing support for the expression of different service features. The *quality* schema supports the description of service qualities, in terms of constraints, measurements and values. The quality schema is then integrated with a *service* schema, that facilitates the description of services and service contracts in terms of functional and non-functional qualities. The quality schema is also integrated with a *strategy* schema, which enables consumers and providers to describe service strategies for their QoS requirements and limits. An example service contract and strategy which validate against these schemas are given in Appendix A. Each of these schemas is now discussed in turn.

4.5.1 Quality Schema

The quality schema provides two key elements for describing a non-functional service quality, as shown in Figure 4.13.

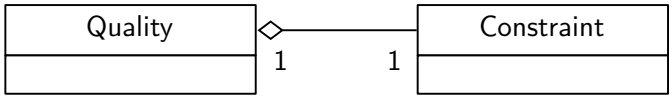


Figure 4.13: Overview of the quality schema ontology elements, used to describe a service quality with a constraint.

The `Quality` element, shown in Figure 4.14, is used to represent a service quality, by associating a basic quality description with a non-functional quality constraint. The quality description is provided by an instance of `QualitySimpleType`, which provides an enumeration of general quality types, such as *availability*, *response time* and *cost* etc.

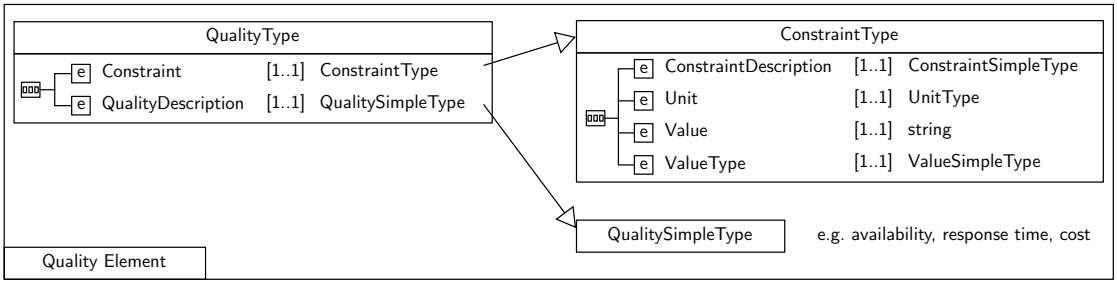


Figure 4.14: The `Quality` element is used to describe a service quality, by associating a basic quality description with a non-functional constraint.

The quality constraint is described by the `Constraint` element, an instance of `ConstraintType`. The `Constraint` element, shown in Figure 4.15, provides attributes for expressing a constraint over a quality.

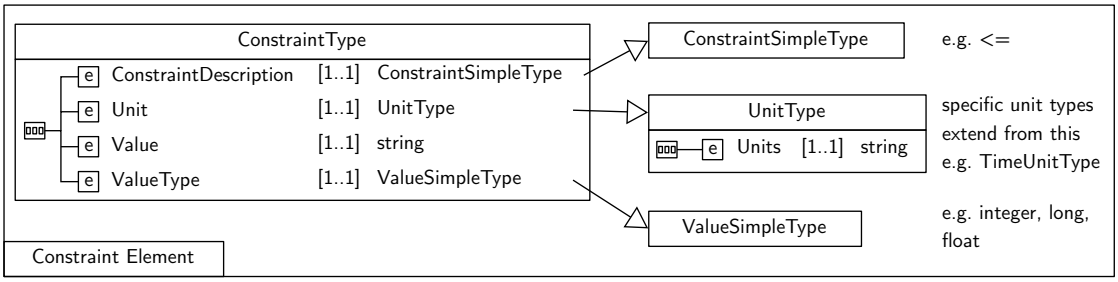


Figure 4.15: The `Constraint` element is used to express a non-functional constraint over a service quality.

For example, the constraint $\leq$1000 ms could be expressed over a *response time* quality. The constraint descriptor ($\leq$) is a value from `ConstraintSimpleType`, which provides an enumeration of common constraint types. The constraint value (1000) is represented as a string by the `Value` element. The value's type is taken from `ValueSimpleType`, which provides an enumeration of common value types, such as *integer*, *long* and *float*. The measurement units for the constraint are described by an element which extends from the base `UnitType` element. The derivative unit element provides an enumeration of measurement units for a specific measurement type. For example, `TimeUnitType` provides time unit descriptors including *ms*, *s*, *min* and *hour*. An example quality which validates against the schema is shown in Figure 4.16.

```
<quality:Quality>
  <quality:Constraint>
    <quality:ConstraintDescription>&lt;=</quality:ConstraintDescription>
    <quality:Unit xsi:type="quality:TimeUnitType">
      <quality:Units>ms</quality:Units>
    </quality:Unit>
    <quality:Value>1000</quality:Value>
    <quality:ValueType>LONG</quality:ValueType>
  </quality:Constraint>
  <quality:QualityDescription>response time</quality:QualityDescription>
</quality:Quality>
```

Figure 4.16: Quality ontology XML example of a *response time* service quality specification. Note that `<=` is the markup language method of representing $\leq$ for the constraint description.

The quality schema is utilised by the service schema, to integrate non-functional service quality information with service descriptions and contracts. The quality schema is also used by the strategy schema, to facilitate the expression of consumer and provider strategies for non-functional QoS requirements and provisions.

4.5.2 Service Schema

The service schema is used to describe a service and a contract, in terms of both functional and non-functional service properties. An overview of the service schema elements and the use of the quality ontology elements is shown in Figure 4.17.

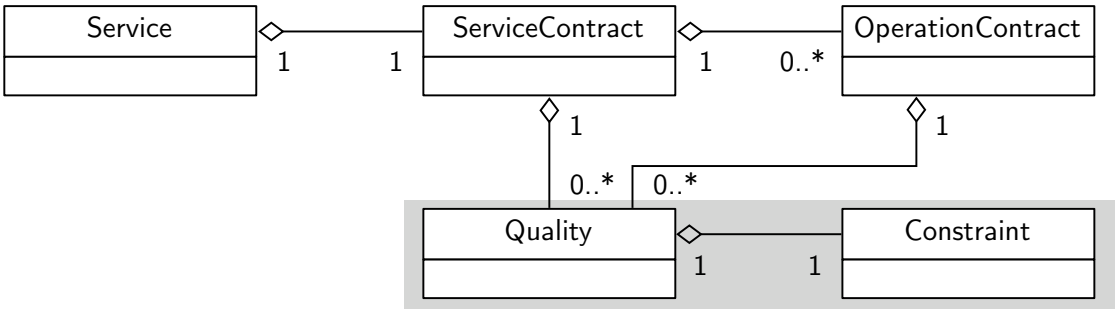


Figure 4.17: Overview of the service ontology elements, used to describe a service and contract in terms of functional and non-functional service properties.

The service schema provides the `Service` element, shown in Figure 4.18, for describing a service. The `Service` element contains a `Lease` element for describing the service lease, which itself provides constraints on the binding between a service consumer and provider. No changes that would affect either party should be made during the period specified by the lease. The `Service` element also contains UUIDs for the provider of the service and the service itself. These UUIDs are used for identification purposes during processes such as service negotiation. The `Service` element then contains the `ServiceContract` element, which describes the functional and non-functional characteristics of the service.

The `ServiceContract` element, shown in Figure 4.19, may contain any number of `ServiceQuality` elements, which are used to specify quality constraints over the service as a whole. The service *lease* is specified separately with the `Lease` element, as a mandatory quality constraint over the service. The fully-qualified service name, which takes a form such as *com.companyname.Service*, is described by the `ServiceType` element. The `ServiceContract` element may then

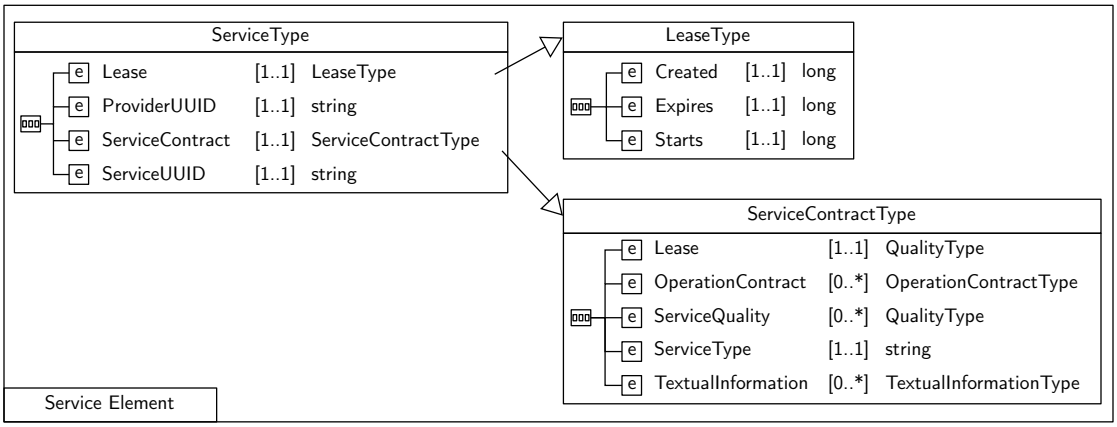


Figure 4.18: The `Service` element is used to describe a service.

contain any number of `OperationContract` elements, which provide functional and non-functional descriptions for each operation provided by a service. As service capabilities are realised as individual service operations, it is necessary to define QoS separately at the operation-level. The `ServiceContract` element may also include any number of `TextualInformation` elements. These information elements are used to provide arbitrary textual descriptions of service issues, such as legal conditions of use, business names and addresses, and technical support details.

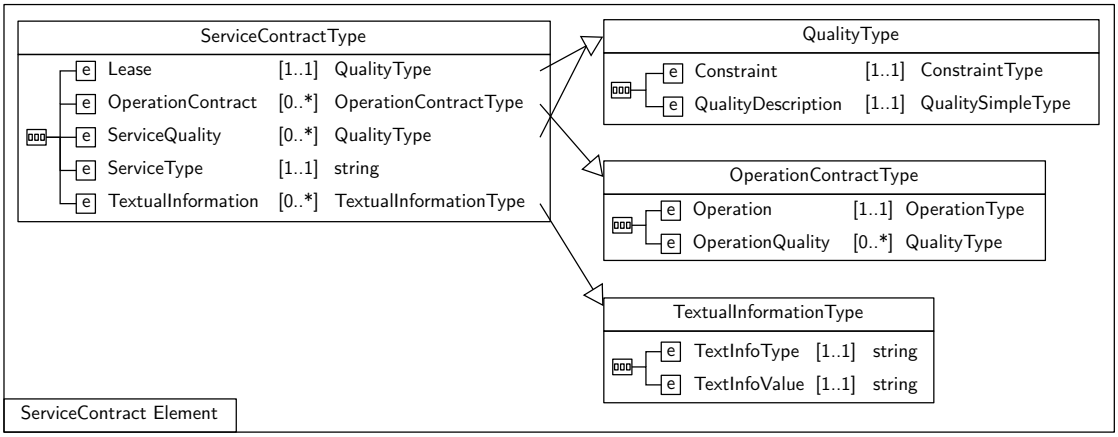


Figure 4.19: The `ServiceContract` element is used to describe the functional and non-functional properties of a service.

The `OperationContract` element, shown in Figure 4.20, is used to describe the functional and non-functional properties of a single invokable service operation. The functional operation is described by the `Operation` element, which provides attributes for the operation's name, signature, input parameters (if any), and

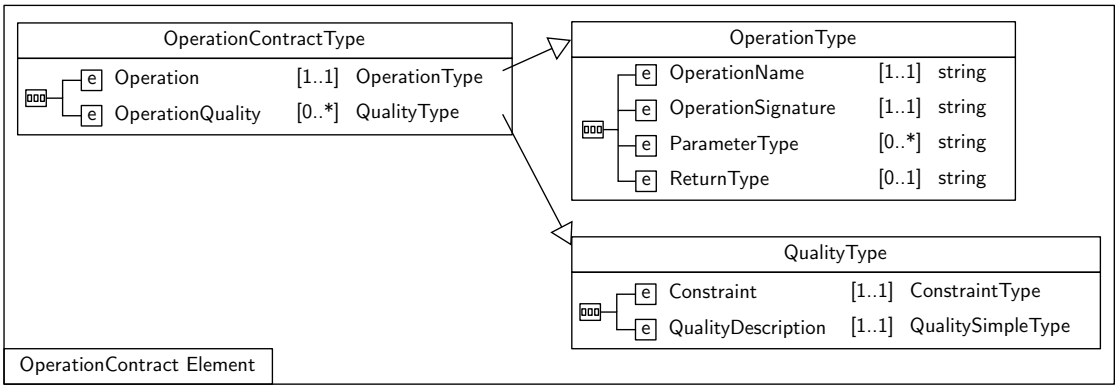


Figure 4.20: The `OperationContract` element is used to describe the functional and non-functional properties of a single service operation.

return type (if any). The `OperationContract` element may then contain any number of `OperationQuality` elements, which are used to specify the non-functional characteristics of the functional operation. An example operation contract which validates against the schema is shown in Figure 4.21. This example is extracted from the larger service contract listing in Appendix A.1.

4.5.3 Strategy Schema

The strategy schema provides elements to express the consumer strategy for a required service or service composition. These elements are also used to describe the provider strategy for negotiating a service and managing service resources. The strategy schema enables both the consumer and provider to express their ideal non-functional service quality requirements, express acceptable limits on these non-functional qualities, and to express relationships between interdependent qualities. An overview of the strategy schema elements is shown in Figure 4.22.

The `Strategy` element is used to describe the strategy for a single service, or a composition of services. The `Strategy` element may contain any number of `ServiceStrategy` elements, which describe individual strategies for each service required or provided by its owner. The `Strategy` element is detailed in Figure 4.23. An example strategy which validates against the strategy schema is provided in Appendix A.2.

The `Strategy` element then provides a series of weight elements, which denote the relative importance of reputation information with the acceptability of a QoS proposal. These weights are applied by the proposal engine when calculating the

```
<service:OperationContract>
  <service:Operation>
    <service:OperationName>add</service:OperationName>
    <service:OperationSignature>
      public abstract int com.xyz.CalculatorService.add(int,
      int) throws java.rmi.RemoteException
    </service:OperationSignature>
    <service:ParameterType>int</service:ParameterType>
    <service:ParameterType>int</service:ParameterType>
    <service:ReturnType>int</service:ReturnType>
  </service:Operation>
  <service:OperationQuality>
    <quality:Constraint>
      <quality:ConstraintDescription>&lt;=</quality:ConstraintDescription>
      <quality:Unit xsi:type="quality:TimeUnitType">
        <quality:Units>ms</quality:Units>
      </quality:Unit>
      <quality:Value>1000</quality:Value>
      <quality:ValueType>LONG</quality:ValueType>
    </quality:Constraint>
    <quality:QualityDescription>response time</quality:QualityDescription>
  </service:OperationQuality>
</service:OperationContract>
```

Figure 4.21: Service ontology XML example, demonstrating how the quality shown in Figure 4.16 is associated with a hypothetical service operation.

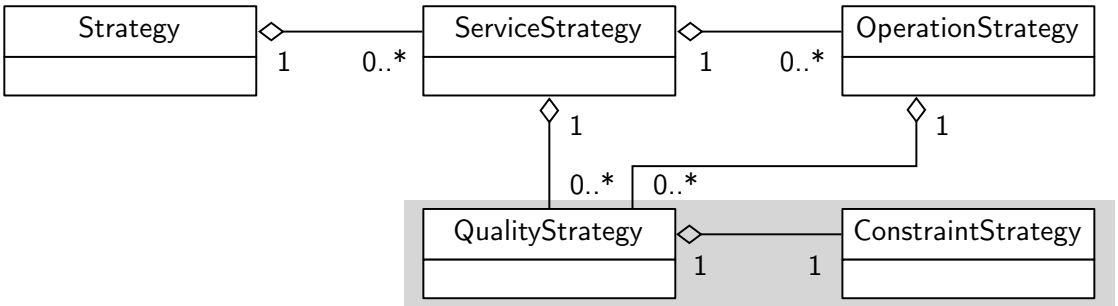


Figure 4.22: Overview of the strategy ontology elements, used to express a consumer or provider service strategy.

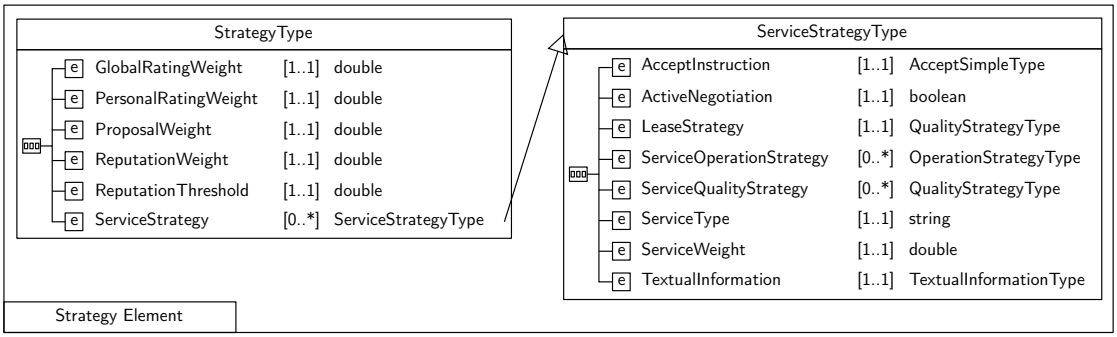


Figure 4.23: The `Strategy` element is used to express how a service or composition of services should be negotiated and managed.

acceptability of a service proposal, which is discussed separately in Section 4.6.

The `GlobalRatingWeight` and `PersonalRatingWeight` elements are both assigned a value between 0.0 and 1.0, so that the sum of their values equals exactly 1.0. These two weights enable the consumer to assign one level of importance to its own experience of a provider, and assign another level to the global experience reported by the reputation system. The overall reputation of a provider is determined by applying these weights to the respective personal and global sources of reputation, and combining the results.

Similarly, the `ProposalWeight` and `ReputationWeight` elements are both assigned a value between 0.0 and 1.0, so that the sum of their values equals exactly 1.0. These two weights enable the consumer to assign one level of importance to the reputation of a provider, and assign another level to the acceptability of the QoS the provider proposes to the consumer. In a consumer strategy, the `ReputationThreshold` element indicates the minimum amount of reputation a provider must have before the consumer will consider its services. For a provider strategy, the `ReputationThreshold` element indicates the minimum rating a consumer must have previously given the provider, for the provider to consider resupplying service to the consumer.

The `ServiceStrategy` element, shown in Figure 4.24, is used to describe the strategy for a single service type. The `ServiceStrategy` contains an `AcceptInstruction` element, which describes the condition for accepting a service proposal. The acceptance condition is an instance of `AcceptSimpleType`, which provides an enumeration of conditions for accepting a service proposal. For example, *best match* means that the best service proposal received should be accepted, regardless of its acceptability to the service strategy. Another instruction, *best acceptable*

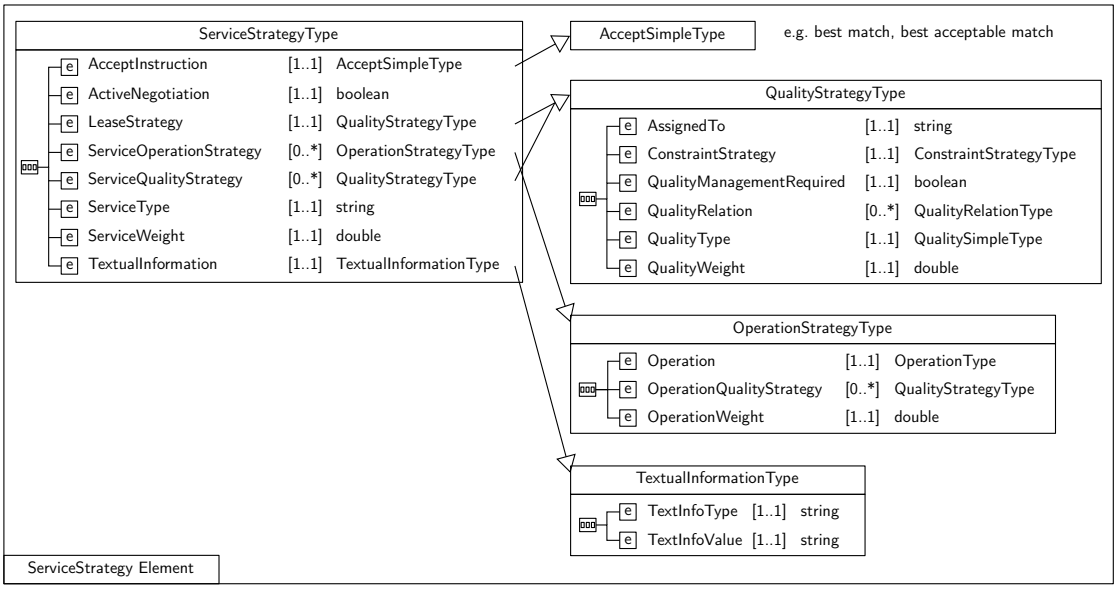


Figure 4.24: The `ServiceStrategy` element is used to describe the strategy for a single service type.

match, means that only acceptable proposals should be considered for acceptance.

The `ServiceStrategy` element also contains an `ActiveNegotiation` element, which holds a simple boolean value to indicate whether the strategy is *active* or *passive*. The value for this element depends on the negotiation model used by the service broker. Typically, the value is *true* for the service consumer, as the consumer actively seeks out services to satisfy its requirements. The value is typically *false* for the service provider, as providers normally wait passively for service consumers to request their services.

The `ServiceStrategy` element may contain any number of `ServiceOperationStrategy` elements, which provide individual strategies for each invokable operation provided by the service. The `ServiceStrategy` element may then contain any number of `ServiceQualityStrategy` elements, which specify strategies for service-level qualities, i.e. those qualities which affect the entire service. The strategy for the service-level *lease* quality is specified separately by the mandatory `LeaseStrategy` element. The `ServiceStrategy` then contains a `ServiceType` element describing the fully-qualified service name the service strategy concerns.

The `ServiceStrategy` contains a `ServiceWeight` element, which reflects the importance of the service in relation to any other services contained within the same overall `Strategy`, i.e. in the case of a service composition. The service weight is specified as a value between 0.0 and 1.0, so that the weights of all `ServiceStrat-`

egy elements contained within a single `Strategy` equal exactly 1.0. These service weights enable the expression of service compositions where certain services are assigned more strategic importance than other services.

The `ServiceStrategy` element may then contain any number of `TextualInformation` elements, which provide textual descriptions of service issues, such as terms of use. These descriptions are copied across into an instance of the `ServiceContract` element, after the non-functional service qualities have been negotiated between the brokers of the service consumer and provider.

The `OperationStrategy` element, shown in Figure 4.25, provides the strategy for a single invokable service operation. The `OperationStrategy` contains an `Operation` element, an instance of `OperationType` from the service ontology, which provides the functional description of the service operation. The `OperationStrategy` may then contain any number of `OperationQualityStrategy` elements, which specify strategies for each non-functional quality constraint over the operation. The `OperationWeight` element holds a value between 0.0 and 1.0, which indicates the operation's importance to any other operations provided by the same service. If the service only provides a single operation, this value is set to 1.0.

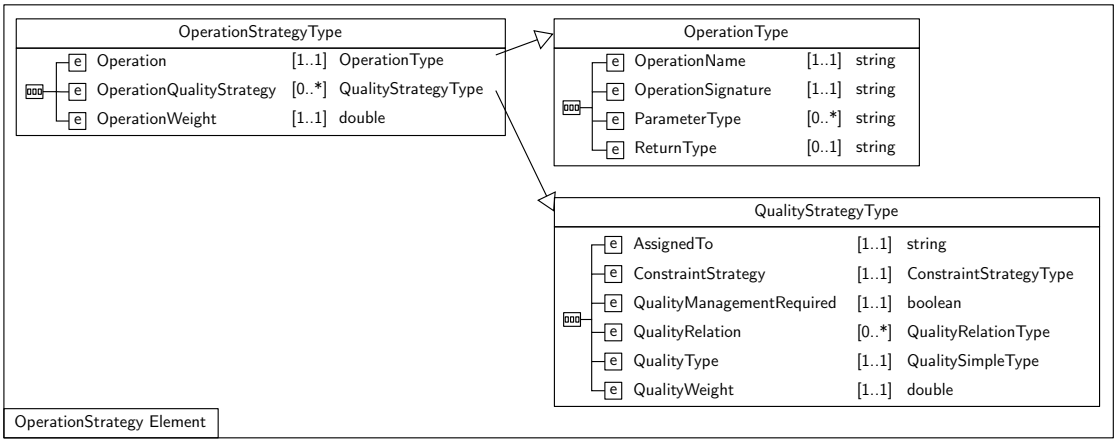


Figure 4.25: The `OperationStrategy` element is used to describe strategy for a single service operation.

The `QualityStrategy` element, shown in Figure 4.26, is used to describe the strategy for a quality associated with a service or service operation. The `AssignedTo` element contains a key for the object the quality is assigned to. This key is either a service type or an operation signature. The `QualityManagementRequired` element provides a flag, which is set to *true* if the quality is a service

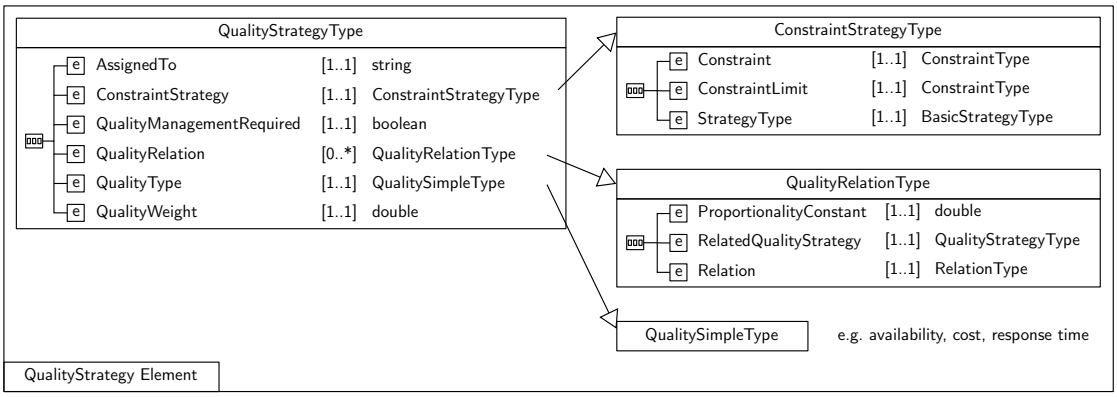


Figure 4.26: The `QualityStrategy` element is used to describe the strategy for a single service or operation quality, and any relationships the quality has with other qualities of the same service or operation.

resource that should be managed by the service broker.

The `QualityStrategy` element includes any number of `QualityRelation` elements, which are used to express interdependent relationships with other service or operation qualities. The `QualityType` element provides a simple description of the quality, from the enumeration provided by `QualitySimpleType`. The `QualityWeight` element contains a value between 0.0 and 1.0, and indicates the quality's importance relative to any other qualities assigned to the same object.

The `QualityStrategy` then contains a `ConstraintStrategy` element, shown in Figure 4.27. The `ConstraintStrategy` element provides a range of acceptable constraints for the quality, and a basic strategy for the quality's value. The `Constraint` element describes the ideal constraint for a particular quality, and the `ConstraintLimit` element describes the least acceptable constraint for the same quality.

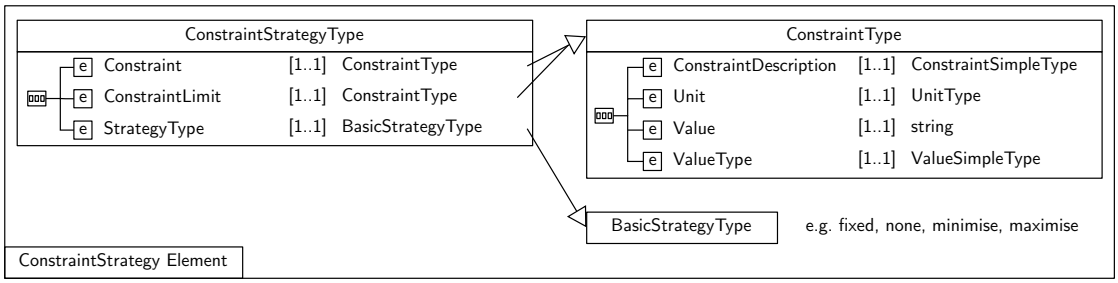


Figure 4.27: The `ConstraintStrategy` element is used to describe the acceptable ranges of constraints for a particular quality.

The `ConstraintStrategy` element contains a `StrategyType` element, which

describes a basic strategy for the quality, from an enumeration provided by `BasicStrategyType`. This enumeration includes strategy type values such as *fixed*, *none*, *minimise*, and *maximise*. The *fixed* strategy means that only the ideal constraint is acceptable, and no limit should be used. The *none* strategy means that any constraint in the range specified by the ideal constraint and constraint limit is acceptable. The *minimise* strategy means that smaller values are preferred for the quality constraint, while the *maximise* strategy means that larger values for the quality constraint are preferred.

4.6 Service Acceptability

The quality assurance framework service brokers are currently configured with utility-based methods for calculating the acceptability of service proposals and provider reputation. These methods operate by comparing QoS proposals and reputation, with the service strategies of consumers and providers.

The strategy template, discussed in Section 4.5.3, provides a description of a consumer's or provider's QoS goals. Each service, service operation and service quality in the strategy template is given a weighting from 0.0 to 1.0, so that the sum of all service- and operation-level qualities is 1.0 (the ideal QoS). The acceptability of a single quality proposal Q_a is calculated as shown in Equation 4.1. This particular formula is based on the acceptability formula given in (Lock, 2006), but extended to factor in the weight Q_w of a single quality, as the quality pertains to the overall QoS of the service or operation to which it is assigned. Q_p is the value proposed for the quality, Q_l is the least acceptable value for the quality, and Q_m is the most acceptable value for the quality.

$$Q_a = \left| \frac{Q_p - Q_l}{Q_m - Q_l} \right| \times Q_w \tag{4.1}$$

The formula is applied to proposed values which fall within the range of acceptable values defined by the quality strategy. If a proposed value falls outside of either side of this range, it is assigned an acceptability of 0.0. Whether high or low values are more or less acceptable, depends on the specific quality type. For example, with the *response time* quality, a greater value is usually less acceptable, and a smaller value more acceptable.

To compute the overall reputation R of the creator of a proposed quality,

any prior local experience R_l of the creator is combined with the global rating R_g provided by the reputation service. The overall reputation of the creator of a proposed quality is calculated as shown in Equation 4.2. G_w is the weight assigned to global experience, and L_w is the weight assigned to local personal experience, so that $0 \leq G_w \leq 1$ and $0 \leq L_w \leq 1$, and $G_w + L_w = 1.0$.

$$R = (R_g \times G_w) + (R_l \times L_w) \tag{4.2}$$

The total acceptability Q_{ta} of a quality is then formed by combining the proposed quality acceptability Q_a, with the total reputation R of the proposal's creator, and applying weights which balance the importance between the quality proposed by the creator, and the creator's reputation. The total quality acceptability is calculated as shown in Equation 4.3, where P_w is the weight of the proposal, and R_w is the weight of the total reputation, so that $0 \leq P_w \leq 1$ and $0 \leq R_w \leq 1$, and $P_w + R_w = 1.0$.

$$Q_{ta} = (Q_a \times P_w) + (R \times R_w) \tag{4.3}$$

However, if no reputation information is available for the proposal's creator, Q_{ta} is simply the same as the proposed quality acceptability Q_a, and the proposal weight P_w is not applied.

Qualities can be assigned to individual service operations. The total acceptability of a single service operation O_{ta} is calculated as shown in Equation 4.4, so that $0 \leq O_{ta} \leq 1$. I represents the total number of qualities assigned to the operation.

$$O_{ta} = \sum_{i=0}^{I} |Q_{ta}|^i \tag{4.4}$$

The total acceptability of a single service S_{ta}, so that $0 \leq S_{ta} \leq 1$, is calculated as shown in Equation 4.5. First, the acceptability of each operation O_{ta} is multiplied by the operation's weight O_w divided by 2. The division by 2 is so that the summed acceptability of all operations is equal in weight to the summed acceptability of all service-level qualities, as the service-level qualities apply across and affect the entire service, and not just a single operation. For this formula, I represents the total number of operations the service provides, and J represents the total number of service-level qualities.

$$S_{ta} = \sum_{i=0}^{I} \left| O_{ta} \times \frac{O_w}{2} \right|^i + \sum_{j=0}^{J} \left| \frac{Q_{ta}}{2} \right|^j \tag{4.5}$$

The total acceptability of a service composition C_{ta}, so that $0 \leq C_{ta} \leq 1$, is calculated as the sum of each total service acceptability S_{ta}, after applying the individual service weight S_w to each service. The composition acceptability calculation is shown in Equation 4.6. Here, I represents the total number of services in the composition.

$$C_{ta} = \sum_{i=0}^{I} |S_{ta} \times S_w|^i \tag{4.6}$$

Calculating the acceptability of every possible composition is an NP-hard problem, with every additional service type in a composition adding another order of magnitude to the number of possible compositions to compare. The basic approach presented here is suitable for comparing compositions that consist of a relatively small number of services. The approach would not scale to more complex compositions, but is suitable for supporting the research objectives of this work. Alternate approaches which specifically address the composition comparison problem, include those based on dynamic programming (Poladian et al., 2004) and heuristics (Berbner et al., 2006).

4.7 Customisable Components

The quality assurance framework provides support for customising certain aspects of the framework systems, so that different brokerage and monitoring schemes can be integrated with one another. The customisable components and the framework subsystems they are associated with are shown in Figure 4.28.

To illustrate how new functionality can be added to the framework, the next section provides an example that discusses how additional forecasting support can be added to the monitoring system. The additional forecasting support enables the monitoring system to assemble service monitors with a forecasting method that wasn't previously available.

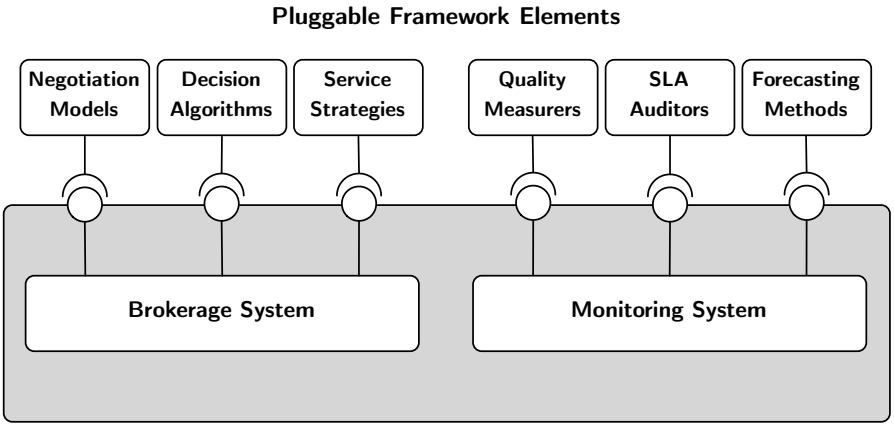


Figure 4.28: The quality assurance framework supports customisable component configurations. Here, ball-and-socket notation is used to represent the notation of pluggability.

4.7.1 Monitor Assembly

The monitoring system provides a customisable architecture for plugging in additional quality measurement instrumentation, SLA auditors and forecasting methods. The monitoring service co-ordinates several builder components, to assemble bespoke service monitors which meet the requirements of the service consumer. An outline of the monitor assembly architecture is given in Figure 4.29.

A similar architecture is used for the construction of custom negotiation engines for service brokers, which implement particular negotiation models, decision algorithms and service strategies. For the purposes of demonstrating the customisation support, this discussion is limited to the service monitoring system and the assembly of service monitors.

4.7.2 Extending Forecast Support

The monitors assembled by the monitoring system support the inclusion of a forecasting component, used to estimate future QoS based on past service performance observations. There are many different forecasting methods, examples of which are discussed in (Wood, 1976) and (Wolski, 1998). For this reason, it is desirable to provide a pluggable architecture that can support different forecasting approaches.

The experiments used to evaluate the framework, discussed in Chapter 5, make use of an exponential moving average (EMA) forecasting model. The EMA forecasting method assigns more importance to recent service performance observa-

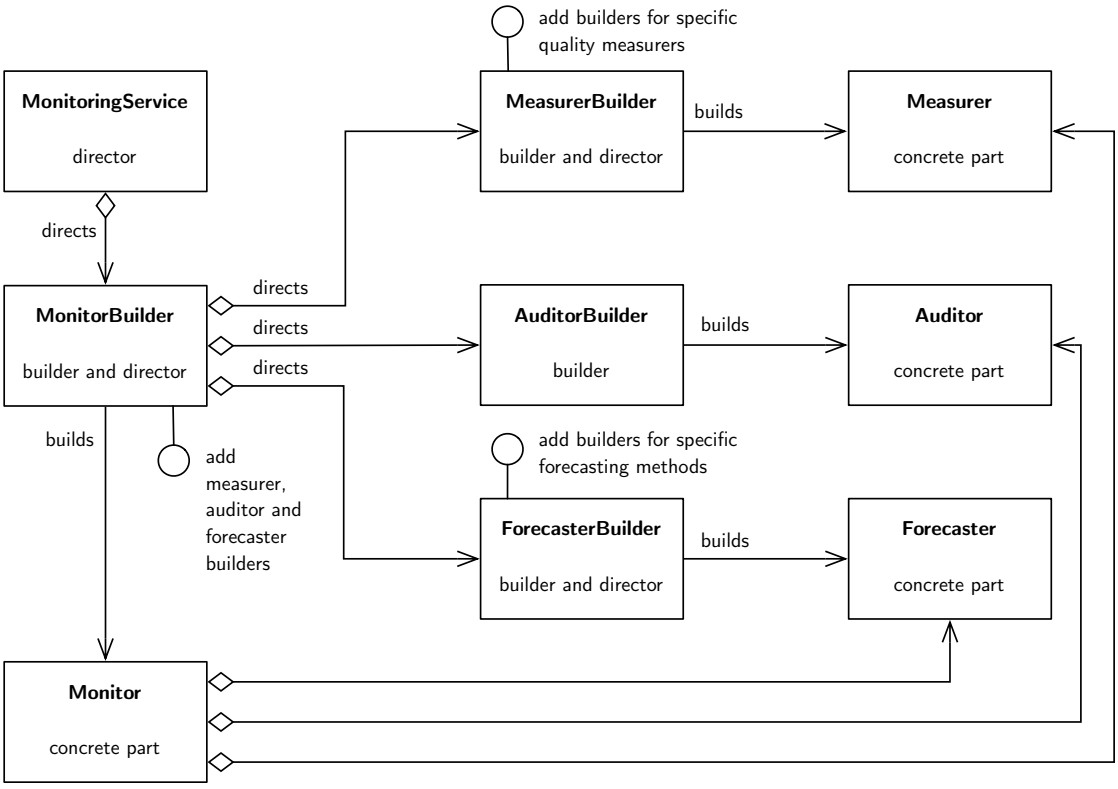


Figure 4.29: A overview of the monitor assembly components.

tions. There follows a discussion describing how support for an additional forecasting method based on a simple moving average (SMA), may be added to the monitoring system. The SMA forecasting method computes a forecast based on the median observed value from a fixed number of earlier observations. The number of earlier observations is sometimes referred to as the *window size*.

Figure 4.30 shows the forecasting component architecture in relation to the monitoring system, the abstract forecast components which are extended by the new SMA forecast components, and the forecaster builder component which creates the forecasting components for the service monitor assembly.

To support the assembly of service monitors with a variety of forecasting methods, the framework provides the abstract `ForecastMethod` class, which in turn provides functionality common to all forecasting methods. The `ForecastMethod` class also provides two abstract methods, which all concrete forecasting methods are responsible for implementing:

```
+addObservedValue(value:Object):void
+getNextForecast():Object
```

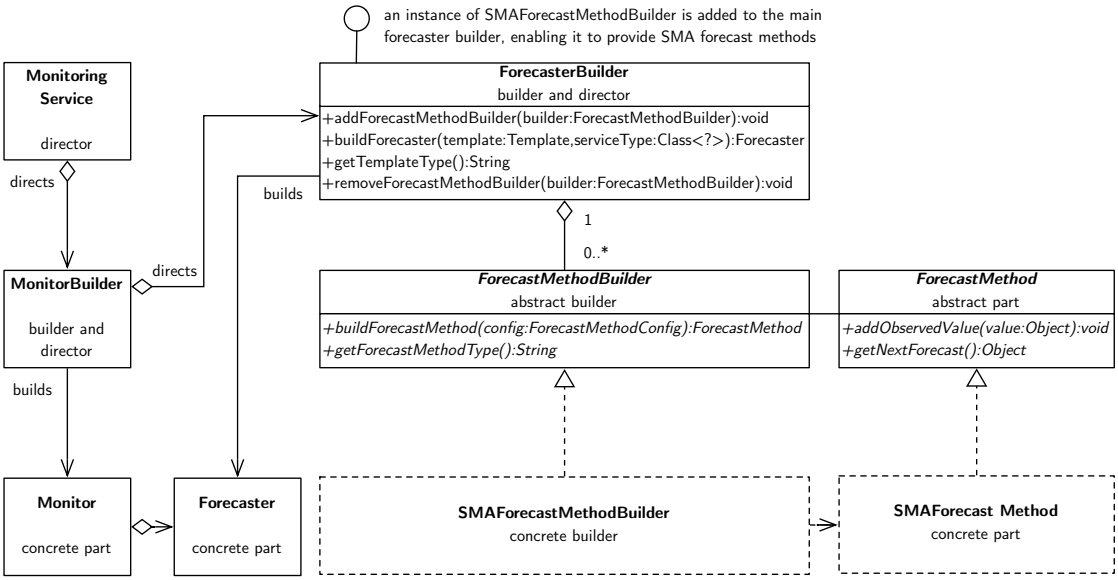


Figure 4.30: The forecaster component architecture. The SMA forecast method functionality extends from the abstract forecasting classes provided by the framework.

The abstract `ForecastMethodBuilder` class provides functionality common to the builders of all forecast methods. The `ForecastMethodBuilder` class also provides two abstract methods, which are invoked by the `ForecasterBuilder` when assembling the forecasting component at runtime. These methods, which all concrete forecast method builders are responsible for implementing, are:

```
+buildForecastMethod(config:ForecastMethodConfig):ForecastMethod
+getForecastMethodType():String
```

To implement the new SMA forecasting method, an `SMAForecastMethod` class is created. The `SMAForecastMethod` class extends the abstract `ForecastMethod` class, and implements its two abstract methods. To enable the assembly of forecasters which use the new SMA method, the `SMAForecastMethodBuilder` support class extends `ForecastMethodBuilder`, and implements its two abstract methods. An instance of `SMAForecastMethodBuilder` is added to the general `ForecasterBuilder` component, providing the builder with the support to assemble a forecasting component that utilises the new SMA forecasting method.

4.8 Summary

This chapter has provided a description of the design and implementation of the integrated quality assurance framework, and the key brokerage, monitoring and reputation systems it provides for quality assurance in service-oriented systems. The chapter discussed the quality ontology the framework provides in order to support the quality assurance processes implemented by the key framework systems. The chapter then provided a description of the utility-based acceptability formulas, used in conjunction with consumer and provider service strategies, for computing the acceptability of services and provided QoS. The chapter concluded with an examination of the support the framework provides for customising certain system components, as a method of integrating a variety of different quality assurance techniques. The following chapter provides a series of service-oriented experiments, for the purposes of demonstrating and evaluating the quality assurance framework.

Chapter 5

Evaluation

This chapter provides an evaluation of the integrated quality assurance framework, using a series of experiments that test identified hypotheses. The experiments run on a simulated service-oriented navigation application composed from multiple services, and involve a series of consumer devices with differing requirements and service strategies. Multiple service providers of each required service type are made available, which offer functionally-equivalent services but with different QoS levels.

Service doping mechanisms are used to alter the QoS provided to the consumer devices at runtime. These doping mechanisms enable the development of different QoS scenarios, which highlight how the framework supports the consumer in maintaining satisfactory quality levels while using the navigation application. The evaluation concludes by discussing scenarios which highlight some limitations of the support provided by the quality assurance framework.

5.1 Service Simulation

The simulated navigation application is composed of location, map, traffic, weather, and information services. The navigation application is location-based, with the location service first queried to obtain the location of the consumer device executing the application. The location is then passed in a request to the map service, to retrieve a map for the current location. The location is also passed to the traffic, weather and information services, which provide data to be visually overlaid on the map. The size and complexity of the simulation data is summarised in Table 5.1.

The simulation system is implemented using the Jini SOA. The brokerage, monitoring and reputation systems of the quality assurance framework are regis-

Service Type	No. Providers	No. Consumers	No. Qualities	No. Doped Qualities
Location	5	3	4	2
Map	5	3	5	3
Traffic	5	3	5	3
Weather	5	3	4	2
Information	7	3	5	3

Table 5.1: Summary of size and complexity of simulation data.

tered with the Jini service registry. Providers of simulated location, maps, traffic, weather, and information services also register with the service registry, and each provider establishes a relationship with a broker from the brokerage system. Each provider offers services with different levels of QoS and reputation, making certain providers more acceptable than others for the different consumer devices.

Three separate consumer devices are simulated for the experiments. These devices, which are an automobile navigation system, an internet tablet and a mobile phone, each have different resource restrictions and QoS requirements from the navigation application. The consumer devices query the service registry to obtain a broker from the brokerage system, obtain monitors from the monitoring system, and provide feedback on services and providers to the reputation system.

The simulation is controlled via a software tool, which visualises different aspects of the system and framework processes at runtime. A walkthrough of the software tool is provided in Appendix B. For the purposes of presenting the experiments in this chapter, the data captured by the tool has been plotted separately.

5.2 Service Doping

To simulate different QoS scenarios, a service doping approach has been developed that enables service qualities to be doped in specific ways, e.g. the *availability* quality of a service can be periodically doped to simulate service outages. The service doping approach facilitates experiments which would prove difficult to perform in the real-world, requiring a suitable service marketplace and the co-operation of commercial service providers. Figure 5.1 shows an overview of the approach.

Each service provider supplies a service doping specification to the service doper builder, which assembles a mechanism to dope the provider's services in a particular manner, affecting the QoS provided to service consumers at runtime. The doper assumes the place of a load-balancing or service resource management component.

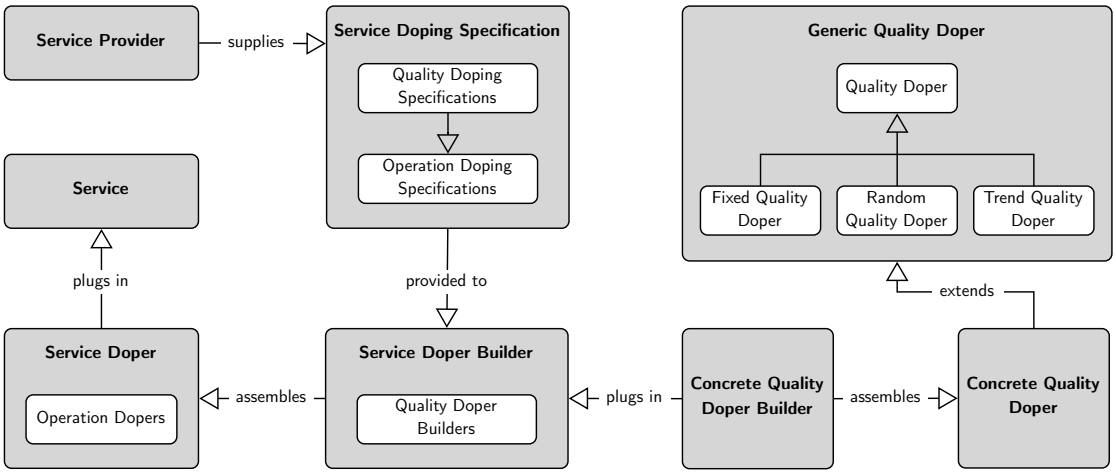


Figure 5.1: Service doping architecture.

Generic quality dopers are provided by the doping framework, supplying common functionality for doping service qualities. *Fixed* quality dopers affect qualities by a fixed level. *Random* quality dopers affect qualities by random levels, and by random levels within a fixed range of values. *Trend* quality dopers affect qualities in a particular direction using a fixed range of values, so as to produce a trend. Trends can be created which have fixed or variable fluctuations in quality levels.

Individual quality doper builders supply the ability to assemble quality dopers for specific quality types, such as *response time*, *memory usage* and *availability*. These concrete quality dopers extend from the generic quality dopers described earlier. The quality doper builders implement a common interface, which enables them to be plugged into the extensible service doper builder. The service doper builder is then used to assemble the final service doping mechanism for a service.

Qualities are doped in periods specified either in terms of time, or in terms of a number of service invocations. All quality dopers are able to alter the time or service invocation periods during which doping is active and inactive. Doping can be performed continuously, for a fixed or random number of invocations, or for a fixed or random period of time. When active, doping can be performed on specific or random service invocations, and at specific or random points in time.

5.3 Service Strategy

Service consumers and providers both specify strategies for the services they respectively require or provide. For the consumer, the strategy describes the ideal

QoS requirements over the functional services it wishes to use. For the provider, the strategy describes the ideal QoS it wishes to offer each consumer, based on the provider's service resource availability. Consumers and providers supply these strategies to their respective service brokers, along with instructions describing which negotiation models to use.

Strategies are described using the service ontology provided by the quality assurance framework. The ontology is shared by service consumers and providers, and supplies a set of standard quality descriptions, constraints and measurement units. If the framework did not provide such an ontology, service consumers and providers would have to agree upon the description terms to use, so as to reduce ambiguity and misunderstandings. As the ontology enables the expression of machine-readable service and strategy descriptions, a high degree of automation can be achieved in processes such as service negotiation, selection and monitoring.

5.3.1 Consumer Strategy

Each consumer has an overall strategy for the service or service composition it requires. For a composition strategy, the consumer assigns each service a weight between 0.0 and 1.0, which indicates the service's importance relative to other services in the composition, so that the weights of all services in the composition add up to 1.0. The consumer strategy also specifies a series of weights which balance the importance of a provider's proposed QoS, either advertised or negotiated, with the reputation of the provider. Provider reputation is itself weighted in terms of a provider's global reputation, and any personal experience the consumer has of a provider. Finally, the consumer strategy contains a reputation threshold, which specifies the minimum level of provider reputation acceptable to the consumer. A summarised consumer strategy for the navigation composition is given in Table 5.2.

The consumer strategy then contains further individual strategies for each service required for the composition. Each service strategy is then broken down into individual quality strategies, which are assigned to specific service operations, or assigned over the service as a whole. An individual consumer strategy for the map service aspect of the navigation composition is given in Table 5.3.

In the example shown in Table 5.3, *availability*, *ram* and *response time* quality strategies are assigned to the operation used to retrieve a map for a given location. The *cost* and *lease* quality strategies are assigned to the map service itself, and apply across all operations the service provides. However, the map service example

Service Type	Service Weight
InformationService	0.30
LocationService	0.20
MapService	0.20
TrafficService	0.20
WeatherService	0.10
Total	1.00

(a) Service weights.

Reputation Property	Weight
Global Reputation Weight	0.30
Personal Experience Weight	0.70
Proposal Weight	0.40
Reputation Weight	0.60
Reputation Threshold	0.25

(b) Reputation weights and threshold.

Table 5.2: Strategy summary for the automobile navigation system.

Quality Type	Assigned To	Quality Weight	Quality Ideal	Quality Limit	Strategy Type	Relations (with Relation Type and Proportionality Constant)
cost	MapService	0.70[a]	≤0.00 €	≤2.00 €	minimise	lease (direct, 1.0) availability (inverse, 0.25) ram (inverse, 0.25) response time (inverse, 0.5)
lease	MapService	0.30	==60 min	≤120 min	minimise	cost (direct, 1.0)
availability	getMap(location: Location):Map	0.40[b]	≥90%	≥75%	maximise	cost (inverse, 0.25)
ram	getMap(location: Location):Map	0.10	≤32 KB	≤48 KB	minimise	cost (inverse, 0.25)
response time	getMap(location: Location):Map	0.50	≤100 ms	≤5000 ms	minimise	cost (inverse, 0.5)

[a] In relation with all service-level qualities (i.e. cost and lease).
[b] In relation with all qualities of the same operation (i.e. availability, ram and response time).

Table 5.3: Map service strategy for the automobile navigation system.

in Table 5.3 features only a single service operation.

With the exception of the *availability* quality, the general consumer strategy is to *minimise* the value of each quality within the constraints describing the ideal quality and the limit for the least acceptable quality. The consumer strategy also specifies any relations between interdependent qualities. For example, the strategy in Table 5.3 describes the relationship between *lease* and *cost* to be directly proportional, with a proportionality constant of 1.0. This means any increase or decrease in the length of the service lease, respectively increases or decreases the cost of the service by a directly proportional amount.

5.3.2 Provider Strategy

A service provider has a strategy for each service it is prepared to provide to one or more service consumers. The strategy describes the standard QoS the provider

wishes to provide each consumer. If a consumer demands an alternative QoS during negotiation, the strategy gives the provider's broker the information required to balance service resources among current and potential consumers. The strategies for the set of map service providers used in the experiments are summarised in Table 5.4. The differences between the provider strategies mean that the consumer will usually reach service proposals of varying acceptability with each provider.

Quality Type	Advertised Constraint	Units	Strategy	Provider 1	Provider 2	Provider 3	Provider 4	Provider 5
cost	$\leq$	€	maximise	0.275 to 4.00	0.50 to 5.00	0.10 to 5.00	0.10 to 10.00	0.25 to 5.00
lease	$\leq$	min	maximise	5 to 240	60 to 720	60 to 120	30 to 720	10 to 480
availability	$\geq$	%	minimise	80 to 95	80 to 90	75 to 95	65 to 95	65 to 95
ram	$<$	KB	maximise	4 to 128	16 to 128	2 to 32	16 to 256	16 to 256
response time	$<$	ms	maximise	1000 to 5000	2500 to 10000	1000 to 5000	2500 to 10000	1000 to 5000

Table 5.4: Summarised service strategies for the map service providers.

The general provider strategy is to minimise resource usage while maximising the price of the services provided. Provider strategies for individual qualities are typically opposite to the consumer strategy. However, it is possible for a provider and consumer to share a common strategy for achieving certain QoS aims, such as completing a transaction within a certain timeframe.

5.4 Initial Service Provider Selection

The hypothesis for the initial service provider selection is that given a number of functionally-equivalent service providers with differing non-functional qualities and reputation, the framework will select the most acceptable provider according to the consumer strategy.

Consumer brokers select service providers based on the service proposals they secure from provider brokers, and the current provider reputation information. Reputation provides additional criteria for the service negotiation and selection processes. The reputation threshold specified in the consumer strategy also enables a consumer broker to limit negotiation to a reputable subset of the available providers for each service; if a provider's reputation is below the acceptable reputation threshold, the consumer broker will not negotiate with the provider's broker.

During negotiation with a provider broker, the consumer broker combines the acceptability of the service proposal received from the provider broker, with the available reputation information for the provider, to determine the overall acceptability of a provider. The overall provider acceptability is calculated by applying proposal and reputation weights specified in the consumer strategy, to the provider's service proposal and reputation, and combining the result.

5.4.1 Initial Service Provider Reputation

The initial overall map service provider reputation information stored in the reputation system is shown in Figure 5.2. Providers have an overall reputation, based on previous ratings made by service consumers to the reputation system provided by the quality assurance framework. Providers also have separate reputations for the individual service qualities they offer, e.g. *response time*.

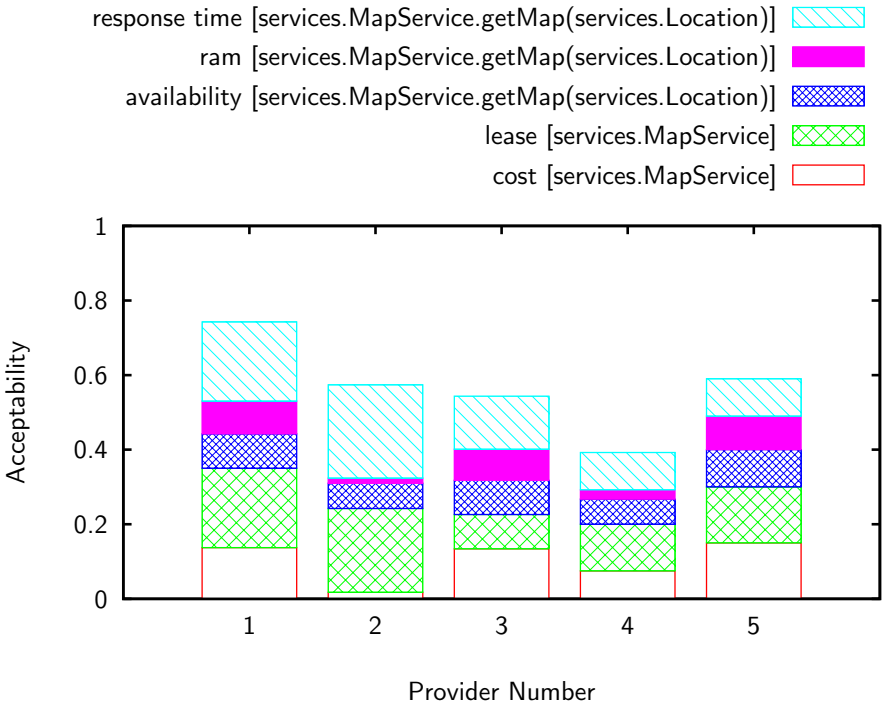


Figure 5.2: Initial map service provider reputation.

The historical ratings received for each map service provider are shown in Figure 5.3. These ratings have been made over a period of time, and are combined together to form the overall reputation of each provider shown in Figure 5.2.

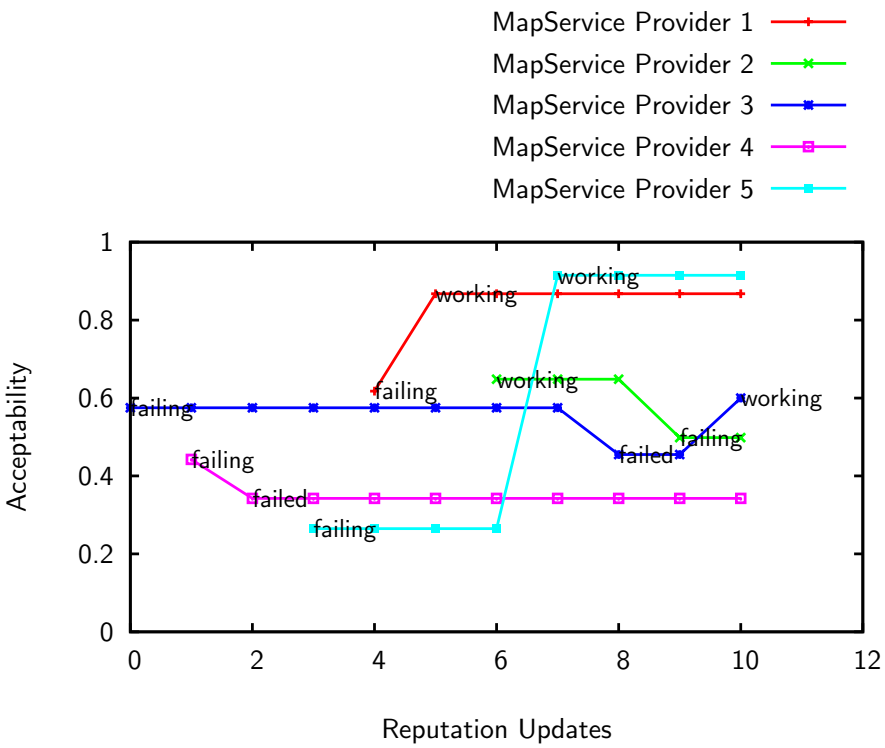


Figure 5.3: Initial historical map service provider ratings.

5.4.2 Initial Service Negotiation

Service brokers discover each other via the brokerage system provided by the quality assurance framework. The consumer broker establishes a negotiation session with the broker of each available reputable provider, for each service type required by the consumer. The consumer specifies a maximum number of negotiation partners, to avoid lengthy negotiations in the event of a multitude of providers.

Provider brokers query the reputation system before negotiating with a consumer broker, to see how the consumer may have rated the provider broker's client in the past. The reputation threshold in the provider strategy is used by the provider broker, to limit negotiation to consumers which have provided the provider with a rating it considers fair, and to those consumers which have not previously used services from the provider.

Negotiation sessions may be unilaterally terminated by either party if an acceptable service proposal is not reached during negotiation. The final service proposal may be unilaterally rejected if another party offers a more acceptable proposal, or has better reputation. Otherwise, the final service proposal is accepted by both parties. The negotiation model is discussed in more detail in Section 4.2.3.

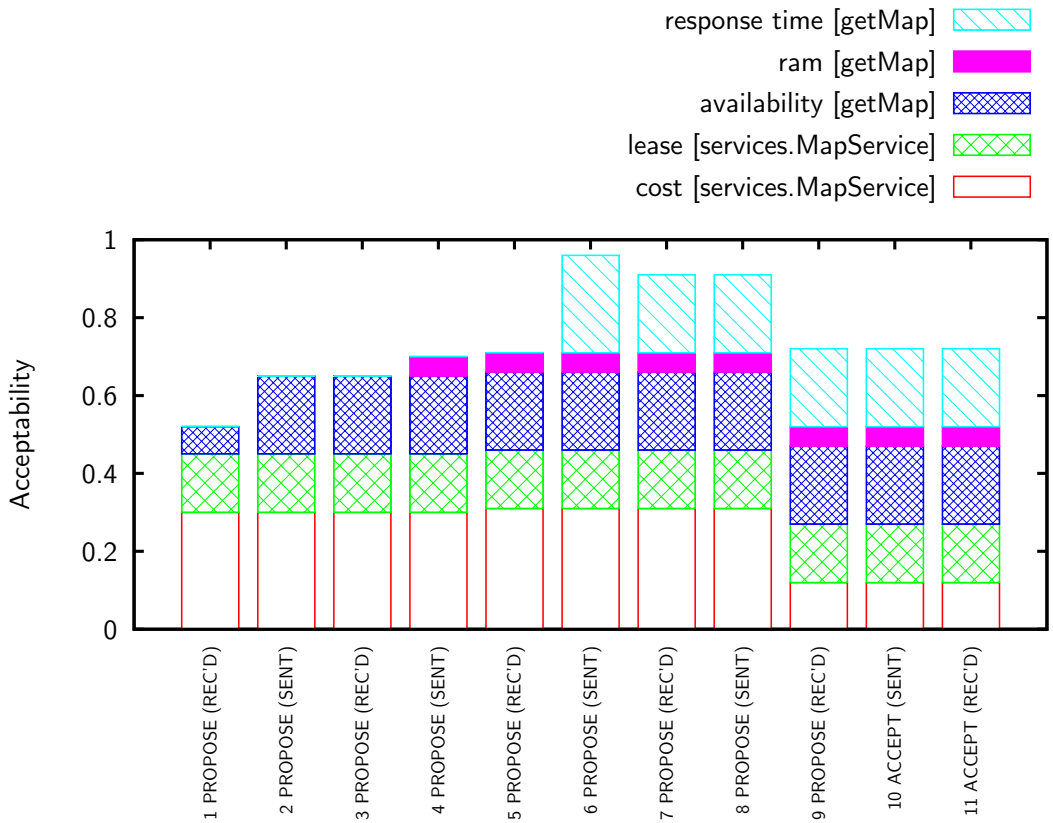


Figure 5.4: Initial negotiation with broker of map service provider 1. Acknowledgement messages (ACKs) are omitted for clarity.

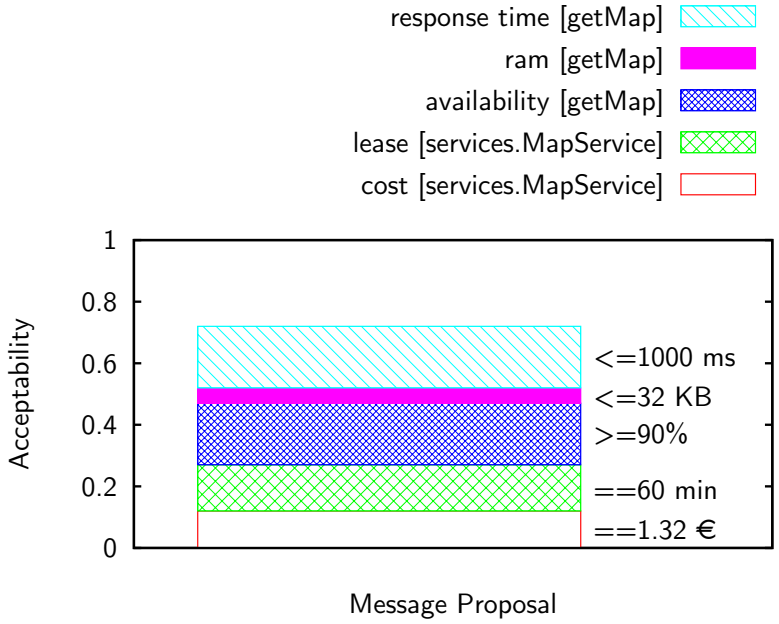
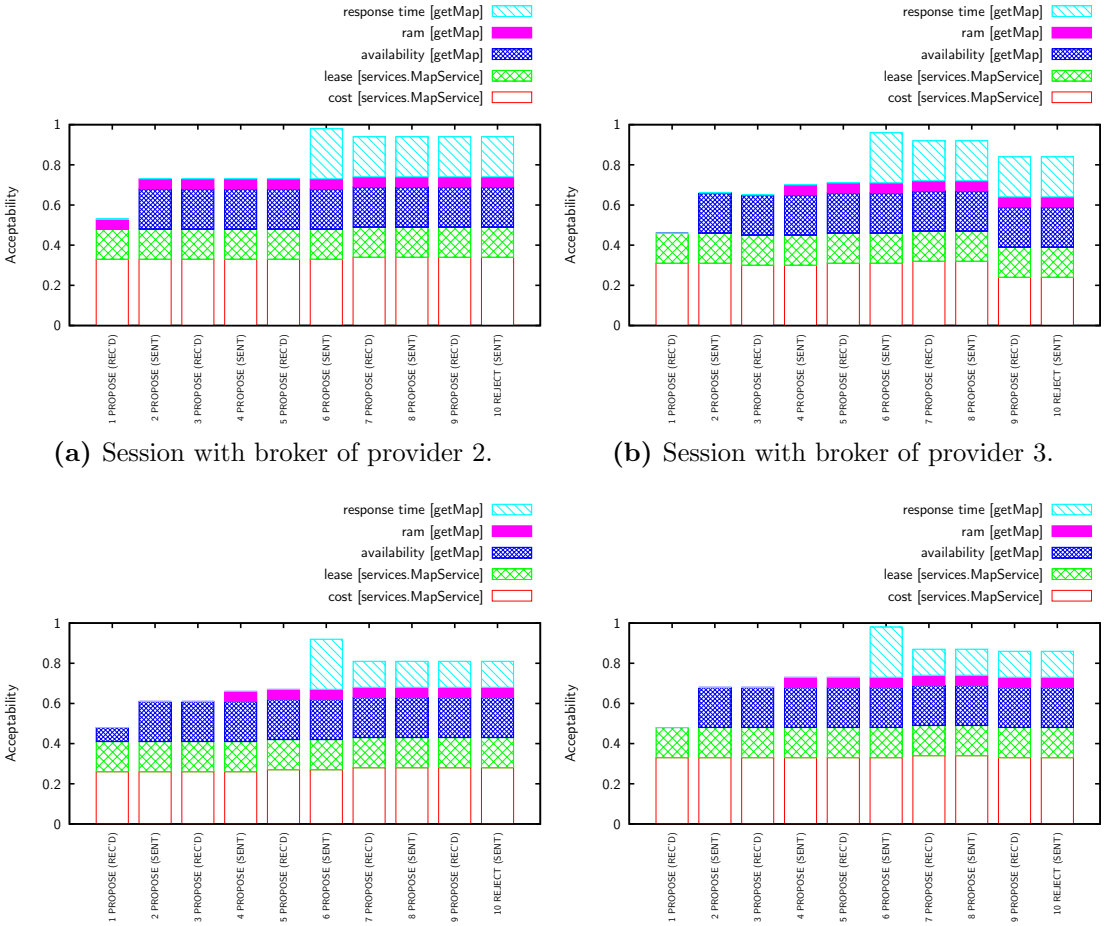


Figure 5.5: Initial accepted map service proposal, corresponding to the final *accept* messages in the negotiation session shown in Figure 5.4.



(a) Session with broker of provider 2.

(b) Session with broker of provider 3.

(c) Session with broker of provider 4.

(d) Session with broker of provider 5.

Figure 5.6: Initial negotiation sessions with other map service provider brokers.

The negotiation session between the consumer broker and the broker of map service provider 1, results in an *accept* decision, as shown in Figure 5.4. The acceptability and values of the accepted service proposal are shown in Figure 5.5. The *accept* proposal shown in Figure 5.5 corresponds to the final proposal in the session shown in Figure 5.4, and forms the basis of the SLA between the consumer and provider of the map service.

Figure 5.6 shows the remaining negotiation sessions with the other map service providers, which result in a *reject* from the consumer broker. Although the other map service providers offer more acceptable service proposals than map service provider 1, their reputation is poorer, as shown in Figure 5.2, leading to rejection.

These negotiation session examples use the bargaining negotiation model and utility-based acceptability formulas described in Chapter 4.

5.4.3 Initial Service Provider Selection Results

A comparison of the map service proposals negotiated with each provider broker is shown in Figure 5.7.

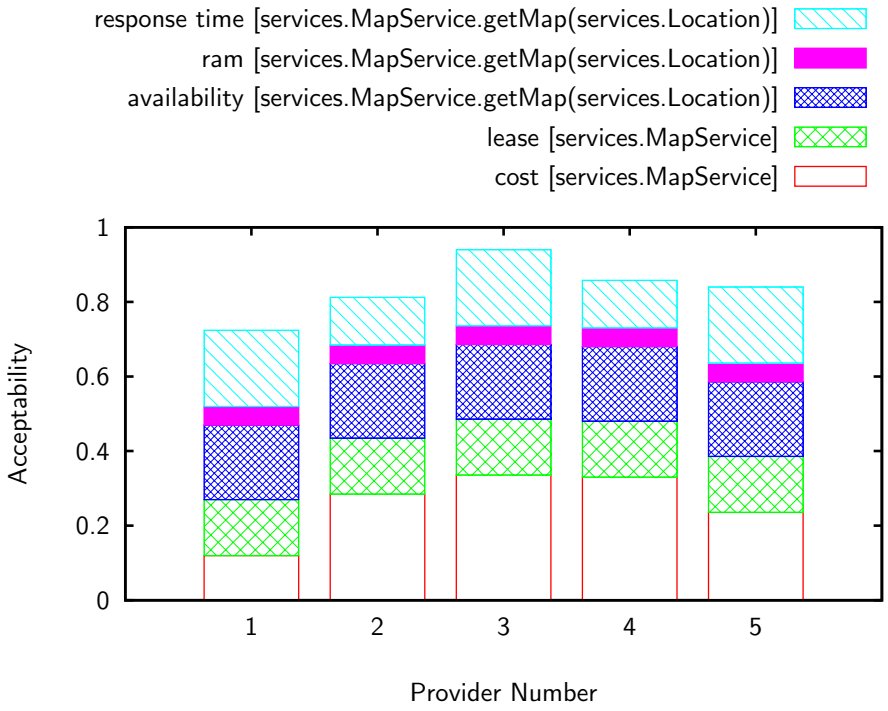


Figure 5.7: Initial map service proposal acceptability.

Using weights specified in the consumer strategy, the consumer broker combines each service proposal with the reputation of the proposal's provider, to determine the overall acceptability of each map service provider to the consumer. These weights give map service provider 1 the highest acceptability to the consumer. The overall acceptability of each map service provider is shown in Figure 5.8.

The selection of the map service provider is not made until acceptable service proposals are reached for each service required for the navigation composition. Each service is individually-weighted in terms of its importance to the consumer, and the consumer broker computes and ranks the acceptability of each possible composition. Once the most acceptable composition is determined, the providers of the composition proposals are accepted, and the remaining providers are rejected.

The overall acceptability of the navigation composition is shown in Figure 5.9, which represents the combined acceptability of the accepted service proposals, and the reputation of the service providers.

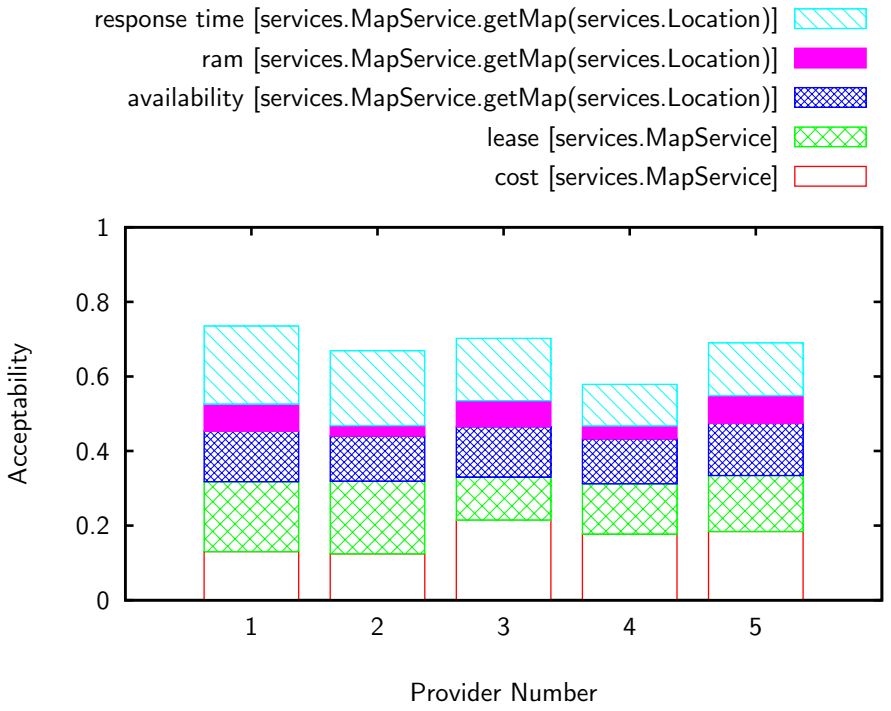


Figure 5.8: Initial overall map service provider acceptability. This represents the combined acceptability of the provider service proposals in Figure 5.7, with the provider reputation in Figure 5.2.

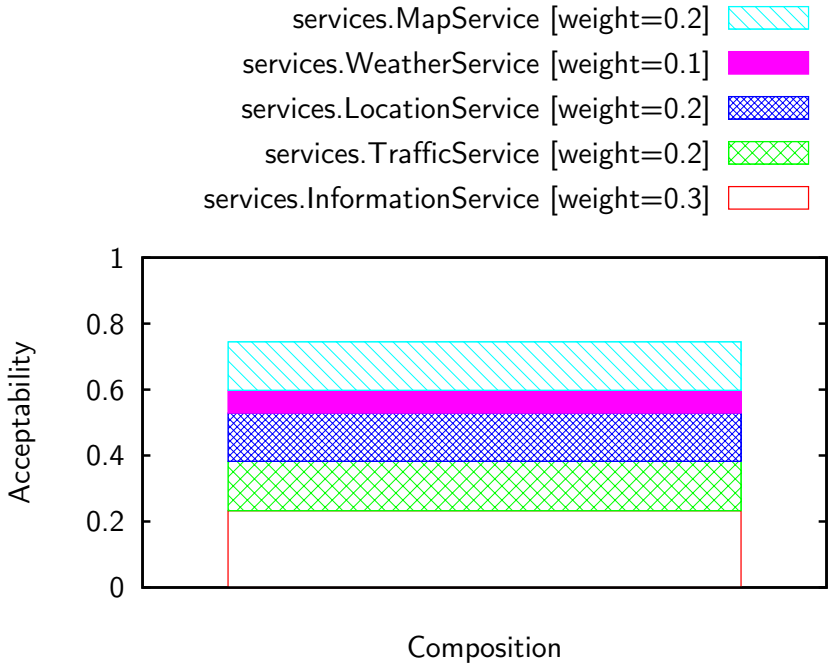


Figure 5.9: Initial overall navigation composition acceptability, inclusive of service proposal acceptability and provider reputation.

This initial provider selection experiment has failed to disprove the hypothesis that given a number of functionally-equivalent service providers with differing non-functional qualities and reputation, the framework will select the most acceptable provider according to the consumer strategy.

5.5 Service Monitoring

After service negotiation is complete, the consumer broker provides the consumer with invokable references to the services it has requested. The consumer supplies a copy of these references to the monitoring system, which assembles monitors for observing, auditing and forecasting runtime service performance. The consumer also supplies the monitoring system with the SLA for each service to be monitored, and a monitoring specification describing which service qualities the consumer requires to be measured, audited and forecast.

The passive monitoring approach provided by the framework reference implementation is used for the evaluation experiments. Service monitors collect pre-invocation and post-invocation measurement data each time a service is invoked by the consumer. The measurement data is fed through an auditor component, which compares the measured service performance with the SLA negotiated between the brokers of the service consumer and provider. The auditor additionally makes use of a forecaster component, for estimating future service quality based on past service performance, and to observe trends in service quality. The forecasting approach used for the experiments is an exponential moving average (EMA) model, which assigns more importance to recent service performance observations.

5.6 Regular Service Performance

The hypothesis for regular service performance is that given no occurrences of service failure, the framework will not need to renegotiate or substitute services for the consumer.

The observed and forecast service quality data received from the map service monitor during normal service invocation, i.e. without any QoS failures, is shown in Figure 5.10. The consumer has requested the monitor audits the runtime *availability*, *ram usage* and *response time* qualities. The remaining *lease* and *cost* qualities are static properties of the SLA, and do not require monitoring.

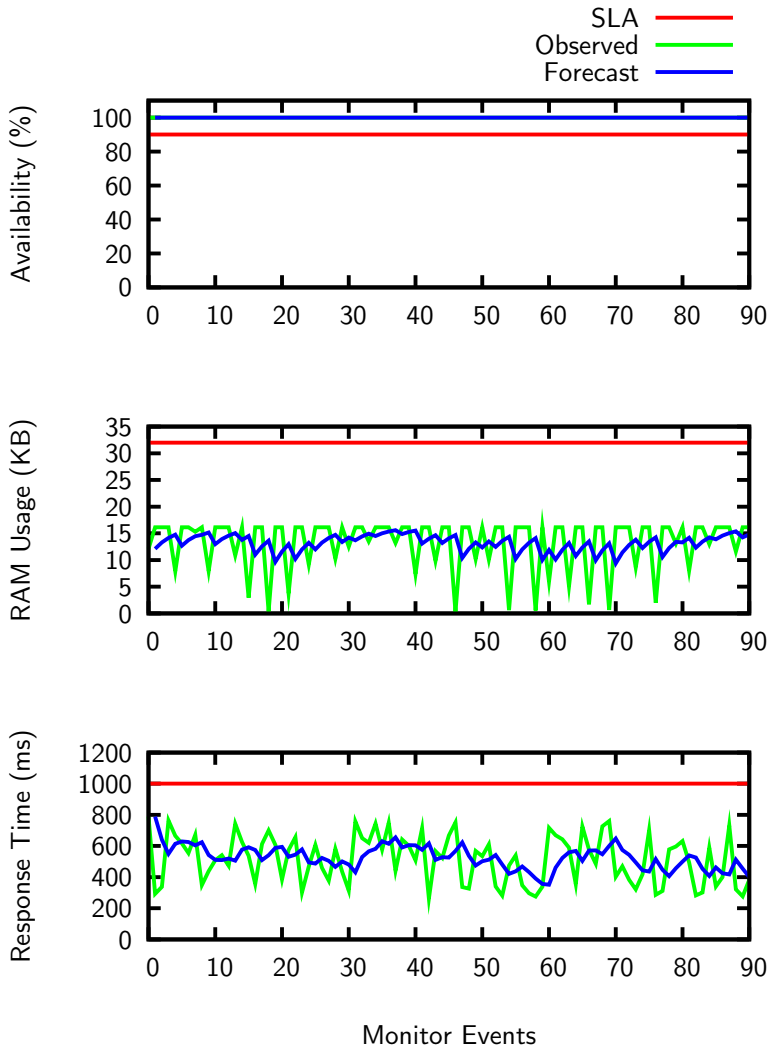


Figure 5.10: Regular service monitor events.

Under regular service performance, the monitored service *availability* remains at 100%, which exceeds the required SLA objective of ≥85% for that quality. The amount of *ram* required to process the map service responses fluctuates between approximately 1 KB and 16 KB, but remains well within the SLA objective of ≤32 KB. The *response time* quality fluctuates between approximately 200 ms and 800 ms, and also remains within the SLA objective of ≤1000 ms.

The monitoring data is translated into the runtime acceptability of individual service and composition invocations, as shown in Figure 5.11 and Figure 5.12.

The acceptability of each quality is calculated by applying the weight assigned to the quality from the consumer strategy. The monitored *availability* at 100% is shown with an acceptability of 0.4, which corresponds to its weight in the consumer

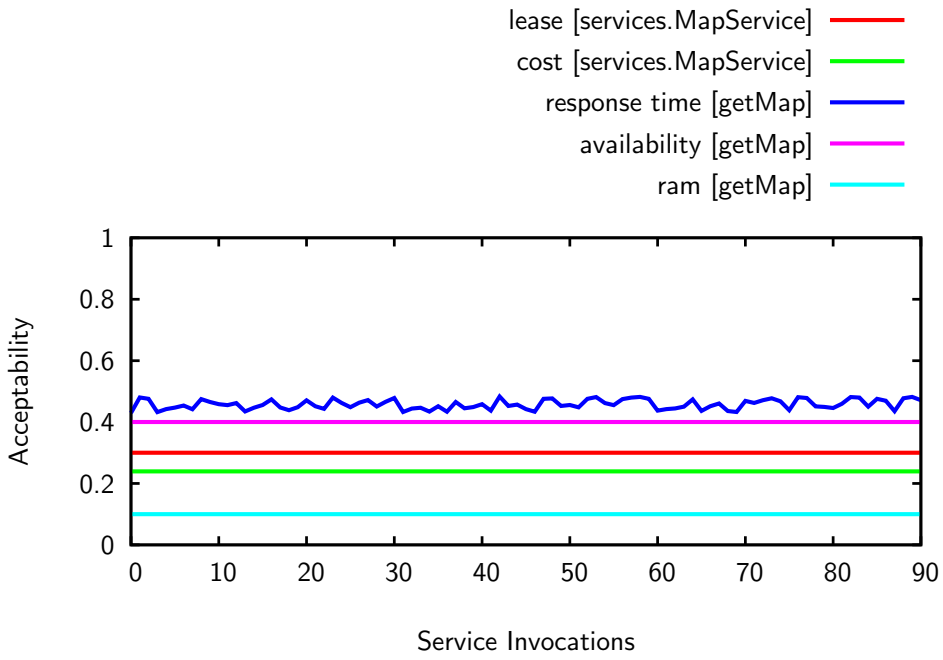


Figure 5.11: Regular service invocations.

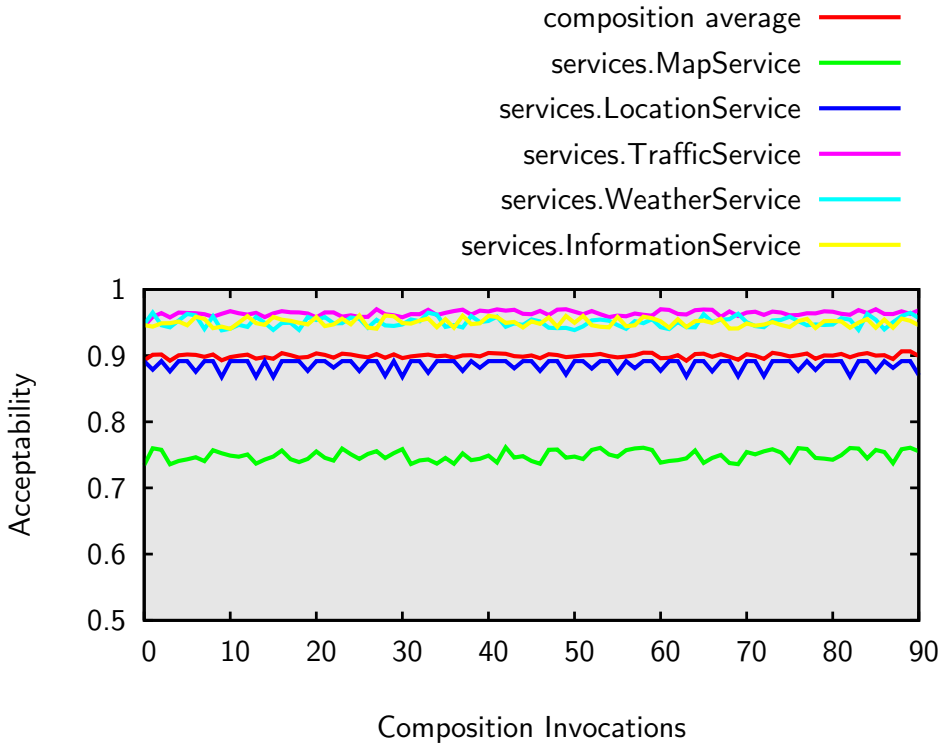


Figure 5.12: Regular composition invocations.

strategy, as shown in Table 5.3. The acceptability of the *response time* quality fluctuates in line with the measurements recorded by the service monitor, which are greater than the consumer ideal of ≤100 ms, but within the SLA objective. The *ram* quality acceptability is constant at 0.1, corresponding to its weight in the consumer strategy, as *ram* measurements are below both the consumer ideal of ≤32 KB and below the SLA objective. The service *lease* and *cost* qualities are not monitored, so the acceptability is based on the SLA values for these qualities.

This regular service performance experiment has failed to disprove the hypothesis that given no occurrences of service failure, the framework will not need to renegotiate or substitute services for the consumer.

5.7 Recurring Service Failure

The hypothesis for a recurring service failure is that given the availability of an acceptable service provider, the framework will renegotiate and substitute the service for the consumer as needed.

The recurring service failure scenario uses the same automobile navigation system consumer strategy and map service providers as the previous experiment. The observed and forecast service quality data from the map service monitor are shown for the recurring service failure scenario in Figure 5.13.

The SLA violations observed by the service monitor are visible on the *response time* plot shown in Figure 5.13. The initial SLA objective of ≤1000 ms for the *response time* quality is violated on the 30th occasion the consumer invokes the map service. The violation is so severe that after renegotiating with the map service provider, the first provider is no longer the most acceptable provider available for the service. The consumer broker consequently switches the consumer to a second map service provider. The second provider also fails during the 60th service invocation, and the consumer is switched to a third provider which maintains acceptable QoS until the consumer unleases the service. The consumer broker is able to maintain an SLA value of ≤1000 ms when transitioning between each map service provider, but each subsequent provider leaves the consumer with a noticeable decrease in overall responsiveness.

The affect of the service failures on the runtime acceptability of the service qualities is shown in Figure 5.14. The acceptability of the *cost* quality increases with the switch to the second provider, and decreases again with the third provider.

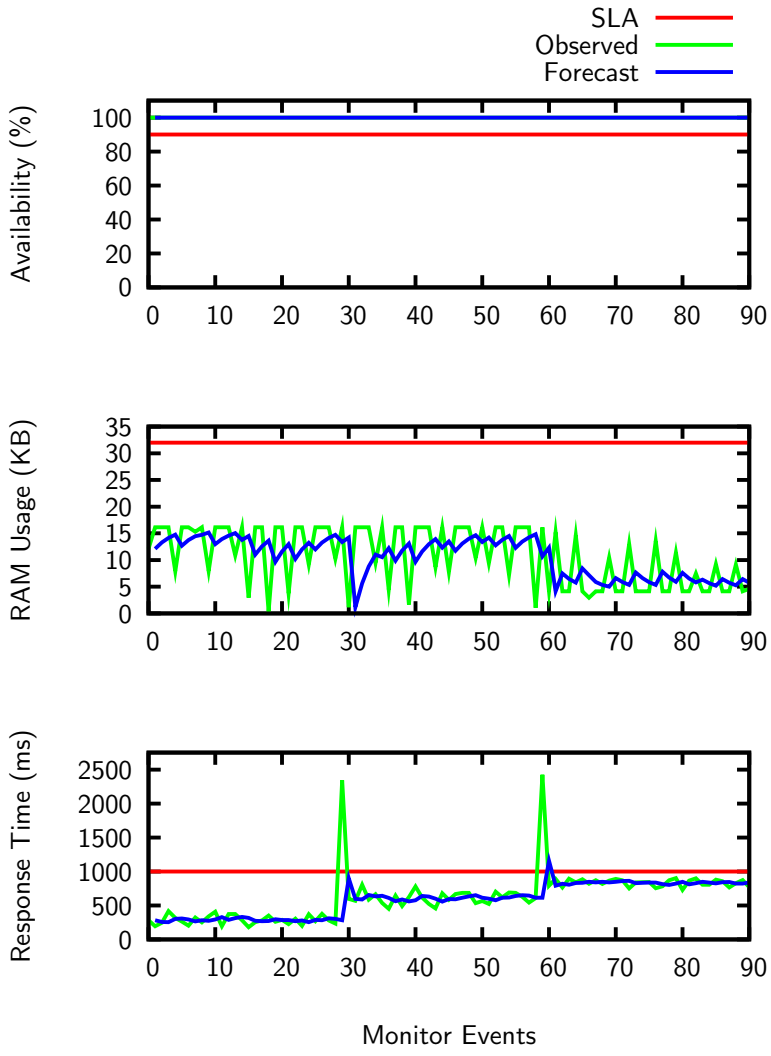


Figure 5.13: Recurring map service response time failure.

The average acceptability of the *response time* quality decreases as the consumer is switched between subsequent providers.

After an SLA violation, and before attempting to renegotiate the problematic service, the service consumer updates the reputation system with a rating for the service provided by the current problematic map service provider. Figure 5.15 shows the additional *failed* rating provided by the consumer for map service provider 1. Figure 5.16 shows how the new rating decreases the overall reputation of the provider. The updated reputation also affects the overall acceptability of map service provider 1, as shown in Figure 5.17.

Figure 5.14: Recurring map service failures and switching providers.

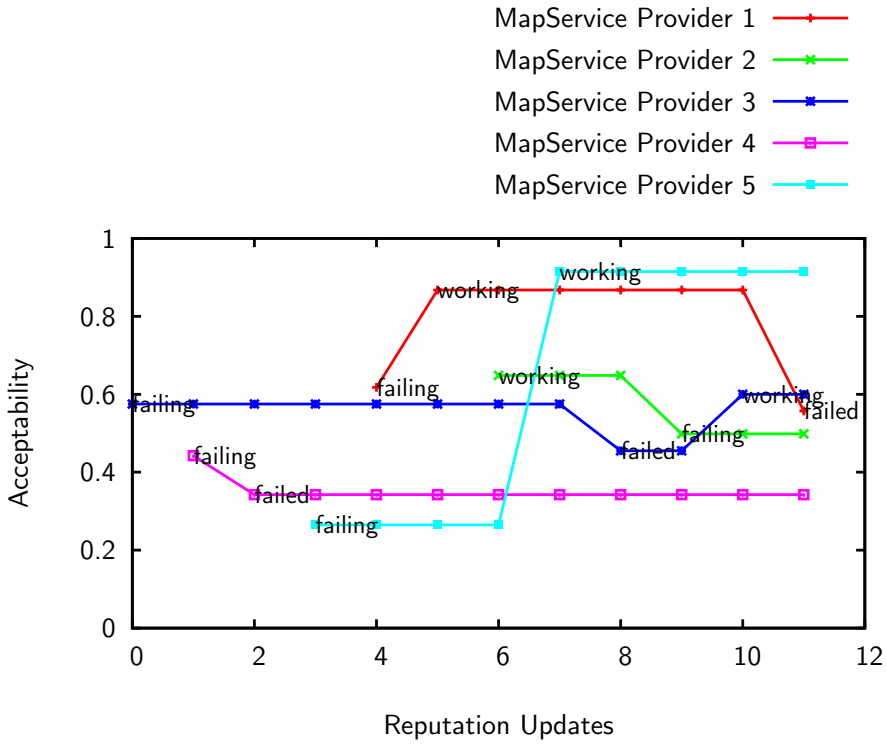


Figure 5.15: Updated map service provider ratings.

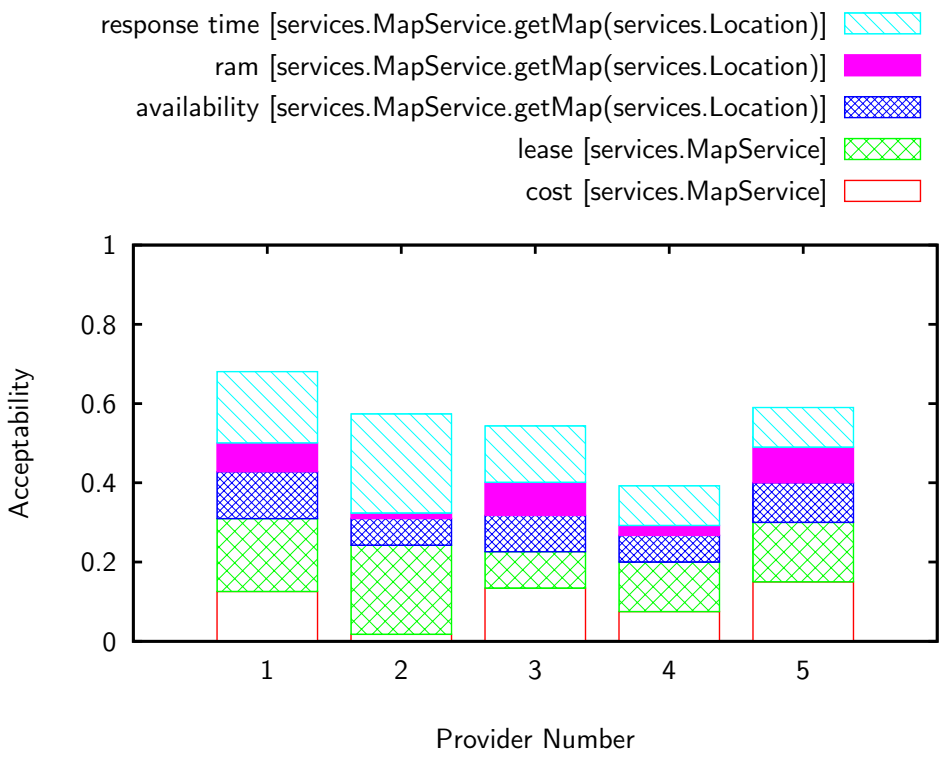


Figure 5.16: Updated map service provider reputation.

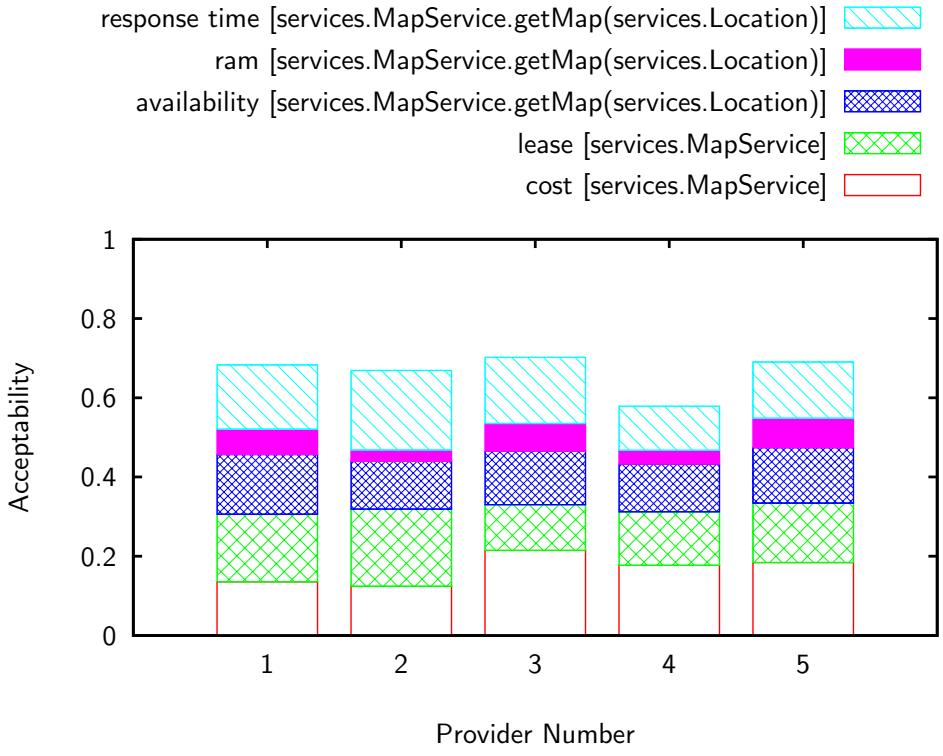


Figure 5.17: Updated overall map service provider acceptability.

The renegotiation with map service provider 1 is shown in Figure 5.18. While the renegotiated service proposal is acceptable, and offers an overall improvement in acceptability over the QoS measured by the map service monitor, when combined with the map service provider reputation, the renegotiated service proposal is no longer the most acceptable from the set of available map service providers.

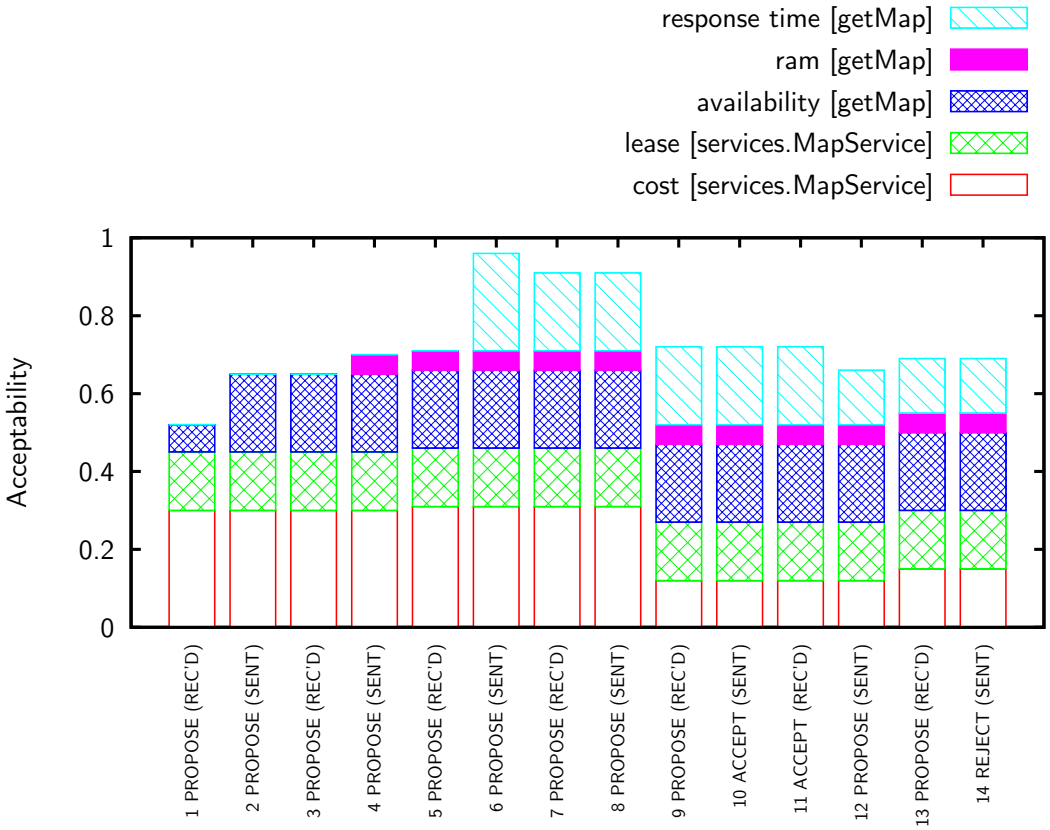


Figure 5.18: Renegotiation with map service provider 1.

The consumer broker switches the consumer to a second provider, map service provider 3, which now offers the most acceptable service. When the second map service provider also later violates the SLA *response time* objective, the renegotiation process is repeated. As with the first provider, the severity of the SLA violation in combination with the provider's decrease in reputation leads the consumer broker to switch the consumer to a third provider, map service provider 5. The third map service provider continues to provide acceptable QoS without SLA violations, until the consumer unleases the navigation services.

Figure 5.19 shows the overall acceptability of the navigation services, and the affect of the map service failures and provider switches on the overall composition.

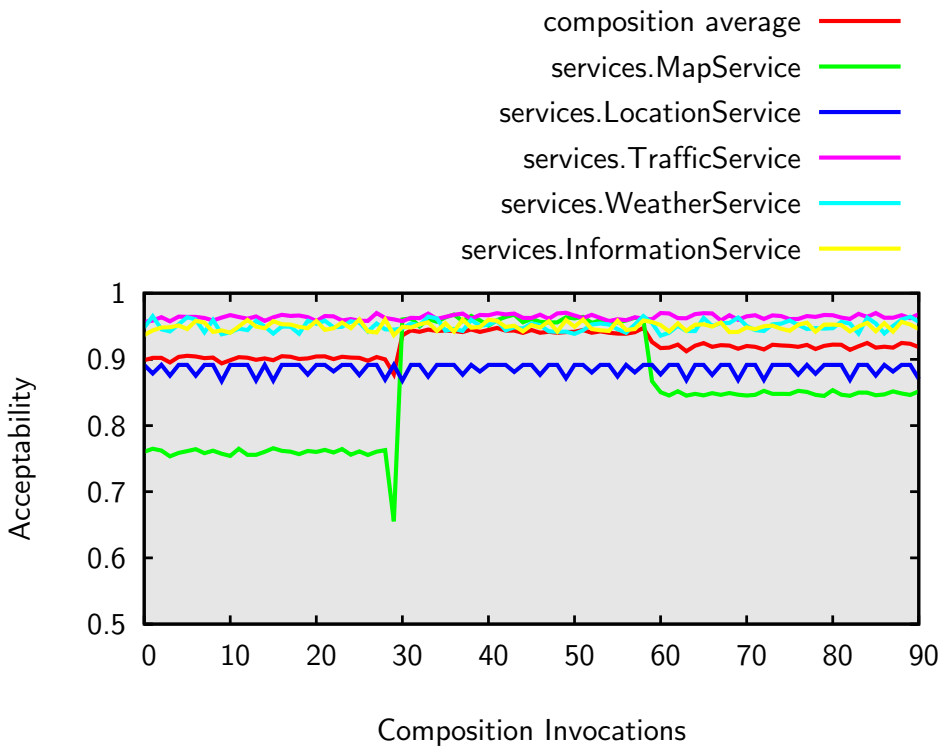


Figure 5.19: Recurring map service failure impact on composition.

This experiment has failed to disprove the hypothesis that given a recurring service failure and the availability of an acceptable service provider, the framework will renegotiate and substitute the service for the consumer as needed.

5.8 Handling SLA Violations

When an SLA objective for a quality such as *response time* is violated, the consumer assumes that the monitored quality level which caused the violation is the best the provider can guarantee. If the monitored quality level is still within the acceptable limits specified by the consumer strategy, the consumer broker will attempt to renegotiate the service with the provider's broker, using the monitored quality level as the starting point. During renegotiation, the consumer broker expects the provider's broker to revise and improve other service qualities, such as *cost*, so as to raise the overall acceptability of the provider to the consumer.

5.8.1 Severe Violations

The hypothesis in the event of a severe SLA violation is that the current service provider will not remain the most acceptable available provider of the service, if its reputation has been damaged badly enough and it does not make enough improvement to the overall acceptability of the service during renegotiation.

The SLA violations discussed in Section 5.7 are so severe that the consumer broker switches the service consumer to another provider of the same service, as shown in Figure 5.14. Due to the decrease in a provider's reputation after a severe SLA violation, the provider's broker must substantially improve the service proposal during renegotiation, in order to retain the highest acceptability out of the available providers for the same type of service. In the recurring failure scenario presented in Section 5.7, the provider brokers, constrained by the service strategies of the providers they represent, are unable to sufficiently improve the service proposal to maintain the highest provider acceptability. As such, the recurring failure scenario has failed to disprove the severe SLA violation hypothesis put forward at the beginning of this section.

5.8.2 Moderate Violations

The hypothesis in the event of a moderate SLA violation is that the current service provider will remain the most acceptable available provider of the service, if its reputation has not been damaged badly enough and it makes enough improvement to the overall acceptability of the service during renegotiation.

When an SLA violation is less severe, the consumer and provider broker can often renegotiate terms of service that preserve the provider's acceptability to the consumer, and enable the provider to retain the consumer's business. This section discusses two alternative scenarios to the recurring failure scenario, which this time feature more moderate SLA violations.

Figure 5.20 shows a more gradual decrease in the acceptability of the *response time* quality, until the SLA objective for the quality is violated. The gradual decrease leads to a moderate SLA violation, and after renegotiation the current map service provider still remains the most acceptable available to the consumer. In response to the violation, the provider broker reduces the service *cost*, increasing the overall acceptability of the provider to the consumer. In this particular example, the service provider manages to stabilise the provided *response time* after the

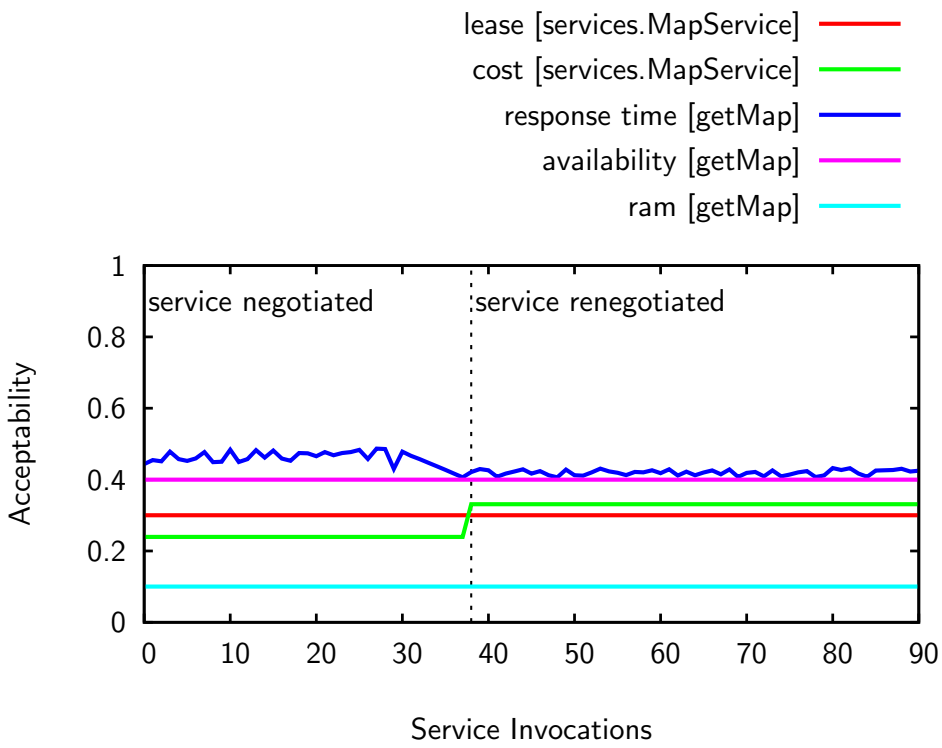


Figure 5.20: Map service failure, followed by provider QoS stabilisation.

service is renegotiated, and prevents further SLA violations.

Another moderate SLA violation is shown in Figure 5.21. On this occasion, the *response time* quality becomes more erratic and the *response time* SLA objective is violated. Renegotiation with the map service provider succeeds, but the renegotiated SLA is immediately violated by the provider a second time. The second violation causes the provider's reputation to fall so low that it no longer offers the most acceptable map service to the consumer. The consumer broker consequently switches the consumer to another provider, which offers a greater acceptability.

Both of the experiments shown in Figure 5.20 and Figure 5.21, have failed to disprove the moderate SLA violation hypothesis put forward at the beginning of this section.

5.9 Service Outage

The hypothesis for an SLA violation caused by a service outage is that given the availability of an acceptable service provider, the framework will renegotiate or substitute the problematic service as required.

Figure 5.21: Map service failure, followed by another failure after renegotiation.

For the service outage scenario, the service concerned is the weather service element of the navigation application, and the consumer is a mobile phone device. The consumer secures an SLA objective of ≥95% for the service *availability* quality. The first four weather service invocations complete successfully, but the consumer experiences a service outage the fifth time it invokes the weather service. The service outage causes the monitored service *availability* to drop to 80%, which is a violation of the negotiated SLA objective.

Figure 5.22 shows the monitor data capturing the service outage, and the drop in the monitored *availability* quality which causes the SLA violation. Due to the service outage, the monitor is unable to measure and audit the service *response time* quality during the fifth service invocation. Unlike the map service, the weather service does not have an negotiated SLA objective for the *ram* quality.

The affect of the service outage on the runtime acceptability of the weather service qualities is shown in Figure 5.23. Due to the severity of the SLA violation, the decrease in the weather service provider's reputation causes the consumer broker to switch the consumer to another provider. The second weather service provider offers a *response time* improvement, but its *cost* is moderately less acceptable

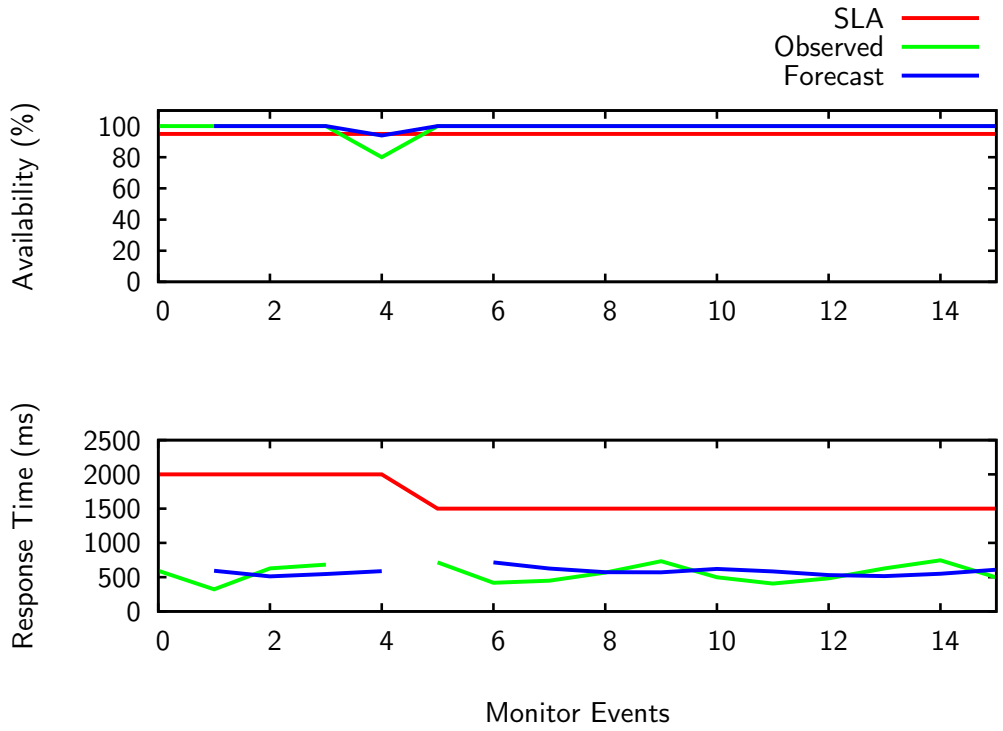


Figure 5.22: Weather service outage and availability failure.

to the consumer. Figure 5.24 shows the runtime acceptability of the navigation services, and the affect of the weather service outage on the acceptability of the overall composition.

This experiment has failed to disprove the hypothesis that given an SLA violation caused by a service outage, and the availability of an acceptable service provider, the framework will renegotiate or substitute the problematic service as required.

5.10 Off-Peak and Peak Service Performance

The hypothesis for an SLA violation during a peak service usage period is that given the availability of an acceptable service provider, the framework will renegotiate or substitute the problematic service as required.

During peak usage periods, service invocations may becomes less responsive due to the increased load on service providers. The peak usage scenario involves the location service element of the navigation application, and the consumer is an internet tablet device.

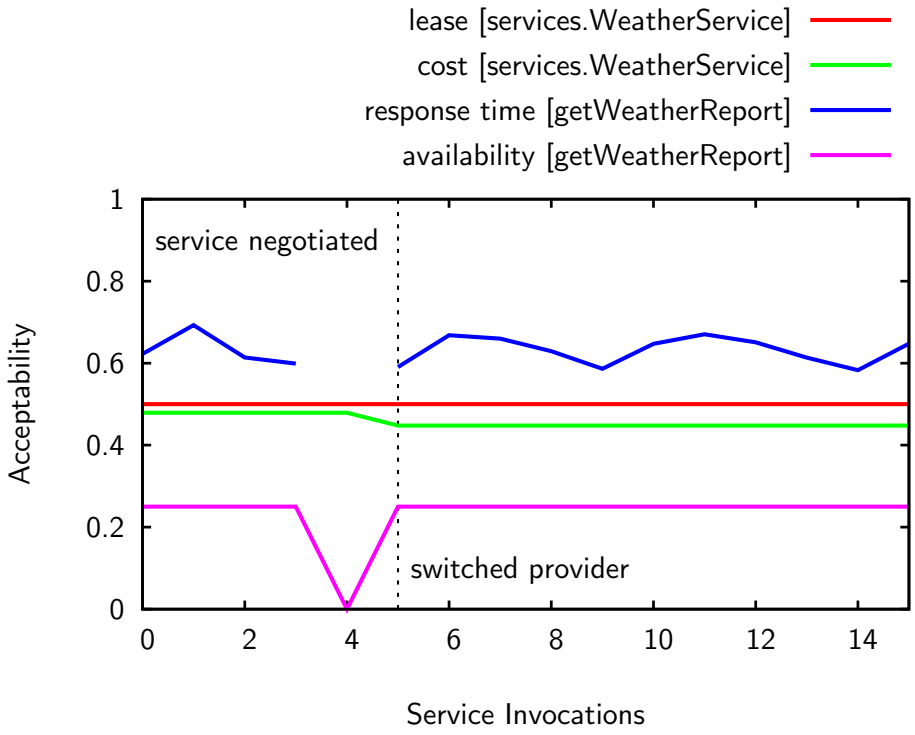


Figure 5.23: Weather service outage and switching provider.

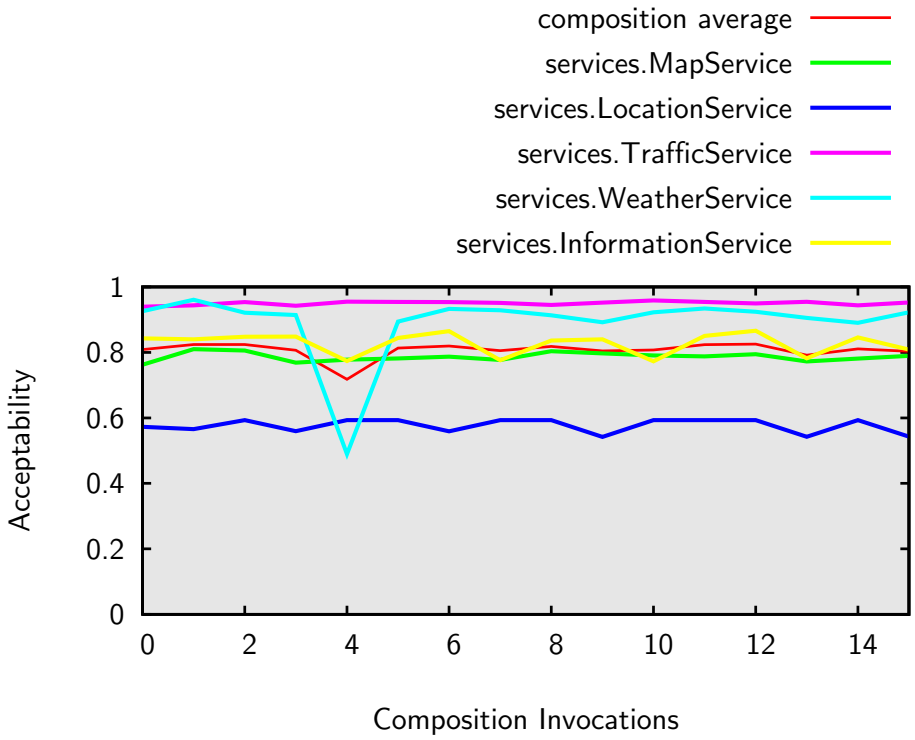


Figure 5.24: Weather service outage impact on navigation composition.

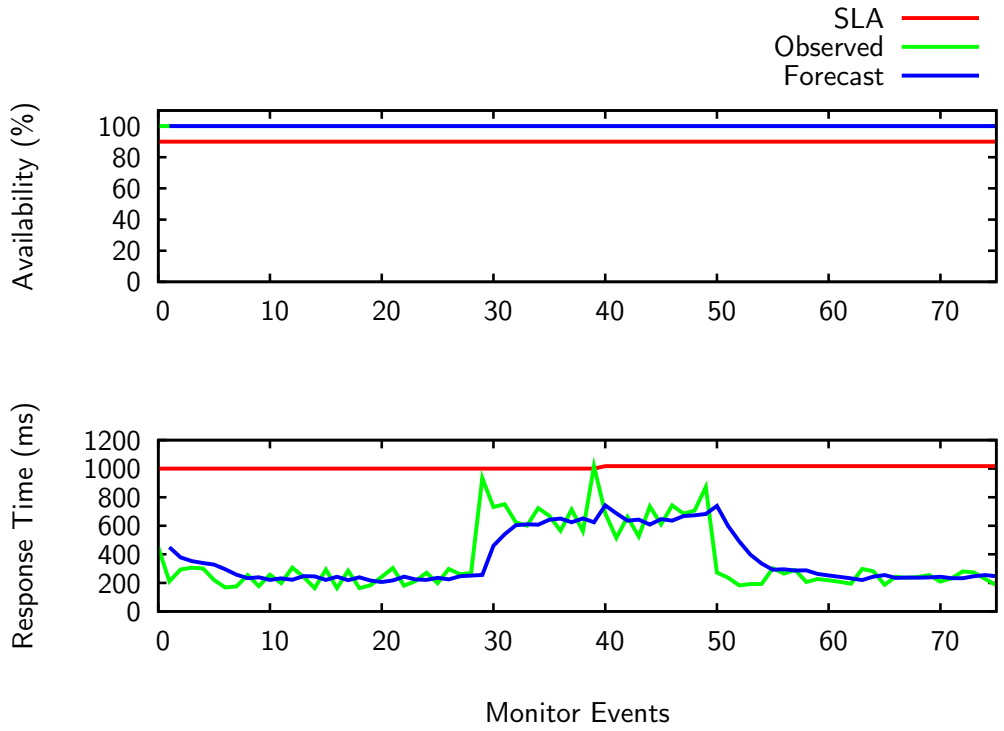


Figure 5.25: Location service response time failure during peak usage period.

The consumer secures an SLA objective of ≤1000 ms for the location service *response time* quality. Halfway through the peak usage period, during the 40th service invocation, the monitored service *response time* violates the SLA objective and the location service is renegotiated by the consumer broker. Figure 5.25 shows the monitor data for the peak service usage period, which is highlighted by the location service *response time* measurements, and the SLA violation.

The location service *response time* SLA violation is moderate, and the consumer and provider brokers successfully renegotiate the service. In response to the SLA violation, the service provider makes a slight reduction to the *cost* of the service, which increases the acceptability of its service to the consumer. The runtime acceptability of the location service qualities is shown in Figure 5.26. The affect of the peak service usage period on the runtime acceptability of the navigation services and the overall composition is shown in Figure 5.27.

This experiment has failed to disprove the hypothesis that given an SLA violation during a peak usage period, and the availability of an acceptable service provider, the framework will renegotiate or substitute the problematic service as required.

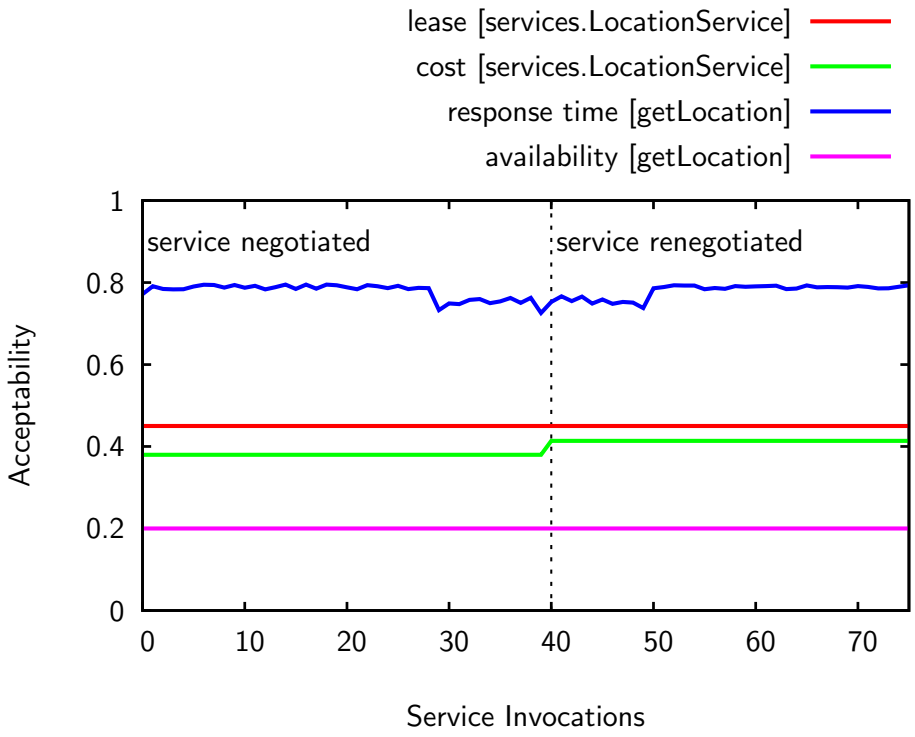


Figure 5.26: Location service failure and renegotiation during peak service usage period.

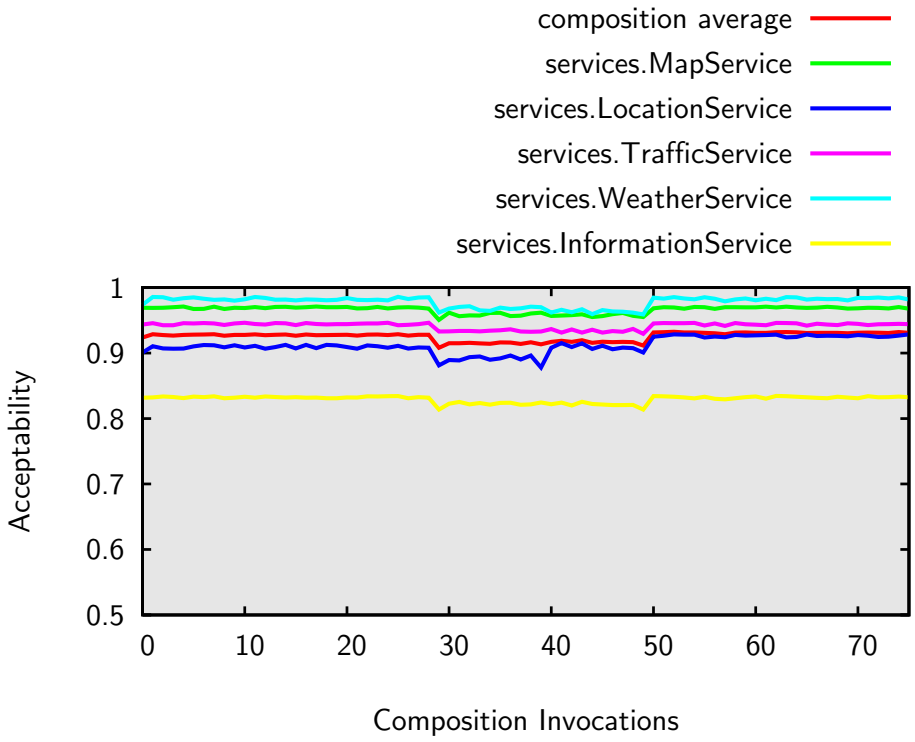


Figure 5.27: Peak service performance impact on navigation composition.

5.11 Consumer Competition

This scenario discusses competition between service consumers. The hypothesis is that it is possible for one consumer to be unable to secure an acceptable service proposal, as a direct result of another consumer competing for the resources of the same provider.

Each service provider has a finite amount of resources to support the SLA guarantees it provides to the consumers of its services. The provider broker negotiates service agreements with an awareness of the provider's current resources, so as not to make SLA guarantees for QoS levels the provider cannot provide in practice. The resource-aware negotiation requires the provider broker to allocate and deallocate portions of the provider's resources during the negotiation process.

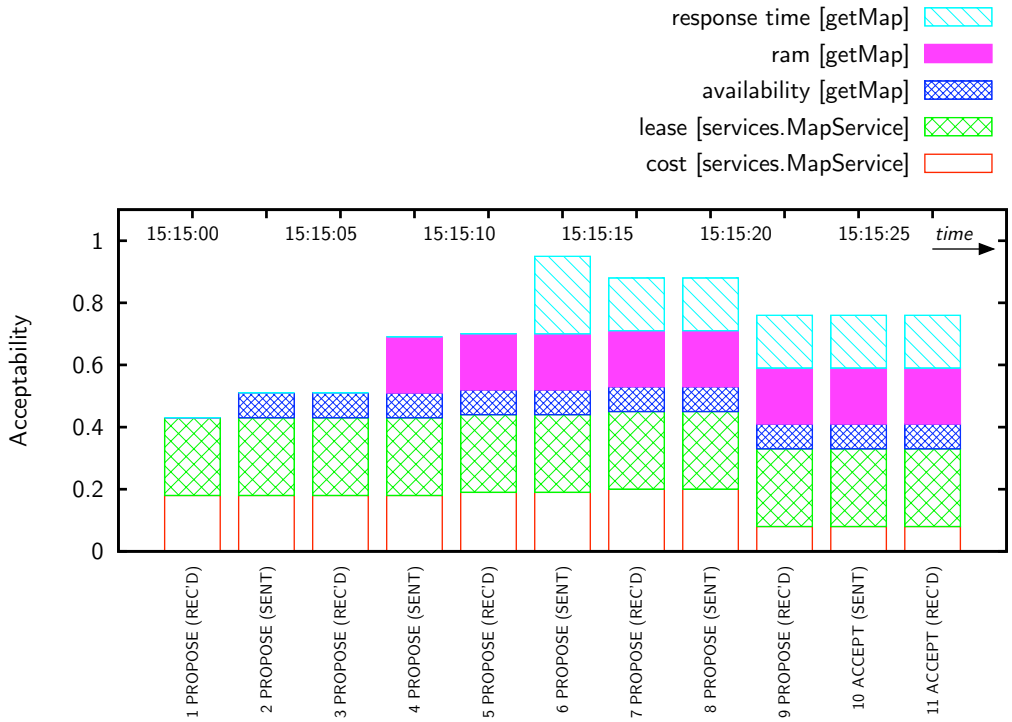
An example of consumer competition is shown in Figure 5.28, where a map service provider is unable to provide an automobile navigation system consumer with an acceptable *response time* guarantee. Shortly before negotiating with the automobile consumer, the provider broker allocates the majority of the provider's remaining *response time* resources to a mobile phone consumer. With few resources left to support a fast *response time*, the provider is unable to offer an acceptable proposal to the automobile consumer. Consequently, the broker for the automobile terminates the negotiation on receiving the unacceptable *response time* offer.

The consumer competition experiment shown in Figure 5.28 has failed to disprove the hypothesis put forward at the beginning of this section.

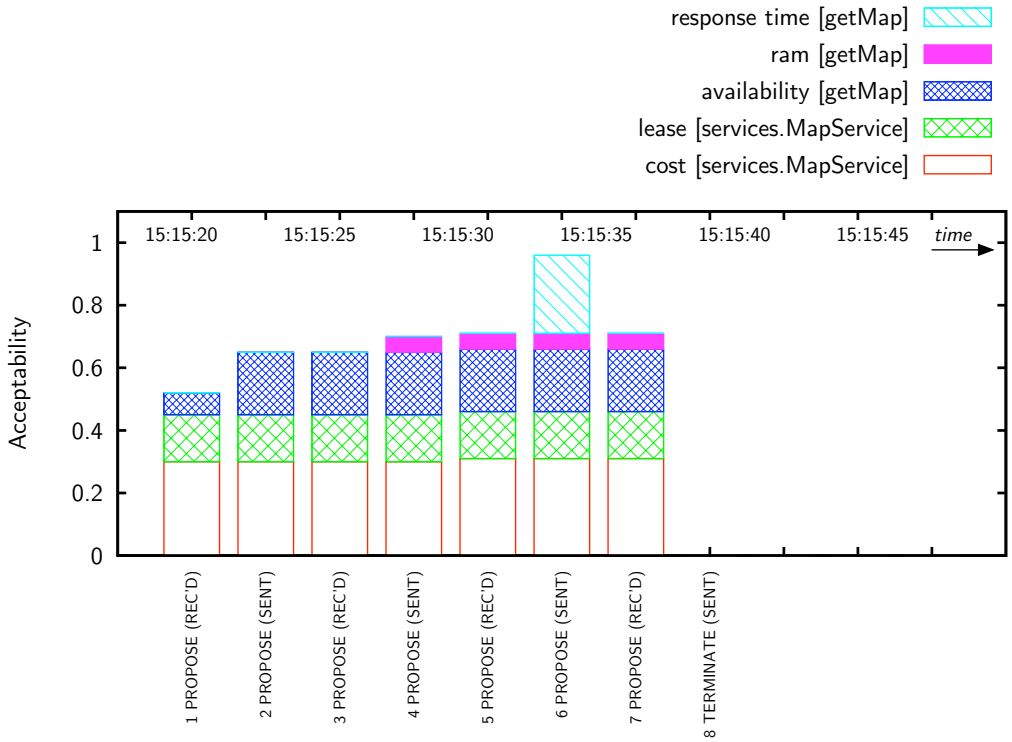
5.12 Framework Limitations

The experiments have so far shown how the quality assurance framework is able to maintain system quality in a variety of QoS scenarios. However, there are situations when the framework is unable to ensure system quality without some consumer intervention.

The first such situation, discussed in Section 5.12.1, arises when the consumer broker is unable to secure an acceptable service proposal for a particular service. If the service is part of a composition, the consumer broker cannot accept proposals for other services in the composition until the situation is resolved. The second situation, discussed in Section 5.12.2, arises when one or more of the framework quality assurance systems becomes unavailable. Depending on system availabil-



(a) Negotiation session with mobile phone consumer completes successfully.



(b) Negotiation session with automobile navigation system consumer fails.

Figure 5.28: Negotiation failure due to consumer competition.

ity, this may require the consumer to handle some parts of the quality assurance process directly, and reduces the overall effectiveness of the framework.

5.12.1 No Acceptable Services

The framework is unable to help the consumer secure a particular service, when no acceptable service proposals can be reached. Figure 5.29 shows an overview of map service provider proposals, in a scenario where the consumer broker is unable to secure a service proposal with a *cost* that is acceptable to the consumer.

Figure 5.29: The consumer broker is unable to secure a map service proposal with a *cost* that is acceptable to the consumer.

The map service is a required element of the navigation composition. The lack of an acceptable map service means the consumer broker cannot accept any proposal for the other four services in the composition, regardless of their acceptability, until an acceptable proposal for the the problematic map service is secured.

The lack of acceptable services is beyond the remit of the framework, as the framework cannot resolve the issue without intervention from the consumer. The consumer broker can only inform the consumer that negotiation for the map service has failed to achieve an acceptable proposal, and provide the consumer with the details of the most acceptable proposal achieved. The consumer must then have

the local ability to decide whether to revise its strategy in order to secure the map service, or go without the navigation application.

If the consumer decides to revise its strategy for the map service, it may increase the chance of securing an acceptable service proposal in two ways. Firstly, the consumer may increase its least acceptable value for the *cost* quality. Secondly, the consumer may reduce its requirements for the other service qualities, with the expectation that the map service providers will lower the value of the *cost* quality for the service accordingly. It is the responsibility of the consumer to perform any strategy revisions in a timely manner, so that existing negotiations between the consumer broker and provider brokers do not expire. Once a strategy revision is complete, the consumer supplies the revised strategy to its broker, which will then reopen negotiation with the map service provider brokers.

5.12.2 Framework System Unavailability

The quality assurance framework consists of three primary systems, each which may be provided as one or more services distributed over the network. It is possible that these systems may become unavailable at some period in time, e.g. due to network problems or high demand. This leads to the possibility of different framework configuration scenarios, as shown in Table 5.5, each providing various levels of support for the quality assurance process.

If the brokerage system is unavailable, the consumer may use the reputation system to secure services based purely on the reputation of the available service providers. Alternatively, the consumer may perform the service negotiation process directly itself, or instead wait for the brokerage system to become available again.

Once the consumer secures agreements for the services it requires, it can pass the agreements directly to the monitoring system. While the brokerage system is unavailable, the consumer will need to assume responsibility for resolving any SLA violations reported by the monitoring system. If the monitoring system then becomes unavailable, the consumer will also need to assume responsibility for measuring service performance. Alternatively, the consumer may choose to lose the ability to diagnose SLA violations, if it does not have the resources to dedicate to the monitoring process.

If either of the brokerage or monitoring systems become unavailable, the framework loses the ability to provide an end-to-end quality assurance solution for the consumer. Depending on the consumer's requirements, the absence of either sys-

Scenario	Brokerage System	Monitoring System	Reputation System	Provided Quality Assurance Support
1	×	×	×	no quality assurance support is provided to the service consumer or provider
2	×	×	✓	the service consumer and provider can query each other's reputation
3	×	✓	×	the service consumer can monitor services for SLA violations and service failures
4	✓	×	×	the service consumer and provider can negotiate service agreements
5	×	✓	✓	the service consumer and provider can query each other's reputation, and the consumer can monitor services for SLA violations and service failures
6	✓	×	✓	the service consumer and provider can negotiate service agreements with an awareness of each other's reputation
7	✓	✓	×	the service consumer and provider can negotiate service agreements, and renegotiate agreements in response to problems detected by the monitoring system
8	✓	✓	✓	the service consumer and provider can negotiate service agreements with an awareness of each other's reputation, and renegotiate agreements in response to problems detected by the monitoring system

Key: (✓) available, (×) not available

Table 5.5: Different framework configuration scenarios.

tem may require the consumer to expend its own resources in implementing the service negotiation and monitoring tasks.

Unlike the brokerage and monitoring systems, the reputation system is not a critical part of the quality assurance solution. Instead, the reputation system enhances the quality assurance solution by providing additional criteria for the service selection process. The reputation system also serves as an additional incentive for service providers to adhere to the SLAs negotiated with consumers.

5.13 Summary

This chapter identified a set of hypotheses and presented a series of experiments, to demonstrate and evaluate the key processes performed by the integrated quality assurance framework. The set of experiments were presented in the form of

several service-oriented QoS scenarios, and demonstrated the quality assurance processes employed by the framework to support the runtime quality of a service-oriented system. However, it is also important to acknowledge the limitations of the study (Kitchenham et al., 2002), such as the use of a simulated system and the use of predetermined data sets for each scenario.

The chapter then discussed the possible consequences of service competition between consumers, and presented scenarios which highlight some limitations of the quality assurance framework.

The following chapter concludes the thesis by comparing the research achievements against the research objectives outlined in the introduction chapter, and discusses further research directions which have arisen from the development of the framework.

Chapter 6

Conclusions

This chapter begins by comparing the research achievements of the integrated quality assurance framework, with the research objectives outlined in the introduction chapter. The chapter then continues with a discussion of future research directions, which have arisen during the development and evaluation of the framework. The chapter concludes the thesis with a summary of the research problem, the issues with existing research efforts, and the key contributions put forward in this thesis.

6.1 Objectives Revisited

This section discusses how the integrated quality assurance framework for service-oriented systems has addressed the research objectives stated in the thesis introduction. These research objectives are to provide the consumer with increased control over service quality, provide support for the expression of quality characteristics, provide a runtime solution for detecting and recovering from SLA violations, provide support for resource-restricted systems, and to provide customisation support for integrating different quality assurance techniques and facilitating experimentation. How each research objective has been addressed is now discussed:

- *Support for increased consumer control over service quality.* The framework components increase the consumer's control over service quality as follows:

    i. The framework's service ontology enables the service consumer to control the specification of service requirements and strategies for realising service requirements.

ii. The framework's brokerage system provides the consumer with the means to achieve SLAs for services, which are closer to meeting the consumer's requirements. The brokerage system also provides the consumer with renegotiation and recovery mechanisms, which are used to resolve SLA violations and improve overall QoS acceptability to the consumer.

iii. The framework's monitoring system provides an automated approach for detecting SLA violations, and provides the consumer with the feedback required to maintain the quality of a system. The forecasting performed by service monitors offers the ability to inform the consumer of potential service failures in advance.

iv. The framework's reputation system enables consumers to collaborate with one another, in order to share and receive advance notifications of unreliable service providers. The reputation system gives the consumer the means to reward service providers that provide services in accordance with SLAs. Service providers which violate SLAs are penalised accordingly.

- *Support for the expression of quality characteristics.* The framework's service ontology facilitates the description of service requirements, and strategies for realising these requirements. The ontology provides a common set of quality description terms for service consumers and providers, to reduce ambiguity and misunderstandings. The ontology also supports the specification of relationships between interdependent service qualities. The ontology is central to facilitating and automating quality assurance processes, such as service negotiation, selection, monitoring, and the rating of services.

- *A runtime solution for quality assurance.* The framework improves support for ensuring runtime quality in service-oriented systems, through the integration of service monitoring with renegotiation and recovery techniques. The framework provides the means to detect SLA violations at runtime, and the means to renegotiate and recover from service failures, in order to maintain an acceptable runtime system quality. The reputation system also provides service consumers with up-to-date information regarding the reliability of service providers.

- *Support for resource-restricted systems.* The framework supports resource-restricted systems by automatically handling complex quality assurance processes on behalf of service consumers and providers. This support is provided as follows:

    i. Resource-restricted systems benefit from having complex service negotiation processes handled on their behalf by the framework's brokerage system. Each service consumer and provider receives a unique service broker, for the purpose of securing acceptable SLAs. The service broker is supplied with service strategies which describe its client's requirements. At this point, the service broker acts autonomously and independently from its client. The tasks of the consumer broker typically involve the discovery of provider brokers of the services required by the consumer, concurrently negotiating SLAs with multiple provider brokers, and performing service selection decisions. The tasks of the provider broker typically involve processing negotiation requests, and performing service resource management.

    ii. Resource-restricted service consumers and providers benefit from the third-party service monitoring system provided by the framework, which independently audits the runtime performance of services. Instead of expending resources measuring and auditing the service quality received, the resources of the service consumer are free to be allocated to consuming services. The consumer is only required to respond to monitoring feedback which indicates a service problem. Providers are also able to focus their resources on the provision of services, rather than monitoring the service performance provided to consumers.

    iii. The framework's reputation system offers an efficient method for service consumers to collaborate and share experience concerning the performance of service providers. The centralised collation of historical provider reputation information performed by the reputation system saves the consumer time and resources, by facilitating the reputation information's efficient dissemination. The reputation system also supplies service providers with the feedback history of specific consumers, so providers can avoid consumers which they deem over-demanding or unreasonable with their feedback.

- *Support for solution integration and customisation.* The quality assurance framework is designed to integrate approaches from different service quality assurance domains. For example, the framework integrates techniques from the service negotiation and monitoring domains, to produce a recovery mechanism for handling SLA violations and service failures. The framework also provides support for customising certain aspects of the framework components, rather than be limited to specific techniques within each quality assurance domain. For example, support for different forecasting models can be added to the monitoring system as required. The integration and customisation support provided by the framework facilitates experimentation with the quality assurance solution.

The framework development also included the design and implementation of a service doping approach, used to simulate different QoS scenarios in service-oriented systems. The approach facilitates significantly complex experiments (see Table 5.1), which would prove difficult to perform in the real-world, requiring a suitable service marketplace and the co-operation of commercial service providers.

6.2 Future Research Directions

This section provides ideas for future research stemming from the development and evaluation of the integrated quality assurance framework for service-oriented systems. Each research direction is now discussed in turn:

- *Experiment with quality assurance techniques.* While the current framework provides support for customising the quality assurance solution, little work has been done in experimenting with *different* approaches from each of the quality assurance domains discussed in the literature review. As a proof of concept, the reference framework implementation has integrated a service ontology with brokerage, monitoring and reputation systems. An experimental approach would integrate multiple techniques from within each quality assurance domain. The experimental approach would then have the ability to switch between techniques at runtime, in response to changes in service requirements and the system runtime environment.

- *Explore and address monitoring concerns.* The monitoring approach currently provided by the framework does not consider problems external to the

system. However, it is possible for problems outside of the service provider's control to cause an SLA violation. For example, complications below the application-level QoS monitored by the framework, such as network congestion and high latency, may lead to an SLA violation. It is argued that QoS cannot be guaranteed when the provider's network connection is supplied by a different ISP to the consumer (Molina-Jimenez et al., 2004). Future research could investigate the distribution and installation of monitors to the same network as the service provider, to mitigate these issues.

The service monitors provided by the framework's monitoring system lack communication between themselves for detecting emergent issues between interdependent services. Future research could investigate the development of a *super monitor*, that is capable of observing a service composition and the interactions between services.

- *Improve forecasting support.* The forecasting models used by the framework's monitoring system are currently weak at detecting trends in service quality. The weakness of the current forecasting models is due to the substantial variability of quality experienced in service-oriented systems. The forecasting model used by a service monitor is currently fixed in-place at the time the monitor is assembled. To provide more accurate QoS estimations, service monitors could be assembled with multiple forecasting models. The monitors would then be provided with the ability to switch between different forecasting models at runtime, in order to determine and use the most accurate model for the current system conditions. In addition, the forecasting models themselves could be more dynamic. For example, a simple moving average (SMA) model could implement a self-adjusting window size. Similarly, an exponential moving average (EMA) model could implement a self-adjusting smoothing factor.

- *Explore and improve upon framework trust issues.* The framework provides security and privacy throughout the domain it oversees, through the use of keys, tokens and SSL encryption. However, on a more general level the framework provides no reason why consumers and providers should trust the results of the framework's brokerage, monitoring and reputation processes. As such, the framework requires that service consumers and providers trust the quality assurance components supplied by the framework.

- *Enhance the framework's reputation system.* The reputation system provided by the framework only considers the reputation of the service provider in terms of the QoS provided to a consumer. The provider reputation can also be extended to incorporate additional criteria, such as communication and negotiation behaviour. For example, a consumer may use communication and negotiation criteria, to avoid a provider that requires an unusually high number of negotiation rounds in order to secure an agreement, when compared to other providers of the same service.

  The framework's reputation system could also provide the consumer with a service selection mechanism, enabling the consumer to alternatively secure service from a provider without the aid of the brokerage system. The reputation selection mechanism could then be used in the event that the brokerage system is unavailable. The service selection decision would be based on the reputation of a provider and its service advertisement.

- *Incorporate fallback mechanisms and dynamic strategies.* The quality assurance solution provided by the framework works on the assumption that the framework's brokerage, monitoring and reputation systems are always available and functional. The service consumers and providers do not currently have an explicit fallback mechanism or self-adapting strategy, in the event that any of the framework systems become unavailable.

- *Provide support for other SOAs, e.g. those based on web services.* The framework has been evaluated using the Jini service technology platform, but other SOA implementations are available. For example, service-oriented systems in industry are commonly realised using web services. It is therefore important to validate the framework's ability to support systems implemented with alternative service platforms, such as those based on web services.

- *Integrate framework with product line engineering.* There is growing research interest in integrating service-oriented techniques to promote agility and flexibility in product line engineering. However, this integration presents challenges such as ensuring the correct runtime quality of product-specific service compositions, maintaining the integrity of systems which use dynamically-composed services, and evaluating runtime service quality levels to provide information for future service selections. The potential of using the quality assurance framework to address these challenges can be explored.

6.3 Concluding Remarks

Ensuring quality in service-oriented systems is a challenging problem, due to the dynamic and complex nature of systems composed from services. Service quality is a combination of the QoS supplied by a service provider, interdependencies between services, constraints imposed by the runtime environment, and other events such as network failures. It is difficult to anticipate the impact of these factors on system behaviour. In addition, the third-party nature of services restricts the consumer's control over quality issues in service-oriented systems.

Traditional software quality assurance approaches are primarily static in nature, and are unable to adequately satisfy the needs of dynamic service-oriented systems. Current service-oriented quality assurance approaches are unsatisfactory, as they fail to provide an end-to-end solution for the consumer. In addition, current solutions offer the consumer little control over service quality, afford limited support for the expression of quality characteristics, lack adequate support for runtime quality assurance, provide limited support for resource-restricted systems, and lack support for integration and experimentation with approaches from different service quality assurance domains.

This thesis has explored the issue of ensuring quality in service-oriented systems, and the particular quality issues which plague such systems. The *software as a service* development model is still in its early stages, and as such there remain many challenges in realising the service-oriented software vision. This thesis has identified these challenges through an extensive literature review, which documents the state of the art in quality assurance solutions for service-oriented systems. The thesis then addresses some of these challenges, with the contribution of a solution that integrates different service-oriented quality assurance approaches, in the form of a software framework, with the potential for future research and development.

Appendix A

Ontology XML Examples

This appendix provides example service contract and strategy XML templates, that validate against the quality, service and strategy schemas of the service ontology provided by the quality assurance framework. The service ontology and schemas are discussed in Section 4.5 of the thesis.

Appendix A.1 provides a basic service contract description, which validates against the *service* XML schema. The contract provides a functional description of a fictional *calculator* service, which provides a single *add* operation for returning the product of two integers. The service contract specifies service-wide *lease* and *cost* quality constraints. The service contract then specifies the functional description of the *add* operation, and associates a *response time* quality constraint with the operation. The service contract ends with some additional textual information, which provides the *email address* of the service's technical support team.

Appendix A.2 provides a basic consumer service strategy example for the same fictional calculator service, which validates against the *strategy* XML schema. The service strategy first provides values for a series of weights, which are used to balance the acceptability of the service proposals made by providers, with the providers' reputation. The strategy then provides individual strategies for the service *lease*, *cost* and *response time* qualities. A quality relationship is specified between the service *lease* and *cost* qualities, so that any increase or decrease in the service *lease* value, respectively increases or decreases the value of the service *cost* directly by the same proportion.

A.1 Example Service Contract

```
<?xml version="1.0" encoding="UTF-8"?>
<service:ServiceContract
  xmlns:quality="http://www.lancs.ac.uk/~robinsdb/Quality"
  xmlns:service="http://www.lancs.ac.uk/~robinsdb/Service"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.lancs.ac.uk/~robinsdb/Service Service.xsd ">
  <service:Lease>
    <quality:Constraint>
      <quality:ConstraintDescription>==</quality:ConstraintDescription>
      <quality:Unit xsi:type="quality:TimeUnitType">
        <quality:Units>min</quality:Units>
      </quality:Unit>
      <quality:Value>60</quality:Value>
      <quality:ValueType>LONG</quality:ValueType>
    </quality:Constraint>
    <quality:QualityDescription>lease</quality:QualityDescription>
  </service:Lease>
  <service:ServiceQuality>
    <quality:Constraint>
      <quality:ConstraintDescription>==</quality:ConstraintDescription>
      <quality:Unit xsi:type="quality:CurrencyUnitType">
        <quality:Units>EUR</quality:Units>
      </quality:Unit>
      <quality:Value>1.00</quality:Value>
      <quality:ValueType>BIGDECIMAL</quality:ValueType>
    </quality:Constraint>
    <quality:QualityDescription>availability</quality:QualityDescription>
  </service:ServiceQuality>
  <service:OperationContract>
    <service:Operation>
      <service:OperationName>add</service:OperationName>
      <service:OperationSignature>
        public abstract int com.xyz.CalculatorService.add(int,
        int) throws java.rmi.RemoteException
      </service:OperationSignature>
      <service:ParameterType>int</service:ParameterType>
      <service:ParameterType>int</service:ParameterType>
      <service:ReturnType>int</service:ReturnType>
    </service:Operation>
    <service:OperationQuality>
      <quality:Constraint>
```

```
42          <quality:ConstraintDescription>&lt;=</quality:ConstraintDescription>
43          <quality:Unit xsi:type="quality:TimeUnitType">
44            <quality:Units>ms</quality:Units>
45          </quality:Unit>
46          <quality:Value>1000</quality:Value>
47          <quality:ValueType>LONG</quality:ValueType>
48        </quality:Constraint>
49        <quality:QualityDescription>response time</quality:QualityDescription>
50      </service:OperationQuality>
51    </service:OperationContract>
52    <service:ServiceType>com.xyz.CalculatorService</service:ServiceType>
53    <service:TextualInformation>
54      <service:TextInfoType>Technical Support</service:TextInfoType>
55      <service:TextInfoValue>support@xyz.com</service:TextInfoValue>
56    </service:TextualInformation>
57  </service:ServiceContract>
```

A.2 Example Service Strategy

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <strategy:Strategy
3    xmlns:quality="http://www.lancs.ac.uk/~robinsdb/Quality"
4    xmlns:service="http://www.lancs.ac.uk/~robinsdb/Service"
5    xmlns:strategy="http://www.lancs.ac.uk/~robinsdb/Strategy"
6    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7    xsi:schemaLocation="http://www.lancs.ac.uk/~robinsdb/Strategy.xsd ">
8    <strategy:GlobalRatingWeight>0.3</strategy:GlobalRatingWeight>
9    <strategy:PersonalRatingWeight>0.7</strategy:PersonalRatingWeight>
10   <strategy:ProposalWeight>0.3</strategy:ProposalWeight>
11   <strategy:ReputationWeight>0.7</strategy:ReputationWeight>
12   <strategy:ReputationThreshold>0.5</strategy:ReputationThreshold>
13   <strategy:ServiceStrategy>
14     <strategy:AcceptInstruction>
15       BEST_ACCEPTABLE_MATCH
16     </strategy:AcceptInstruction>
17     <strategy:ActiveNegotiation>true</strategy:ActiveNegotiation>
18     <strategy:LeaseStrategy>
19       <strategy:AssignedTo>com.xyz.CalculatorService</strategy:AssignedTo>
20       <strategy:ConstraintStrategy>
21         <strategy:Constraint>
22           <quality:ConstraintDescription>==</quality:ConstraintDescription>
23           <quality:Unit xsi:type="quality:TimeUnitType">
```

```
24            <quality:Units>min</quality:Units>
25          </quality:Unit>
26          <quality:Value>60</quality:Value>
27          <quality:ValueType>LONG</quality:ValueType>
28        </strategy:Constraint>
29        <strategy:ConstraintLimit>
30          <quality:ConstraintDescription>&gt;=</quality:ConstraintDescription>
31          <quality:Unit xsi:type="quality:UnitType">
32            <quality:Units>min</quality:Units>
33          </quality:Unit>
34          <quality:Value>30</quality:Value>
35          <quality:ValueType>LONG</quality:ValueType>
36        </strategy:ConstraintLimit>
37        <strategy:StrategyType>MAXIMISE</strategy:StrategyType>
38      </strategy:ConstraintStrategy>
39      <strategy:QualityManagementRequired>
40        false
41      </strategy:QualityManagementRequired>
42      <strategy:QualityRelation>
43        <strategy:ProportionalityConstant>1.0</strategy:ProportionalityConstant>
44        <strategy:RelatedQualityStrategy>
45          <strategy:AssignedTo>com.xyz.CalculatorService</strategy:AssignedTo>
46          <strategy:ConstraintStrategy>
47            <strategy:Constraint>
48              <quality:ConstraintDescription>==</quality:ConstraintDescription>
49              <quality:Unit xsi:type="quality:CurrencyUnitType">
50                <quality:Units>EUR</quality:Units>
51              </quality:Unit>
52              <quality:Value>1.00</quality:Value>
53              <quality:ValueType>BIGDECIMAL</quality:ValueType>
54            </strategy:Constraint>
55            <strategy:ConstraintLimit>
56              <quality:ConstraintDescription>
57                &lt;=
58              </quality:ConstraintDescription>
59              <quality:Unit xsi:type="quality:CurrencyUnitType">
60                <quality:Units>EUR</quality:Units>
61              </quality:Unit>
62              <quality:Value>5.00</quality:Value>
63              <quality:ValueType>BIGDECIMAL</quality:ValueType>
64            </strategy:ConstraintLimit>
65            <strategy:StrategyType>MINIMISE</strategy:StrategyType>
```

```
66            </strategy:ConstraintStrategy>
67            <strategy:QualityManagementRequired>
68              false
69            </strategy:QualityManagementRequired>
70            <strategy:QualityType>cost</strategy:QualityType>
71            <strategy:QualityWeight>0.5</strategy:QualityWeight>
72          </strategy:RelatedQualityStrategy>
73          <strategy:Relation>DIRECT</strategy:Relation>
74        </strategy:QualityRelation>
75        <strategy:QualityType>lease</strategy:QualityType>
76        <strategy:QualityWeight>0.5</strategy:QualityWeight>
77      </strategy:LeaseStrategy>
78      <strategy:ServiceType>com.xyz.CalculatorService</strategy:ServiceType>
79      <strategy:ServiceWeight>1.0</strategy:ServiceWeight>
80    </strategy:ServiceStrategy>
81  </strategy:Strategy>
```

Appendix B

System Visualisation Tool

This appendix presents a walkthrough of the system visualisation tool, used for capturing and visualising system behaviour at runtime. The system visualisation tool was used in the development of the evaluation experiments discussed in Chapter 5, and makes extensive use of the JFreeChart (Gilbert, 2009) application programming interface (API) for chart generation.

B.1 Initial State

Before a simulation can be started, a series of command-line scripts are used to launch the Jini service registry, to launch and register the core brokerage, monitoring and reputation framework services, and to launch and register the service providers required for the simulation. Each different experiment has its own set of service provider scripts, which configure the providers with particular service doping mechanisms.

Once the service providers are registered with the Jini service registry, the system visualisation tool is used to load the consumer service strategies. Before starting the simulation, the only data available to the visualisation tool is the initial reputation data, obtained by discovering and querying the service provided by the framework reputation system. The negotiation and monitoring process viewers provided by the tool contain no data at this point. When ready, the user of the tool is able to launch the simulation from the menu, and may later pause and resume the simulation from the same menu, in order to take snapshots of the system. The initial state of the visualisation tool is shown in Figure B.1.

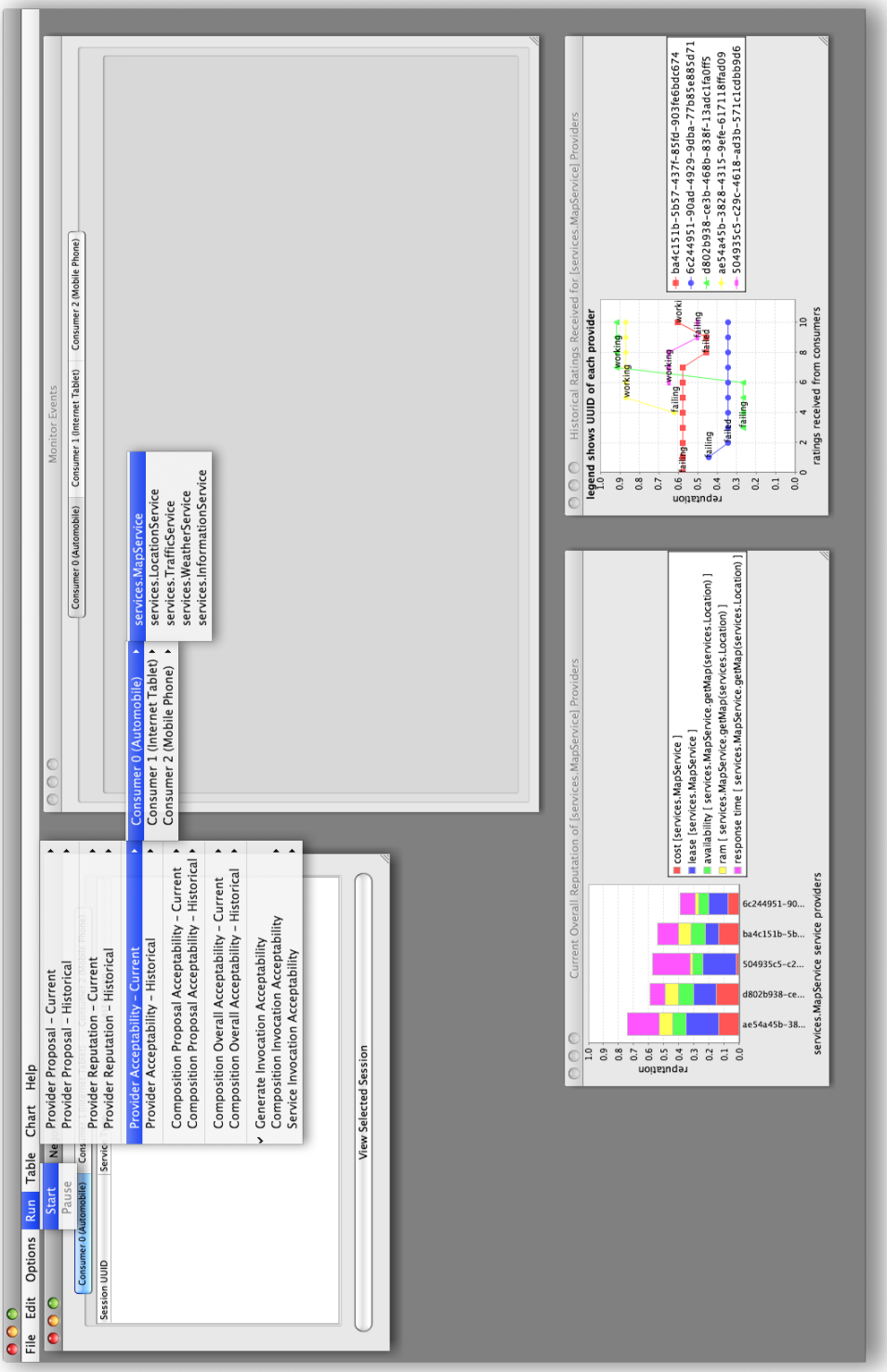


Figure B.1: When visualisation tool is initialised, only reputation data is available; the negotiation and monitoring views are empty. The tool provides a menu to start, pause and resume the current simulation.

B.2 Negotiation Visualisation

The system visualisation tool provides several views of the framework negotiation process. The tool first provides a negotiation session list viewer, which shows the current state of each negotiation session for each service consumer. The negotiation session list viewer is shown in Figure B.2.

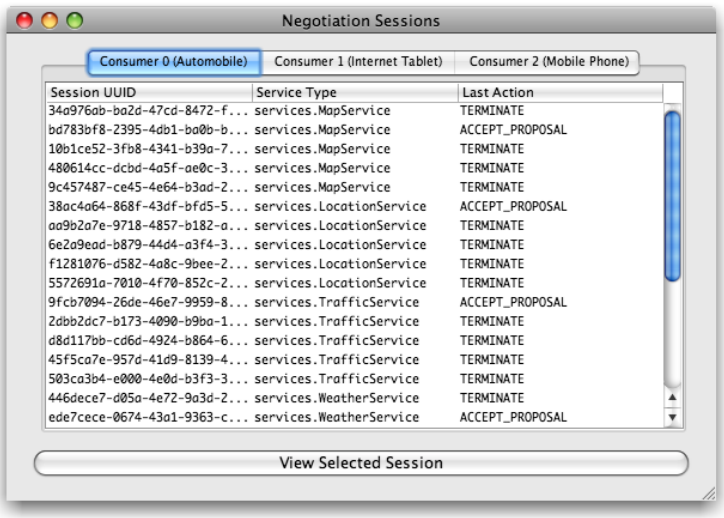


Figure B.2: The negotiation session list viewer shows the latest state of each negotiation session for each consumer.

Using the negotiation session list viewer, the user can select and view any active or terminated negotiation session. The contents of a negotiation session are presented by the negotiation session viewer, as shown in Figure B.3. From the negotiation session viewer, the user is able to view the contents and service proposal of any negotiation message, as shown in Figure B.4. In the example shown in Figure B.4, the negotiation message corresponds to the final *accept* message from the negotiation session shown in Figure B.3.

For each service consumer, the visualisation tool provides views of the current acceptability of the available providers for the services required by the consumer. Figure B.5 shows a service proposal viewer, which compares the latest proposals from each service provider for a particular service type. The example shows the acceptability of the available map service providers to the automobile consumer.

The user of the visualisation tool is also able to view the reputation of the available providers for the services required by each consumer. Figure B.6 shows the service provider reputation viewer, which compares the reputation of each

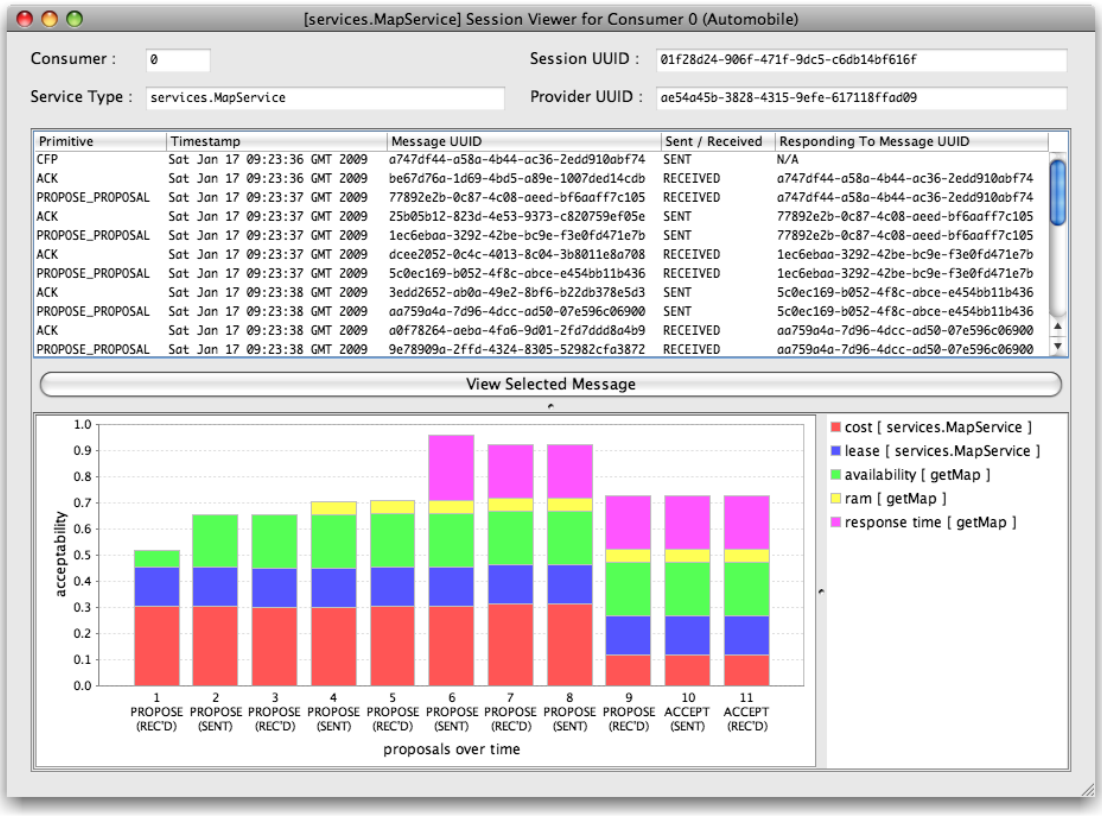


Figure B.3: The negotiation session viewer provided by the visualisation tool, is used to display the contents of a negotiation session between any consumer and provider.

service provider for a particular service type. The example shows the reputation of the same set of map service providers, whose proposals are shown in Figure B.5.

The user is then able to view the overall acceptability of the available providers for the services required by each consumer. The overall provider acceptability is the combination of the proposed acceptability data, shown in Figure B.5, with the reputation data shown in Figure B.6. The reputation and proposal data is combined according to the weights in the consumer strategy (see Appendix A.2). The overall acceptability viewer is shown in Figure B.7, for the same set of map service providers and the automobile consumer already discussed.

Once a service composition has been negotiated for a service consumer, the user is able to view the acceptability of the service composition proposals, as shown in Figure B.8. Figure B.9 provides a similar overview of the composition, but reflects the combination of the service composition proposal acceptability, with the reputation of the providers for each service in the composition.

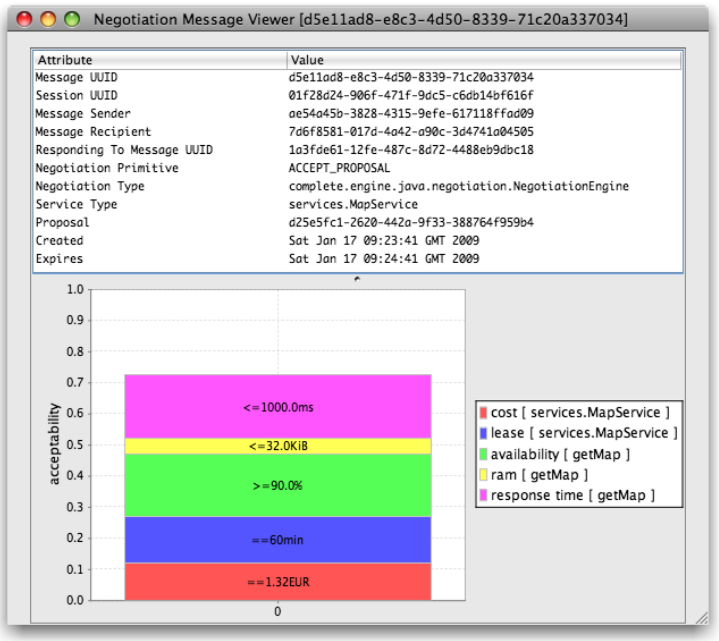


Figure B.4: The negotiation message viewer provided by the visualisation tool. The example message shown corresponds to the final negotiation message from the session in Figure B.3.

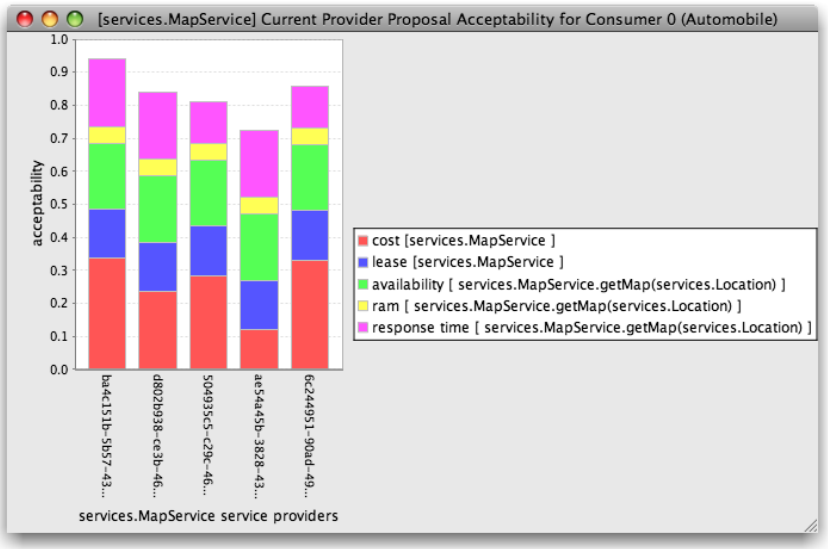


Figure B.5: The visualisation tool provides an overview of the current service proposals received for each service consumer and service type. This example shows the current map service proposals received by the broker of the automobile consumer.

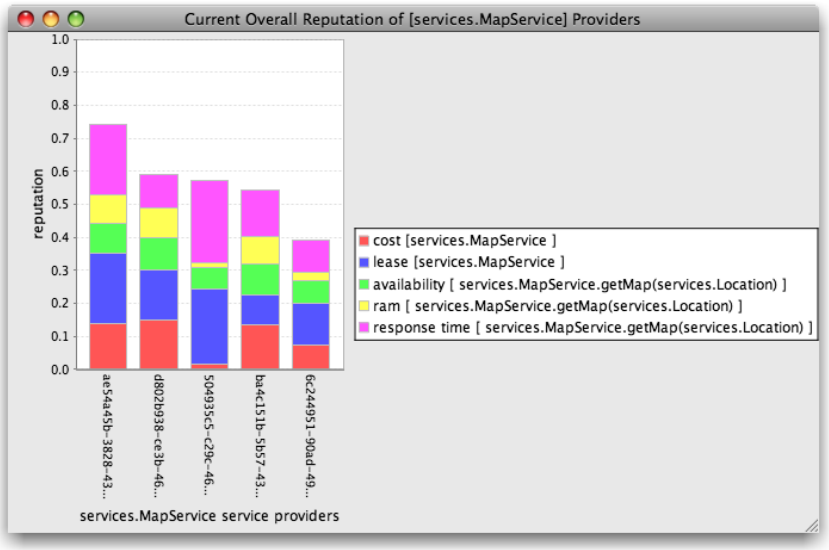


Figure B.6: The visualisation tool provides an overview of the current reputation for each service provider of each service type. This example shows the current reputation of the available map service providers.

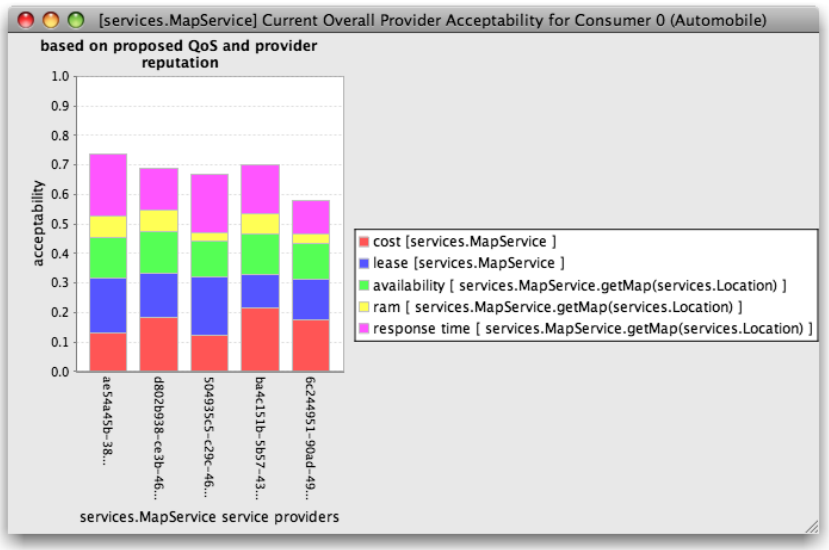


Figure B.7: The visualisation tool provides an overview of the overall provider acceptability for each service consumer and service type. This acceptability is a combination of the provider's proposal acceptability and reputation. This example shows the current overall acceptability of the available map service providers to the automobile consumer.

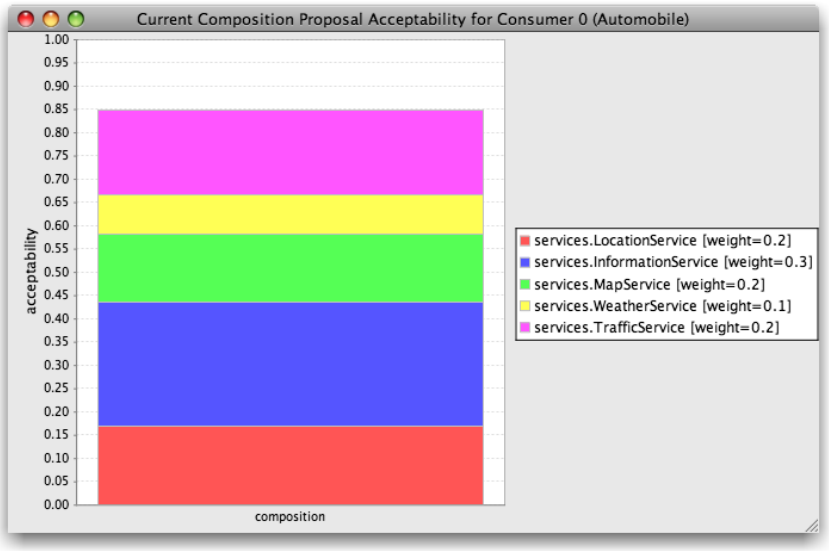


Figure B.8: The visualisation tool provides an overview of the negotiated composition acceptability.

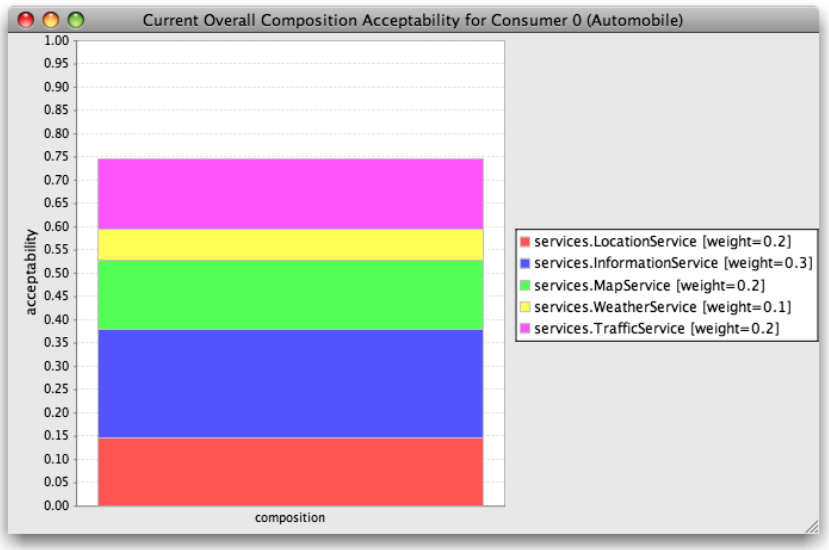


Figure B.9: The visualisation tool provides an overview of the overall composition acceptability, which incorporates the acceptability of the negotiated service proposals and the reputation of the service providers.

B.3 Service Performance Visualisation

The system visualisation tool provides views which show the runtime performance of the services involved in the system. The tool provides a monitor event viewer, shown in Figure B.10, which displays the data received from the monitors attached to the services invoked by each consumer. The monitor event viewer logs the monitor event data in a table, and provides individual charts which display the observed and forecast values for each monitored service quality. In the example shown in Figure B.10, the monitor event viewer is displaying the map service performance for the automobile consumer.

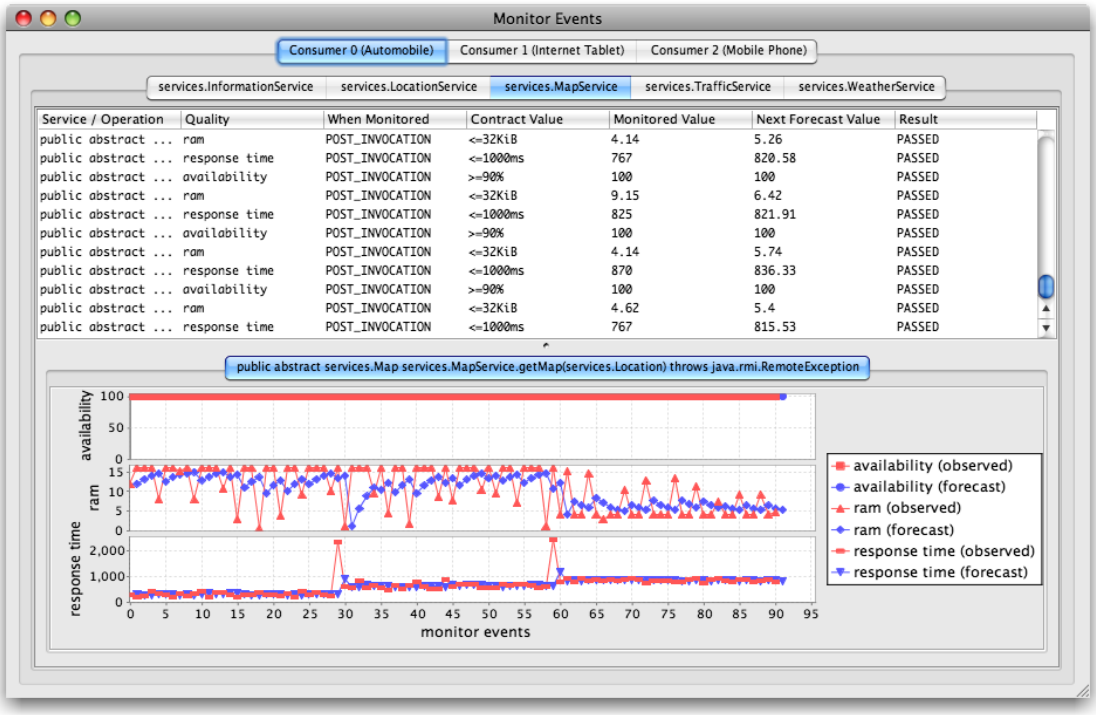


Figure B.10: The visualisation tool provides a monitor event viewer for each active consumer and service that is invoked in the simulation.

The monitor event data is combined with the service consumer strategies, to produce views which show the runtime acceptability of the services for each consumer. The visualisation tool provides a service invocation viewer, shown in Figure B.11, that shows the individual acceptability of monitored qualities for each service invocation. The tool also provides a composition invocation viewer, shown in Figure B.12, that shows the overall acceptability of each service in the composition, and also shows the overall acceptability of the composition itself.

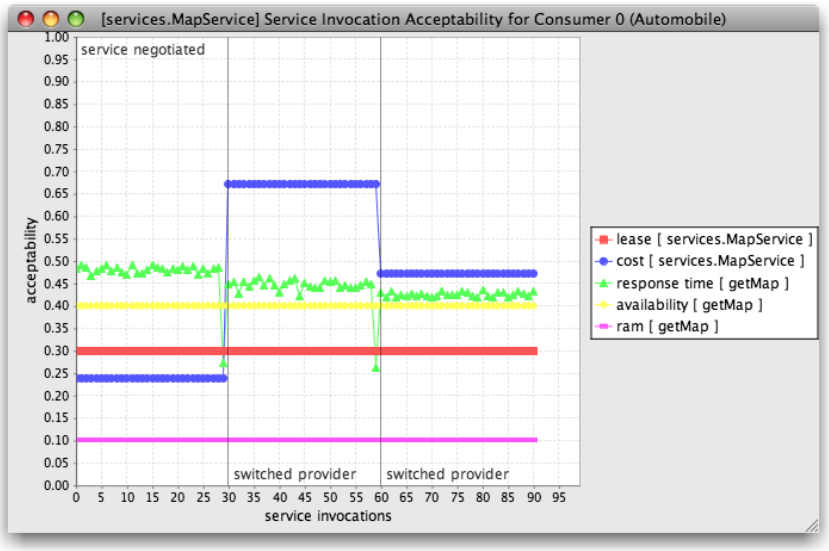


Figure B.11: The visualisation tool provides a service invocation acceptability viewer for each active consumer and service that is invoked in the simulation.

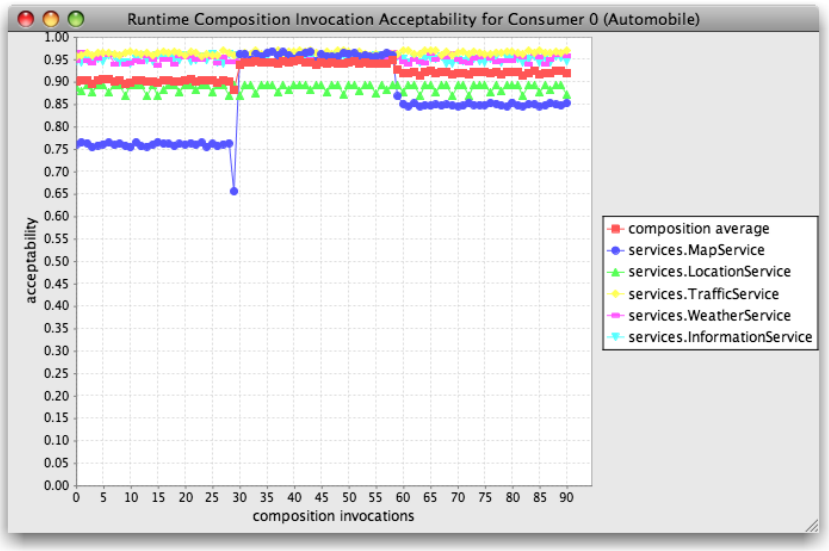


Figure B.12: The visualisation tool provides a composition invocation acceptability viewer for each active consumer invoked in the simulation.

References

Abran, A., & Moore, J. W. (Eds.). (2004). *Guide to the software engineering body of knowledge, 2004 version.* IEEE Computer Society.

Ali, A. S., Majithia, S., Rana, O. F., & Walker, D. W. (2006). Reputation-based semantic service discovery. *Concurrency and Computation: Practice & Experience*, *18*(8), 817–826.

Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., et al. (2006). *Web services agreement specification (WS-Agreement), version 2006-09-07* (Tech. Rep.). Global Grid Forum.

Apache Software Foundation. (2009). *Apache Axis.* Retrieved March 17th, 2009, from http://ws.apache.org/axis/

Baresi, L., Ghezzi, C., & Guinea, S. (2004a). Smart monitors for composed services. In *ICSOC '04: Proceedings of the 2nd international conference on service oriented computing* (pp. 193–202). New York, NY, USA: ACM Press.

Baresi, L., Ghezzi, C., & Guinea, S. (2004b). Towards self-healing service compositions. In *PRISE '04, first conference on the principles of software engineering.* Buenos Aires, Argentina.

Baresi, L., & Guinea, S. (2005). Towards dynamic monitoring of WS-BPEL processes. In *ICSOC 2005, third international conference of service-oriented computing, volume 3826 of lecture notes in computer science* (pp. 269–282). Springer.

Benatallah, B., Dumas, M., Fauvet, M.-C., Rabhi, F. A., & Sheng, Q. Z. (2002). Overview of some patterns for architecting and managing composite web services. *SIGecom Exchanges*, *3*(3), 9–16.

Benjamim, A. C., Sauvé, J., Cirne, W., & Carelli, M. (2004). Independently auditing service level agreements in the grid. In *Proceedings of the 11th HP OpenView university association workshop, HPOVUA 2004.*

Bennett, K., Layzell, P., Budgen, D., Brereton, P., Macaulay, L., & Munro, M.

(2000). Service-based software: the future for flexible software. *Asia-Pacific Software Engineering Conference*, *0*, 214.

Bennett, K., Munro, M. C., Gold, N., Layzell, P. J., Budgen, D., & Brereton, P. (2001). An architectural model for service-based software with ultra rapid evolution. In *ICSM '01: Proceedings of the IEEE international conference on software maintenance (ICSM '01)* (p. 292). Washington, DC, USA: IEEE Computer Society.

Berbner, R., Spahn, M., Repp, N., Heckmann, O., & Steinmetz, R. (2006). Heuristics for QoS-aware web service composition. In *ICWS '06: Proceedings of the IEEE international conference on web services* (pp. 72–82). Washington, DC, USA: IEEE Computer Society.

Bianculli, D., & Ghezzi, C. (2007). Monitoring conversational web services. In *IW-SOSWE '07: 2nd international workshop on service oriented software engineering* (pp. 15–21). New York, NY, USA: ACM.

Brereton, P., Budgen, D., Bennnett, K., Munro, M., Layzell, P., MaCaulay, L., et al. (1999). The future of software. *Communications of the ACM*, *42*(12), 78–84.

Brogi, A., Corfini, S., & Popescu, R. (2008). Semantics-based composition-oriented discovery of web services. *ACM Transactions on Internet Technology (TOIT)*, *8*(4), 1–39.

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). *Web services description language (WSDL).* Retrieved 4th February, 2009, from http://www.w3.org/TR/wsdl

Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (1999). *Non-functional requirements in software engineering* (Vol. 5). Springer.

Clement, L., Hately, A., Riegen, C. von, & Rogers, T. (2004). *UDDI spec technical committee draft 3.0.2* (OASIS Committee Draft). OASIS. Retrieved February 9th, 2009, from http://uddi.org/pubs/uddi_v3.htm

Comuzzi, M., & Pernici, B. (2005). An architecture for flexible web service QoS negotiation. In *EDOC* (p. 70-82). IEEE Computer Society.

Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kuebler, D., et al. (2004). Web services on demand: WSLA-driven automated management. *IBM Systems Journal*, *43*(1), 136–158.

Dobson, G., Lock, R., & Sommerville, I. (2005). QoSOnt: a QoS ontology for service-centric systems. In *EUROMICRO-SEAA* (p. 80-87). IEEE Computer

Society.

Elfatatry, A., & Layzell, P. (2005). A negotiation description language. *Software: Practice and Experience*, *35*(4), 323–343.

Erl, T. (2005). *Service-oriented architecture: Concepts, technology, and design.* Upper Saddle River, NJ, USA: Prentice Hall PTR.

Erl, T. (2007). *SOA: Principles of service design.* Upper Saddle River, NJ, USA: Prentice Hall PTR.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley Professional.

Gilb, T. (1988). *Principles of software engineering management.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Gilbert, D. (2009). *JFreeChart Java chart library.* Retrieved 13th April, 2009, from http://www.jfree.org/jfreechart/

Gimpel, H., Ludwig, H., Dan, A., & Kearney, R. (2003). PANDA: Specifying policies for automated negotiations of service contracts. In M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, & J. Yang (Eds.), *ICSOC* (Vol. 2910, p. 287-302). Springer.

Grassi, V., & Patella, S. (2006). Reliability prediction for service-oriented computing environments. *Internet Computing, IEEE*, *10*(3), 43-49.

Heineman, G. T., Loyall, J. P., & Schantz, R. E. (2004). Component technology and QoS management. In I. Crnkovic, J. A. Stafford, H. W. Schmidt, & K. C. Wallnau (Eds.), *CBSE* (Vol. 3054, p. 249-263). Springer.

Herssens, C., Faulkner, S., & Jureta, I. (2008). Context-driven autonomic adaptation of SLA. In A. Bouguettaya, I. Krüger, & T. Margaria (Eds.), *ICSOC* (Vol. 5364, p. 362-377).

Hoffman, B. (2005). Monitoring, at your service. *ACM Queue*, *3*(10), 34–43.

ISO. (2000). *ISO 9001:2000, quality management systems – requirements.*

Jennings, N. R., Norman, T. J., Faratin, P., & Odgers, B. (2000). Autonomous agents for business process management. *Journal of Applied Artificial Intelligence*, *14*, 145–189.

Jordan, D., & Evdemon, J. (2007). *Web services business process execution language version 2.0.* Retrieved February 11th, 2009, from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

Jøsang, A., Ismail, R., & Boyd, C. (2007). A survey of trust and reputation systems for online service provision. *Decision Support Systems*, *43*(2), 618–644.

Josuttis, N. M. (2007). *SOA in practice: The art of distributed system design.* Sebastopol, CA, USA: O'Reilly.

Jouve, W., Lancia, J., Consel, C., & Pu, C. (2006). A multimedia-specific approach to WS-Agreement. In *ECOWS '06: Proceedings of the European conference on web services* (pp. 44–52). Washington, DC, USA: IEEE Computer Society.

Jurca, R., Faltings, B., & Binder, W. (2007). Reliable QoS monitoring based on client feedback. In *WWW '07: Proceedings of the 16th international conference on world wide web* (pp. 1003–1012). New York, NY, USA: ACM Press.

Kitchenham, B., Pfleeger, S. L., Pickard, L., Jones, P., Hoaglin, D., Emam, K. E., et al. (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, *28*, 721–734.

Klein, M., König-Ries, B., & Müssig, M. (2005). What is needed for semantic service descriptions? A proposal for suitable language constructs. *International Journal of Web and Grid Services*, *1*(3/4), 328–364.

Kraus, S. (2001). Automated negotiation and decision making in multiagent environments. In M. Luck, V. Marík, O. Stepánková, & R. Trappl (Eds.), *Easss* (Vol. 2086, p. 150-172). Springer.

Kretzschmar, F. (2006). Negotiations in service-oriented architectures. *IBIS - International Journal of Interoperability in Business Information Systems*, *1*(3), 73-84.

Kritikos, K., & Plexousakis, D. (2007). Requirements for QoS-based web service description and discovery. *31st Annual International Computer Software and Applications Conference, 2007 (COMPSAC 2007)*, *2*, 467-472.

Küster, U., & König-Ries, B. (2007). Supporting dynamics in service descriptions – the key to automatic service usage. In *Proceedings of the fifth international conference on service oriented computing (ICSOC '07).* Vienna, Austria.

Lamanna, D. D., Skene, J., & Emmerich, W. (2003). SLAng: A language for defining service level agreements. In *Proceedings of the the ninth IEEE workshop on future trends of distributed computing systems (FTDCS '03)* (p. 100). Washington, DC, USA: IEEE Computer Society.

Lausen, H., Polleres, A., & Roman, D. (2005). *Web service modeling ontology (WSMO).* Retrieved 6th February, 2009, from http://www.w3.org/Submission/WSMO/

Lazovik, A., Aiello, M., & Papazoglou, M. (2004). Associating assertions with business processes and monitoring their execution. In *ICSOC '04: Proceedings of the 2nd international conference on service oriented computing* (pp. 94–104). New York, NY, USA: ACM Press.

Litoiu, M., Mihaescu, M., Ionescu, D., & Solomon, B. (2008). Scalable adaptive web services. In *SDSOA '08: Proceedings of the 2nd international workshop on systems development in SOA environments* (pp. 47–52). New York, NY, USA: ACM.

Lock, R. (2006). Automated negotiation for service contracts. *30th Annual International Computer Software and Applications Conference, 2006 (COMPSAC '06)*, *2*, 127-134.

Lüders, F., Flemström, D., & Wall, A. (2005). Software component services for embedded real-time systems. In *Proceedings of the fifth conference on software engineering research and practice in Sweden* (p. 123-128). Västerås, Sweden: Mälardalen University.

Ludwig, H., Dan, A., & Kearney, R. (2004). Cremona: an architecture and library for creation and monitoring of WS-Agreements. In *ICSOC '04: Proceedings of the 2nd international conference on service oriented computing* (pp. 65–74). New York, NY, USA: ACM Press.

Ludwig, H., Keller, A., Dan, A., King, R. P., & Franck, R. (2003). *Web service level agreement (WSLA) language specification.* Retrieved 31st January, 2009, from http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

Mahbub, K., & Spanoudakis, G. (2004). A framework for requirements monitoring of service based systems. In *ICSOC '04: Proceedings of the 2nd international conference on service oriented computing* (pp. 84–93). New York, NY, USA: ACM Press.

Martin, D. L., Paolucci, M., McIlraith, S. A., Burstein, M. H., McDermott, D. V., McGuinness, D. L., et al. (2004). Bringing semantics to web services: The OWL-S approach. In J. Cardoso & A. P. Sheth (Eds.), *SWSWPC* (Vol. 3387, p. 26-42). Springer.

Maximilien, E. M., & Singh, M. P. (2004a). A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, *08*(5), 84-93.

Maximilien, E. M., & Singh, M. P. (2004b). Toward autonomic web services trust and selection. In *ICSOC '04: Proceedings of the 2nd international conference on service oriented computing* (pp. 212–221). New York, NY, USA: ACM

Press.

McGuinness, D. L., & Harmelen, F. van. (2004). *OWL web ontology language overview.* Retrieved 6th February, 2009, from http://www.w3.org/TR/owl-features/

McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems*, *16*(2), 46–53.

Menascé, D. A. (2002). QoS issues in Web services. *Internet Computing, IEEE*, *6*(6), 72–75.

Menascé, D. A., & Casalicchio, E. (2004). QoS in grid computing. *IEEE Internet Computing*, *08*(4), 85-87.

Menascé, D. A., & Dubey, V. (2007). Utility-based QoS brokering in service oriented architectures. *ICWS*, *0*, 422-430.

Menascé, D. A., Ruan, H., & Gomaa, H. (2007). QoS management in service-oriented architectures. *Performance Evaluation*, *64*(7-8), 646–663.

Milanovic, N., Richling, J., & Malek, M. (2004). Lightweight services for embedded systems. In *Workshop on software technologies for embedded and ubiquitous computing systems (WSTFEUS)* (Vol. 00, p. 40). Los Alamitos, CA, USA: IEEE Computer Society.

Milanovic, N., Stantchev, V., Richling, J., & Malek, M. (2003). Towards adaptive and composable services. In S. Stefan (Ed.), *Proceedings of the international IPSI conference (IPSI 2003).* Montenegro.

Molina-Jimenez, C., Shrivastava, S., Crowcroft, J., & Gevros, P. (2004). On the monitoring of contractual service level agreements. In *WEC '04: Proceedings of the first IEEE international workshop on electronic contracting* (pp. 1–8). Washington, DC, USA: IEEE Computer Society.

Moser, O., Rosenberg, F., & Dustdar, S. (2008). Non-intrusive monitoring and service adaptation for WS-BPEL. In *WWW '08: Proceedings of the 17th international conference on world wide web* (pp. 815–824). New York, NY, USA: ACM.

Müller, C., Cortés, A. R., & Resinas, M. (2008). An initial approach to explaining SLA inconsistencies. In A. Bouguettaya, I. Krüger, & T. Margaria (Eds.), *ICSOC* (Vol. 5364, p. 394-406).

Object Management Group, Inc. (2004). *Common object request broker architecture: Core specification.* Retrieved May 20th, 2009, from http://www.omg.org/docs/formal/04-03-12.pdf

Oldham, N., Verma, K., Sheth, A., & Hakimpour, F. (2006). Semantic WS-Agreement partner selection. In *WWW '06: Proceedings of the 15th international conference on world wide web* (pp. 697–706). New York, NY, USA: ACM Press.

OSGi Alliance. (2009). *OSGi Alliance specifications.* Retrieved January 28th, 2009, from http://www.osgi.org/Specifications/HomePage

O'Sullivan, J., Edmond, D., & Hofstede, A. T. (2002). What's in a service? Towards accurate description of non-functional service properties. *Distributed and Parallel Databases*, *12*(2-3), 117–133.

Paolucci, M., Kawamura, T., Payne, T. R., & Sycara, K. (2002). Semantic matching of web services capabilities. In *First international semantic web conference Sardinia, Italy, June 9–12, 2002* (Vol. 2342/2002, p. 333-347). Springer Berlin / Heidelberg.

Poladian, V., Sousa, J. P., Garlan, D., & Shaw, M. (2004). Dynamic configuration of resource-aware services. In *ICSE '04: Proceedings of the 26th international conference on software engineering* (pp. 604–613). Washington, DC, USA: IEEE Computer Society.

Pouyllau, H., & Haar, S. (2007). A protocol for QoS contract negotiation and its implementation using web services. In *2007 IEEE international conference on web services (ICWS 2007)* (p. 168-175). Salt Lake City, Utah, USA.

Robinson, D., & Kotonya, G. (2008a). A runtime quality architecture for service-oriented systems. In A. Bouguettaya, I. Krüger, & T. Margaria (Eds.), *International conference on service-oriented computing (ICSOC)* (Vol. 5364, p. 468-482).

Robinson, D., & Kotonya, G. (2008b). A self-managing brokerage model for quality assurance in service-oriented systems. In X. Li, C. S. Smidts, & J. Xu (Eds.), *High assurance systems engineering (HASE)* (p. 424-433). IEEE Computer Society.

Saunders, S., Ross, M., Staples, G., & Wellington, S. (2006). The software quality challenges of service oriented architectures in e-commerce. *Software Quality Control*, *14*(1), 65–75.

ShaikhAli, A., Rana, O., Al-Ali, R., & Walker, D. (2003). UDDIe: an extended registry for web services. *Proceedings of the Symposium on Applications and the Internet Workshops, 2003*, 85-89.

Shaw, M. (1996). Truth vs knowledge: The difference between what a component

does and what we know it does. In *IWSSD '96: Proceedings of the 8th international workshop on software specification and design* (p. 181). Washington, DC, USA: IEEE Computer Society.

Skene, J., Skene, A., Crampton, J., & Emmerich, W. (2007). The monitorability of service-level agreements for application-service provision. In *WOSP '07: Proceedings of the 6th international workshop on software and performance* (pp. 3–14). New York, NY, USA: ACM.

Sommerville, I. (2006). Software engineering (8th edition). In (pp. 743–769). Addison Wesley.

Su, S. Y. W., Huang, C., Hammer, J., Huang, Y., Li, H., Wang, L., et al. (2001). An internet-based negotiation server for e-commerce. *Very Large Databases (VLDB) Journal*, *10*(1), 72-90.

Sun Microsystems, Inc. (2009a). *The Java language specification.* Retrieved February 20th, 2009, from http://java.sun.com/docs/books/jls/index.html

Sun Microsystems, Inc. (2009b). *Jini specifications and API archive.* Retrieved January 28th, 2009, from http://java.sun.com/products/jini/

Tian, M., Gramm, A., Ritter, H., Schiller, J., & Winter, R. (2004). A survey of current approaches towards specification and management of quality of service for web services. *Praxis der Informationsverarbeitung und Kommunikation*, *3*(4).

Toma, I., Foxvog, D., & Jaeger, M. C. (2006). Modeling QoS characteristics in WSMO. In *MW4SOC '06: Proceedings of the 1st workshop on middleware for service oriented computing* (pp. 42–47). New York, NY, USA: ACM Press.

Tournier, J.-C., Babau, J.-P., & Olive, V. (2005). An evaluation of Qinna, a component-based QoS architecture for embedded systems. In *SAC '05: Proceedings of the 2005 ACM symposium on applied computing* (pp. 998–1002). New York, NY, USA: ACM Press.

Truex, D. P., Baskerville, R., & Klein, H. (1999). Growing systems in emergent organizations. *Communications of the ACM*, *42*(8), 117–123.

Turner, M., Budgen, D., & Brereton, P. (2003). Turning software into a service. *Computer*, *36*(10), 38-44.

W3C Working Group. (2004). *Web services architecture.* Retrieved May 20th, 2009, from http://www.w3.org/TR/ws-arch/

Wang, X., Vitvar, T., Kerrigan, M., & Toma, I. (2006). A QoS-aware selection

model for semantic web services. In A. Dan & W. Lamersdorf (Eds.), *ICSOC* (Vol. 4294, p. 390-401). Springer.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, *265*(3), 66-75.

Wishart, R., Robinson, R., Indulska, J., & Jøsang, A. (2005). SuperstringRep: reputation-enhanced service discovery. In *ACSC '05: Proceedings of the twenty-eighth Australasian conference on computer science* (pp. 49–57). Darlinghurst, Australia: Australian Computer Society, Inc.

Wolf, W. (2001). *Computers as components: principles of embedded computing system design.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Wolski, R. (1998). Dynamically forecasting network performance using the network weather service. *Cluster Computing*, *1*(1), 119–132.

Wood, D. (1976). *Forecasting for business.* New York: Longman.

Woodside, C. M., & Menascé, D. A. (2006). Guest editors' introduction: Application-level QoS. *IEEE Internet Computing*, *10*(3), 13-15.

World Wide Web Consortium (W3C). (2001). *XML Schema.* Retrieved March 19th, 2009, from http://www.w3.org/XML/Schema

Xu, Z., Martin, P., Powley, W., & Zulkernine, F. (2007). Reputation-enhanced QoS-based web services discovery. In *2007 IEEE international conference on web services (ICWS 2007)* (p. 249-256). Salt Lake City, Utah, USA.

Yan, J., Kowalczyk, R., Lin, J., Chhetri, M. B., Goh, S. K., & Zhang, J. (2007). Autonomous service level agreement negotiation for service composition provision. *Future Generation Computer Systems*, *23*(6), 748–759.