A Runtime Quality Architecture for Service-Oriented Systems

Daniel Robinson and Gerald Kotonya

Computing Department, Lancaster University, Lancaster, LA1 4WA, UK
{robinsdb, gerald}@comp.lancs.ac.uk

Abstract. System quality aspects such as dependability, adaptability to a changing runtime environment, and concerns such as cost and provider reputation, are increasingly important in a competitive software service market. Service-oriented system quality is not just a function of the quality of a provided service, but the interdependencies between services, the resource constraints of the runtime environment and network outages. This makes it difficult to anticipate how these factors might influence system behaviour, making it difficult to specify the right system environment in advance. Current quality management schemes for service-oriented systems are inadequate for ensuring runtime system quality as they focus on static service properties, rather than emergent properties. They also offer the consumer only limited control over the quality of service. This paper describes a novel consumer-centred runtime architecture that combines service monitoring, negotiation, forecasting and vendor reputation, to provide a self-managing mechanism for ensuring runtime quality in service-oriented systems.

Keywords: Service-Oriented Architecture, Negotiation, Monitoring, Quality of Service, Software Composition.

1 Introduction

Service-oriented architectures support dynamic composition and reconfiguration of software systems by making advertised functionality and behaviour available on an "as-needed" basis [1]. This model of software deployment offers significant benefits over the traditional model of software deployment as a product, including reduced capital investment, dynamic integration and rapid deployment of platform and network-independent systems [2,3]. However, as the nature of service-oriented applications continues to vary and the demands on them grow, features such as dependability, adaptability to a changing runtime environment, and concerns such as cost and provider reputation are becoming increasingly important consumer quality considerations.

Current service quality management schemes are largely concerned with predicting system properties based on the static properties of its components [4]. However, the dynamic nature of a system composed from services requires a dynamic runtime approach which is able to detect and respond to emergent problems in the service execution environment, and to the problems that may

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 468–482, 2008.

arise as a result of different services being composed together. Secondly, current quality schemes offer the consumer only limited control over the quality of a service and therefore the system. In summary, the current software service quality frameworks offer the consumer:

- *Limited consumer control over service quality.* The third-party nature of software services means that a consumer has little control over the quality of services outside the static service level agreement (SLA). SLAs are intended to define the scope, level, and quality of an externally provided service together with associated responsibilities. However, they are difficult to enforce, hard to integrate with specific consumer quality strategies and provide no obvious way of ensuring runtime system quality.
- *Poor support for runtime quality.* Whilst there are initiatives for monitoring and reporting service quality failings [5,6], monitoring alone is inadequate for ensuring runtime quality. To ensure runtime quality, monitoring must be supported with effective (re)negotiation and recovery strategies.
- *Limited support for customisation.* Current quality assurance approaches for service-oriented systems are restricted to specific quality assurance schemes, limiting their scope for experimentation and customisation (i.e. variability in quality contexts).
- *Poor support for resource-restricted systems.* Quality assurance is particularly challenging for systems that operate in resource-restricted environments [7]. Not only must a service have an acceptable level of quality; it must be possible to integrate and orchestrate it within the constraints of the runtime environment.

We have developed a self-configuring quality framework that uses an adaptable service brokerage architecture, to integrate consumer strategies with monitoring, (re)negotiation, forecasting and provider reputation as a means for ensuring runtime quality. The rest of this paper is organised as follows: Section 2 reviews current approaches for addressing quality in service-oriented systems. Section 3 describes the architecture of our quality framework. Section 4 describes service strategy formulation and management. Section 5 uses a small case study to illustrate the framework. Section 6 reviews the framework and provides some conclusions.

2 Background

Service-oriented systems are distributed and composed from numerous services which can be discovered and replaced at runtime. It is possible for several service providers to offer services with common functionality, but with different non-functional qualities. Qualities can be considered as constraints over the functionality of a service [8].

A characteristic of distributed systems is the volatility of service quality [9]. It is therefore important that mechanisms are in place for managing the overall

system quality [10]. Traditionally, quality of service (QoS) has been associated with telephony and computer networking, specifying requirements on the data flowing across the network (such as latency, jitter, number of dropped packets etc.). To ensure quality in service-oriented systems, application-level QoS must also be considered [11].

2.1 Service Description, Discovery and Selection

There are several initiatives to improve the characterisation of services by including non-functional aspects in their description. These include semantic approaches, such as the Web Service Modeling Ontology (WSMO) [9] and Ontology Web Language for Services (OWL-S) [12], and non-semantic approaches such as WS-Agreement [13].

WSMO and OWL-S both share a similar goal, which is to aid the automation of service discovery, selection, composition, substitution, and invocation through richer semantics [14]. WS-Agreement is a Web service protocol used in industry for establishing an agreement between a service provider and consumer.

When integrated with the service discovery process [12], these initiatives enable service providers to differentiate themselves from other providers of similar services. Service consumers are then able to discover and select providers that best satisfy their non-functional requirements. However, such initiatives are limited if there are no services which satisfy consumer requirements, as consumers must either select the closest match or go without service. Consumers are also required to trust that providers will provide services as advertised.

2.2 Service Reputation Systems

Reputation systems, such as feedback mechanisms used by online auction sites, are designed to address issues of trust between parties who have not dealt with one another before. Reputation systems can be used to help manage quality in service-oriented systems, by helping to distinguish between low and high quality service providers [15].

A reputation-based approach to service selection is described in [16] which uses software agents that share QoS information with one another, based on their interactions with the services they are attached to. Initially, each provider has the same (or no) reputation. Over time, poor service providers develop a poor reputation which makes them less likely to be selected for use by the agents. A reputation-enhanced service discovery protocol is discussed in [17]. This enables service consumers to consider QoS issues when making service selection decisions, with fewer assumptions about the trustworthiness and reliability of providers.

Reputation systems enhance service discovery and selection processes, by incorporating feedback on providers as part of the service selection criteria. This requires consumers to expend valuable resources auditing the received QoS of consumed services, and then providing feedback to a reputation system.

Reputation systems are also limited when there are no services which satisfy consumer requirements.

2.3 Service Negotiation

Service negotiation can bring software composed from services closer to meeting consumer requirements, through the formation of SLAs between service providers and consumers. Service providers can also benefit from negotiation, by utilising spare resources to provide a better QoS to those consumers who are prepared to pay an additional cost.

The negotiation of SLAs has been an active research area in the Web service community for several years. WS-Agreement [13] enables the specification of an agreement between a service provider and consumer, and provides a protocol for the creation of an agreement using agreement templates. The Web Service Level Agreement (WSLA) [18] is a similar initiative for defining SLAs, and describes how SLAs may be monitored for compliance. SLA negotiation has also seen considerable interest in the agent [16,19] and grid [20] communities.

Current initiatives are primarily concentrated on the negotiation of single services, and have not focused much on the negotiation of end-to-end QoS constraints [19]. Current initiatives also lack the ability themselves to effectively monitor service agreements for compliance.

2.4 Service Monitoring

Monitors are required to determine if services actually meet the terms and conditions agreed between service consumers and providers [21]. Monitors are also used to detect emergent properties that arise as a consequence of services interacting with each other through composition. Service providers can also impose conditions of use upon a service consumer, which may be monitored for compliance. The motivation for monitoring is to enable the quality management of services and service compositions, in response to problems such as networking issues, changes in the environment and emergent system properties.

Monitoring approaches used in service-oriented systems include: open and closed-loop control systems [6], assertion-based techniques [22] and approaches using late-binding and reflection [5]. Open and closed-loop techniques are used by service providers to stabilise service-oriented software, by collecting runtime information on services and feeding it to service controllers. In assertion-based approaches, pre-conditions and post-conditions are asserted on services and their non-functional properties, such as business processes, communication protocol preferences, organisational licensing and authentication.

Current initiatives to monitoring are largely manual activities. For example, service compositions specified as BPEL processes and annotated by a system designer with comments describing the monitoring to be performed [22], make it difficult to support quality management in a meaningful way. Such approaches are limited in handling problematic services, and do not support advanced techniques such as service renegotiation.

3 Quality Architecture

Fig. 1 shows the architecture of our proposed quality framework. The framework has been developed using the Jini[1] SOA, but is flexible enough to be applied to other SOAs such as Web services. This flexibility is achieved through implementation-specific connection interfaces. The Jini SOA was chosen primarily for its service discovery mechanism, relatively small footprint, and to facilitate the evaluation of the framework in resource-constrained environments.

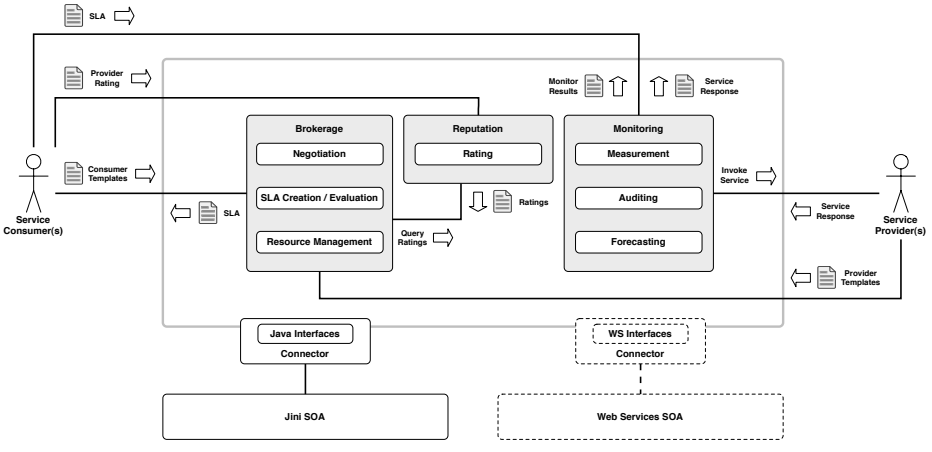


Fig. 1. Framework overview

The quality framework comprises mechanisms for discovering, brokering, monitoring and rating services and their providers (see Fig. 1). The next sections discuss each of these in turn.

3.1 Brokerage Architecture

Existing brokerage models [23,19] focus on particular methods of negotiation, and are not engineered to be integrated with monitoring and reputation processes. We have developed a brokerage approach which provides a structural framework for integrating different methods of negotiation, monitoring and reputation, and supporting the requirements of automated service negotiation and renegotiation in SOA. Our brokerage model, shown in Fig. 2, is based on a factory architecture which creates individual brokers for service consumers and providers on demand.

A service provider uses the service discovery mechanism to locate a brokerage service provider, which in turn supplies it with a broker. The service provider supplies the broker with templates describing the negotiation models to use, decision algorithms and strategies for creating and evaluating proposals. These

[1] Jini – Sun Microsystems SOA: `http://www.jini.org/`

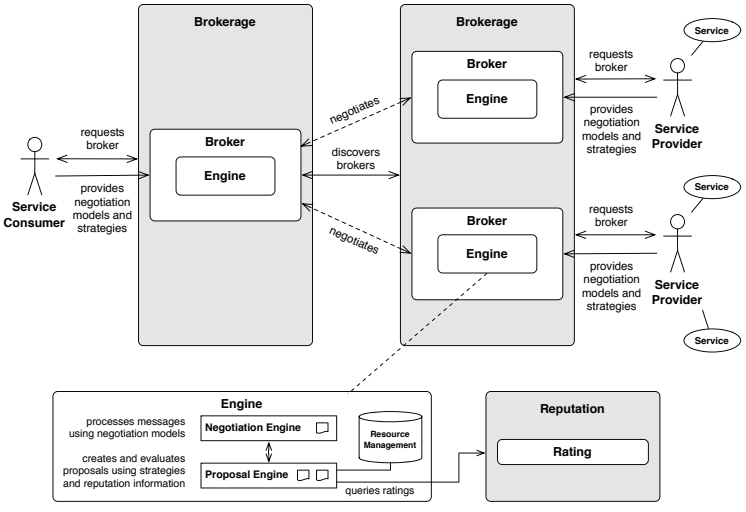


Fig. 2. Service brokerage architecture

templates are provided to an engine builder interface, which provides the broker with an engine for processing negotiation messages and service proposals. Providers also provide additional information enabling their brokers to perform service resource management on their behalf. For most types of negotiation, provider brokers enter a *passive* waiting state until they receive negotiation requests from consumer brokers.

Consumers locate service brokers similarly to service providers. However, there is no service resource management performed on the consumer side. Once initialised, consumer brokers typically enter an *active* state and use the service discovery mechanism to seek out brokerages that contain brokers of the service types required by the consumer, up to a specified limit. There are many different models of negotiation and types of negotiation decision algorithms. Fixed-pricing, auction, reverse auction and bargaining negotiation models are identified in [24]. Barter/bargaining models, request for quotes (RFQs) and auctions are identified in [25].

The framework architecture is pluggable, enabling a variety of negotiation models, decision algorithms and proposal strategies to be used. The framework currently supports two negotiation models. The first is a fixed-price negotiation model (cf. catalogue shopping) with a decision algorithm that accepts only if all qualities are within their respective range as specified by the strategy. The second type is a bargaining model based on static strategies. With the bargaining model, consumer brokers negotiate a single quality at a time. The counter proposal from the provider broker contains not only its offer for that quality, but offers for any other qualities which are related to that quality. To avoid deadlock, consumer brokers must determine any other qualities which have changed since the last proposal, and agree not to negotiate them later in the session. Once the consumer broker has finished negotiating with the set of discovered provider

brokers for each service, each possible composition is ranked, and the service proposals for the most acceptable composition are accepted. The remaining proposals are rejected but are recorded in a negotiation cache. The negotiation cache enables brokers to record the negotiation behaviour and proposals provided by other brokers they have previously interacted with. The negotiation process is initiated and led by the service consumer (see activity diagram in Fig. 3).

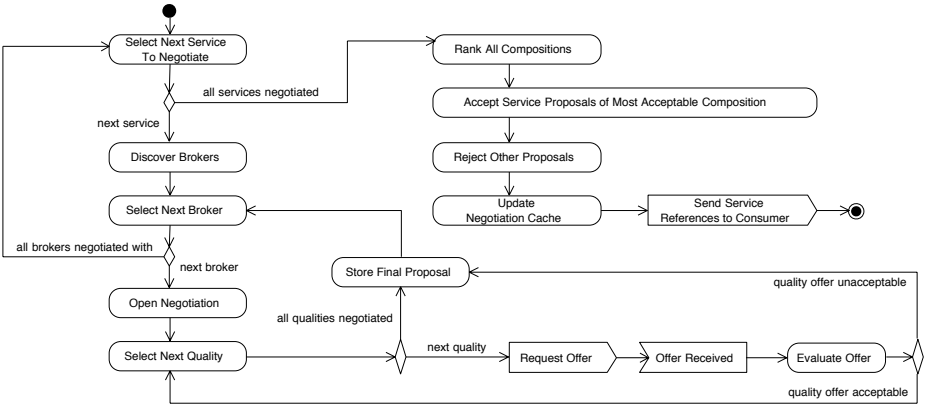


Fig. 3. Negotiation process implemented by engine in Fig. 2

Brokers share a common negotiation protocol for exchanging service proposals. The negotiation protocol currently supported by the framework is based on the primitives and protocol described in [24].

3.2 Monitoring Process

The framework actively monitors the quality of negotiated services for violations and failings at runtime. Changes in service quality are continuously evaluated against system composition acceptability levels, and an early renegotiation and replacement automatically initiated for failing services. Monitors are implemented as dynamic proxies (cf. decorator pattern), allowing for the creation of monitors at runtime which transparently intercept requests and responses between consumers and providers.

We have adopted a passive monitoring mechanism, which has the advantage that no additional load is placed on the consumer or provider of a service. In addition, the provider cannot differentiate between consumer and monitor requests (see Fig. 4). The monitoring service also provides a mechanism for auditing data collected by another party. The advantage with this approach is that the consumer does not have to expend additional resources performing auditing. However, the consumer expends additional resources in collecting data and providing it to the auditing service. Another provided approach enables a service to be monitored independently (cf. probed) from the service consumer. This provides an advantage for the consumer, but places additional load on the

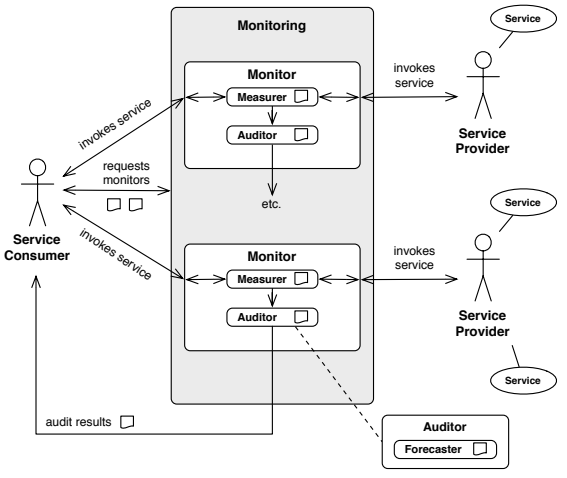


Fig. 4. Service monitoring architecture

provider. Furthermore, the provider may be able to distinguish monitor requests from consumer requests, and respond to each differently.

Auditors compare measured service qualities to those specified in the service contract. If a measured quality does not conform to its contracted value and constraints, the audit signals a *failed* quality contract violation. When consumers are informed of detected problems, they instruct their broker to renegotiate a new contract. If renegotiation fails, brokers attempt to secure service from an alternate provider (see Fig. 5).

The auditing data provided to the consumer has an overall result. If a pre-invocation service audit has passed, the consumer informs the monitor to invoke the service. If it has failed, the consumer can elect not to invoke the service and request its broker to renegotiate the problematic service. If a post-invocation service audit has passed, the consumer informs the monitor to continue monitoring the service. If it has failed, the consumer can again request its broker to renegotiate. When requesting renegotiation, the consumer provides its broker with a service contract created from the auditing data, which forms the basis for any renegotiation attempt.

When renegotiating a failed quality, the consumer broker assumes the provider broker is unable to guarantee a value better than the value which caused the audit to fail. Instead, the consumer broker expects some offer of improvement in another service quality or qualities. Improvements in other qualities should raise the overall acceptability of the renegotiated service to a more acceptable level. The consumer broker compares the acceptability of the renegotiated service proposal, with the proposals made by any other provider brokers in its negotiation cache. If the renegotiated service proposal is still the most acceptable, the consumer broker accepts the renegotiated proposal and continues using the service.

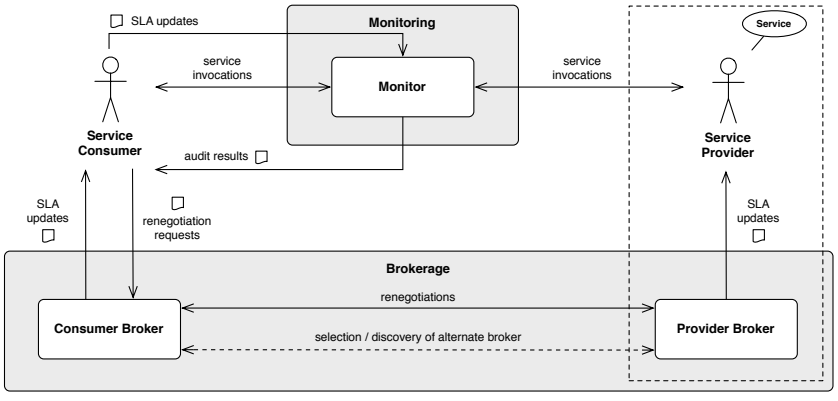


Fig. 5. Monitoring and renegotiation

If the renegotiated service proposal is no longer the most acceptable, the consumer broker attempts to gain service from an alternate provider, and rejects the renegotiated service proposal if successful.

Forecasting is an additional process which complements the auditing performed by service monitors. Service consumers specify the type of forecasting model to be used, and any additional parameters and values. During the audit, trends in measured service qualities are forecast. This enables the auditor to estimate in advance when a given service quality is about to fail. If a particular quality is estimated to fail, the audit signals a *failing* quality. If no contract violation is detected, and no problems are forecast, the audit signals an acceptable service. The framework currently provides forecasting models based on moving averages and exponential smoothing (such forecasting techniques are discussed in [26]).

3.3 Reputation Process

The reputation service provides a method for service consumers to rate the services of providers they have used. A service instance is rated once by a consumer, and is done once the service has been unleased. Services are unleased when either a contract is violated, or the service lease expires. All ratings received for the same provider and service type are combined, to develop the overall reputation for the provider's ability to supply that particular service type in accordance with negotiated contracts.

The reputation service provides a query method to determine the overall rating of a provider for a given service type. Consumer brokers use this method to limit negotiation sessions to those providers which have an acceptable level of reputation (as defined in the strategy provided by the consumer). If a consumer has previously used the service(s) of a particular provider, the consumer's own rating is combined with the global rating provided by the reputation service, according to weights specified in the consumer's strategy. Provider brokers also

query for any rating a consumer may have given its provider in the past, before agreeing to provide a service to the consumer.

4 Service Strategy and Management

We have implemented strategy templates, based on a quality ontology which enables services to be described and negotiated in terms of their non-functional qualities and constraints. Non-functional attributes are specified using metadata interfaces, which enable a *design by contract* [27] approach.

Both consumer and provider strategies include three attributes which describe how any available reputation information should be used. The *reputation threshold* is used by consumer brokers to limit negotiation to those providers who have a level of reputation above a certain value. The threshold is also used by provider brokers, to limit negotiation to those consumers who have previously rated the provider above a certain value. The other two attributes are *proposal weight* and *reputation weight*, which are used when computing the overall acceptability of an offer. These attributes respectively indicate the importance of a service proposal, when compared to the reputation of the consumer or provider which made the proposal. Consumer strategies include two further attributes, *personal experience* and *global experience*, to weight the consumer's own experience of a particular provider and service against the experience provided by other consumers.

4.1 Service Acceptability

Each quality, operation and service in a 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 using the following formula, based on the acceptability formula given in [25], but extended to factor in the weight of a single quality Q_w as it pertains to an overall QoS. Let Q_p be the value proposed for the quality, Q_l the least acceptable value for that quality and Q_m the most acceptable value for that quality.

$$Q_a = \left| \frac{Q_p - Q_l}{Q_m - Q_l} \right| * Q_w$$

The formula is used for proposed values which fall within the range of acceptable values defined by the least and most acceptable values. Whether these are high or low values depends on whether greater or lesser values are more acceptable or less acceptable e.g. for a response time quality, a greater number may be less acceptable and a smaller number more acceptable. This equation is extended if reputation information is available for the creator of the proposal. If there exists any prior personal experience of the proposal creator, the reputation R is computed from both the global rating R_g provided by the reputation service, and the local personal experience R_l. Let G_w be the weight assigned to global experience and L_w be the weight assigned to local personal experience,

so that $0 \leq G_w \leq 1$ and $0 \leq L_w \leq 1$, and $G_w + L_w = 1.0$. The overall rating R is then defined as:

$$R = (R_g * G_w) + (R_l * L_w)$$

The total quality acceptability Q_{ta} is then calculated as follows. Let P_w be the weight of the proposal and R_w the weight of the reputation information, so that $0 \leq P_w \leq 1$ and $0 \leq R_w \leq 1$, and $G_w + P_w = 1.0$.

$$Q_{ta} = (Q_a * P_w) + (R * R_w)$$

4.2 Service Composition

The framework is capable of negotiating a composition, but is not responsible for the actual composing of services (this activity is left to the service consumer). Each service in the required composition is individually-weighted, enabling the specification of compositions where one service is more critical than another. Calculating the acceptability of every possible composition is an NP-hard problem. For example, if a composition with 3 different services is required and there are 10 providers for each type of service, 10^3 composition comparisons are required in order to calculate the acceptability of every possible composition. Provider reputation can be used to limit negotiation to a subset of the available providers, but the linear programming approach still does not scale. Every additional service in a composition introduces another order of magnitude to the number of compositions which must be compared. There are several proposed solutions to this optimisation problem [23].

5 Case Study

We have developed a small case study to evaluate the framework and to visualise the framework processes and system quality at runtime. Simulated consumer devices, each with different resource constraints and requirements, execute a navigation application comprised of location, traffic, weather, street maps, and information services.

First, the system invokes the location service to obtain the location of the consumer. The location data is passed on to the maps, weather and traffic services to obtain graphical maps, weather and traffic information within an n metre radius of the consumer. Traffic and map information are integrated to highlight traffic conditions on the roads. The map information can also be used to request information on places of interest (e.g. restaurants, shops, parking, tourist attractions etc.). The consumers of this application each simulate a navigation device, such as a mobile phone, internet tablet, or automobile navigation system. Variances in consumer requirements mean that different providers are more acceptable to different consumers. In addition, the runtime environments of the different consumers vary from remote locations to busy metropolitan areas, meaning that invoking services will result in widely different responses.

Several providers of each service type are registered with the Jini discovery service. Each provider offers services with arbitrary differences in levels of QoS and cost, making some providers more acceptable than others for the different consumer devices. Once SLAs have been negotiated for each service type in the navigation composition, monitors are attached and each service in the navigation process is invoked as required.

Providers are doped in a variety of ways, so that they occasionally violate negotiated SLA qualities. Monitors detect these violations by auditing service invocations and forecasting trends. Once notified of a violation, consumers instruct their brokers to renegotiate the SLA if the monitored QoS is still within acceptable limits. If the monitored QoS is not within acceptable limits, if renegotiation is unsuccessful or if the service continues to deteriorate, the consumer may switch to an alternate provider if available, depending on its strategy.

5.1 Visualising Framework Processes

Software has been developed for visualising the framework processes at runtime. The negotiation viewer displays the consumers actively negotiating a service or service composition. The viewer provides a means to view negotiation sessions between each consumer broker and the provider brokers they negotiate with.

Fig. 6 shows a negotiation session between a consumer device and a *MapService* provider. The final proposal was rejected as another provider was more

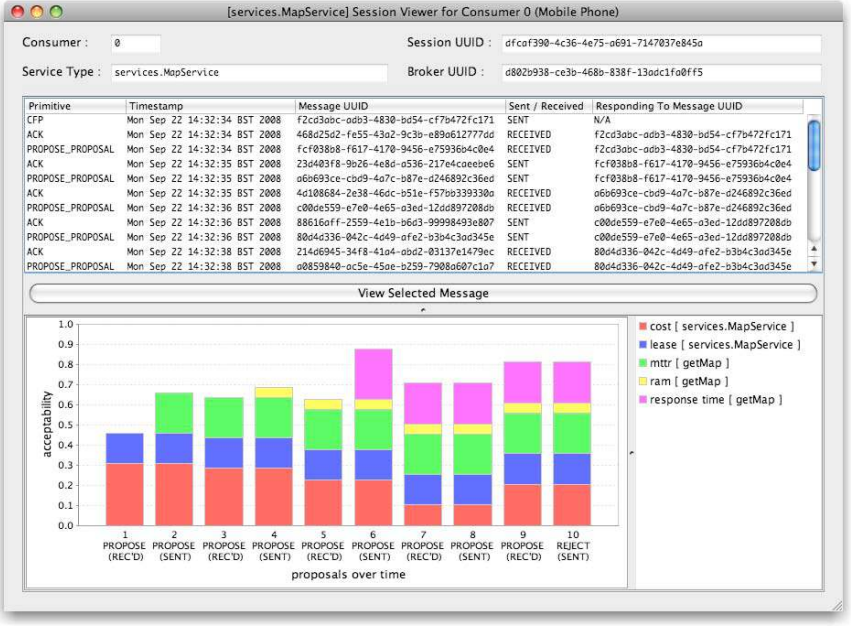
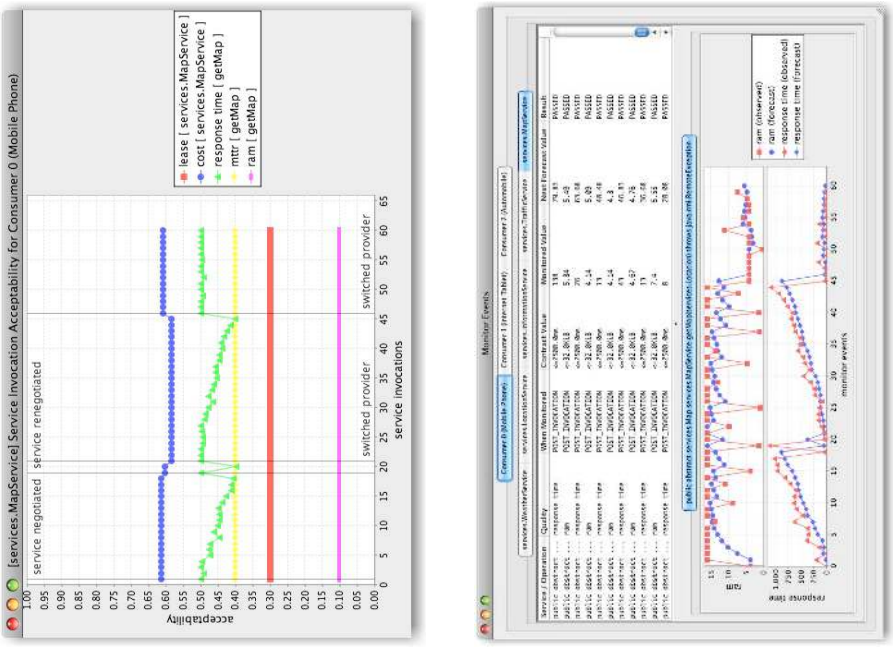


Fig. 6. Negotiation session viewer



(a) Service invocation acceptability **(b)** Monitor event viewer

Fig. 7. Example simulation views

acceptable. The first table column contains the negotiation primitive of each negotiation message; the second column contains the timestamp; the third column contains the universally-unique identifier (UUID) of the message; the fourth column indicates whether the message was sent or received by the consumer broker; and the fifth column contains the UUID of the message being responded to (if any). The user can view the contents of any message and service proposal.

The user can also view and compare negotiated service proposals from each provider of a particular service, and view the invoked acceptability of individual services (see service example in Fig. 7a) and service compositions.

The monitor viewer (see Fig. 7b) shows each consumer currently executing services. Each table entry represents a single monitoring event. The entry includes the UUID and type of the service monitored, the service operation invoked, the quality audited (profiled RAM usage and response time in the example), the SLA quality value, the observed quality value, when the audit occurred, and audit result. The observed and forecast values for different qualities are plotted on a chart where trends in service qualities can be observed over time. Consumers react to failed/failing services by requesting their brokers to renegotiate, and may switch provider if renegotiation fails (as shown on Fig. 7a).

The user is also able to view the current and historical reputation of each provider of each service type, based on the ratings from consumers.

6 Conclusions

This paper has presented a runtime quality architecture for service-oriented systems, that uses a novel service brokerage model to provide a framework for integrating consumer strategies with different negotiation and monitoring techniques. The framework enables consumers to negotiate service agreements which are closer to their requirements, and compensates providers accordingly. Services are monitored during runtime for compliance with the negotiated agreements. Problematic services are renegotiated, or alternate sources of service provision are sought. Our approach does not provide a definitive solution to all quality assurance problems that plague service-oriented systems. However, we believe it offers a real alternative to current quality frameworks and provides some answers to the problems that we set out at the beginning of this paper.

Current quality management initiatives allow the consumer only limited control over the quality of provided services; we have developed a framework that allows consumers to specify quality-weighted services and to associate these with consumer strategies. Effective runtime quality assurance must combine monitoring with effective recovery and self-management strategies. We have developed a portable self-managing quality architecture, that uses a lightweight service brokerage model to integrate pluggable monitoring and negotiation with consumer quality strategies for ensuring runtime quality. Lastly, we have demonstrated the effectiveness of the quality architecture with the simulation of a small resource-constrained example. We are currently investigating improvements to the framework to support runtime quality more efficiently in resource-constrained system environments.

We are also looking at better ways of expressing and incorporating fallback mechanisms as part of the consumer strategy, and better ways of integrating a quality of service ontology. Fallback mechanisms would provide the consumer with the ability to function in the event that certain framework components or services are unavailable.

References

1. Turner, M., Budgen, D., Brereton, P.: Turning software into a service. Computer 36(10), 38–44 (2003)
2. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River (2005)
3. Sommerville, I.: 31. In: Software Engineering, 8th edn. Addison Wesley, Reading (2006)
4. Lüders, F., Flemström, D., Wall, A.: Software component services for embedded real-time systems. In: Proc. Fifth Conference on Software Engineering Research and Practice in Sweden, Västerås, Sweden, Mälardalen University, October 2005, pp. 123–128 (2005)
5. Baresi, L., Ghezzi, C., Guinea, S.: Smart monitors for composed services. In: ICSOC 2004: Proceedings of the 2nd international conference on Service oriented computing, pp. 193–202. ACM Press, New York (2004)
6. Hoffman, B.: Monitoring, at your service. Queue 3(10), 34–43 (2005)

7. Milanovic, N., Richling, J., Malek, M.: Lightweight services for embedded systems. Wstfeus 00, 40 (2004)
8. O'Sullivan, J., Edmond, D., Hofstede, A.T.: What's in a service? Towards accurate description of non-functional service properties. Distrib. Parallel Databases 12(2-3), 117–133 (2002)
9. Toma, I., Foxvog, D., Jaeger, M.C.: Modeling QoS characteristics in WSMO. In: MW4SOC 2006: Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006), pp. 42–47. ACM Press, New York (2006)
10. Menascé, D.A., Ruan, H., Gomaa, H.: QoS management in service-oriented architectures. Perform. Eval. 64(7-8), 646–663 (2007)
11. Woodside, C.M., Menascé, D.A.: Guest editors' introduction: Application-level QoS. IEEE Internet Computing 10(3), 13–15 (2006)
12. Martin, D.L., et al.: Bringing semantics to web services: The OWL-S approach. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 26–42. Springer, Heidelberg (2005)
13. Andrieux, A., et al.: Web services agreement specification (WS-Agreement), version 2006-09-07. Technical report, Global Grid Forum (2006)
14. O'Sullivan, J.: Towards a Precise Understanding of Service Properties. PhD thesis, Queensland University of Technology (2006)
15. Jøsang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decis. Support Syst. 43(2), 618–644 (2007)
16. Maximilien, E.M., Singh, M.P.: Toward autonomic web services trust and selection. In: ICSOC 2004: Proceedings of the 2nd international conference on Service oriented computing, pp. 212–221. ACM Press, New York (2004)
17. Wishart, R., Robinson, R., Indulska, J., Josang, A.: SuperstringRep: reputation-enhanced service discovery. In: ACSC 2005: Proceedings of the Twenty-eighth Australasian conference on Computer Science, pp. 49–57. Australian Computer Society, Inc., Darlinghurst (2005)
18. Ludwig, H., et al.: Web service level agreement (WSLA) language specification (2003), `http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf`
19. Yan, J., et al.: Autonomous service level agreement negotiation for service composition provision. Future Gener. Comput. Syst. 23(6), 748–759 (2007)
20. Czajkowski, K., Foster, I.T., Kesselman, C., Sander, V., Tuecke, S.: Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2002. LNCS, vol. 2537, pp. 153–183. Springer, Heidelberg (2002)
21. Benjamim, A.C., Sauvé, J., Cirne, W., Carelli, M.: Independently auditing service level agreements in the grid. In: Proceedings of the 11th HP OpenView University Association Workshop, HPOVUA (2004)
22. Baresi, L., Ghezzi, C., Guinea, S.: Towards self-healing service compositions. In: PRISE 2004, First Conference on the PRInciples of Software Engineering, Buenos Aires, Argentina (November 2004)
23. Menascé, D.A., Dubey, V.: Utility-based QoS brokering in service oriented architectures. ICWS 0, 422–430 (2007)
24. Li, H.: Automated E-business Negotiation: Model, Life Cycle and System Architecture. PhD thesis, University of Florida (2001)
25. Lock, R.: TRANSACT (Tool for Real-time Automated Negotiation of Secure Authorisation ContracTs). PhD thesis, Lancaster University (2005)
26. Wolski, R.: Dynamically forecasting network performance using the network weather service. Cluster Computing 1(1), 119–132 (1998)
27. Meyer, B.: Applying "design by contract". Computer 25(10), 40–51 (1992)