Journal of King Saud University – Computer and Information Sciences (2015) 27, 323–333

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

Bootstrapping quality of Web Services

Zainab Aljazzaf *

Department of Information Science, College of Computing Sciences and Engineering, Kuwait University, Kuwait

Received 18 June 2014; revised 13 November 2014; accepted 9 December 2014
Available online 29 June 2015

KEYWORDS

Services;
Web Services;
Quality of services;
Bootstrapping;
Service Oriented Architecture

Abstract A distributed application may be composed of global services provided by different organizations and having different properties. To select a service from many similar services, it is important to distinguish between them. Quality of services (QoS) has been used as a distinguishing factor between similar services and plays an important role in service discovery, selection, and composition. Moreover, QoS is an important contributing factor to the evolution of distributed paradigms, such as service-oriented computing and cloud computing. There are many research works that assess services and justify the QoS at the finding, composition, or binding stages of services. However, there is a need to justify the QoS once new services are registered and before any requestors use them; this is called bootstrapping QoS. Bootstrapping QoS is the process of evaluating the QoS of the newly registered services at the time of publishing the services. Thus, this paper proposes a QoS bootstrapping solution for Web Services and builds a QoS bootstrapping framework. In addition, Service Oriented Architecture (SOA) is extended and a prototype is built to support QoS bootstrapping. Experiments are conducted and a case study is presented to test the proposed QoS bootstrapping solution.

1. Introduction

To build a distributed application from Web Services, the application developer, or service requestor, may need to select Web Services from different service providers. Because there are many Web Services with similar *functionalities*, service requestors need to differentiate among them. The only differentiating factor between similar Web Services may be their *non-functional* properties, which can be considered as criteria for service selection. Quality of services (QoS) has been used as a non-functional property for selecting services (Papazoglou et al., 2006; Maximilien and Singh, 2004; Dragoni, 2009; Ying-Feng et al., 2006; Huhns and Singh, 2005; Zhang et al., 2012; Kalepu et al., 2003; Kim and Doh, 2007; Liu et al., 2004; Zheng et al., 2014; Rajeswari et al., 2014).

For example, one approach in service selection (Ran, 2003) involves the case where the Web Service registry can capture the QoS provided by the service provider and the QoS required by the service requestor and accordingly match the two while discovering the service, to select the best match from services with similar functionality. In this scenario, a service requestor may need a service that has a low response time, which is considered a QoS property of the service. Therefore, a service with

* Tel.: +965 24633223.
E-mail address: dr.zainab@ku.edu.kw.
URL: http://www.isc.ku.edu.kw/.
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2014.12.003
1319-1578

a lowest response time will be selected by the requestor from many similar services with different response times.

Many studies have been conducted to examine QoS compliance by monitoring Web Services or collecting quality ratings from the users (Papazoglou et al., 2006; Kim and Doh, 2007; Zhang et al., 2012; Liu et al., 2004; Huhns and Singh, 2005; Kalepu et al., 2003; Zheng et al., 2014). Thus, considering the issues related to the Web Services and distributed paradigms, QoS is an important contributing factor to the evolution of distributed paradigms.

Service providers may register their Web Services claiming the services' QoS. However, the Web Service brokers need to justify the conformance of the QoS to the published specification. Many studies have been conducted to justify the QoS at the time of selecting the services or at run time of services (Maximilien and Singh, 2004; Dragoni, 2009; Zhang et al., 2012; Kalepu et al., 2003; Kim and Doh, 2007; Liu et al., 2004; Zheng et al., 2014). To the best of our knowledge, few attempts were done to justify the QoS once new services are registered and before any requestors use them. Therefore, there is a need to ensure the QoS, especially for new services (new comer) that no service consumer has tried using before, and for which the justification of the QoS measurements is not available a priori, so called Bootstrapping QoS. Bootstrapping is the process of evaluating the QoS of the newly registered services at the time of publishing the services.

In this paper, a solution for bootstrapping QoS is introduced that assesses the QoS attributes for the newly registered Web Services. The main contribution lies in the automated approach for QoS bootstrapping at the Publish time and before any requestor requests the Web Service. As a result, the justification of QoS for new Web Services will be available at the time of publishing the Web Services, thus:

- there is no need to test the Web Services at the time of Finding the service;
- the Finding operation will be faster;
- increases the opportunity for new Web Services to be selected by requestors;
- increases the level of trust in such services.

Accordingly, this work proposes a QoS bootstrapping solution for Web Services that includes a QoS model, QoS bootstrapping framework, SOA extension to support bootstrapping QoS, and prototype. Consequently, experiments are conducted to evaluate the bootstrapping solution.

The rest of the paper is organized as follows. The background is presented in Section 2. Section 3 presents the related work. The proposed QoS bootstrapping technique is included in Section 4 and it covers QoS model and QoS bootstrapping framework. Section 5 covers the SOA extension to support bootstrapping QoS. The prototype is presented on Section 6. The experiment, evaluation, and case study are demonstrated in Section 7. Section 8 concludes the paper.

2. Background

This section provides the background about Web Services and QoS and service-oriented computing paradigm, as given below.

2.1. Web Services and quality of service

The development of a distributed software system requires the interaction of services and the use of resources from diverse organizations throughout the Web. A service is "a discrete unit of business functionality that is made available through a service contract" (Rosen et al., 2008), which includes a service interface, service documents, service policies, and QoS. Services perform functions from simple requests to complicated business processes.

Services can be implemented using Web Service technology. Web Services are an emerging technology that enables applications running from different machines over the Web to integrate and exchange data regardless of their platform, hardware, operating system, and languages (Papazoglou, 2012).

A Web Service is defined by the World Wide Web Consortium (W3C) as "a software application identified by a URI, whose interface and binding are capable of being defined, described and discovered by XML artefacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols" (Ferris et al., 2004). Similarly, IBM defines a Web Service as "self-contained, self-described, dynamically discovered applications with Internet-based interfaces" (Yuan and Long, 2002).

Accordingly, a Web Service is a new breed of Web applications, which is modular and can be published, located, and invoked across the Web. Once a Web Service is deployed, other applications can discover and invoke it. Web Services are loosely coupled, platform-neutral, reusable, and distributed software components (Yuan and Long, 2002).

Web Services are based on common standards, such as Extensible Markup Language (XML), and existing technologies, such as Hypertext Transfer Protocol (HTTP). The key to Web Services' success is the open standards that facilitate the interoperability among different parties (Yuan and Long, 2002). Web Services technology's main protocols include Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI) (Papazoglou, 2012).

SOAP is an XML-based standard messaging protocol, using HTTP as a means of transport and for circumventing the firewalls. WSDL is the service representation language used to describe the interface of and access to Web Services. This description includes the operations and parameters, location, and invocation protocol of the Web Services. UDDI is a cross-industry directory standard for the description, publication, and discovery of Web Services. This standard stores the Web Service interfaces described by WSDL, categorizes Web Service information, and allows searching the directory for Web Services.

Regarding quality and QoS, the international quality standard ISO 8402 (part of the ISO 9000) describes quality as "the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs". Hoyle (2005) defines quality as "the degree to which a set of inherent characteristics fulfills a need or expectation that is stated, general implied or obligatory". The author elaborates that "quality is thought of as conformance to specification regardless of whether the specification actually meets the needs of the customer or society" Hoyle (2005). The W3C describes the

QoS requirements for Web Services as the quality aspect of a Web Service (Lee et al., 2003).

2.2. Service-Oriented Computing

Service Oriented Computing (SOC) is "a computing paradigm that utilizes services as fundamental elements to support rapid, low-cost development of distributed applications in heterogeneous environments" (Papazoglou and Georgakopoulos, 2008). To realize the potential of SOC, Service Oriented Architecture (SOA) is developed to overcome many enterprise challenges, including designing complex distributed services, managing business processes, ensuring transaction QoS, complying with agreements, and leveraging different computing devices such as personal computers and cell phones (Papazoglou and Georgakopoulos, 2008). SOA is "an architectural style for building enterprise solutions based on services" (Rosen et al., 2008).

Because SOA is concerned with an enterprise scope beyond a single application (Rosen et al., 2008), the design principles of SOA are independent of any technology (Papazoglou and Georgakopoulos, 2008). SOA can be implemented using many distributed computing technologies such as Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), and Web Services. In particular, Web Services have gained more popularity as a technology for implementing SOA because of their important features, especially their interoperability and self-description interfaces, as well as the fact that they base their development on existing ubiquitous infrastructures, such as HTTP and XML (Papazoglou and Georgakopoulos, 2008).

There are three role interactions in SOA, as shown in Fig. 1 (Papazoglou, 2012): *Web Service provider*, an organization or platform that owns, implements, and controls access to the Web Service; *Web Service requestor*, an application, Web Services, or the client who is looking for and invoking a Web Service; and *Web Service registry*, a searchable directory where the description of the Web Services is published by Web Service providers and searched by Web Service requestors.

The Web Service broker facilitates operations in SOA. First, in the Publish phase, Web Service providers register and publish their Web Services into the Web Service registry. Next, in the Find or Discovery phase, Web Service requestors query the registry for Web Services according to their functional properties. Finally in the Bind phase, the Web Service requestors bind and invoke the selected Web Service (Papazoglou, 2012). The brokerage handles the integration, customization, and governance of the access to the Web Services, all in an effort to reduce end-user complexity.

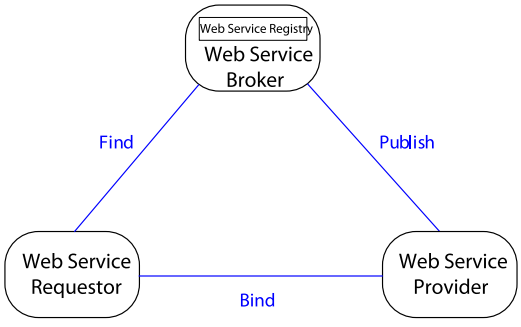

Figure 1 Web Services roles and operations in SOA Papazoglou and Georgakopoulos, 2008.

3. Related work

The research works in QoS encompass different research domains, such as network, software engineering, and service-oriented computing paradigms. Many research efforts are dedicated to the area of QoS; however, to the best of our knowledge, most of the existing work leaves open the way for how QoS parameters are bootstrapped or pre-evaluated for new published services at Publish time of services.

Many researchers assess the QoS at the Find time of services or at run time, or check the QoS compliance (Zou et al., 2009; Kalepu et al., 2003; Zhengping et al., 2007; Zhang et al., 2010; Sherchan et al., 2006; Zhang et al., 2012). For example, Zou et al. (2009) present a service selection and ranking framework. The QoS are not assessed a priori, and the services are ranked based on the published QoS by the service provider. Kalepu et al. (2003) calculate the compliance values, which involve the difference between the projected and achieved parameter values, for all of the services' Service Level Agreement (SLA) parameters. The local and global rankings of the services and their providers are evaluated based on the compliance levels of SLA parameters. Zhengping et al. (2007) monitor the behavior of services at run time, and Zhang et al. (2010) present a Web Service search engine to find desired Web Services at Find time of services. Specifically, the engine ranks Web Services by monitoring the non-functional QoS characteristics of Web Services. Sherchan et al. (2006) measure the compliance of QoS attributes by comparing the projected values agreed-upon in the SLA and the delivered values obtained from the performance monitoring system after using the service. Zhang et al. (2012) present a QoS based computational model for Web Service selection through an improved normalization method. An appropriate service would be recommended to the service requestor by the QoS registry, which is responsible for collecting the QoS. In those models, however, the QoS were not assessed at Publish time of the services, and thus, they did not consider bootstrapping the QoS.

In contrast to QoS bootstrapping, QoS prediction aims at providing personalized QoS value prediction for service users, by employing the historical QoS values. The research problem of QoS prediction is how to accurately predict the missing QoS values by employing the available previous collected QoS values from other users' experience (Zheng et al., 2014). There are many works in QoS prediction for new users (Wang et al., 2013; Baraki et al., 2013; Ge et al., 2010; Yu, 2012). For example, Wang et al. (2013) propose a model that integrates QoS prediction and leads to a composition result that fulfills and maintains the user requirement. Baraki et al. (2013) present two algorithms to predict QoS and Quality of Experience (QoE), where the predictions are based solely on previously collected QoS and QoE data. Yu (2012) presents a strategy that uses decision tree learning to bootstrap service recommendation systems. However, the above methods recommend services to users based on previous collected QoS values from

other users' experiences and do not bootstrap QoS of new services.

Zheng et al. (2014) mention that although QoS of Web Services has been investigated intensively, but there is lack of services' pre-evaluation and real-world QoS data sets. Accordingly, the availability of the comprehensive real-world Web Service QoS data sets is important in validating various QoS-based approaches. Thus, pre-evaluating services and providing the evaluated QoS in advance helps the service providers and consumers of such services. The authors conduct several large-scale QoS evaluations of real-world Web Services. However, the authors did not build a framework and they focused on investigating three user-observed QoS properties only, which are failure probability, response time, and throughput.

Regarding the bootstrapping framework in the literature, the only bootstrapping approach was proposed by Rosenberg et al. (2006). The authors proposed an evaluation approach for bootstrapping QoS attributes of Web Services that provides a set of up-to-date QoS attributes for Web Service selection. The authors mainly address performance and dependability related QoS attributes, which are response time, latency, and execution time, availability, and robustness. Although the authors build a framework, but they did not address service selection based on QoS or apply it to service computing applications such as SOA.

Therefore, the work in this paper overcomes the limitations in the literature and provides a full bootstrapping solution that includes a QoS model, QoS bootstrapping framework, SOA extension to support service selection based on the bootstrapped QoS, and prototype.

4. QoS bootstrapping technique

This section describes the QoS Bootstrapping solution, which starts by identifying the QoS model followed by the QoS bootstrapping framework, as given below.

4.1. The QoS model

This section presents the QoS model used in this work. A QoS model expresses QoS attributes for Web Services, as shown in Table 1.

Table 1 The QoS model.

QoS	
Latency	Security
Execution time	Transaction ACID
Response time	Regulatory
Throughput	Exception handling
Transaction time	Interoperability
Availability	Competence
Reliability	Honesty
Scalability	Usability
Integrity	Testability
Capacity	Stability
Robustness	Supported standards
Accuracy	Modifiability
Accessibility	Execution price
Timeliness	Other QoS

The following presents evaluation approaches for some of the QoS (Lee et al., 2003; Mathijssen, 2005; Yu et al., 2007):

- Latency ($QoS_{Latency}$):
  Latency is "the round-trip delay (RTD) between sending a request and receiving a response" Lee et al., 2003.
- Execution time ($QoS_{Execution}$):
  The execution time of a service is the time taken by the service to execute and process its sequence of activities.
- Response time ($QoS_{Response}$):
  The response time of a service is the time required to process and complete a service request; the response time includes the execution time and the latency. The following is the formula to evaluate the response time:$QoS_{Response} = QoS_{Execution} + QoS_{Latency}$Fig. 2 shows a graphical representation of the three time frames, latency time, execution time, and response time, and identifies the relations between them. Moreover, the figure demonstrates the evaluation technique for each QoS. In this work, the wrapping time is not considered.
- Throughput ($QoS_{Throughput}$):
  The throughput of a service refers to the number of requests a service can process per unit of time. Throughput depends on the power of the service machines and it is measured by sending many requests over a period of time and then counting the number of responses. The following is the formula to evaluate the throughput:

$$QoS_{Throughput} = \frac{\text{Number of requests}}{\text{time period}}$$

- Availability ($QoS_{Availability}$):
  Services should be available for direct invocation. The availability of a service is the probability that a service is up, present, and accessible to use. The following is the formula to evaluate the availability: $QoS_{Availability} = \frac{\text{Uptime}}{\text{Total time}}$
- Reliability ($QoS_{Reliability}$):
  The reliability or success rate of a service means the ability of a service to perform its function under the stated conditions correctly with either "no fail" or "response failure to the user" for a specific interval of time, and is related to availability (Lee et al., 2003). Reliability can be evaluated as follows (Yu et al., 2007):$QoS_{Reliability} = 1 - \frac{n}{N*t}$where t denotes the total time a service is monitored for recording the number of failures, n is the number of failures encountered during that period, and N is the total number of

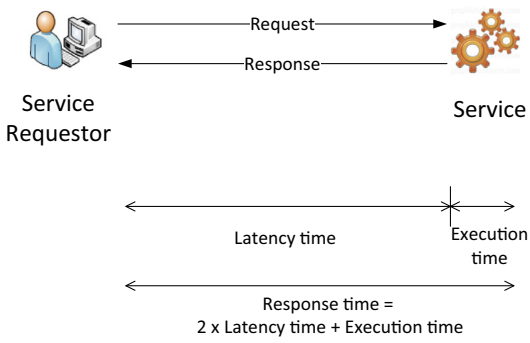

Figure 2 Latency, execution, and response time frames.

events (number of successful events plus number of failures); thus, the reliability or the success rate in one day can be derived.

- Accessibility ($QoS_{Accessibility}$):
  Accessibility refers to the service capability to serve the client's requests (Lee et al., 2003). Mathijssen (2005) denoted accessibility as follows:

$$QoS_{Accessibility} = \frac{\text{The number of user's requests that succeeded}}{\text{Total requests done by the user}}$$

  Similarly, Yu et al. (2007) calculate accessibility as "a ratio of the number of successful acknowledgements received to the total number of requests sent", as shown below:

$$QoS_{Accessibility} = \frac{\text{Number of acknowledgements received}}{\text{Total number of requests sent}}$$

4.2. QoSBF: QoS bootstrapping framework

This section presents the QoS Bootstrapping Framework, called QoSBF. Fig. 3 shows the proposed framework. QoSBF consists of several components that perform different functions. In the following, QoSBF components and their functions will be explored more.

- Web Service preprocessing:
  This component obtains the information about a Web Service that is necessary for the invocation component. The required Web Service information can include the service operation, input and output parameters of the operation and their data types, and binding information.
- Web Service invocation:
  The Web Service invocation component invokes services; specifically, it requires services' operations, input and output parameters, data types, and binding information, which is obtained by the previous component, the Web Service preprocessing component.
- QoS evaluation:
  This component is responsible for evaluating the QoS. In this work, the monitoring approach is proposed as a method for bootstrapping the QoS. In particular, a monitor bootstraps QoS, such as response time. The collected information is stored in the QoS registry and used by the matching component.
- Matching:
  The matching component supports service selection based on the bootstrapped QoS and the requestors' QoS preferences. In particular, it returns services that match requestor's preferences on a set of QoS by matching the preferences with services' bootstrapped QoS. As a result, the service broker returns a number of services with a similar functionality and different QoS. For example, a requestor may require a weather service with a low $QoS_{Response}$. The service broker may return many weather services each with different $QoS_{Response}$. Consequently, the requestor will select one with the lowest $QoS_{Response}$ as his/her preference.
- QoS registry:
  The QoS registry stores the bootstrapped QoS that is evaluated by the QoS evaluation component. In addition, it can provide the bootstrapped QoS to the matching component. Consequently, it supports searching, matching, and selecting services based on the requestors' QoS preferences.

Moreover, the QoS bootstrapping process is conducted periodically to update the bootstrapped QoS values and detect any changes in their values.

The QoS bootstrapping process works as follows: Once the Web Service is registered into a Web Service registry, the registry will obtain the necessary information about the Web Service, such as Web Service interface and parameters. Then, the Web Service invocation component will invoke the Web Service. Consequently, the QoS evaluation component will evaluate the QoS. Accordingly, the bootstrapped QoS will be stored in the QoS registry for later use by the matching component. Consequently, the QoS will be bootstrapped periodically before any requestor starts searching for services; i.e., before the Find operation of services.

5. SOA extension to support QoS bootstrapping

This section presents the SOA extension for supporting QoS bootstrapping. Fig. 4 shows the proposed SOA extension; the architecture adheres to that of the SOA model and has the same SOA roles and operations with additional component and link interaction.

The additional component is the QoS bootstrapping framework, which is the QoSBF. QoSBF is added on the service broker side; accordingly, the broker is responsible for conducting the QoS bootstrapping process at the Publish time of services. The additional link interaction includes a QoS bootstrapping link between the service broker and service provider to bootstrap the QoS of registered services. In addition, the Find link

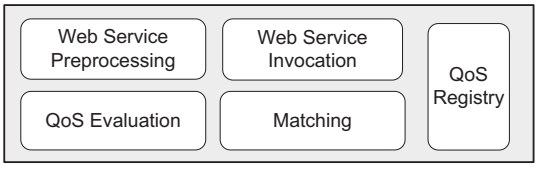

Figure 3 QoSBF: QoS Bootstrapping Framework.

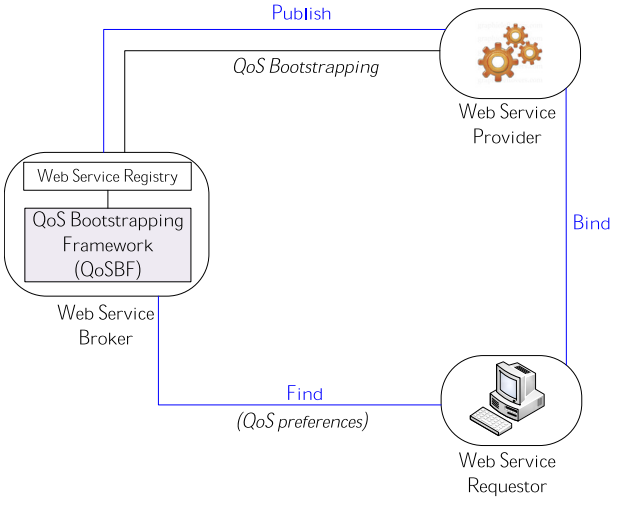

Fig. 4 SOA Extension for Supporting QoS Bootstrapping.

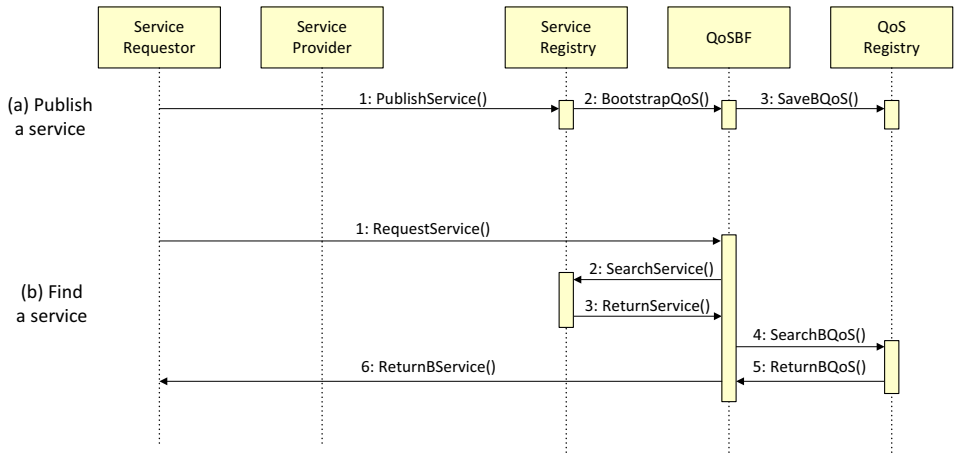

Figure 5 Publish and request operation of a service.

support selecting services based on requestors' QoS preferences.

Fig. 5 shows the sequence diagram of the Publish and Find operations of a service in the extended SOA. Service providers need to publish their services in a service registry. In the Publish operation (part a in the figure), the service provider publish services in the service registry. Subsequently, QoSBF starts the bootstrapping process and stores the bootstrapped QoS in the QoS registry. In the Find operation (part b in the figure), the service broker supports the selection of services based on the services' QoS properties based on a set of requestors' QoS preferences. In this operation, QoSBF obtains the find request from the requestor and first discovers the service registry for services that match the requestor's functional preferences, and then the QoS registry for services that match the requestor's QoS preferences. Consequently, the services that satisfy the requestor's preferences are returned to the requestor.

6. QoS bootstrapping framework prototype

Analytical and empirical studies are the basis of any modeling effort. Accordingly, the analytical QoSBF model discussed in this work requires empirical analyses to evaluate the QoS bootstrapping solution, show its practical uses, and obtain the optimal outcome. This section presents the empirical studies that include a prototype of the QoSBF. Experimentation, evaluation, and case study of the QoS bootstrapping solution will be presented in Section 7.

Fig. 6 shows the QoSBF prototype based on the proposed extension of SOA. The prototype presents the elements and their relations as well as the software tools; it consists of service providers, service requestors, and a service broker. The service broker contains a service registry and the QoSBF, which conducts the QoS bootstrapping process.

In the current implementation, services are deployed on Windows machines (running Windows 8) with the following software tools and technologies: Web Service technology to implement services, WSDL to describe the Web Services, SOAP as a messaging standard, Structured Query Language (SQL) database, and SoapUI (SoapUITool, 2014) for testing and monitoring Web Services. In addition, different ping tools are used to test the network connections. The Web Services are deployed into GlassFish Server 4.

SoapUI is a functional testing tool for testing and monitoring Web Services. SoapUI parse the Web Service WSDL, invoke Web Services, and monitor Web Services. A description of bootstrapping QoS at SOAP level is studied by Rosenberg et al. (2006). However, SoapUI tool is used in this work to parse WSDL and invoke Web Services. XML, XPath, Groovy, and Java Database Connectivity (JDBC) are used within the SoapUI tool to write different scripts, including collecting the QoS, monitoring Web Services, and connecting to the database.

The QoS bootstrapping process works as follows (see Fig. 6): When a service provider publishes a service, the framework starts the QoS bootstrapping process by parsing the WSDL (1). Next, the service will be invoked (2) and then monitored (3) to evaluate the QoS. Then, the evaluated QoS will be stored in the QoS registry (4).

When a service requestor requests a service (5), the matching component returns services based on the services' *functional properties*, which is discovered from the service registry (6), and *QoS* from the QoS registry (7) based on the requestor's QoS preferences. The matched services are returned to the requestor for the selection of a specific service (8).

7. Experiment and evaluation

This section presents the experimental results of the proposed QoS bootstrapping solution with respect to some QoS. A use case and scenarios are presented. Accordingly, the experiments are conducted and the results are analyzed to show the effectiveness of the proposed approach.

The experimental methodology is as follows: A number of online Web Services are discovered from online registries. Some examples of Web Services are WeatherSoap Service, stockQuote service, IP2Geo service, shop service, and article service. Some Web Services have similar functional properties and are provided by different service providers. Subsequently, a list of QoS are selected for the experiment, which are $QoS_{Response}, QoS_{Latency}, QoS_{Execution}, QoS_{Throughput}, QoS_{Availability},$

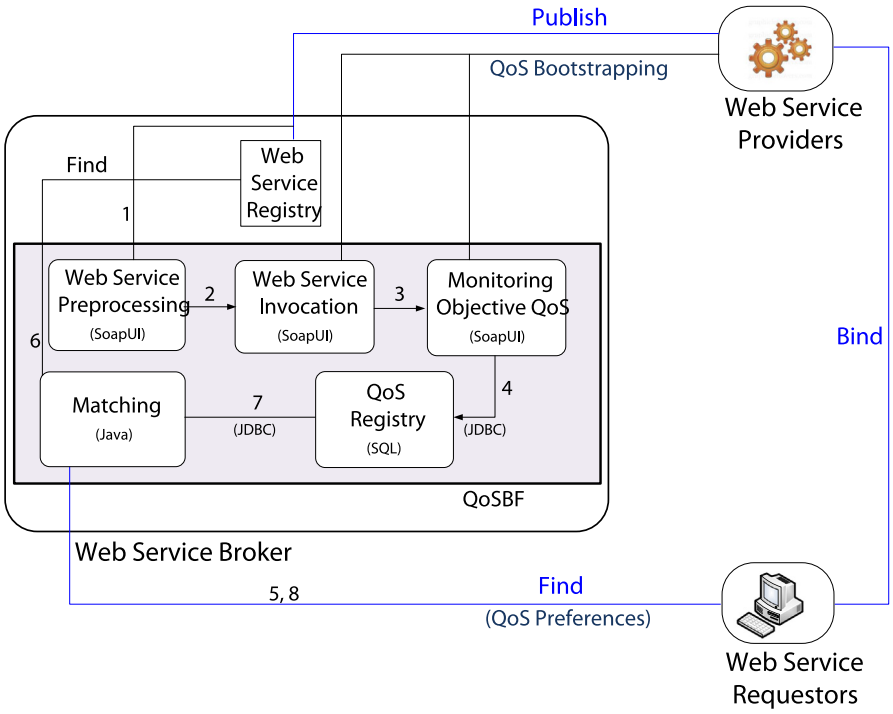

Figure 6 QoS Bootstrapping Framework Prototype.

Table 2 The monitored Web Services and their bootstrapped QoS.

WS	QoS						
	$QoS_{Response}$	$QoS_{Latency}$	$QoS_{Execution}$	$QoS_{Throughput}$	$QoS_{Availability}$	$QoS_{Reliability}$	$QoS_{Accessibility}$
WS1	193.5*	182*	11.5*	1.428	94.62	99.995	99.156
WS2	311.82	260	51.82	1.455	94.97	99.990	97.487
WS3	379.4	208	171.4	1.094	98.24	99.990	98.006
WS4	1500	1100	400	0.8	87.48	99.988	87.195
WS5	586.18	207	379.18	1.712	97.57	99.98	96.564
WS6	475.4	207	268.4	1.526	87.77	99.982	96.363
WS7	663.78	331	332.78	1.528	93.54	99.994	98.187
WS8	241.07	214	27.07	1.422	97.13	99.993	98.659
WS9	975	860	115	0.9	91.35	99.992	93.359
WS10	345.3	288	57.3	1.352	57.53	99.994	98.323
WS11	392	179	213	1.176	100	99.987	97.777
WS12	350.34	316	34.34	1.518	96.94	99.995	98.431
WS13	777.45	317	460.45	1.692	96.81	99.994	98.263
WS14	300	246	54	1.1	72.3	99.867	67.424
WS15	205	174	31.05	1.372	100	99.993	98.901
WS16	897	453	444	1.77	97.79	99.994	97.402

WS: Web Services, * Times are in ms.

$QoS_{Reliability}$ and $QoS_{Accessibility}$. Then, the QoS are bootstrapped at Publish time by invoking and monitoring the Web Services to evaluate their QoS.

Table 2 shows some of the monitored Web Services and their bootstrapped QoS. In the table, the Web Services are abbreviated and numbered for simplicity. Monitoring the QoS is conducted several times for duration of hours throughout days; and each time the average value is evaluated and the final value in the QoS database is updated.

Fig. 7 shows the bootstrapped $QoS_{Response}$ and $QoS_{Throughput}$ for the monitored Web Services. The figure shows variations on the value of QoS for services provided by different providers, which can assist requestors in their selection decision.

For example, a requestor may select a service with the highest $QoS_{Throughput}$ from a list of services with similar functionalities. In the following subsections, a case study and scenarios are presented.

7.1. A case study: city information

This section presents a case study where requestors search for Web Services from a service registry to build an enterprise application. The application is a 'City Information' application that provides requestors with the information about cities. Specifically, a requestor provides a city name and the application returns information about the city, such as address, weather, and zip code.

Fig. 8 presents the case study application, which is constructed as a composition of Web Services, including the following: find 'Address', 'Calculate' the coordinator, find the 'Weather', and finally 'Get' more information about the city, such as zip code. However, upon searching the registry, one can find that there are many such Web Services that have the same functionality, e.g., weather Web Service, provided by different service providers. Thus, to build such enterprise application, the developer needs to select Web Services that match his/her QoS preferences.

Following the case study, an application developer wants to select Web Services to build the City Information application that is composed of four Web Services. The Web Services are Address, Calculator, Weather, and Get Web Services. Upon searching the service registry by the matching component of the QoSBF, it will return three address Web Services, four Calculator Web Services, four Weather Web Services, and five Get Web Services, each from different service providers, as shown in Table 3. Subsequently, the matching component will search the QoS registry for the QoS of such Web Services that were already bootstrapped. Table 3 shows the bootstrapped QoS of the returned Web Services.

The following are two scenarios that present the importance of the bootstrapped QoS in building applications.

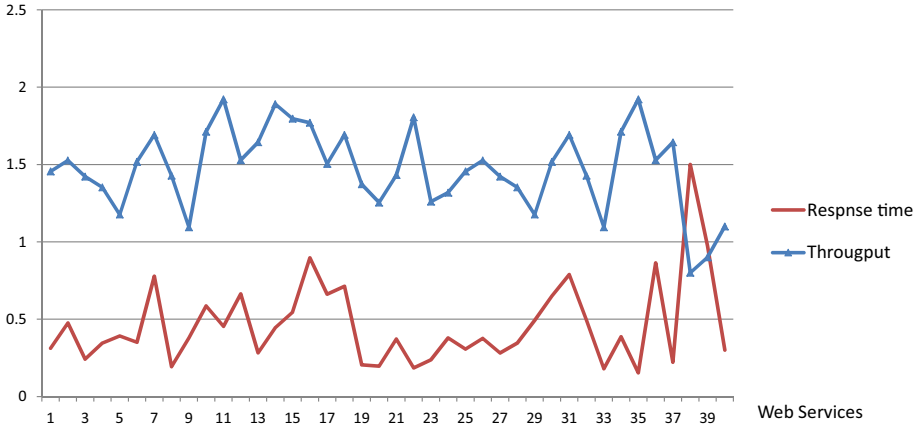

Figure 7 The bootstrapped $QoS_{Response}$ and $QoS_{Throughput}$.

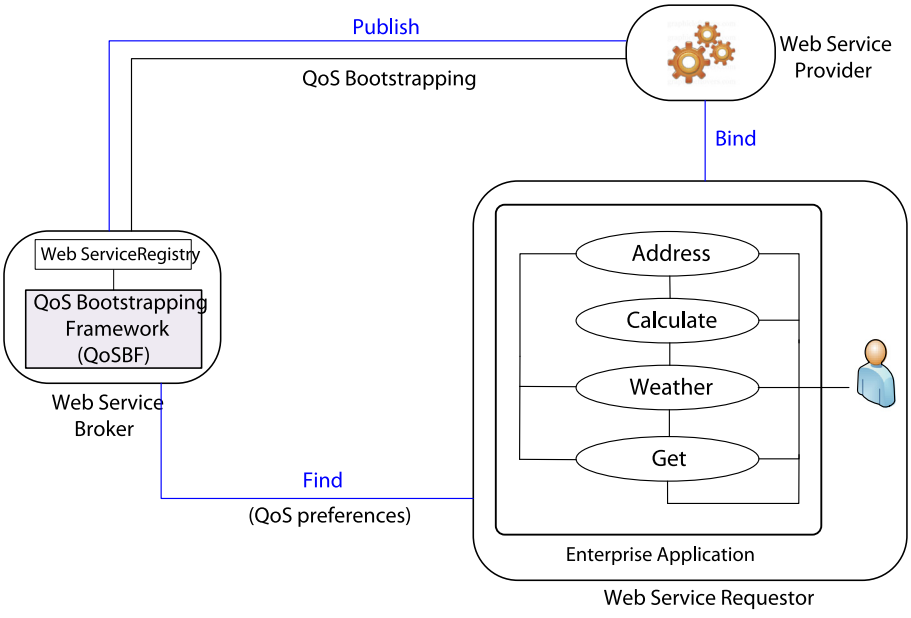

Figure 8 QoS bootstrapping in SOA, City Information Case Study.

Table 3 The returned Web Services and their bootstrapped QoS.

WS	QoS						
	$QoS_{Response}$	$QoS_{Latency}$	$QoS_{Execution}$	$QoS_{Throughput}$	$QoS_{Availability}$	$QoS_{Reliability}$	$QoS_{Accessibility}$
Address1	282.6*	238*	44.6*	1.644	98.25	99.996	99.04
Address2	371.95	164	207.95	1.431	93.94	99.990	98.501
Address3	1500	1100	400	0.8	87.48	99.988	87.195
Calculator1	241.07	214	27.07	1.422	97.13	99.993	98.659
Calculator2	627	489	138	0.81	68.33	99.958	79.904
Calculator3	193.5	182	11.5	1.428	94.62	99.995	99.156
Calculator4	379.4	208	171.4	1.094	98.24	99.990	98.006
Weather1	311.82	260	51.82	1.455	94.97	99.990	97.487
Weather2	350.34	316	34.34	1.518	96.94	99.995	98.431
Weather3	300	246	54	1.1	72.3	99.867	67.424
Weather4	379.11	37	342.11	1.318	100	99.990	96.65
Get1	475.4	207	268.4	1.526	87.77	99.982	96.363
Get2	897	453	444	1.77	97.79	99.994	97.402
Get3	205.05	174	31.05	1.372	100	99.993	98.901
Get4	196.03	163	33.03	1.254	81.03	99.978	96.440
Get5	237.76	166	71.76	1.26	94.16	99.981	96.969

WS: Web Services, * Time in ms.

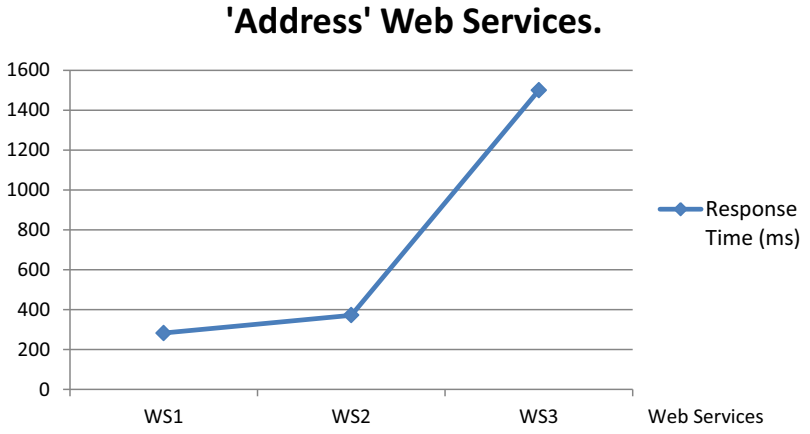

Figure 9 The $QoS_{Response}$ for the Address Web Services.

7.1.1. Scenario 1: selecting non-bootstrapped Web Services

In this scenario, the developer wants to select the desired Web Services to build the application. However, the Web Services' QoS is not bootstrapped; thus, the QoS is not available priory. Therefore, the developer will select Web Services randomly.

Accordingly, the developer may select the following Web Services: Address3 Web Service with high response time ($QoS_{Response} = 1500$ ms); Calculator2 Web Service with low availability ($QoS_{Availability} = 68.33$); Weather3 Web Service with low availability and accessibility ($QoS_{Availability} = 72.3, QoS_{Accessibility} = 67.424$); and Get2 Web Service with high response time ($QoS_{Response} = 897$ ms). Accordingly, the built application will have a high response time and low availability and accessibility, which result in low-performance application.

7.1.2. Scenario 2: selecting bootstrapped Web Services

In the following, a scenario is presented to show the importance of the availability of the bootstrapped QoS, and its benefit in distinguishing between similar Web Services.

The developer wants to select Web Services. The bootstrapped QoS is available; thus, the developer can select Web Services based on QoS that best match his/her preferences. For example, the developer wants to select Web Services with the lowest response time as his/her preference. The matching component will return Web Services and their corresponding $QoS_{Response}$ values. Figs. 9–12 show the $QoS_{Response}$ values for the returned Web Services.

Subsequently, the requestor can select a Web Service that has the lowest $QoS_{Response}$ value, which, for the given case, are Address1 Web Service with $QoS_{Response} = 282.6$ ms; Calculator3 Web Service with $QoS_{Response} = 193.5$ ms; Weather3 Web Service with $QoS_{Response} = 300$ ms; and Get4 Web Service with $QoS_{Response} = 196.03$ ms. Thus, the result will be a high-performance application with low response time.

Similarly, the requestor can select a Web Service that has the highest $QoS_{Availability}$ value, which are as follows (see Table 3): Address1 Web Service with $QoS_{Availability} = 98.25$; Calculator4 Web Service with $QoS_{Availability} = 98.24$; Weather4

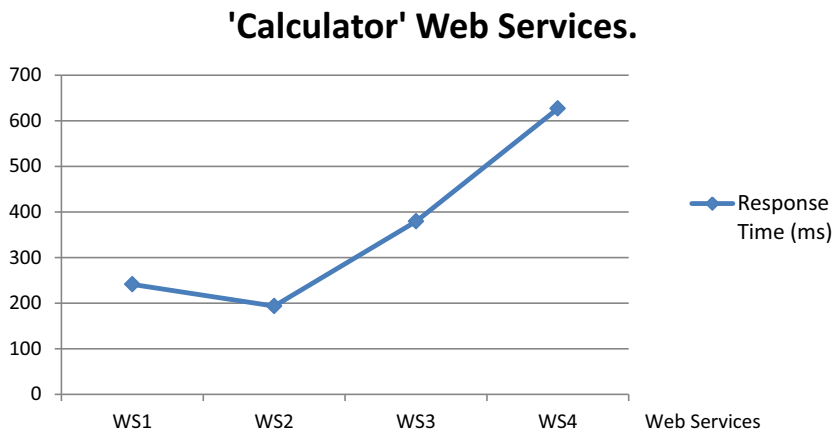

Figure 10 The $QoS_{Response}$ for the Calculate Web Services.

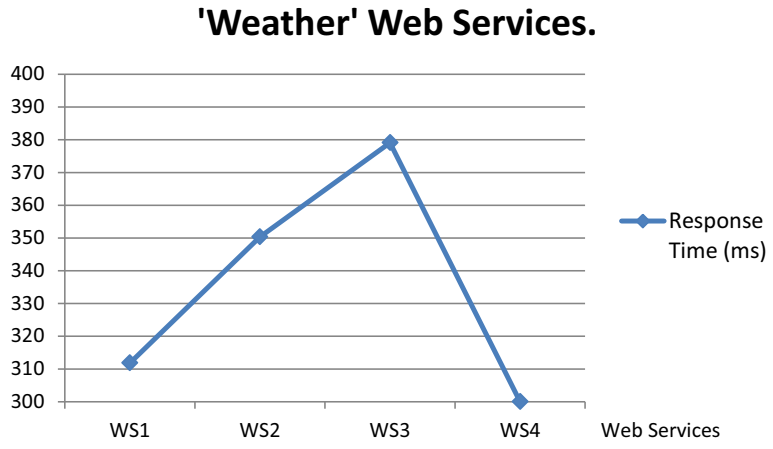

Figure 11 The $QoS_{Response}$ for the Weather Web Services.

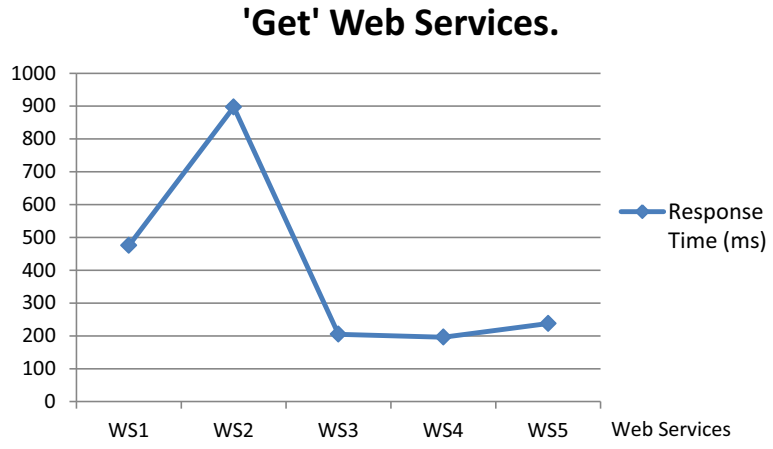

Figure 12 The $QoS_{Response}$ for the Get Web Services.

Web Service with $QoS_{Availability} = 100$; and Get3 Web Service with $QoS_{Availability} = 100$.

8. Conclusion

QoS has been used as a distinguishing factor between similar services and as a criterion for service selection. Bootstrapping QoS justifies the QoS once new services are registered. This paper proposes a QoS bootstrapping solution that includes QoS model, QoS bootstrapping framework, SOA extension to support QoS bootstrapping, and prototype. The experiments and evaluations show the importance of the bootstrapping techniques in developing high-performance applications. Accordingly, bootstrapping QoS plays an important role in the evolution of distributed paradigms. Further research is needed in this area. Specifically, there is a need to extend the work to bootstrap other QoS in the QoS model. Particularly, future work will seek to build methodologies for bootstrapping security.

Acknowledgement

This work was supported by Kuwait University, Research Grant No. [QI04/13].

References

Baraki, H., Comes, D., Geihs, K., 2013. Context-aware prediction of QoS and QoE properties for web services. In: Conference on Networked Systems (NetSys), pp. 102–109.

Dragoni, N., 2009. Toward trustworthy web services – approaches, weaknesses and trust-by-contract framework. In: IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, vol. 3, pp. 599–606.

Ferris, C., Barbir, A., Garg, S., Austin, D., 2004. Web services architecture requirements. W3C note, W3C, <http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211> last accessed April, 2014.

Ge, J., Chen, Z., Peng, J., Li, T., Zhang, L., 2010. Web service recommendation based on QoS prediction method. In: 9th IEEE International Conference on Cognitive Informatics (ICCI), pp. 109–112.

Hoyle, D., 2005. Automotive Quality Systems Handbook, second ed. Elsevier Ltd.

Huhns, M.N., Singh, M.P., 2005. Service-oriented computing: key concepts and principles. IEEE Internet Comput. 9, 75–81.

Kalepu, S., Krishnaswamy, S., Loke, S., 2003. Verity: a QoS metric for selecting web services and providers. WISEW '03: Proceedings Fourth International Conference on Web Information Systems Engineering Workshops, pp. 131–139.

Kim, Y., Doh, D., 2007. A trust type based model for managing QoS in web services composition. In: International Conference on Convergence Information Technology, pp. 438–443.

Lee, K., Jeon, J., Lee, W., Jeong, S., Park, S., 2003. QoS for web services: requirements and possible approaches. Tech. rep., W3C, Web Services Architecture Working Group. <http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/>.

Liu, Y., Ngu, A., Zeng, L., 2004. QoS computation and policing in dynamic web service selection. In: Proceedings of the 13th International World Wide Web conference on Alternate Track Papers & Posters. ACM, New York, NY, USA, pp. 66–73.

Mathijssen, S., 2005. A fair model for quality of web services. 3rd Twente Student Conference on IT, Enschede.

Maximilien, E., Singh, M., 2004. Toward autonomic web services trust and selection. In: Aiello, M., Aoyama, M., Curbera, F., Papazoglou, M.P. (Eds.), Proceedings of Second International Conference in Service-Oriented Computing, ICSOC '04. ACM, New York, NY, USA, pp. 212–221.

Papazoglou, M., 2012. Web Services and SOA: Principles and Technology. Pearson Education, Essex, England; New York.

Papazoglou, M.P., Georgakopoulos, D. (Eds.), 2008. Service-Oriented Computing. The MIT Press, Cambridge, MA.

Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., Krmer, B.J., 2006. Service-oriented computing research roadmap. In: Dagstuhl Seminar Proceedings, 05462, pp. 1–29.

Rajeswari, M., Sambasivam, G., Balaji, N., Saleem Basha, M., Vengattaraman, T., Dhavachelvan, P., 2014. Appraisal and analysis on various web service composition approaches based on qos factors. J. King Saud University – Comput. Inf. Sci. c26, 143–152.

Ran, S., 2003. A model for web services discovery with QoS. ACM SIGecom Exchanges 4 (1), 1–10.

Rosen, M., Lublinsky, B., Smith, K.T., Balcer, M.J., 2008. Applied SOA: Service-Oriented Architecture and Design Strategies. Wiley Publishing.

Rosenberg, F., Platzer, C., Dustdar, S., 2006. Bootstrapping performance and dependability attributes of web services. IEEE Computer Society, 205–212.

Sherchan, W., Loke, S.W., Krishnaswamy, S., 2006. A fuzzy model for reasoning about reputation in web services. In: Haddad, H. (Ed.), SAC '06: Proceedings of the 2006 ACM symposium on Applied computing. ACM, pp. 1886–1892.

SoapUITool, 2014. <http://www.soapui.org/>, last accessed September 2014.

Wang, H., Sun, H., Yu, Q., 2013. Reliable service composition via automatic QoS prediction. In: 2013 IEEE International Conference on Services Computing (SCC), pp. 200–207.

Ying-Feng, Z., Pei-Ji, S., 2006. The model for consumer trust in C2C online auction. In: ICMSE '06: International Conference on Management Science and Engineering, pp. 125–129.

Yu, Q., 2012. Decision tree learning from incomplete QoS to bootstrap service recommendation. In: IEEE 19th International Conference on Web Services (ICWS), pp. 194–201.

Yu, W.D., Radhakrishna, R.B., Pingali, S., Kolluri, V., 2007. Modeling the measurements of QoS requirements in web service systems. Simul. J. 83 (1), 75–91.

Yuan, M., Long, J., 2002. Securing wireless j2me. Tech. rep., IBM.

Zhang, J., Zhong, F., Yang, Z., Liu, H., 2012. A QoS-aware computation model for dynamic web service selection. In: IEEE 12th International Conference on Computer and Information Technology (CIT), pp. 230–235.

Zhang, Y., Zheng, Z., Lyu, M., 2010. Wsexpress: a qos-aware search engine for web services. In: ICWS '10: IEEE International Conference on Web Services, pp. 91–98.

Zheng, Z., Zhang, Y., Lyu, M., 2014. Investigating QoS of real-world web services. IEEE Trans. Serv. Comput. J. 7 (1), 32–39.

Zhengping, L., Xiaoli, L., Guoqing, W., Min, Y., Fan, Z., 2007. A formal framework for trust management of service-oriented systems. In: SOCA '07: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications. IEEE Computer Society, Washington, DC, USA, pp. 241–248.

Zou, G., Xiang, Y., Gan, Y., Wang, D., Liu, Z., 2009. An agent-based web service selection and ranking framework with QoS. In: ICCSIT 2009: 2nd IEEE International Conference on Computer Science and Information Technology, pp. 37–42.