A Model for Managing and Discovering Services Based on Dynamic Quality of Services

Hao-peng Chen*
School of Software, Shanghai Jiao Tong University, Shanghai, China, 200240
Email: chen-hp@sjtu.edu.cn

Can Zhang and Guang Yang
School of Software, Shanghai Jiao Tong University, Shanghai, China
Email: { zhangcan05, allenmacyoung }@gmail.com

***Abstract*—In this paper, we analyze the problems of existing Service Computing model and propose a model for managing and discovering services based on dynamic quality of services. This model consists of a Quality Measurement Center (QMC), which receives and analyzes the feedbacks of dynamic qualities of services to evaluate and rank the services according their dynamic qualities, and enables service consumer to search and discover services by the functional and/or QoS requirements; a QoS Spy, which feedbacks the real-time dynamic qualities of services to QMC and processes the fault detection and substitution of services; a Service Quality Calculator, which calculates the qualities of composite services based on their atomic services. With this model, we can evaluate and discover services on the Internet based on both functional and qualitative constraints. Meanwhile, this model is compatible with the basic principles of Service Computing, which are loosely coupled, protocol independent and location transparent. This model is the foundation of dynamic service discovery and substitution, and can improve the availability, scalability, and modifiability of service-based applications.**

***Index Terms*—service computing; dynamic quality; service managing; service discovering; feedback**

I. INTRODUCTION

With the great development of Web Service technology and the application of Service-Oriented Architecture (SOA), Service Computing has been a hotspot of Distributed Computing and made a significant progress in recent years. As more and more services are available on the Internet, we have a lot more choices when composing a service-based application. However, for a service-based application which is composed by a certain number of services, the quality of each service may affect the quality of the whole system. Therefore, in order to build an application of good quality, we have to face the following problems:

The first problem is how to discover those services which meet the application's functional requirements. This can be solved by querying the service registry which identifies the functionality of each service by parsing the description file of the service. The UDDI (Universal Description Discovery and Integration) specification has specified what functions a service registry should have, such as the function for registering the WSDL (Web Service Description Language) file of a web service to the registry, the function for searching a web service in yellow pages, green pages, and white pages, and the function for searching a web service by the keyword given by user [1]. Some UDDI specification complaints are available, such as Apache jUDDI, which is an open source Java implementation of UDDI [2]. Since this problem is not our focus, we suppose that the existing UDDI implementations can solve this problem well.

The second problem is how to discover those services which meet the application's QoS (Quality of Service) requirements. With the development of Service Computing theory, technology and tools, there are more and more services on the Internet. As a result, when a developer of a SOA application search a service on the Internet based on a functional requirement, he can find several candidates. Thus, developers are more concerned with the quality attributes of services than ever before. They want better and more suitable services.

One way to solve this problem is to extend the description file of the service to include the Quality of Service (QoS) attributes of the service. Thus, later we can obtain the QoS attributes of the service by parsing the description file. However, this solution has several limitations. First, the descriptions are not objective. People who use the service may have quite different opinions and feelings from the person who writes the description file. Second, these descriptions are static. They cannot reflect the dynamic quality of the service. In other words, even if the description claims that the service is good, it is not persuasive enough for developers to choose this service. The dynamic quality of a service is composed by a set of attributes which have real-time status, such as performance, availability, security, reliability, and so on.

The third problem is how to automatically and dynamically substitute service when service fault occurs. Even if we can ensure that the services selected can meet

This paper is supported by the National High-Tech Research Development Program of China (863 program) under Grant No. 2007AA01Z139.

doi:10.4304/jnw.5.8.888-895

the QoS requirements when building the system, we can never guarantee that the composition of these services is still the best choice after running for a long time. As a result, if one component service fails, a mechanism is needed to ensure that the fault can be detected, the running process shall not be interrupted, and the failed service can be quickly and efficiently substituted. The faults can be classified into two types: the faults of functionality and the faults of quality. We focus on the latter, especially on the faults of dynamic quality.

The fourth problem is how to build a comprehensive closed loop SOA governance infrastructure, where the service registry can monitor, rank, classify and recommend services according to their dynamic qualities. If we can provide a model for managing and discovering services based on dynamic quality of services, which supports service publication, discovery and location, binding and invoking, monitoring, evaluation, ranking, and recommendation, a service consumer will get the most suitable service among all the candidates which satisfy his functional and QoS requirements.

In this paper, we show how our model solves these problems. In general, our model involves three parts: a Quality Measurement Center (QMC), which receives and analyzes the feedbacks of dynamic qualities of services to evaluate and rank the services according their dynamic qualities, and enables service consumer to search and discover services by the functional and/or QoS requirements; a QoS Spy, which feedbacks the real-time dynamic qualities of services to QMC and processes the fault detection and substitution of services; a Service Quality Calculator, which calculates the qualities of composite services based on their atomic services.

The rest of this paper is organized as follows. Section 2 surveys some related work. Section 3 presents the overview of our model. Section 4.A shows the architecture of the QMC. Section 4.B presents how we design the QoS Spy. Section 4.C describes the design of the Service Quality Calculator. Finally, the paper is concluded in Section 5.

II. RELATED WORK

Many service computing models have been proposed by various companies and organizations in recent a couple of years. Among these service computing models, the mode proposed by IBM is the most comprehensive one which includes two main parts: Service Container Architecture [3] and Service Data Object [4]. SCA and SDO have been adopted as specifications by leading companies, such as Oracle, Sybase, SAP, and Iona. IBM also implements WebSphere Integration Reference Architecture (WIRA) in its WebSphere series products. WIRA provides self-contained service collection and supports integration of large-scale enterprise application [5]. Also, Microsoft has released its own SOA solution: BizTalk Server, an integrated business processing server, which acts as the backbone of SOA systems. Just like the architecture of enterprise service bus (ESB), the BizTalk Server contains adapters, pipelines and business rules engine [6]. Furthermore, Microsoft provided WorkFlow 3.0 and 3.5 in .NET framework as an alternative of BizTalk since WorkFlow is much more convenient for .NET users compared with BizTalk [7]. Oracle also has released Oracle SOA Suite, which aims to be a self-contained hot-pluggable software suite for building, deploying and managing SOA systems [8].

These reference models, platforms and tools distinguish themselves from each other in many ways. However, they have some common characters, such as they all use Web Service as the specific mechanism of service implementation. However, WSDL, the common description language for Web Service, has its limitation upon describing QoS attributes. In despite of the prosperity of different SOA-related products, they still cannot satisfy the requirements of building a highly available service network since they cannot describe real-time qualities of services and lack of the ability of dynamic service substitution.

Many researchers have also worked on the Service Computing model and related techniques. For examples, in [9], authors proposed algorithms for web services discovery and composition by parsing syntactic and semantic service descriptions; in [10], authors adapted A* algorithm to effectively search Web Services; in [11], authors designed and developed an agent-based Web Service composition framework; in [12], authors proposed a Web Service discovery mechanism based on Quality of Service, which classifies Web Services according to their QoS descriptions beforehand. In addition, OWL-S has become a W3C standard for describing Semantic Web Services [13]. The OWL-based language is composed of three main parts, namely service profile, process model, and grounding. It has been praised for its computer-interpretable semantic markup of services, but when it comes to describe QoS, OWL-S only includes general and non-quantitative descriptions like quality rating in the service profile. Consequently, to make up OWL-S's limitations in describing quality of services, in [14], authors presented OWL-Q, a novel, rich and extensible ontology-based approach for describing QoS of Web Services.

We find that most researches are based on static QoS descriptions rather than dynamic quality information. Thus, these QoS descriptions can hardly reflect the real-time status of dynamic qualities of services. Consequently, the dynamic service discovery and substitution becomes impossible. Without dynamic service discovery and substitution, the availability of service-based applications will be reduced and beyond the control.

In summary, we need to design a model to dynamically manage and discover services according to their real-time status of dynamic qualities. With this model, developers of service-based applications can find the most suitable services to build the applications. Meanwhile, the run-time qualities of the applications can be improved by dynamic service discovery and substitution. This model also should be compatible with existing service computing models.

III. RATIONALE OF THIS MODEL

The rationale of the model for managing and discovering services based on dynamic quality of services can be outlined by the collaborative framework shown in Figure 1. It is not supposed to be a replacement of the existing service computing model. In fact, our framework extends the existing model by adding new collaborations, so it is compatible with the existing one. Figure 1 shows the specific collaborative framework.

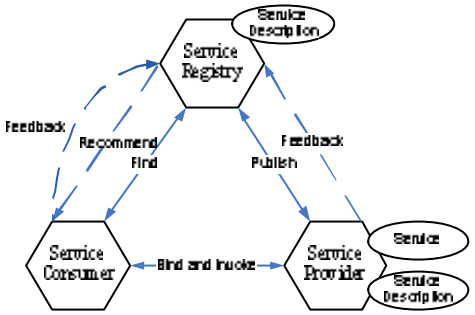

Figure 1. Collaborative framework

Figure 1 contains three roles involved in the Service Computing model: Service Consumer, Service Provider, and Service Registry. Service Consumer, which can be an application or a software module, invokes the query module of the Service Registry to find services it needs, and then binds and invokes these services with specific transfer protocols. Service Provider, which is an addressable entity on the Internet, publishes service descriptions in the Service Registry, and also receives and processes Service Consumer' s requests. Service Registry, which contains all the service descriptions published by different Service Providers, provides the Service Consumer with service discovery function. In our model, we simply inherit these definitions of roles from the existing model, but we add new collaborations between these roles.

Solid lines in Figure 1 represent collaborations that are quite similar to the standard collaborations supported by existing models, including Publish, which means Service Provider publishes descriptions of its services to make them known to Service Consumer; Find, which means Service Consumer queries Service Registry to find and locate appropriate services; Bind and Invoke, which means Service Consumer could invoke the services found in the Find process according to the service descriptions.

In our model, we add some new meanings to these three collaborations. For Publish, the service descriptions published will also include the static QoS attributes in a certain specification, such as in OWL-Q[15], to make automated processing possible. For Find, besides finding services according to functional requirements, we also support finding services according to QoS requirements which include the constraints on both static and dynamic QoS attributes. For Bind and Invoke, we bind Service Consumer with services which have the same functionality rather than one specific service, that is, we provide a logic binding but not a physical binding.

In addition, as you can see in Figure 1, we add three new collaborations in dotted lines: Feedback and Recommend between Service Registry and Service Provider, and Feedback between Service Registry and Service Consumer. By Feedback between Service Registry and Service Consumer, we mean that Service Consumer feedbacks Service Registry with the real-time status of dynamic quality of the services it invokes. The feedback is in terms of the experience of Service Consumer with which the Service Registry can recommend suitable services to Service Consumer according to the similarity of Service Consumers. By Recommend, we mean that by collecting statistics of real-time feedback, Service Registry can evaluate, rank, and classify services which have the same functionality. Therefore, Service Registry can recommend services of better quality to Service Consumer while there are a lot of services with similar functions. By Feedback between Service Registry and Service Provider, we mean that Service Provider feedbacks Service Registry with the real-time status of dynamic quality of the services it provides, with which the Service Registry can evaluate each service globally. We had once added a Feedback collaboration between Service Provider and Service Consumer in an early version of our model. We had considered that Service Provider also should send feedback to Service Consumer, which would help Service Consumer dynamically detect fault and recompose services. Later, however, we realized that this feedback can be replaced with the measurement of Service Consumer, and the latter is much more objective and meaningful for evaluating qualities of services than the former. Thus, we deleted this collaboration in newer versions of the model.

In summary, by extending the existing Service Computing model with new collaborations, the new framework brings us closer to the goal of high availability and reliability of Service Computing model. In the following sections, we will explain each part of this model in details.

IV. DETAILS OF THIS MODEL

The model for managing and discovering services based on dynamic quality of services has three parts: Quality Measurement Center, QoS Spy, and Service Quality Calculator. This section will describe these parts in detail.

A. Quality Measurement Center

In our model, since Service Registry takes the responsibility of collecting real-time status of dynamic quality of services, evaluating and ranking services, and recommending services of good quality, we call it an Intelligent Service Registry, which is an extended UDDI-complied service registry by adding QMC into it. Figure 2 shows the architecture of the Intelligent Service Registry.

In the architecture shown in Figure 2, the part in dotted frame is QMC, which runs either as an independent center, or as a part of an integrated Intelligent Service

Registry. The Intelligent Service Registry exposes four external interfaces, namely Service Publish Interface, Service Discovery Interface, Registry P2P Interface, and Service Feedback Interface. Each interface takes on a different role in the architecture. The Service Publish Interface is an interface of a UDDI-complied service registry, which enables service providers register their services into the Registry. The Service Discovery Interface is an interface of QMC. With the Service Discovery Interface, those service-based applications can find and locate services they are interested in. The Registry P2P Interface is an interface of both UDDI-complied service registry and QMC. It is used for interaction between different intelligent service registries for sharing real-time status of dynamic quality of services so as to manage the whole distributed system. The Service Feedback Interface is an interface of QMC, which is designed for collecting feedback from Service Customers and Service Providers. In the following paragraphs, we will use four scenarios to explain the usage of each interface separately.

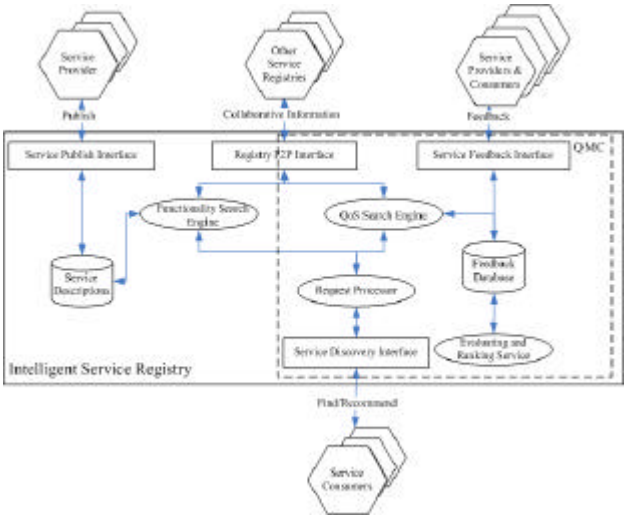

Figure 2. The architecture of Intelligent Service Registry

The first scenario is that a Service Provider registers the descriptors of its services into Registry by Service Publish Interface. It is a basic and necessary function for any service registry, especially for UDDI-complied registry, to allow service provider to register services. The registering task is accomplished by saving service descriptors into Service Descriptors Database.

The second scenario is that a Service Consumer proposes a service discovery request and the receiving registry center finds the suitable services without help of other centers. First, the request is dispatched to the Request Processor, which is a part of QMC. The Request Processor divides the request into functional requirements and QoS requirements, which will be passed to Functionality Search Engine and QoS Search Engine separately. Second, the Functionality Search Engine, a part of UDDI-complied service registry, will find a set of services which satisfy the functional requirements. Third, the QoS Search Engine, a part of QMC, sorts the services found by the Functionality Search Engine by the evaluation and ranking of their QoS attributes, and then returns the services that match the specific QoS requirements of the Service Consumer. If the Service Consumer provides little or no description of its QoS requirements, the Request Dispatcher will recommend the most qualified service. If QMC runs as an independent center, then it would only support the service finding based on QoS constraints.

The third scenario makes a nice complement to the second scenario, namely the situation that the receiving registry center does not find the suitable services. For a distributed Service Registry architecture, several registry centers reside on the Internet, and each registry center holds a part of the whole registry information. Thus, these registries form a physical P2P ring and use chord [16] protocol as the distributing protocol of service descriptors. Furthermore, there are several logic P2P rings built on this physical P2P ring into each of which a specific quality attribute, such as performance, availability, reliability, and so on, is mapped. We have established such a multiple-logic-ringed prototype of distributed registry and designed an algorithm for finding service in this prototype based on multiple QoS constraints.

Therefore, to complete a query request from the Service Consumer, the service center who receives the request sometimes has to communicate with others to return a complete and optimized result. It works like this: after one registry center receives a query request, it will first search in its database to find suitable services. If no services are found, the request will then be passed to other registry centers by the Register P2P Interface. This procedure won't stop until the services which satisfy the functional requirements and QoS requirements are found. Also, the registry center where these suitable services are found will be connected with the Service Consumer and then evaluate and monitor the real-time status of dynamic quality of those services chosen by the Service Consumer. The Registry P2P Interface is an interface of both UDDI-complied service registry and QMC, since both of them are support P2P structure.

The fourth scenario is about monitoring and collecting real-time status of dynamic quality of the registered services. Periodically, feedback on real-time status of dynamic quality of the registered services is sent to Service Registry through the Service Feedback Interface. Then, the feedback will be saved into Feedback Datebase. The Evaluating and Ranking Service, periodically refreshes the data about evaluation and rank of services in the Feedback Database according to the feedback data received. The data stored in the Feedback Database includes average response time, average failure time and so on. In addition, the Evaluating and Ranking Service also makes validation of newly registered services. By validation, we mean that the Evaluating and Ranking Service compares the data collected by testing the service and the QoS attributes the service declares to be, and decides whether the descriptions deviate from the real situation. All the components involved in this scenario are the proprietary components of QMC which are not parts of UDDI-complied service registry.

Distributed Intelligent Service Registry plays an important part in our model. The architecture mentioned here only captures the basic design of the Service Registry. In fact, the whole architecture is much more complicated, and further research on the specific techniques is needed.

QMC also can runs as an independent center to evaluate, rank, and classify services according to their qualities, and provide searching function for Service Consumers. If QMC runs as such a third-party service search engine, it also can coordinate with UDDI-complied service registry to provide comprehensive search capability. In this scenario, the P2P ring of QMC is independent of the one of UDDI-complied service registry.

The core of QMC is the hash algorithm for distributing the feedbacks into a P2P ring of nodes of QMC. We establish an improved B+ tree as the basis for accomplishing this distribution. Please see more details of this algorithm in reference [17].

In QMC, the QoS attributes are classified into two categories according to their measurability. The attributes that can be quantificationally evaluated are measurable QoS attributes. For each attribute of this category, we evaluate it by a certain algorithm for analyzing the feedbacks. The attributes that can hardly be measured accurately are unmeasurable attributes. We use reputation to represent the general quality of all the unmeasurable attributes as a whole. We have designed an algorithm for evaluate the reputation of a service. Please see more details of this algorithm in reference [18].

Although the load of the nodes of QMC can be effectively and efficiently reduced due to the P2P structure of QMC, the load of each node of QMC will be still very heavy when it is deployed on Internet. We designed a multiple-level-cached structure of the node of QMC. In this structure, a cache tree is set up to cache the result of service queries. Service Consumers are grouped by their similarity and each group has a corresponding cache. When a Service Consumer wants to find a service according to certain constraints, it will send a service query to the cache of its group at first. If there is a service which can satisfy the constraints in the cache, this service will be the query result delivered to Service Consumer. Otherwise, the query will be sent to an upper level cache. The experiment has demonstrated this structure can greatly reduce the load of each node of QMC. The details of this multiple-level-cached structure are in reference [19].

QMC is the core of our model. Although it is rather complex, we have developed a prototype of QMC and designed the necessary algorithm. We need to continuously improve the existing implementation.

B. QoS Spy

We design and develop a QoS interceptor, named QoS Spy, to enable the feedback and dynamic service substitution. The feedback is used in two ways. First, QMC will use the feedback data to evaluate and rank services. Second, service-based applications will use the feedback data to monitor the real-time status of dynamic quality of services used and decide if dynamic substitution of services is needed. Figure 4 shows the structure of QoS Spy.

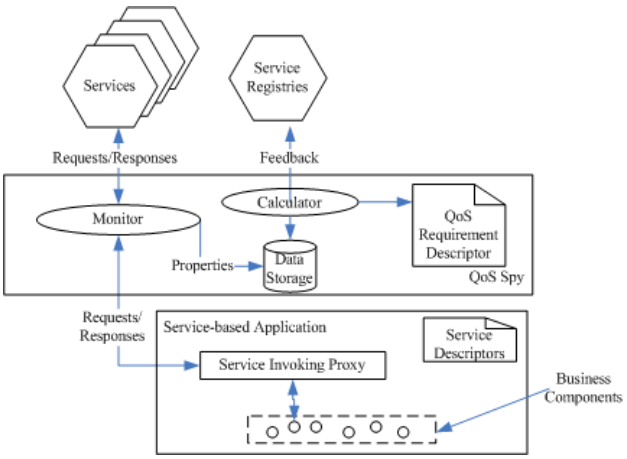

Figure 3. The Structure of QoS Spy

In the structure shown in Figure 3, the QoS Spy is composed of a Monitor, a Calculator and a lightweight Data Storage. It runs in the hosting environment of service-based application. In the following paragraphs, we will explain the functions of each part of the model.

The Monitor acts as an interceptor of each service invocation. The business components of a service-based application send service invoking requests and receive service invoking responses via the Service Invoking Proxy. Service Invoking Proxy transfers a local call into a request of service invocation on the Internet over some well-known protocols, like XML-based SOAP message, and dispatch the request to target service. After the target service returns the result, Service Invoking Proxy will parse the standard response package, like a SOAP message, into a return value of a local call, and send it to the business component.

All the requests and responses are intercepted by QoS Spy, and then forwarded to services or business components by QoS Spy. At the time of intercepting one request, the Monitor records information like exact intercepting time and target service, and calculates the average request frequency, and then stores them in the Data Storage. Then, the request is forwarded to the target service. At the time of intercepting one response, the Monitor also records information like execution time and response time, and then forwards the response to business component.

We have designed and developed three kinds of interceptors of Monitor:

a) Proxy-based interceptor. This kind of interceptors is a proxy hosting in the environment of the service-based application. With this kind of interceptor, Service Consumer doesn't invoke services directly. Instead, it calls the proxy and passes it the information of the target service, including the URL of target service, the target operation, and its parameters. The proxy takes the responsibility of delivering invocation to proper service. That is, Service Consumer has to modify its code if it wants

to use this kind of interceptors. However, this kind of interceptors is the simplest and most accurate one.

b) AOP-based interceptor. With this kind of interceptors, the service-based application needs to import the library of this interceptor and compile its code with an AOP compiler. The AOP complier interweaves the code of the library of this interceptor into the compiled class files to provide functions of interception and feedback. With this kind of interceptors, we needn't to modify any code of the application. However, this kind of interceptors is dependent on the approach of implementation of the service-based application.

c) Port-based interceptor. This kind of interceptors is independent of the implementation of service-based application and runs as a separate process. It monitors the ports used by the service-based application, such as 8080 for HTTP and 21 for FTP, and analyzes the packets transported via these ports. The packets about invocation requests and responses of target services are intercepted and one of their copies is saved into Data Storage of QoS Spy for calculating. This kind of interceptors, however, is dependent on the type of hosting Operating System.

Each of these three kinds of QoS interceptors has its advantages and disadvantages. Please see more details in reference [20].

As a part of QoS Spy, the QoS interceptor usually runs on the hosting environment of service-based application. As Service Providers and Service Consumers both need the QoS interceptor, it also can separately run as an independent tool in the hosting environment of services to feedback the global real-time status of dynamic qualities of services.

Since the number of service invocations is rather great, sending feedback per invocation will sharply increase the load of QMC. So the feedbacks are needed to be preprocessed. We designed a circular queue as the data structure of Data Storage to cache the feedbacks, which can avoid the leakage of memory [21]. The Calculator periodically accesses data in Data Storage, calculates the QoS attributes, and then sends them to Service Registry. Thus, the amount of feedbacks and the number of times of communication between QoS Spy and QMC can be reduced to a reasonable range.

The Calculator also compares the calculated QoS attributes with the QoS constraints described in the QoS Requirement Descriptor. The QoS Requirement Descriptor is edited by developers at the design time to describe the QoS constraints on the composite services and whole business flow. If the calculated QoS attributes cannot satisfy the QoS constraints, the Calculator will adopt an algorithm to determine whether there is any fault in the application. We have designed more than three algorithms for detecting faults. The first One is based on Queueing Theory [22][23]. In this algorithm, a service is modeled as a queue. We can detect the fault by compare the features of the queue with the calculated QoS attributes. The second one is based on Artificial Neural Network (ANN) [24]. In this algorithm, we can detect fault by compare the predicted value generated by ANN with the calculated QoS attributes. The third algorithm is based on Markov Chain [25]. In this algorithm, we can detect fault by compare the predicted value generated by Markov Chain with the calculated QoS attributes. Each of these three algorithms has its advantages and disadvantages. The experiments have demonstrated that they all take effect on fault detection.

If a fault is detected, the Calculator will determine how to recover the application from this fault. The Calculator analyzes the structure of composite service and business flow to find a suitable solution to service substitution. We designed an algorithm for evaluating the impact factor of each component service of a composite service. The impact factor indicates what extent of impact a component service imposes on the composite service. It is important that different QoS attributes should be calculated differently. We designed an algorithm for calculating the impact factors [26]. It naturally that we prefer choose the service with larger impact factor to be substituted, since the effect of improving QoS is more positive if we do so.

At beginning, the Calculator chooses the service with max impact factor to be substituted. The Calculator sends queries to QMC to find alternative(s). If the alternatives are more than one, the Calculator will compare the improving effect of them and determine which one is chosen. If there is not any improving effect of all the alternatives which can satisfy the QoS constraints in QoS Requirement Descriptor, the Calculator will choose the service with second max impact factor to be substituted, and so on. If the Calculator find there is no solution to effectively improve the QoS of composite service by substituting a single component service, it will try to substitute two component services, and so on. Finally, the Calculator will either find a solution to effectively improve the QoS of composite service by substituting a certain number of component services, or fail to find a solution. If the solution is found, the Calculator will inform the service-based application to modify its business logic descriptor in order to substitute the component services. Otherwise, it will inform the service-based application that the automatic fault tolerance is impossible and artificial intervention is necessary.

When the Calculator compares the improving effect of the candidates of a service, it will call the function of Service Quality Calculator to accomplish this task.

The QoS Spy is a run-time tool and the source of real-time status of dynamic qualities of services. It is also a key for ensuring the high availability of service-based applications.

C. Service Quality Calculator

Service Quality Calculator is a design time tool for developers to calculate the quality of composite service based on the qualities of its component services. It also can be invoked by QoS Spy to search the solution to service substitution.

To calculate the quality of composite service, we designed an XML schema to describe QoS constraints, named Web Service QoS Constraints (WSQC). It mainly

has three parts: BPEL (Business Process Execution Language) Reference, Metric Definition, and Metric Constraint. Since we use a non-invasive way for BPEL to describe the constraints rather than extend the BPEL, we need the BPEL Reference which refers to the BPEL of composite service. The Metric Definition describes the features of the QoS attributes, such as the name and measurability. The Metric Constraint is the specific constraints on QoS. This is a novel and extensible ontology-based approach for description of the QoS constraints [27]. With this approach, developers can describe the QoS constraints unambiguously.

The Service Quality Calculator classifies the QoS attributes into three categories: measurable direct attributes which can be measured directly by QMC, measurable derived attributes which cannot be measured directly but can be derived by measurable direct attributes, and unmeasurable attributes which can only be evaluated as reputation.

The Service Quality Calculator also classifies the patterns of the structure of component services into four categories: sequence pattern, parallel pattern, choice pattern, and iteration pattern. A developer recursively applies the four patterns on the component services to build a composite service.

There are 12 cases generated by orthogonalizing the three categories of QoS attributes and the four categories of patterns of structure of component services. For each case, we designed a specific formula to calculate the quality of the component services as whole. By recursively applying the formulae on the component services, the Service Quality Calculator can calculate each QoS attribute of a composite service [27].

The Service Quality Calculator sends queries to QMC to get the qualities of component services. So it is also a client of QMC. With this tool, the developers can choose the most suitable candidates to build a composite service. In design time, the developer builds the logic flow of a composite service at first. Then, he finds all the possible candidates of each component service by querying QMC. If the candidates are not unique, the solutions to build the physical flow of the composite service are also not unique. The Service Quality Calculator iterates the solution space and calculates the quality of each solution. Finally, it can give developer a recommended solution.

The Service Quality Calculator is also invoked by the QoS Spy at run time. So it can either run as a service, or be packaged as a library.

V. CONCLUSION

In this paper, we have presented a model for managing and discovering services based on dynamic quality of services. In particular, we focused on the dynamic features of the model, such as dynamic service discovery and composition, dynamic service substitution and fault detection. What's more, this model can rank, classify and recommend services according to their real-time performance.

We believe that our model has accomplished the goal of high availability of service-based application. It is based on and consequently compatible with the existing Service Computing model. Hence, it can be integrated and used in the existing Internet environment.

We have designed and developed all the necessary parts involved in this model, including Quality Measurement Center, QoS Spy, and Service Quality Calculator. The experiments have demonstrated these tools are effective and feasible.

In future, we will extend our model by adding semantic based service managing and discovering. Thus, our model will can not only support fault tolerance on QoS, but also support fault tolerance on functionality. As a result, the availability of service-based applications can be improved more greatly.

ACKNOWLEDGMENT

This paper is supported by the National High-Tech Research Development Program of China (863 program) under Grant No. 2007AA01Z139.

REFERENCES

[1] UDDI Spec Technical Committee Draft, Dated 2004.10.19, http://uddi.org/pubs/uddi_v3.htm

[2] jUDDI 3.0 - Developer Guide, Dated 2009.09.15, http://ws.apache.org/juddi/

[3] Added by Graham Barber (IBM), last edited by Graham Barber (IBM), "Service Component Architecture", Nov 07, 2007, http://www.osoa.org/display/Main/Service+ Component+ Architecture+Home

[4] Added by Graham Barber (IBM), last edited by Mike Edwards (IBM) "Service Data Objects", Dec 21, 2007 , http://www.osoa.org/display/Main/Service+Data+Objects +Specifications

[5] S. Simmons, "Introducing the WebSphere Integration Reference Architecture: A Service-based Foundation for Enterprise-Level Business Integration", IBM WebSphere Developer Technical Journal, Aug. 17, 2005, http://www-128.ibm.com/developerworks/websphere/techjournal/0508_simmons/0508_simmons.html.

[6] Daniel Rubio, "BizTalk Server: Microsoft's SOA building block", Jan. 24, 2006, http://searchwebservices. techtarget.com/tip/0,289483,sid26_gci1161311,00.html

[7] Windows Workflow Foundation Overview, http://msdn.microsoft.com/en-us/library/ms734631.aspx

[8] Oracle SOA Suite Datasheet, "Oracle SOA Suite", October 2006, http://www.oracle.com/technologies/soa/ oracle-soa-suite-datasheet.pdf

[9] Seog-Chan Oh, Hyunyoung Kil, Dongwon Lee, and Soundar R. T. Kumara, Algorithms for web services discovery and composition based on syntactic and semantic service descriptions, In *Proceedings of the 8th IEEE International Conference on E-Commerce Technology and the 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*, pp. 66 – 66, June 2006.

[10] Seog-Chan Oh, Byung-Won On, Eric J. Larson, Dongwon Lee, BF*: web services discovery and composition as graph search problem, In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service*, pp.784 – 786, April 2005.

[11] Bin Li, Xiao-yan Tang, Jian Lv, The research and implementation of services discovery agent in web services composition framework, In *Proceedings of the Fourth*

International Conference on Machine Learning and Cybernetics, vol. 1, pp.78 – 84, August 2005.
[12] Yannis Makripoulias, Christos Makris, Yiannis Panagis, Evangelos Sakkopoulos, Poulia Adamopoulou , Athanasios Tsakalidis, web service discovery based on quality of service, In *Proceedings of IEEE International Conference on Computer Systems and Applications,* pp.196-199, March 2006,.
[13] David Martin, et al., OWL-S: Semantic Markup for Web Services, November 2004, http://www.w3.org/Submission/OWL-S/
[14] Kyriakos Kritikos, Dimitris Plexousakis, Semantic QoS Metric Matching, In *Proceedings of ECOWS 2006: Fourth European Conference on Web Services*, pp.265-274, 2006
[15] Kyriakos Kritikos and Dimitris Plexousakis, OWL-Q for semantic QoS-based web service description and discovery, In *Proceedings of the SMR2 2007 Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web*, pp.123-137, November 2007.
[16] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM 2001*, pp. 149–160, August 2001.
[17] Siming Xiong, Haopeng Chen, QMC: a service registry extension providing QoS support, In *Proceedings of 2009 International Conference on New Trends in Information and Service Science*, pp.145-151, July 2009.
[18] Guang Yang, Hao-peng Chen, An extensible computing model for reputation evaluation based on objective and automatic feedbacks, In *Proceedings of International Conference on Advanced Language Processing and Web Information Technology 2008*, pp. 585 – 592, July 2008.
[19] Shu-jia Wang, Hao-peng Chen, A web service selecting model based on measurable QoS attributes of client-side, In *Proceedings of 2008 International Conference on Computer Science and Software Engineering,* pp.385-389, December 2008.
[20] Cheng Zhou, Haopeng Chen, A mechanism for collecting and feedbacking the real-time quality of web service, In *Proceedings of the 1st International Conference on Information Science and Engineering*, in press.
[21] Cheng Zhou, Haopeng Chen, An objective and automatic feedback model for QoS evaluation, In *Proceedings of International Conference on Computer Sciences and Convergence Information Technology*, in press.
[22] Hao-peng Chen, Cheng Zhang, A queueing-theory-based fault detection mechanism for SOA-nased applications, In *Proceedings of IEEE Joint Conference on E-Commerce Technology and Enterprise Computing, E-Commerce and E-Services*, pp.265 – 269, July 2007.
[23] Hao-peng Chen, Cheng Zhang, A fault detection mechanism for service-oriented architecture based on queueing theory, In *Proceedings of IEEE 7th International Conference on Computer and Information Technology*, pp. 1071-1076, October 2007.
[24] Hao-peng Chen, Zhi-yong Wang. A fault detection mechanism for fault-tolerant SOA-based applications, In *Proceedings of the sixth IEEE International Conference of Machine Learning and Cybernetics*, pp.3777-3781, August 2007.
[25] Jiang Ma, Hao-peng Chen, A reliability evaluation framework on composite web service, In *Proceedings of IEEE 4th International Symposium on Service-Oriented System Engineering*, pp.123-128, December 2008.
[26] Jinbo Du, Haopeng Chen, Can Zhang, A heuristic approach with branch cut to service substitution in service orchestration, In *Proceedings of The 4th International Conference on Frontier of Computer Science and Technology,* pp.59-67, December 2009.
[27] Meng Li, Hao-peng Chen, Nan Wang, Description and Calculation of Qualities of Composite Services, *In Proceedings of 2009 IEEE Asia-Pacific Services Computing Conference,* pp.385-390, December 2009.

Hao-peng Chen.
Anhui Province, China.
Sept. 4th, 1975.
Ph.D's degree in computer software and theory 2001 at Department of Computer Science and Engineering, Northwestern Polytechnical University, Xian, China. Master's degree in computer application 1999 at Department of Computer Science and Engineering, Northwestern Polytechnical University, Xian, China. Bachelor's degree in computer software 1996 at Department of Computer Science and Engineering, Hangzhou Institute of Electronic Engineering, Hangzhou, China.

He is an ASSOCIATE PROFESSOR at School of Software in Shanghai Jiao Tong University. His research interests include distributed computing, software engineering, and software architecture.

Can Zhang.
Jiangsu Province, China.
Dec. 5th, 1987.
Bachelor's degree in software engineering 2009 at School of Software, Shanghai Jiao Tong University, Shanghai, China.
Master's degree in software engineering expected (2012) at School of Software, Shanghai Jiao Tong University, Shanghai, China.

She is currently a postgraduate student in School of Software, Shanghai Jiao Tong University. She got a summer internship with CRDC (Cisco China Research & Development Center) in July 2008. And her major research interests are service computing, software architecture.

Guang Yang.
Shanghai, China.
Mar. 27th, 1984.
Master's degree in software engineering 2009 at School of Software, Shanghai Jiao Tong University, Shanghai, China.
Bachelor's degree in software engineering 2006 at School of Software, Shanghai Jiao Tong University, Shanghai, China.

He is currently a full time employee with Microsoft (China) Server and Tools Business as a Program Manager working on workflow. His major research interests include SOA, business process management and internet application.