A Closed-loop Mechanism for Service Evaluating and Discovering on the Internet [1]

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

Guang Yang
School of Software
Shanghai Jiao Tong University
Shanghai, China
e-mail: allenmacyoung@gmail.com

Can Zhang
School of Software
Shanghai Jiao Tong University
Shanghai, China
e-mail: zhangcan05@gmail.com

***Abstract*—In this paper, we analyze the problems of existing Service Computing model and propose a closed-loop mechanism for service evaluating and discovering on the Internet which is not only compatible with the basic principles of Service Computing—loosely coupled, protocol independent and location transparent, but also can rank, classify and recommend services based on their real-time performance. This mechanism facilitates dynamic service discovery and substitution which is the core of service computing since it improves the availability, scalability, and modifiability of service-based applications.**

Keywords-QoS; closed-loop; SOA; evaluation; discovery; dynamic substitution

I. INTRODUCTION

Service Computing is emerging as a new discipline in the Distributed Computing field. Thanks to the development of Web Service technology and the application of Service-Oriented Architecture (SOA), Service Computing has 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 an application which is based on different services, the quality of each service may affect the performance of the whole system. Therefore, in order to build an application of good quality, we have to face the following problems:

a) 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. In fact, this has already been supported by a lot of modeling tools.

b) How to discover those services which meet the application's QoS requirements. With the development of Service Computing theory and technology, there're more and more services on the Internet which provide almost the same function. As a result, developers of SOA applications are more concerned with the application's QoS attributes than ever before. They want faster and better services.

One way to solve this 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 runtime performance 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.

c) How to automatically substitute service when service failure occurs. Even if we can ensure that the services selected can meet the QoS requirements when building the system, we can never guarantee that the

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

978-0-7695-3769-6/09 $26.00 © 2009 IEEE
DOI 10.1109/NCM.2009.145

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 running process is not interrupted and the failed service is quickly and efficiently substituted.

d) How to build a comprehensive closed loop SOA governance infrastructure, where the service registry can monitor, rank, classify and recommend services according to their runtime performance. If we can enable a closed loop from service publication, discovery and location, binding and invoking to service monitoring, evaluation, ranking, recommendation, the service consumer will get the most suitable service among those services which satisfy the functional and QoS requirements of the system.

In this paper, we show how our Closed-loop Mechanism solves these problems. In general, our Closed-loop Mechanism involves three parts: the Intelligent Service Registry, QoS interceptors of services and service processes, and the fault-tolerant application front-end. Each part plays an importance part in the mechanism. First, Intelligent Service Registry stores the static descriptions and dynamic feedbacks of registered services and enables Service Consumer to search and discover services by the functional and/or QoS requirements. Also, Intelligent Service Registry evaluates, ranks, and recommends services according to their real-time QoS attributes. Second, QoS interceptors of services and service processes make their real-time QoS attributes describable and feedback these attributes to the intelligent registry. Third, in our mechanism, the application front-end has the ability of dynamic service discovery, dynamic service composition, and fault -tolerance.

The rest of this paper is organized as follows. Section 2 surveys some related work. Section 3 presents the overview of our mechanism. Section 4.1 shows the architecture of the Intelligent Service Registry. Section 4.2 presents how we design the QoS interceptors of services and service processes. Section 4.3 shows the framework of application front-end. 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 [1] and Service Data Object [2]. 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 [3]. 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 [4]. 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 [5]. 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 [6].

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 [7], authors proposed algorithms for web services discovery and composition by parsing syntactic and semantic service

descriptions; in [8], authors adapted A* algorithm to effectively search Web Services; in [9], authors designed and developed an agent-based Web Service composition framework; in [10], authors proposed a Web Service discovery mechanism based on Quality of Service, which classifies Web Services according to their QoS descriptions beforehand. Many other researches can be found in various journals and proceedings.

However, we find that almost all these researches are based on static QoS descriptions rather than real-time performance data. Thus, these QoS descriptions can hardly reflect the real-time qualities of services. Consequently, the dynamic service discovery and substitution becomes impossible and nonsense.

In summary, we need to design a closed-loop mechanism to dynamically describe and discover services according to their real-time qualities. This mechanism also should be compatible with existing service computing models.

III. RATIONALE OF CLOSE-LOOP BASED MECHANISM

The collaborative framework shown in Figure 1 outlines the closed-loop mechanism for service evaluating and discovering. 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.

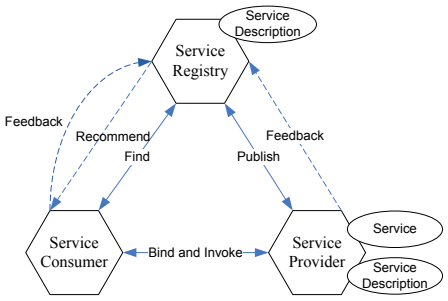

Figure 1. Collaborative framework of the closed-loop mechanism for service evaluating and discovering

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 mechanism, we add some new meanings to these three collaborations. For Publish, the service descriptions published will also include the QoS attributes in formal language to make automated processing possible. For Find, besides finding services according to functional requirements, we also support finding services according to QoS requirements. For Bind and Invoke, we bind Service Consumer with services which have the same functional requirements rather than one specific service.

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 runtime performance data 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 runtime 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 runtime performance data 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 mechanism. 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 mechanism.

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 mechanism in details.

IV. DETAILS OF CLOSED-LOOP BASED MECHANISM

There are three parts in the close-loop based mechanism for service evaluating and discovering. They are intelligent service registry, QoS interceptors, and the framework of application front-end. This section will describe these parts in detail.

A. The Architecture of Intelligent Service Registry

In our mechanism, since Service Registry takes the responsibility of collecting real-time performance data of services, evaluating and ranking services, and recommending services of good quality, we call it an intelligent Service Registry. Figure 2 shows the architecture of the Intelligent Service Registry.

In the architecture shown in Figure 2, 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 compatible with existing service registry to enable service providers register their services into the Registry. With the Service Discovery Interface, those service-based applications can find and locate services they are interested in. The Registry P2P Interface is used for interaction between different service registry centers for sharing real-time information so as to manage the whole distributed system. The Service Feedback Interface 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.

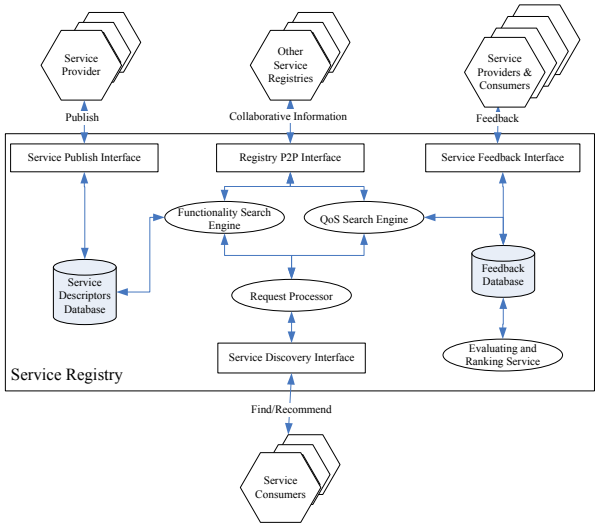

Figure 2. The architecture of Intelligent Service Registry

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(Universal Description, Discovery, and Integration)-complied registry, to allow service provider to register services. We also designed such an interface in our registry in order to be compatible with existing registries. 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. 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 will find a set of services which satisfy the functional requirements. Third, the QoS Search Engine 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.

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 [11] protocol as the distributing protocol of service descriptors. Furthermore, there are multiple 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 runtime performance of those services chosen by the Service Consumer.

The fourth scenario is about monitoring and collecting real-time performance of the registered services. Periodically, feedback on the runtime performance of 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.

Distributed Intelligent Service Registry plays an important part in our mechanism. 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.

We also consider that we can design a separate Quality Measurement Center (QMC) to evaluate, rank, and classify

services according to their qualities, and provide searching function for Service Consumers. QMC can be a third-party service search engine since it is independent of any service registry. We have also established a prototype of QMC, which need to be integrated with service registry in the future.

B. QoS interceptors

We have to design QoS interceptors for services in our mechanism to enable the feedback mechanism. The feedback is used in two ways. First, Service Registry will use the feedback data to evaluate and rank services. Second, service-based applications will monitor the runtime performance of services used and decide if dynamic replacement of services is needed according to the feedback data. Figure 3 shows the QoS interceptor of service with feedback function.

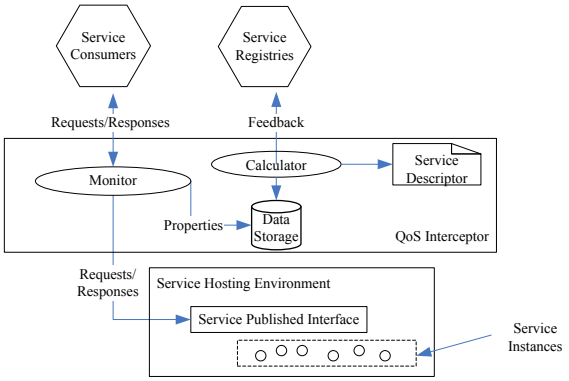

Figure 3. QoS interceptor of service

In the model above, the QoS interceptor is composed of a Monitor, a Calculator and a lightweight Data Storage. 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. At the arrival time of one request, the Monitor records information like arrival time and calculates the average request frequency, and then stores them in the Data Storage. Then, the request is passed to the Service Hosting Environment which transfers the request and response of each service invocation on the Internet over some well-known protocols, like XML-based SOAP message, into a local call, and dispatch the request to proper Service Published Interface which is the interface exposed to Service Consumers. Finally, the request is delivered to some instance of proper service. After the instance returns the result, Service Hosting Environment will pack it up into a standard response package, like a SOAP message, and send this package to the Service Consumer. At the response time of the invocation, the Monitor also records information like execution time, and response time.

The Calculator periodically accesses data in Data Storage, calculates the QoS attributes, modifies the relevant part of service descriptor, and then sends the calculated QoS attributes to Service Registry.

We have designed three kinds of QoS interceptors:

a) Proxy-based interceptor. This kind of interceptors is a proxy hosting in the environment of its target services. With this kind of interceptor, Service Consumer should explicitly invoke the proxy in its code. The proxy takes the responsibility of delivering invocations to proper services. That's to say, Service Consumer has to modify its code it wants to use this kind of interceptors. However, this kind of interceptors is the simplest and most accurate one.

b) AOP-based interceptor. This kind of interceptors also hosts in the environment of its target services. Unlike the proxy-based interceptor, Service Consumer does not have to modify its code explicitly. The AOP complier interweaves the necessary codes into the compiled class file to provide functions of interception and feedback. However, this kind of interceptors is dependent on the approach of implementation of target services.

c) Port-based interceptor. This kind of interceptors is independent of the implementation of target services and runs as a separate process. It monitors the ports used by target services, 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 interceptor

for calculating. This kind of interceptors, however, is dependent on the type of hosting Operating System.

In summary, each of these three kinds of QoS interceptors has its advantages and disadvantages. The QoS interceptor can be built in the service computing based application front-end or run in the hosting environment of service. As Service Providers and Service Consumers both need the QoS interceptor, they may choose the most suitable one according to their specific requirements.

C. *Framework of Application Front-end Based on Service Computing*

The application front-end based on service computing is supposed to have the ability of dynamic service discovery, dynamic service replacement, and fault discovery. Since all the services of a service computing based application are distributed on the Internet, the application front-end runs as a mediator and controller of these services. To accomplish this goal, we propose a framework for the application front-end in Figure 4.

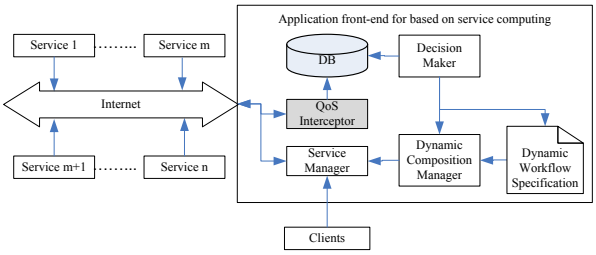

Figure 4. A framework of application front-end based on Service Computing

In the framework shown above, the application front-end has two main functions. First, it is responsible for managing session status and controlling business process. Second, it determines whether there are some services should be substituted with other service according to their real-time quality and executes the substitution.

On one hand, the application front-end accepts client requests. When one client request arrives, the Service Manager takes the responsibility of managing the client session, and processing the client request according to the Dynamic Workflow Specification. This part is almost the same as the common service computing model.

On the other hand, the built-in QoS interceptor in the application front-end keeps on collecting runtime status of all the services involved in the application in order to monitor the service endpoint behavior and the QoS attributes. What's more, all the data collected are stored in a database. These data are used later in two ways. First, we can detect failure in the application with these data. Second, with these data, the Decision Maker periodically calculates the end-to-end QoS attributes of the whole system according to some service measurement model like queuing model or queuing network model. The Dynamic Workflow Specification describes the quality requirements about business processes as well as the business logics. Decision Maker compares the calculated real-time quality of business process with the quality requirements described in Dynamic Workflow Specification. If the QoS attributes no longer satisfy the requirements, the Decision Maker will notify the Dynamic Composition Manager that it should try to find alternative services. To find new services, the Dynamic Composition Manager interacts with Service Registry via Service Manager, and then modifies the Dynamic Process Specification to include the new services. As a result, following requests will be processed according to the new specification.

We have designed more than three algorithms for detecting fault services[12][13][14] and an algorithm for substituting fault service[15]. We keep researching on the improvement or these algorithms. In the future, we will design and implement a prototype of this front-end.

V. CONCLUSION

In this paper, we have presented a close-loop mechanism for service evaluating and discovering on the Internet. In particular, we focused on the dynamic features of the mechanism, such as dynamic service discovery and composition, dynamic service substitution and fault detection. What's more, this mechanism can rank, classify

and recommend services according to their real-time performance.

In summary, we believe that our mechanism 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.

REFERENCE

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

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

[3] 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, available at: http://www-128.ibm.com/developerworks/websphere/techjournal/0508_simmons/0508_simmons.html.

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

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

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

[7] 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", 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 (CEC/EEE'06), June 2006, pp. 66 - 66

[8] Seog-Chan Oh, Byung-Won On, Eric J. Larson, Dongwon Lee, "BF*: Web Services Discovery and Composition as Graph Search Problem", Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE '05), 29 March-1 April 2005, pp.784 - 786

[9] Bin Li, Xiao-yan Tang, Jian Lv, "The Research and Implementation of Services Discovery Agent in Web Services Composition Framework", Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou (ICMLC'05), 18-21 August 2005, Volume 1, pp.78 - 84

[10] Yannis Makripoulias, Christos Makris, Yiannis Panagis, Evangelos Sakkopoulos, Poulia Adamopoulou , Athanasios Tsakalidis, "Web Service discovery based on Quality of Service", IEEE International Conference on Computer Systems and Applications (ICCSA'06), March 2006, pp.196 – 199

[11] 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, pages. 149–160,San Diego, CA, 2001.

[12] Hao-peng Chen, Cheng Zhang, A Fault Detection Mechanism for Service-Oriented Architecture Based on Queueing Theory, IEEE 7th International Conference on Computer and Information Technology 2007 (CIT2007), PP. 1071-1076, 2007.10, ISBN 978-0-7695-2983-7.

[13] Hao-peng Chen, Zhi-yong Wang. A Fault Detection Mechanism for Fault-Tolerant SOA-Based Applications, The sixth IEEE International Conference of Machine Learning and Cybernetics (ICMLC 2007), PP.3777-3781, 2007.8, ISBN 1-4244-0972-1.

[14] Hao-peng Chen, Cheng Zhang, A Queueing-Theory-Based Fault Detection Mechanism for SOA-Based Applications, IEEE Joint Conference on E-Commerce Technology (CEC'07) and Enterprise Computing, E-Commerce and E-Services (EEE '07), PP.265 – 269, 2007.7, ISBN 0-7695-2913-5.

[15] Jiang Ma, Hao-peng Chen, A Reliability Evaluation Framework on Composite Web Service, IEEE 4th International Symposium on Service-Oriented System Engineering (SOSE 2008), PP.123-128, 2008.12, ISBN 978-0-7695-3499-2/08.