Open Framework Supporting Multimedia Web Services

Jia Zhang
Computer Science Department
Northern Illinois University
DeKalb, IL 60115
jiazhang@cs.niu.edu

Jen-Yao Chung
IBM T.J. Watson Research
Yorktown Heights, New York 10598
jychung@us.ibm.com

Abstract

With the rapid emergence of web services, more and more web services are published on the Internet as resources for web application development. There may exist some relationships among different web services, such as exact match, plug-in match, and irrelevant. In this paper, we discuss the issues related to multimedia web services, and propose a three-tier framework in order to establish an open environment supporting multimedia web services. With the description of the architecture and its implementation, we can make a multimedia web services oriented system more transparent, interoperable, and fault-tolerate.

1. Introduction

Web services are broadly regarded as self-contained, self-describing, and modular applications that can be published, located, and invoked across the Internet [7]. This emerging paradigm opens a new way of web application design and development to quickly develop and deploy web applications by integrating other independently published web services components to conduct new business transactions. However, since the web services components are actually integrated at run time through the Internet, one essential problem arising is how to guarantee that a web service can be obtained dynamically with transparency and fault tolerance. Furthermore, when web services contain multimedia elements, only the services that satisfy the quality of service (QoS) at the time should be taken into consideration.

Roy [10] summarizes a typical architectural model for web services am
service brokers,
publish web
requesters dem
then service req
providers. This
of web services
interoperable. A
services to co
Figure 1 illu

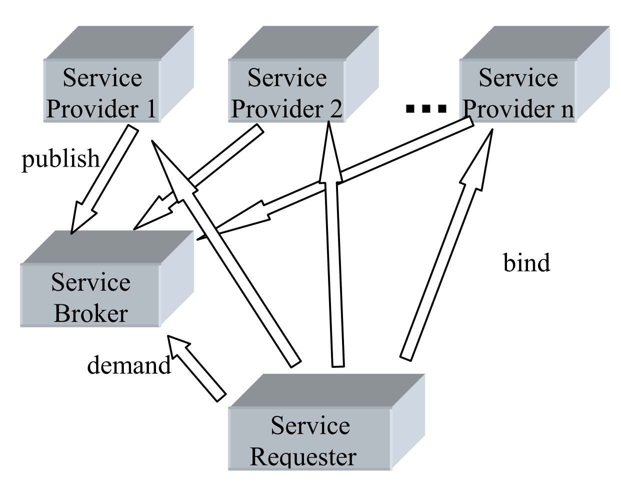

Figure 1. More sophisticated situation

architecture. Suppose one web application needs to integrate three published web services, each from a different service provider 1, 2, 3, respectively. These three service providers publish their services on the same service broker. When a service requester obtains the locations of these three services from the service broker, it needs to invoke the three web services separately from different service providers. Therefore, the service requester needs to be aware of the detailed access information of each service provider, such as the location, and even port number of the desired service. When an unexpected accident occurs on the web, say, service provider 1 crashes. The service requester has to reaccess the service broker for a substitute, and reestablish the connection to the new service provider. As a result, the service requester has to end up handling invocation and error handling of every web services needed, which is

Table 1. Requirements and mechanisms

Requirements	**Mechanisms**
transparency of locating and invoking web services	encapsulation of the details of location and invocation
management of relationships between web services	definition of the service relationships
dynamic selection and composition of web services	dynamic selection and dynamic binding
QoS awareness	support of the expression of QoS parameters
fault tolerance	dynamic selection and dynamic binding
distribution of web service registration	trading between service brokers, caching

prototype implementation of the framework. In Section 7 we present our evaluation. In Section 8 we draw a conclusion and discuss future work.

2. Problem Domain Definition

As the first step of the design process, we identify the requirements in order to support web services. We believe that there are six essential requirements: transparency, management of service relationships, dynamic selection and composition, quality of service, fault tolerance, and distribution of web services registration. Table 1 summarizes these requirements and the mechanisms that we believe to be capable of fulfilling each requirement. For brevity, in this paper we omit the discussion for other possible requirements, such as resource localization, multicast support, support for continuous media, real time synchronization, security, latency tolerance, etc.

The first is the transparency of locating and invoking web services. Similar to network transparency [9], web services transparency means that web services behave in the same way independent of their distributed locations and execution environments. To achieve the transparency, a mechanism needs to be provided to encapsulate the details of location and invocation of web services.

The second is the management of relationships between published web services. A large amount of web services have already been published on the Internet, and the number and types of web services grow rapidly [4]. How to select an appropriate web service, and how to select a substitute when one web service is unavailable, are of paramount importance. Therefore, mechanism needs to be provided to define the relationships between web services, such as exact matching, substitutable matching, etc.

The third is the dynamic selection and composition of web services into a new business transaction. Due to the unpredictable feature of Internet, some pre-selected web services may be temporarily unavailable at some time; therefore other compatible web services should be selected as replacements. As a result, applications based on web services should be able to choose and compose the web services to be used at run time. Furthermore, since web services will be dynamically selected, static binding at compiling time and linking time is not practical. Therefore, dynamic binding needs to be supported.

The fourth is the quality of service (QoS) awareness. More and more web services contain multimedia elements that require timeliness of transmissions [6]. In addition, a web service may become overloaded at some point and stop responding in a timely fashion, which will violate QoS requirements. Therefore, the selection of web services should not only be based upon availability, but also on QoS characteristics. In order to select the web service that fulfills the QoS requirements, mechanisms need to be provided to support the expression of QoS parameters, so that web services can be selected based upon their QoS values if so desired. In this paper we do not discuss the mechanism to ensure the QoS through the Internet transmission.

The fifth is fault tolerance. The fault tolerance here refers to the ability of a web services oriented system to respond gracefully to an unexpected web services failure. Considering an application that is composed of several web services and is executed the second time, from the first time of execution, the set of web services used will be cached. Since each web service will be invoked remotely from its resident site at the time of invocation, it is possible that one web service crashes without warning after the first invocation. The system needs to be able to make some special arrangements to find a new web service to replace the failed one, so that a service requester will still obtain the whole application even a certain web service is crashed. Dynamic selection and binding of web services can be the mechanism to achieve this goal.

The sixth is the distribution of web services registration. As more and more web services are published on the Internet, it is infeasible to have one central service broker that handles the entire pool of published web services. As a result, there will be many service brokers on the web, each managing some web services. When a web service is requested, the closest service broker will be first checked. If the expected web service is not found, the service broker should automatically contact with other service brokers for the appropriate service. If the service is found elsewhere, the original service broker should duplicate the service information to its local storage for future usages. Mechanisms should be provided to support the trading between service brokers described here.

3. Related Work

Remote Procedure Call (RPC) is a powerful mechanism in distributed computing, which enables software to make procedure calls over the Internet onto another procedure running on distributed machines. XML-RPC [8] utilizes the standard eXensible Markup Language (XML) [15] encoding strategy so that systems can be loose coupled and highly interoperable; the issue of argument marshaling existing with the traditional RPC is resolved due to the fact that all data is encoded as text before transmission [1]. Apache XML-RPC [2] is a Java implementation of XML-RPC. Although XML-RPC is simple to understand and use, its goal of simplicity decides that it cannot handle complex data types. Simple Object Access Protocol (SOAP) [11], on the other hand, is a more comprehensive and powerful transportation protocol, which can handle complex data types such as user defined data types, and have the ability to have each message define its specific processing control and recipient. Becoming ad-hoc standard of web services field, the SOAP specification defines a convention to represent RPC calls and responses. Therefore, SOAP covers XML-RPC and provides more power of supporting web services oriented system. As a result, in our research, we decide to adopt SOAP RPC to access remote web services. Meanwhile, our previous work enhances SOAP in order to improve the ability and flexibility of the ad hoc standard SOAP protocol to serve for multimedia web services, by supporting batch facility and carrying on QoS requirements [16]. Consequently, we utilize our enhanced SOAP to transfer request messages.

Researchers, especially those from the field of web services discovery, have been interested on identifying QoS features as requirements of web services location. UX [3] suggests three QoS parameters: response time, cost, and reliability. Vinoski [13] summarizes five QoS parameters: latency as the average time for an operation to return the results after its invocation, fees as the money needed to be paid to invoke operations, availability as the probability that the web service is present and ready to be invoked, accessibility as the degree of being capable of serving a request, and reliability as the degree of being capable of maintaining the service and service quality. In our work, we choose to adopt several multimedia-related QoS parameters: response time, reliability, availability, and accessibility.

Meanwhile, a powerful language to formally and precisely define a web service is of paramount importance. Web service description language (WSDL) [14] from W3C is becoming the ad hoc standard for web services publication. However, WSDL can only specify limited information of a web service as the function names and limited input and output information [4]. Gao and colleagues [4] propose a web service capability description language (SCDL) to describe, advertise, request, and match web services capabilities precisely. SCDL defines four types of atomic web service capability matches: exact match, plug-in match, relaxed match, and not relevant. The paper provides a theoretical basis to define web service capability matching. However, the paper does not provide any information about the implementation of the SCDL language; its previous version SDL [5] is still at early development stages [12]. Therefore, the usage of SCDL in web service applications is still unclear.

4. Basic Definitions

To address the issues discussed above, a three-tier framework supporting web services is proposed. To facilitate our discussion of the framework, however, we need to define some basic concepts first. For brevity, in this paper, we will use the term web service and service interchangeably.

Definition 1: Web service

In this paper, each web service is defined as a 6-tuple (hostId, ontoDes, Sigs, Pre, Post, QoS), where:

- hostId: is the unique identifier of the hosting server machine of the web service;
- ontoDes: is the ontological description. This element defines the concept of the context and its meaning description [4].
- Sigs: is the set of RPC methods exposed by the service:

  Sigs ::= {M1 V M2, …, V Mi…}, $i \geq 1$

  Each RPC method can be defined as follows:

  M ::= (N, i_1, i_2,… , i_m, o_1, o_2,…o_n) where:

  N: the name of the method;

  i_1, i_2,… , i_m: the list of the types of the input parameters;

  o_1, o_2,…o_n: the list of the types of output parameters;
- Pre: is the pre-condition of the web service;
- Post: is the post-condition of the web service;
- QoS: is the QoS feature of the web service.

Definition 2: Signature match

Considering two RPC methods A and B exposed by web services, method A is regarded as signature matching to method B if:

1. The input parameters of A are super types of those of B;
2. The output parameters of A are subtypes of those of B.

Definition 3: Plug-in match web service

Plug-in match defines a substitute relationship between web services. If a web service X is a plug-in match service of web service Y, it means that web service X can be plugged into the place where web service Y is to be used as a substitute, but not vice versa. A web service X is a plug-in match service of web service Y, if:

1. The method set defined by X is a superset of that defined by Y;
2. For each mutual method between X and Y, the signature of the method signature of X signature matches that of Y.
3. The specification of X semantically matches the specification of Y. In other words, (pre-X => pre-Y) ^ (post-Y => post-X) [4].
4. For every QoS requirement defined in Y, X fulfils the same QoS requirement with super type

(i.e. for every QoS specification, X satisfies with stronger features).

Definition 4: Exact match web service

Exact match web service defines that two web services are potentially interchangeable. Two web services, say X and Y, are considered to be exact match if X is plug-in match with Y and Y is plug-in match with X.

Definition 5: Related web services

Related web services defines that two web services are either exact match or plug-in match. Otherwise two web services are considered irrelevant services.

Definition 6: Service domain

A service domain in this paper is a conceptual term for the purpose of the management of web services. A service domain is defined based upon the distribution of web services registration. All service providers who register on a service broker form a service domain together with the service broker. As new services register onto the service broker, or some old services remove from the service broker, the boundary of the service domain alters accordingly. Therefore, the concept of a service domain represents a set of published web services.

5. Three-tier Framework Supporting Multimedia Web Services

Based upon our previous discussions, in this section, we present a three-tier framework for web services, which aims to support the integration of the mechanisms we propose to fulfill the requirements. As illustrated in Figure 2, there are three layers in the framework: service providers, service broker, and service requesters. Multiple service providers register onto a same service broker; and multiple service requesters access the same service broker. Containing still three components in the model, our framework differs from the traditional one in the following ways. In the normal architecture, as we discussed in the first section, when a service requester asks for a web service, the service broker finds the expected service and returns the service provider's information to the service requester, then the service requester will connect to the specific service provider for the service by itself. In our model, on the other hand, the service broker serves as the middle tier between service providers and service requesters; and service requesters will connect to the service providers through the service broker. In addition, a service broker will not only serve for service registration and management, but also serve for dynamic service selection and binding, caching, service trading, etc.

The service broker is wrapped by an XML layer. This XML layer can be implemented by different XML-based technologies, such as SOAP, UDDI, and WSDL. This paper will not discuss the related technologies. Therefore, we use *XML layer* merely represents that the communication between the service broker and the service providers and service requesters are all based on XML technology. The internal structure of a service broker contains the following five functional components: service registrar, service manager, service binder, service trader, and service analyzer.

Service registrar handles service registration for service providers and the service trader component. Service providers register new web services onto the service broker, or remove old services from it. Commonly, the service registrar normally maintains one repository of registered web services, and also provides an operation engine over the repository, e.g. adding/removing a service entry, and query functionality for service binder, which will be discussed below. Through the service registrar, the details of web services are encapsulated from service requesters. The service trader component can also register web services into the service registrar - the scenario will be discussed later.

Service manager is meant to manage the registered web services in order to realize the dynamic selection and binding of QoS-aware web services with transparency and fault tolerance. Managing the relationships between registered web services, service manager selects an appropriate service from the service pool at run time, based on the functionality and QoS requirements. Then the service binder will try to bind to the selected service provider. If the chosen service is not available, or cannot satisfy the QoS requirements at the moment, the service manager will look into its service pool again for a related service. If there is no related service available at the time, the service manager will notify the service requester to try at later time. The service manager normally maintains one repository of registered services grouped with relationships, e.g. exact matching and plug-in matching.

Service binder dedicates to provide dynamic binding service for service requesters. Either a service requester or the service manager can invoke the service binder for service. On receiving requests, the service binder will query the registered service repository for the proper binding properties, such as the service host machine identifier and service name, etc. If the service binder can not set up the connection, the service binder will notify the service manager for a substitute web service, and try to build the connection again.

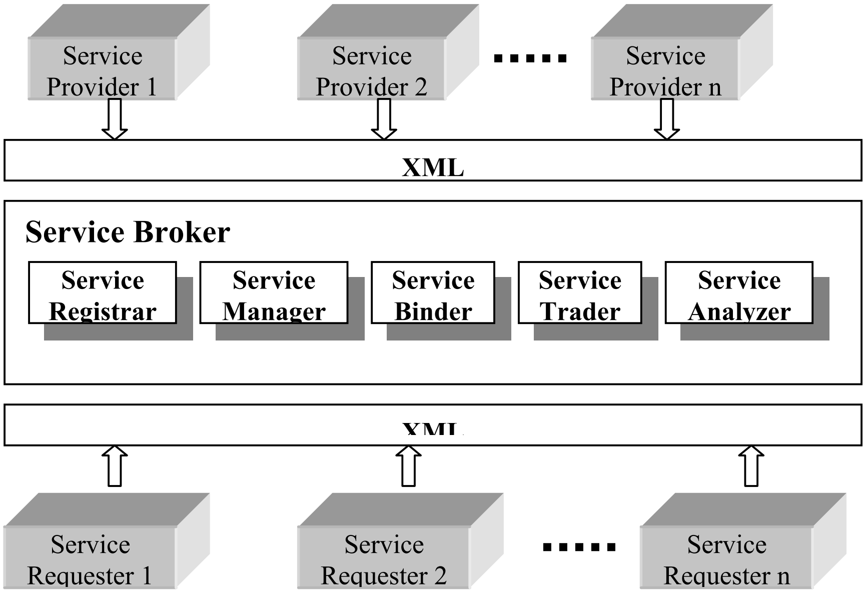

Figure 2. Three-tier framework supporting web services

Service trader manages the trading facility among different service domains. A service requester sends a request to a service broker, maybe because it is a member of the service broker, or the specific service broker is in its local area, etc. It is possible that a requested web service can not be found on one service trader, the service trader needs to forward the request to other service brokers, or service domains, for the particular web service. The QoS requirements associated with the request should be sent together as well. Therefore, the service trader needs to maintain a pool of other service brokers. If a service is found from another service broker, the original service trader will register the web service through the service registrar component in the same service broker. When a service registration is copied over, the associated popularity, which will be discussed below, will as well be copied. Therefore this duplicate copy of the service registration can serve for future requests in the original service domain. Meanwhile, when a service trader receives a request from another service broker, it will check the service manager component for the requested service.

Service analyzer is a utility component, which provides statistical analysis on the popularity of registered web services in a service domain. For example, whenever a binding between the service binder and a service provider is successfully set up, the counter associated with the specific web service will be increased by one. The higher the counter is, the higher popularity the service provider will be. The popularity will be one of the essential criteria for the service manager to select appropriate web services, when multiple registered web services provide similar functionality and QoS parameters.

6. Implementation

We have implemented a prototype system based on our three-tier framework. Figure 3 illustrates the architecture of the prototype system. In our system all service providers expose their web services with RPC interfaces. Here we will focus on the structure of service broker. We utilize our enhanced SOAP protocol [16] to serve for the communication channels between the service broker and service providers, due to its ability to transfer QoS parameters and facilitate multimedia transportation. As we have not discovered an ideal description language for web services publication, as we discussed in the previous section, in this prototype system we implement a registrar component together with interfaces for service providers to register their web services with QoS requirements. This is certainly just a temporary solution, but it can help us prove the concept of our framework; furthermore this module can be easily upgraded with a description language integrated in.

In the service broker, we implemented five main functional modules and three repositories. The five modules are service registrar, service manager, service binder, service trader, and service analyzer. The functionality of each module follows our framework. The three repositories are: service repository, broker repository, and historical repository. These repositories store service information, other service brokers' information, and historical successful access information, respectively. Figure 3 also shows the relationships among them and the access paths among them. Service broker contains SOAP operator to generate and interpret to and from SOAP messages. Notice that there are three SOAP operators in Figure 3. As a matter of fact, there is only one SOAP operator exists. The reasons to have multiple SOAP operators are, one just for display purposes for easier painting, the other one is to emphasize that SOAP translation or generation are necessary at three places. To be brief, in the following description, we omit the steps of translation/generation of SOAP messages, as they are always necessary when the service broker communicate externally.

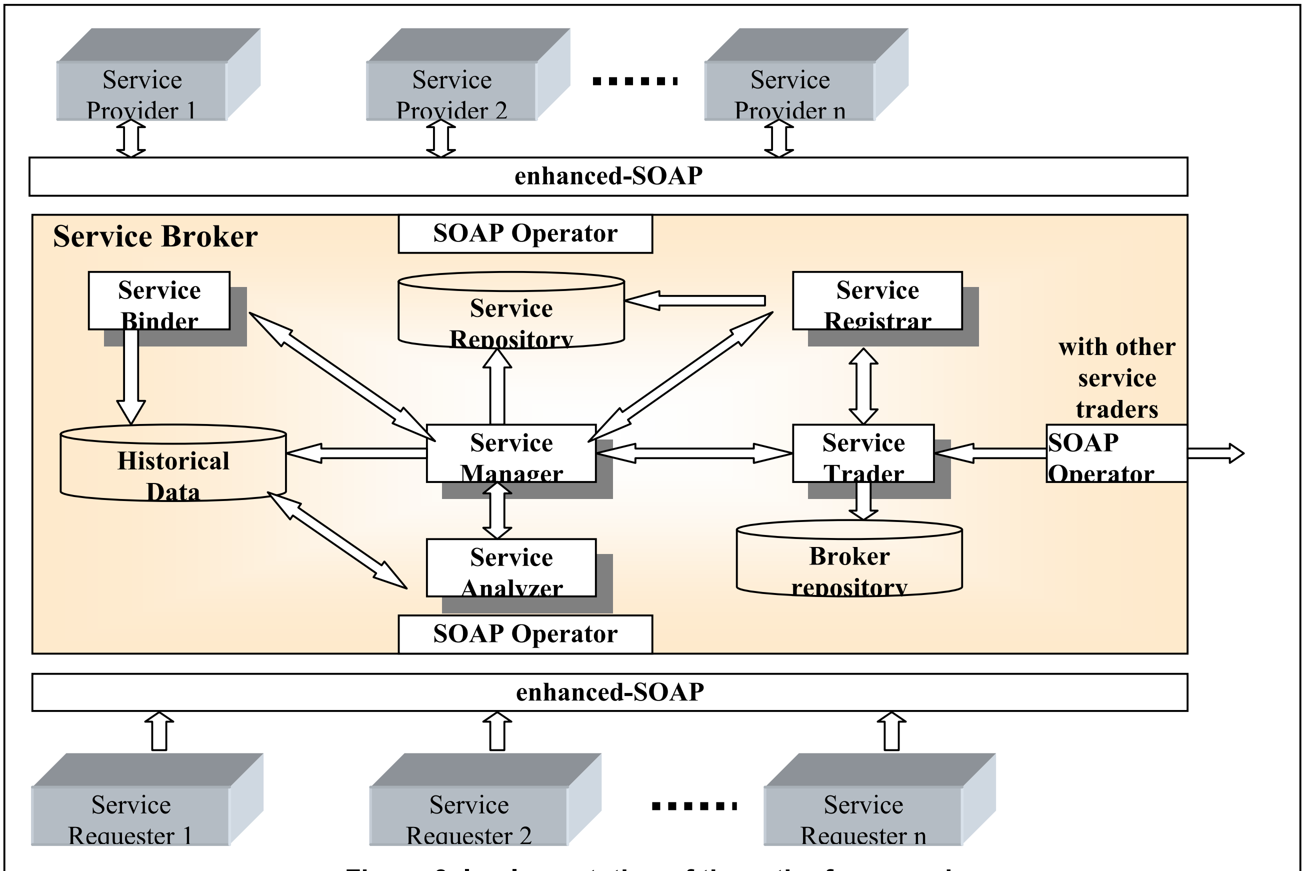

Figure 3. Implementation of three-tier framework

Service providers publish their services to the service broker through the service registrar module. The registration information follows our definition of a web service in previous section, which includes functionality, host machine identifier, QoS parameters, etc. The service registrar then stores the service information in the service repository.

When a service requester asks for a specific service, the request should contain the desired functionality of the service and QoS requirements. The request will be forwarded to the service manager, and the latter one searches the service repository and historical repository for an appropriate web service registered. Notice here that the service repository contains different views and categories based on the matching relationships between the web services, as we discussed in previous section. In our prototype system, we provide two options for service requesters. One is that service requesters give the service broker full right to automatically decide which web service to choose. The other one is to let the service broker provides candidate services, and leave to service requesters to decide which one will be invoked. To facilitate our discussion, here we assume to adopt the first option. The service manager will then coordinate with the service analyzer to decide the most appropriate web service. We will also skip here the algorithm to choose a web service, and simply assume that the service broker always selects the most appropriate web service, based on the QoS requirements and popularity.

If a service is found successfully from the service repository, the service binder will try to establish the connection to the host machine containing the chosen web service. If succeeded, the connection will be passed back to the service requester, and the service binder will record the successful access information into the historical repository. Otherwise, the service manager will repeat the previous action to search for a replacement, and the service binder will continue to bind to the new host machine with the newly selected service. If the service is not found in the service repository, the request will be forwarded to the service trader, and the service trader will in turn search for the broker repository to find other service domains for the expected service. Service brokers from different service domains may interact with each other to share the recorded web services. If one of the associated service brokers finds the desired service in its own service domain, the registration information will be passed back to the original service broker, and the information will be registered to its service repository through the service registrar. As a new web service is registered, the range of the service domain is correspondingly enlarged.

7. Evaluation

The main evaluation of our work was conducted to examine the effectiveness of our framework against the research issues we discussed in the Section 2 problem domain specification. For each issue, we scrutinize what solution our framework proposes, which components of our framework are involved, and check whether the issue has been fully solved or partially solved. The evaluation result is summarized in Table 2. For other issues related to web services, we do not discuss in this paper.

- Transparency of locating and invoking web services: The service manager, service binder, and service trader are involved. The service manager selects the appropriate service; the service binder connects to the chosen service. If the desired service does not register in the service domain, the service trader will be involved to find one in other service domains. Therefore, the detailed information about the location and the invocation of a proper service is masked from the service requester. This issue is solved by our framework.
- Management of relationships: The service manager is involved. The service manager decides the matching relationships between registered web services, and groups them accordingly to provide different views of the service repository. Consequently the service relationships are maintained. However, in reality, due to the fact that we have not found an ideal web service description language that is powerful enough to describe the characteristics of a QoS-required web service, this issue will remain partially solved until we find better solutions.
- Dynamic selection and composition: The service manager, service binder, and service trader are involved. The service manager conducts run-time selection of the appropriate web service based on the availability and popularity. The service binder helps to guarantee the availability of the web service. The service trader helps to locate a service. However, due to the same reason as above, this issue cannot be fully solved unless a powerful description language appears. In addition, in this paper we do not discuss the issue of service composition.
- QoS awareness: The service manager and the service binder are mainly involved. The service manager utilizes QoS parameters as a criterion to select services; and the service binder connects to the host machine of the service to check whether the QoS parameters remain the same at the moment. However, due to the same reason as above, this issue cannot be fully solved unless a powerful description language appears.
- Fault tolerance: The service manager and the service

Table 2. Requirements solving results

Requirements	Components involved	Result
transparency of locating and invoking web services	service manager, service binder, service trader	solved
management of relationships between published web services	service manager	partly solved
dynamic selection and composition of web services	service manager, service binder, service trader	partly solved
QoS awareness	service manager, service binder	partly solved
fault tolerance	service manager, service binder	solved
distribution of web service registration	service trader	solved

binder are mainly involved. The service manager and the service binder cooperate to achieve dynamic service selection. When a service is not available, the service manager will research for a substitute. Therefore, this issue is solved.

- Distribution of web service registration: The service trader is involved. The service trader facilitates web services to be registered at different service brokers locally. At the run-time, service traders can interact with each other to share the web service registration information. Therefore, the issue is solved.

Based on our evaluation result, we can see that our framework to large degree solves several essential issues related to web services.

8. Conclusions and Future Work

This paper discusses the issues related to web services oriented environment, and proposes a three-tiered framework in order to achieve transparent dynamic QoS-enabled web services with fault tolerance. We also discuss our implementation of the framework. This research work leads to establishing an open environment supporting web services.

Our future work includes the following directions. First we will explore a web service description language in order to support our requirements. Second, we will investigate the notification services between service domains. Third, we need to examine the effect of the security issue in our framework.

9. Acknowledgments

We would like to acknowledge our deep appreciation to Dr. Rodney Angotti for his valuable feedback and proof reading our manuscript.

10. References

[1] M. Allman, “An Evaluation of XML-RPC”, *ACM SIGMETRICS Performance Evaluation Review*, 30(4), Mar. 2003, pp. 2-11.
[2] http://ws.apache.org/xmlrpc.
[3] Z. Chen, L.-T. Chia, B. Silverajan, and B.-S. Lee, “UX – An Architecture Providing QoS-Aware and Federated Support for UDDI”, Proceedings of the International Conference on Web Services (ICWS’03), Las Vegas, NV, USA, Jun. 23-26, 2003, pp. 171-176.
[4] X. Gao, J. Yang, and M.P. Papazoglou, “The Capability Matching of Web Services”, Proceedings of the IEEE International Symposium on Multimedia Software Engineering (MSE’02), Newport Beach, CA, USA, Dec. 11-13, 2002, pp. 56-63.
[5] W.J. van den Heuvel, J. Yang, and M. Papazoglou, “Service Representation, Discovery and Composition for E-marketplaces”, Proceedings of the 6th International Conference on Cooperative Information Systems (COOPIS’01), Trento, Italy, Sep. 5-7, 2001, pp.
[6] D.W. Hong and C.S. Hong, "A QoS Management Framework for Distributed Multimedia Systems", *International Journal of Network Management*, 13(2), Mar./Apr. 2003, pp. 115-127.
[7] “IBM Web Services tutorial”, http://www-106.ibm.com/developerworks/webservices.
[8] S. Laurent, J. Johnston, and E. Dumbill, *Programming Web Services with XML-RPC,* O’Reilly, 2001.
[9] D.P. Reed, J.H. Saltzer, and D.D. Clark, “Comment on Active Networking and End-to-End Arguments”, *IEEE Network,* 12(3), May/Jun. 1998, pp. 69-71.
[10] J. Roy and A. Ramanujan, “Understanding Web Services”, *IEEE IT Professional,* Nov./Dec. 2001, pp. 69-73.
[11] http://www.w3.org/TR/SOAP.
[12] A. Tsalgatidou and T. Pilioura, “An Overview of Standards and Related Technology in Web Services”, *Distributed and Parallel Databases,* 12, 2002, pp. 135-162.
[13] S. Vinoski, “Service Discovery 101”, *IEEE Internet Computing,* 7(1), Jan./Feb. 2003.
[14] http://www.w3.org/TR/wsdl.
[15] T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maker, “Extensible Markup Language (XML) 1.0, 2nd Ed. W3C, 2000, http://www.w3.org/TR/2000/REC-xml-20001006.pdf.
[16] J. Zhang and J.-Y. Chung, “A SOAP-oriented Component-based Framework Supporting Device-independent Multimedia Web Services”, Proceedings of IEEE 4th International Symposium on Multimedia Software Engineering (MSE’02), Newport Beach, CA, USA, Dec.11-13, 2002, pp. 40-47.