A Portable Interceptor Mechanism on SOAP for Continuous Audit

Chen-Liang Fang[1], Deron Liang[2], Fengyi Lin[3],
Chien-Cheng Lin[2], William Cheng-Chung Chu[4]
[1]Jin-Wen Inst. of Tech., [2] National Taiwan Ocean University, [3]Chihlee Inst. of Tech., [4]Tunghai University
Email: fang@jwit.edu.tw, drliang@iis.sinica.edu.tw, linfengyi.tw@yahoo.com.tw, hammerlin@hotmail.com

Abstract

Web Services has become popular in modern distributed applications, and the SOAP technology currently is the most used in Web Services. Recent middleware research works widely use interception approach for many problem domains, for example Fault Tolerance, Continuous Audit (CA), security etc. Instead of providing a Portable Interceptor as CORBA does, SOAP 1.2 provides an intermediary mechanism. Due to backward compatibility issue, we found intermediary mechanism is not a feasible solution for interception. Furthermore, our use case analysis found that CORBA Portable Interceptor functionality does not fulfill all requirements of Continuous Audit. This motivates us to develop a new Portable Interceptor Mechanism (PIM) on SOAP for CA. In this paper, we propose a PIM on SOAP to meet the interception requirements for Web Services. Our PIM includes portable interceptor management in SOAP engine and portable interceptor interface definitions.

1. Introduction

SOAP technology has several advantages over other middleware technologies when it comes to building a distributed application system [8]. Our previous SOAP related works [4][6] had shown that an interceptor mechanism is needed to fulfill the requirements on logging, client fault transparency, and redundant nested invocation problem in a FT system. Based on our recent survey, we notice that recent middleware research works use interception approach in many other problem domains. For example, FT-SOAP [6] uses interception approach in logging/recovery mechanism; [9], [18] uses an interceptor to audit an accounting information system; and dSniff 16 uses interception approach in computer forensics auditing.

Instead of providing portable interceptor, SOAP 1.2 provides intermediary mechanism. Based on the discussion in [8], a SOAP 1.1 client might not be able to route message to intermediary node by adding routing information, <via> tag, to the SOAP message. This is so called a backward compatibility issue. We found intermediary approach is not a feasible solution for interception due to backward compatibility issue. Many SOAP 1.2 compliant products provide ways that a request can be pre-processed or post-processed by a third-party application. Examples are SOAP Handler of Apache Axis 1.3[1] and WSE filter of Microsoft [12]. Using these proprietary interceptor mechanisms might cause portability problem.

We notice that SOAP and CORBA have many architectural technologies in common. For example, the services are defined in WSDL [20] in SOAP and IDL in CORBA [13]. On the other hand, they differ in many ways. For example, SOAP 1.2 doesn't support portable interceptor mechanism that CORBA does [13]. This motivates us to propose portable interceptor on SOAP by referring to Portable Interceptor of CORBA [13]. Based on our use case analysis, presented in Section 2, for the system requirements, the specification of CORBA Portable Interceptor is not enough to fulfill the requirements of some interception applications. For example, there are no administrative functions which are needed in Continuous Audit (CA) application.

In this paper, we propose a **portable interceptor mechanism** (PIM) on SOAP to meet the interception requirements for Web Services. Our PIM includes portable interceptor management in SOAP engine and portable interceptor interface definitions.

The remainder of this paper is structured as follows. Section 2 analyzes the PIM feature by use cases. In Section 3, we discuss our PIM design in

XIII ASIA PACIFIC SOFTWARE ENGINEERING CONFERENCE (APSEC'06)
0-7695-2685-3/06 $20.00 © 2006 IEEE

detail. The prototype implementation is given in Section 4 and then followed by Evaluation in Section 5. Our conclusions are detailed in Section 6.

2. The Functional Requirements of PIM

In this section, we would like to analyze the basic functional requirements of the proposed portable interceptor (PI) in SOAP. We analyze the functional requirements by use cases. The use cases not only show the requirements of portable interceptor in SOAP but also show why the proposed interceptor is completely compliant to current SOAP standard [21].

According to Liang's work on continuous audit (CA)[9], we suggest that the PIM has to fulfill the following requirements:

R1 Automated monitoring procedures will provide most of the audit evidence necessary to opine on the subject of the tertiary monitoring at real time or closed to real time opinion real-time or near real-time monitoring

R2 The monitoring mechanism and the platform should be easy to use.

R3 The subject of the monitor entity has suitable characteristics necessary to conduct the tertiary monitoring.

R4 The concurrent technique designed under continuous monitoring can be adjusted in seconds by those tertiary monitoring, with limited technical background.

R5 When a continuous monitoring system is not functional, it could not affect the reviewee's EDP systems

Manage a Portable Interceptor

The auditor uses an admAP to manage an auditing mechanism, implemented as a portable interceptor, as shown in Figure 1. The admAP invokes a plug-in service call in the SOAP engine. The SOAP engine finds the location (or URI) of the pluggable auditing mechanism (a portable interceptor) and loads (or plug) into the SOAP engine, as shown in the step 1 of Figure 1. After plugging this portable interceptor, the auditor sets the desired initial auditing parameters in the auditing interceptor shown in Step2. Then the auditor activates the plugged portable interceptor via the same admAP, as shown in the Step 3 of Figure 1. Furthermore, the administrator can deactivate the functions of the plugged portable interceptor by invoking a deactivation function to the SOAP engine, as shown in Step 4 of Figure 1. Furthermore, the auditor may alter runtime auditing options via SOAP engine service after the portable interceptor is plugged.

Based on the above use case, we find that the **plug-in** feature fulfills the requirement R2 because it can install interceptor by simply providing the location of the monitoring module; the **set property** feature supports Requirement R3 and R4; and the **activate/deactivate** feature support Requirement R4 and R5.

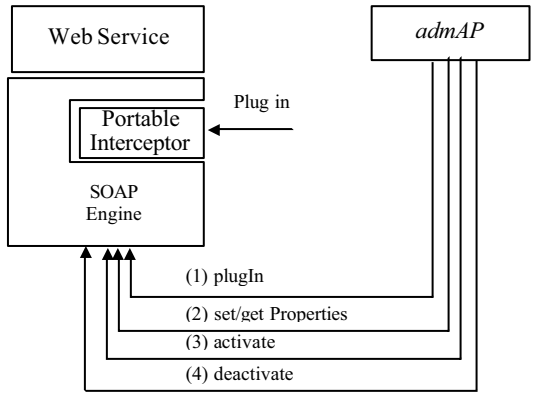

Figure 1. Interceptor management on Web Services server.

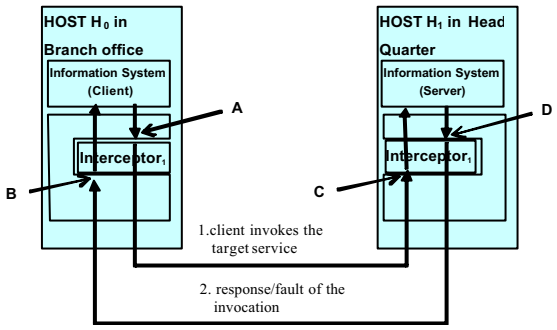

Figure 2. The four interception points of interceptors in a Web Services.

Runtime Phase of a Portable Interceptor

Suppose that a multi-national corporation use Web Services to integrate the information systems in branch office and head quarter. The information system in branch is a Web Services client and the one in head quarter is a Web Services server. The auditor may install the client side interceptor in the system located either in branch office or a server side

0-7695-2685-3/06 $20.00

interceptor in the system located in head quarter. The both client and server auditing interceptor intercepts transactions at two interception points. A Web Services client side interceptor can intercept outgoing requests (interception point A) and requested response/fault (interception point B) when the Web Services response. Furthermore, a server side interceptor can intercept arrival requests before the Web Services serves the request (interception point C) and/or intercept the response (interception point D) after the Web Services application return the response/fault to the client.

Based on the above use case, we observe that the portable interceptor is able to intercept all transactions in the system and fulfills the requirement R1. Table 1 summarizes what the requirements are fulfilled by theses features.

Feature \ Requirement	R1	R2	R3	R4	R5
Client (Server) side interceptor	X				X
Plug-in		X			
Activate/deactivate				X	X
Set/get property			X	X	

Table 1. The features of PIM fulfill the requirements.

3. The Portable Interceptor Design

Based on the use case analysis, we propose portable interceptor architecture to fulfill the requirements of portable interceptor. The requirements of portable interceptor are categorized into two folds: the portable administration features in SOAP engine; and a four-interception-point interception mechanism. We shall present the administration features design in SOAP engine first and then followed by the portable interceptor design in the following paragraphs.

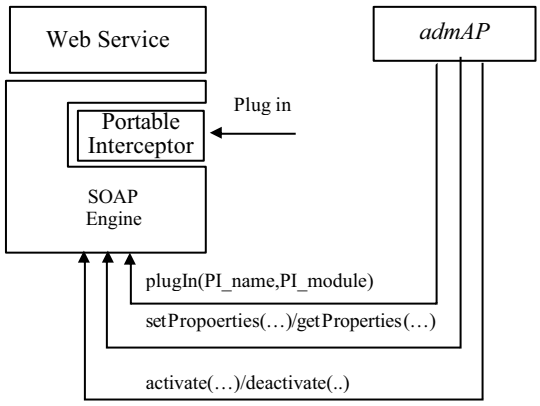

Figure 3. The portable interceptor management design: An usage example.

Portable Interceptor Administration API Design

As shown in Figure 3, the SOAP engine is implemented five portable interceptor administration functions. The Web Services system administrators can implement an administration application, admAp of Figure 3, to manage all portable interceptors by using the administration APIs. The four administration functions are named as *plugIn*(), *setProperties*(), *getProperties*(), *activate*(), and *deactivate*(). A portable interceptor is loaded(or plugged) into SOAP engine by using *plugIn*(*PI_name*, *PI_module*) function. The advantage of *plugIn*() design is no-interruption installation. The *plugIn*() provides a non-stop PI installation mechanism for a critical web service. As a result, the Web Services will not be interrupted during PI installation. The system administrator then invoke *setProperties*(*PI_name, Properties*) on the SOAP engine to setup all desired operation parameters. After setting all desired parameters for interception operation, the PI is able to be activated by calling *activate*(*PI_name*) on the SOAP engine. And vice versa, the installed PI can be deactivated by calling *deactivate*(*PI_name*) if the system administrator want to deactivated a specific PI at desired time.

Figure 4 depicts the excerpt of portable interceptor definition in WSDL format. The WSDL is not very readable for presentation purposes. Fortunately, OMG proposed a mapping specification between OMG's IDL and WSDL [14]. We convert

0-7695-2685-3/06 $20.00 © 2006 IEEE

WSDL of the PI manager into OMG IDL base on [14]. The excerpt of portable interceptor manager in IDL format is shown Figure 5. In the remainder of this paper, we present the portable interceptor interface using OMG IDL.

```
<?xml version="1.0" ?>
<definitions name="PI_Manager" …>
   <types>
     <schema …>
       <xsd:simpleType name="PI_module">…
       <xsd:simpleType name="PI_name"> …
     </schema>
  </types>
  <message name="plugIn">
    <part name="PI" type="xsd:PI_name" />
    <part name="pim" type="xsd:PI_module" />
  </message>…
  <message name="setPropertiess">
    <part name="props" type="xsd:Properties" />
  </message>…
  <message name="getProperties">
  </message>…
  <message name="activate">
    <part name="PI" type="xsd:PI_name" />
  </message>…
  <message name="deactivate">
    <part name="PI" type="xsd:PI_name" />
  </message>…
  <portType name="PI_ManagerPortType">
     <operation name="plugIn">…
     <operation name="setPropertiess">…
     <operation name="getProperties">…
     <operation name="activate">…
     <operation name="deactivate">…
  </portType>
  <binding …>
    …..
  </binding>
  <service name="PI_ManagerService">
     …
  </service>
</definitions>
```

Figure 4. The excerpt of portable interceptor manager interface definition (WSDL format)

```
typedef string PI_name;
typedef string PI_module;

interface PI_Manager {
   void plugIn(in PI_name PI, in PI_module pim);
   void setProperties(in PI_name PI, in Properties props);
   Properties getProperties(in PI_name PI);
   boolean activate(in PI_name PI);
   boolean deactivate(in PI_name PI);
}
```

Figure 5. The excerpt of interceptor manager interface definition (OMG IDL format)

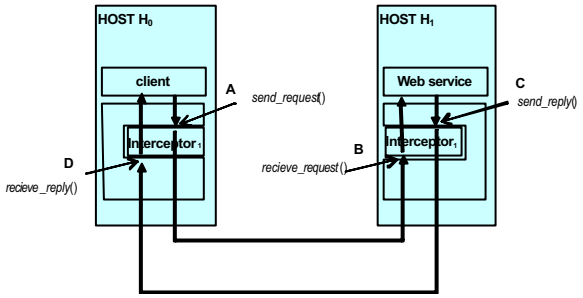

Figure 6. The API design of Portable Interceptor

Portable Interceptor API Design

The Figure 7 depicts our potable interceptor interface design. We define a basic interceptor, a client side interceptor, and a server side interceptor. The detail design of the three interceptor interfaces are presented in the following paragraph in order.

A basic interceptor interface, named **Interceptor**, is defined in Line 2 to Line 10 of Figure 7. The *Interceptor*::*initialize*(), Line 3 in Figure 7, is called by SOAP engine after loading the portable interceptor in initialization phase. The interceptor programmers may set up all necessary running environment parameters for all desired purposes. The *Interceptor*::*destroy*(), Line 4 in Figure 7, allows SOAP engine to clean up a unused portable interceptor for management purpose in terminal phase. The *Interceptor*::*setPropoerties*(*PI_name,Properties*) and *Interceptor*::*getPropoerties*(*PI_name*) allows administrator to set additional properties and query the running properties during runtime. Furthermore, the SOAP engine calls *Interceptor*::*activate*() to activate the interceptor and *Interceptor*::*deactivate*() to temporary deactivate the interceptor when the SOAP engine is invoked *PI_Manager*::*activate*() and *PI_Manager*::*activate*() by *admAP* when the interceptor administrator desires to do so. The basic interceptor is inherited by **ClientInterceptor** and **ServerInterceptor**. As a result, the **ClientInterceptor** and **ServerInterceptor** have these basic features.

```
1 : //IDL definitions of portable interceptor
2 : local interface Interceptor {
3 :     void initialize();
4 :     void destroy();
5 :     void setProperties(in PI_name PI, in Properties props);
6 :     Properties getProperties( in PI_name PI);
7 :     void activate();
8 :     void deactivate();
9 :       …
10 : };
11 : local interface RequestInfo {
12:      string target;
13:      string operation;
14 :    …
15 : };
16 : local interface ClientInterceptor : Interceptor {
17:      void sendRequest (in RequestInfo ri);
18:      void receiveReply (in RequestInfo ri);
19: };
20:
21: local interface ServerInterceptor : Interceptor {
22:      void receiveRequest  (in RequestInfo ri);
23       void sendReply (in RequestInfo ri);
24 };
```

Figure 7. The excerpt of portable interceptor interface definition (OMG IDL format).

The definition of **ClientInterceptor** is shown in Line 16-19 in Figure 7. The *ClientInterceptor*::*sendRequest*(*RequestInfo*) is invoked by SOAP engine before the client request is sent to server. The interceptor programmers can implement their desired intercepting code in this function. Similarly, the *ClientInterceptor*::*receiveReply*(*RequestInfo*) is invoked by SOAP engine after the SOAP engine receiving response from server. The interceptor has chance to inspect the response.

The proposed **ServerInterceptor** is defined in Line 21-24 of Figure 7. This interface define the two server side interception points *ServerInterceptor*::*receiveRequest*() and *ServerInterceptor*::*sendReply*(). The behavior of this interface is similar to **ClientInterceptor**.

4. The Implementation of Portable Interceptor Mechanism

As discussed in previous section, our portable interceptor mechanism in SOAP has two parts: a SOAP engine featured portable interceptor management API and portable interceptor. We adopt two open source SOAP engine, Apache Axis 1.3[1] and Codehaus XFire [3], to implement our portable interceptor mechanism design. We will first present the implementation of our portable interceptor management first and then the proposed portable interceptor Java class for Apache Axis and Codehaus XFire.

The Figure 8 depicts the excerpt of interface definition of the portable interceptor in Java. We assume our portable interceptor mechanism is part of SOAP standard. We can define our portable interceptor mechanism as Package **org.w3c.PortableInterceptor**. This interface is directly converted from the IDL definition of PI_Manager. A possible implementation of PI_Manager is shown in Figure 9. The Line 7 and 8 show how to load and manage the given portable interceptor in *PI_Manager*::*plugIn*().

```
package org.w3c.PortableInterceptor;
public interface PI_Manager extends java.rmi.Remote{
   void plugIn(PI_name PI, PI_module pim);
   void setProperties(PI_name PI, Properties props);
   Properties getProperties(PI_name PI);
  boolean activate(PI_name PI);
  boolean deactivate(PI_name PI);
}
```

Figure 8. The excerpt of PI_Manager interface definition.

```
1 : package org.apache.axis.PIM;
2 : ...
3 : public java PI_ManagerImpl implements PI_Manager {
4 :    private Hashtable plugedInterceptor = new HashTable ;
5 :    void plugIn(PI_name PI, PI_module pim) {
6 :          …
7 :          ServerInterceptor si=Class.forName(pim);
8 :          plugedInterceptor.put(PI, si);
9 :          ..
10:    };
11:   void setProperties(PI_name PI, Properties props) {
12:         …
13:    };
14:   Properties getProperties(PI_name PI) {
15:         ...
16:    };
17:  boolean activate(PI_name PI) {
18:         ...
19:    };
20:  boolean deactivate(PI_name PI) {
21:         …
22:    };
23 :}
```

Figure 9. The code excerpt of PI_Manager implementation for Apache Axis .

Figure 10, Figure 11, and Figure 12 show the excerpt of interface definitions that are directly converted from the related IDL definition in Figure 7. A portable interceptor developer should implement these interfaces with their desired interception feature than they may to load and manage their portable interceptor by using PI_Manager.

```
package org.w3c.PortableInterceptor;
/* All potable interceptors implement this interface
public interface Interceptor
{
   void initialize() ;
   void destroy() ;
    void activate() ;
   void deactivate() ;
    ...
} // interface Interceptor
```

Figure 10. The excerpt of Interceptor interface definition in Java.

```
package org.w3c.PortableInterceptor;
public interface ClientInterceptor extends Interceptor
{
   void sendRequest(...) ;
   void receiveReply(...) ;
} // interface ClientInterceptor
```

Figure 11. The excerpt of ClientInterceptor interface definition in Java.

```
package org.w3c.PortableInterceptor;
public interface ServerInterceptor extends Interceptor
{
   void receiveRequest(...) ;
   void sendReply(...) ;
} // interface ServerInterceptor
```

Figure 12. The excerpt of ServerInterceptor interface definition in Java.

5. Portability Test and Performance Evaluation

There are two parts of evaluation for this work: portability test on Apache Axis and XFire; and the performance evaluation of our PIM. We use an interceptor implementation example based on the PIM to examine whether this interceptor can be installed on both Apache Axis and XFire or not. We design a series of performance experiments to evaluate the performance of our PIM. The results show the overhead of our PIM is insignificant to the SOAP engine.

Figure 14 depicts the experiment environment for testing portability of PIM. We implemented a dummy portable interceptor based on the server side portable interceptor interface **org.w3c.PortableInterceptor.ServerInterceptor** and store in Host H_1 and Host H_2. The code excerpt of the dummy portable interceptor is shown in Figure 13. The dummy portable interceptor will dump the SOAP message (Line 6 in Figure 13) when the interception function receiveRequest() is called. The Host H_1 and H_2 are installed the modified open source SOAP engine Axis and XFire respectively, as shown in the Figure. An administration AP *admAP* and a client are installed on Host H_0. The admAP invokes the plugin() and activate() to plug in and activate our test portable interceptor **PI** (the Step 1 to 4 in the figure). The client then invokes an echoString() request to the web services on Host H_1 and H_2. The dumped SOAP messages on Axis and XFire engine console show that the portable server interceptor is functioning on both SOAP engine. That is, our portable interceptor is portable to these SOAP engine.

0-7695-2685-3/06 $20.00 © 2006 IEEE

```
1:  package tw.edu.ntou.cs.dms.PortableInterceptor;
2 :  import javax.xml.soap.SOAPMessage;
3 :  import org.w3c.PortableInterceptor.ServerInterceptor;
4 :  public class ServerPI01 implements ServerInterceptor {
5 :  public void receiveRequest(SOAPMessage reqMsg) {
6 :          System.out.println(reqMsg.toString());
7 :  }
...
     }
}
```

Figure 13. The code excerpt of the dummy service portable interceptor example.

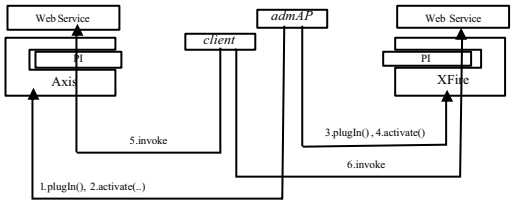

Figure 14. The environment to test the portability of PIM

Our PIM contributes runtime overhead when the PIM looks up the PI chain and call the interception functions in the interceptor. In order to examine the overheads of PIM, we use a dummy PI to examine the overhead on Axis and XFire respectively. The experiment results show the overheads on both Axis and XFire are all insignificant to the SOAP system.

The two set of experiments are conducts in the same experiment environment as shown in Figure 15. The PIM are implemented in both Apache Axis 1.3 and XFire 1.0 SOAP engine on a Pentium IV 3.2Ghz with 1GB memory PC. The host is running on MS Windows XP platform. The client is implemented in MS C# and invokes echoString() to the Web Services on the same host. The purpose of the experiment is to measure the response time delay doe the working PIM. Note that all results reported in this section have a 95% confidence interval with interval half-widths of less than 3% of average measurements.

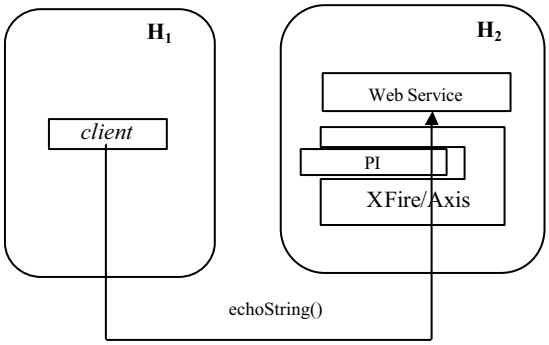

Figure 15. The environment of overhead experiment.

Based on the overhead analysis, we notice that the size of the input/output parameter in each nested invocation (or the message size) could affect the round trip time (RTT) of an invocation. Therefore, we measure the net processing time occurred in RAD for message size varying from 4 bytes to 500 KB. The experiment result, shown in Table 2, shows the client round trip time is increasing as the message size. We conclude that the overhead on Axis and XFire are all negligible.

Data Size	4 Bytes	1 KB	2 KB	5 KB	50 KB	500 KB
RTT with PIM (ms)	1.64	1.94	2.18	3.08	14.53	116.90
RTT w/o PIM (ms)	1.62	1.89	2.13	3.01	14.37	116.61
PIM overhead	0.02	0.04	0.05	0.07	0.15	0.29

(A) Axis

Data Size	4 Bytes	1 KB	2 KB	5 KB	50 KB	500 KB
RTT with PIM (ms)	1.10	1.11	1.19	1.53	6.18	98.55
RTT w/o PIM (ms)	1.09	1.11	1.19	1.52	6.18	99.01
PIM overhead	0.01	0.00	0.00	0.01	0.00	-0.46

(B) XFire

Table 2. The experiment data of client RTT delay.

6. Conclusions

In this paper, we have concluded the limitations of current SOAP standard in our previous fault tolerant Web Services research works and other works need interception mechanism as well. We take the advantages of SOAP to propose a Portable Interception Mechanism on SOAP. We also discussed the impacts on interception design due to the architectural differences in SOAP and CORBA. Our new PIM make SOAP interceptor portable like CORBA does. We believe that our experience on developing PIM can be applied to applications, such as security, etc., especially the communities familiar

0-7695-2685-3/06 $20.00

with other middleware.

References

1. Apache, Axis architecture Guide, Apache Axis 1.3 Documents, 2005.
2. Brown, K. and Kindel, C., Distributed Component Object Model Protocol – DCOM/1.0, 1996 http://www.microsoft.com/oledev/olecom/draft-brown-dcom-v1-spec-01.txt .
3. Codehaus, XFire user's guide, XFire 1.0 Document, 2006.
4. Fang, C.L., Liang, D., Chen, C., and Lin, P., A Redundant Nested Invocation Suppression Mechanism for Active Replication Fault Tolerant Web Service, in Proc. Of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE' 04), pp. 9-16, Taipei, Taiwan, March 2004.
5. Anand Krishnan, Morgan Deters, Venkita Subramonian, Interceptor. cs562, Jan. 2002.
6. Liang, D., Fang, C., Chen, C., and Lin, F., 2003, Fault tolerant web service, 10th Asia-Pacific Software Engineering Conference (APSEC'03)
7. Liang, D., Fang, C., Yuan, S., Chen, C., A fault-tolerant object service on CORBA, The Journal of System and Software, vol. 48, pp.197-211, Nov, 1999.
8. Liang, D., Fang, C., Chen, C., Lin F., Fault tolerant web service, in Proc. of IEEE APSEC, 2003, pp. 310-321, Chiang Mai, Thailand, December 2003.
9. Lin, F., Liang, D., and Wu, S., A Study on Interceptor in Supporting Continuous Monitoring, to appear in Proc. Of 2006 Annual Meeting of the American Accounting Association, August, 2006.
10. Microsoft Inc., DCOM Technical documents, http://msdn.microsoft.com/library/, 2003.
11. Microsoft, ASP.NET Web Services or .NET Remoting: How to Choose, 2002. http://msdn.microsoft.com/library/default.asp?url=/ library/en-us/dnbda/html/bdadotnetarch16.asp.
12. Microsoft, Logging Application Block Introduction, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/logging-ch01.asp, 2003.
13. Object Management Group, Common Object Request Broker Architecture: Core Specification, V 3.0.3, 2004. OMG Technical Committee Document formal/04-03- 01.
14. Object Management Group, WSDL-SOAP to CORBA Interworking, OMG Technical Committee Document mars/03-05-07, 2003.
15. Salamone, S., Secure E-Markets Emerge: Extranet VPNs Mean EBusiness, Internet Week, May 8 2000 issue, 2000.
16. Song, D., dSniff Document, dSniff 2.3, 2001. http://www.monkey.org/~dugsong/dsniff/
17. Sun Microsystems, Java RMI Specification, 2003. http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html
18. Vasarhelyi, M., and Halper, F., The continuous audit of online systems. Auditing: A Journal of Practice and Theory, Vol. 10, No. 1, pp.110-125, 1991.
19. W3C, Simple Object Access Protocol (SOAP) 1.1, 2000, http://www.w3.org/TR/SOAP/.
20. W3C, Web Services Description Language (WSDL) 1.1, 2001. http://www.w3.org/TR/wsdl
21. W3C, Simple Object Access Protocol (SOAP) 1.2 recommendation, 2003. http://www.w3.org/TR/2003/PR-soap12-part0-20030624/
22. Zwass, V., Electric Commerce: Structures and Issues, International Journal of Electric Commerce, pp. 3-23, 1996.

0-7695-2685-3/06 $20.00 © 2006 IEEE