International Journal of Computer Science and Applications, ©2009 Technomathematics Research Foundation
Vol. 6, No. 1, pp. 40 – 52

ARCHITECTURE FRAMEWORK PROPOSAL FOR DYNAMIC AND UBIQUITOUS SECURITY IN GLOBAL SOA

DEVEN SHAH

Sardar Patel Institute of Technology, Mumbai University.
Mumbai, Maharastra 400058, India
devenshahin@yahoo.com

DR. DHIREN PATEL

S.V. National Institute of Technology, Surat.
Surat, Gujrat 395007, India
dhiren29p@gmail.com

Abstract: Global Service Oriented Architecture (Global SOA) is about the entire Web being a reusable, shareable, public SOA. This work (in progress) presents a detailed analysis of the security requirements for Global SOA. The main problem in seamless ubiquitous integration of distributed network of web services into one Global Service oriented Architecture is that of security. Our strategy is to work on SOAP message interceptor (or Handler) for providing message level security in SOA. For global SOA, we are proposing architecture for ubiquitous integration of security using handlers without any pre-configuration required at service requester side. Our attempt is to introduce simple handlers supporting dynamic security features in Global SOA with interoperability and flexibility.

Keywords: Global SOA, handlers, SOA, Security, Web Services.

1. Introduction

The world today is seeing the power of community intelligence and thoughts through the medium of blogs, wiki and other online communities. Wouldn't it be great if like humans, applications could also work together intelligently to give the users what they require more efficiently and effectively? Most service-oriented architectures (SOA) are still conceptually trapped inside an organization's firewall or VPN. Global SOA envisions the Web as the global stage upon which to act out grand visions of constructing vast supply chains of data and global application-to-application communication. The security is a major barrier for migrating SOA from enterprise network to web. Various Security mechanisms for SOA require static binding between communicating web services. Our architecture proposal is based on the ubiquitous requirements of flexible security for Global SOA[Cotroneo *et.al*(2004)].

1.1. *SOA for EAI*

SOA is the exposure of software resources in the form of services, which can be accessed over a network. When SOA is used for EAI (Enterprise application integration) where diverse applications in an enterprise communicate and collaborate to achieve a business objective, binding between the web services is pre-configured and the interaction is static[Alonso *et al.* (2004)]. The UDDI used is private and is accessible to organization, its business partners only. Figure 1 shows basic SOA architecture. Figure 2 shows implementation of SOA architecture for EAI where static binding between web services is mandatory.

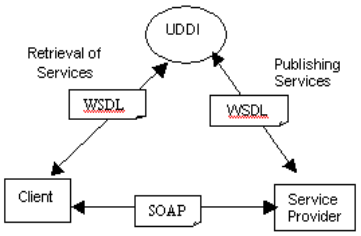


Fig.1. SOA Architecture

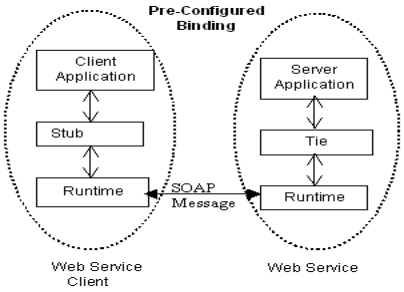


Fig. 2. Static web service interaction

1.2. *Global SOA for web*

As businesses become global there is a need for these applications to become available globally[Shah and Patel (2008)]. Thus the pre configured binding between web services becomes obsolete. Suppose an application for an online shopping chain accesses a service broker that specializes in shipping. The broker locates services from public UDDI registry that meet certain criteria such as fast delivery time and invokes them at run time. Thus the binding between broker and web services is dynamic. Figure 3 shows the dynamic binding between client and web services in Global SOA.

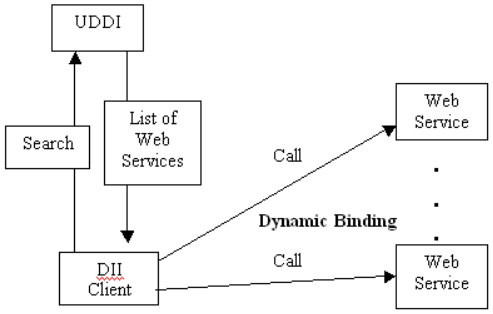


Fig. 3. Dynamic Invocation of multiple web services

1.3. *REST style based global SOA*

REST is the emerging style for web service development. This approach concentrates on simplicity as all functions are formed on the basic CRUD (create, retrieve, update, delete)[Panstasso *et.al*(2008)]. Of late, REST has won a lot of support from enterprises.

(1) Asynchronous behavior is the major strength of this approach. Since REST builds on HTTP 1.1, integration with Web 2.0 technologies is achieved easily.
(2) The major disadvantage of REST is the security aspects that are essential for ubiquitous deployment. SSL standards prevalent for standard web applications can be used, but they don't satisfy custom, dynamic security requirements.
(3) Dynamic binding has not been implemented with REST yet. This is due to the absence of any web service discovery mechanism.
(4) REST is less functional in certain areas; Reliable messaging is the best example. HTTP is not reliable protocol. There are simply no mechanisms built in that allow HTTP requests to be reliably delivered using once-and-only-once or retry-until success semantics.

Our proposed Dynamic and Security Architecture is based on SOAP based Global SOA. It can also work with REST style SOA, provided REST based web services can be describe using HTTP binding in WSDL2.0.

Rest of the paper is organized as follows: In Section 2 and 3, we discuss the security strategy of SOAP message interception using handlers for the SOA and Global SOA. In Section 4 we propose architecture for ubiquitous integration of security using handlers. In Section 5 and 6 we discuss implementation detail of propose architecture. Section 7 discusses how interoperability issues about security can be solved. In section 8, we evaluate the perormance of proposed architecture with conclusions and references at the end.

2. Security for SOA

2.1. Message-level security

Security is one of the major concerns of SOA-based implementations, especially when it spans outside the boundaries of enterprise. We identify message level security as the best approach for securing web services. Message-level security [Rahaman (2006)] includes all the benefits of SSL (Transport Level Security), but with additional flexibility and features. Message-level security is end-to-end, which means that SOAP message is secure even when the transmission involves one or more intermediaries. The SOAP Message it self is digitally signed and encrypted, rather than just the connection. And finally you can specify that only parts of the message be signed or encrypted. One strategy to implement message level security is to embed the security processing logic in the application code. But this leads to complexity in testing of both business functionality and security requirements. When the business application grows larger in scale or becomes highly distributed, the maintenance effort and support to manage a change in the security processing logic is enormous. Also as the tools for creating web services directly

from application code are available, this strategy puts additional burden on developer to study SOAP Message structure.

2.2. Handlers- SOAP message interceptors

A better strategy would be to use Message Handlers. Also known as SOAP interceptors they provide a way of modifying the SOAP Request/Response. A simple example of using handlers is to encrypt and decrypt secure data in the body of a SOAP message. A client application uses a handler to encrypt the data before it sends the SOAP message request to the Web service. The Web service receives the request and uses a handler to decrypt the data before it sends the data to the back-end component that implements the Web service. This provides the advantage of making security independent of business functionality.

2.3. Handler chain

Handlers allow you to intercept a SOAP message at various timing during a service invocation. Handler process a SOAP request message right before it goes on the network and you can process the response message before it is returned to the client. Several handlers can be combined into what is called a "handler chain".

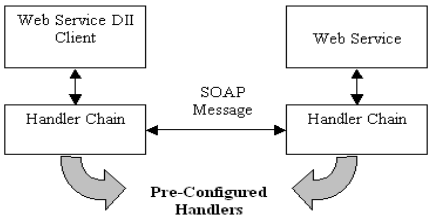


Fig. 4. Dynamic Invocation of web service with pre-configured handlers

The concept of message handler is now gaining a stand in context of web service security and is a best choice in SOA for EAI where handlers are pre-configured as shown in figure 4. But using message security handlers with global SOA raises many concerns.

3. Global SOA Security concerns with handlers

- Dynamic configuration of handlers: In Global SOA, Client dynamically invokes web services based on the result returned from the UDDI. Hence security cannot be pre-established. This necessitates dynamic configuration of security handlers. Also these handlers form a chain as per the order specific to web services. This chain should be configured at run time for each web service being called.
- Identification of handlers: Now issues remains, how client identifies handlers that are to be incorporated at its side. These include, Number of handlers, type of handlers, Sequence of handlers for creating handler chain etc.
- Security handler information exchange: As the configuration of handlers should be dynamic, handler information should be conveyed to the client in some manner. This

becomes difficult, as it requires one more level of interaction, which conflicts, with request-response model of SOAP. Currently UDDI and WSDL is the only way that service information can be conveyed to the client.

- Key Info exchange: When web service uses encryption handler or digital signature handler, then key information needs to be conveyed to the client. As handlers are configured dynamically at the client side, key information should also be conveyed dynamically.
- Different security requirements for operations in web service: It is possible that two operations in a Web Service have different level of sensitivity and hence has different security requirements. Hence, it is required to extend the concept of message handler to 'operation level'.

4. Proposed Architecture

- Dynamic discovery and invocation of web services: In global SOA, Client dynamically searches for the web services from UDDI and then sends SOAP request to that web services. Server-side Web Services has incorporated various handlers as per its security requirement (as shown in figure 5).
- Unique identification of handlers: We have proposed unique identification for handlers i.e. we have created number of Handlers based on various security requirements for e.g. handler for authentication, handler for encryption, handler for identifying content-based DoS attack etc. All handlers are part of security package and can be bundled with application server software or be part of J2EE or .NET Framework, or can be freely downloadable from Internet. Client will identify server-side handlers based on this identification, and then create handler chain from the same package to modify the SOAP message.
- Security handler information exchange at runtime: We identify UDDI and WSDL as the only means to convey the web service handler info to the client, as they are available prior to the interaction with web service. But as UDDI is a global database used for discovery of web services, inserting handler info in UDDI leads to additional burden. On the other hand, WSDL is located on the Application server where the web service is present and is associated with each web service, hence is a better place to insert handler info.
- Types of Universal Handlers: Based on above proposed architecture we have identified and designed handlers for following security concerns namely Authentication, Authorization, Confidentiality, Message integrity, Non-repudiation, Denial of Service, XML Injection, and XML Rewriting.
- Key Information Exchange: When web service configures handler it publishes key info like X509 certificate in WSDL so that when client configures handlers, these handler can extract the certificate, which includes the public key, and then encrypt the message[Abdalla (2005)]. Entire scenario can be explained as follows:

  Web service configures encryption handler and digital signature handler and publishes its info in WSDL.

Key info, here X509 certificates should also be published in WSDL. As CA signs certificate no one can modify it.

Client dynamically configures handlers on its side. Encryption handler needs recipient trusted certificate. This handler retrieves the certificate from WSDL and gives it to CA for validation. If validation is successful then client trusts service. It retrieves public key from certificate encrypt the message and sends it.

Now issue remains how service trusts client. Digital signature handler signs the message using his certificate, which is issued by CA and then sends this certificate through SOAP message header or attachment.

Server side handlers intercepts the SOAP message retrieves the certificate from the SOAP message validates it using CA. If validation is successful then it trusts service and passes the SOAP message to service.

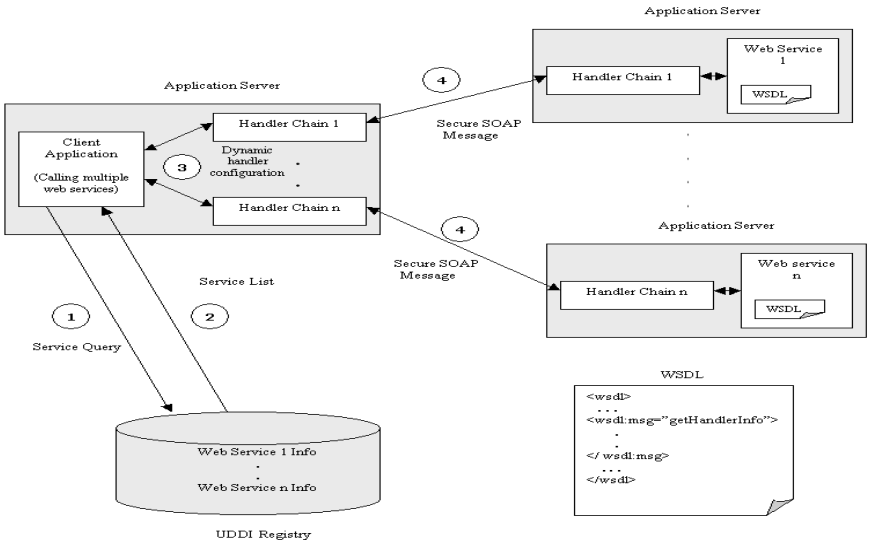


Fig. 5. Dynamic and Ubiquitous Security Architecture

- Different handler configuration for each operation within a web service: Let a Stock web service expose two methods: 'getQuote' and 'buyStock' (as shown in figure 6). Let's assume that the 'getQuote' method is meant for public use: anyone can access this method to get a stock quote price of any organization. On the other hand, the 'buyStock' method is meant to be strictly private for registered members. The concept of message handler can be extended to method level, where each method is configured with different handlers or no handlers at all as per the security requirements. Each method has a separate handler chain, and incoming SOAP message will be routed through specific handler chain based on the method. As we discussed dynamic configuration of handler chain at the client side for Global SOA, handler information published in WSDL is associated with each method. This can be

done by specifying a namespace for handler tag and adding this tag as an extensibility element. This tag is placed in the 'operation' node in WSDL. For example, authentication handler can be added for a method as follow:

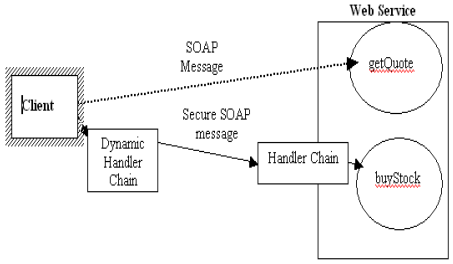


Fig. 6. Operation level security configuration

-<wsdl:operation name="buyStock">
<handler:h1>security.authentication.handler.clientAuthenticationHandler</handler:h1>
……..
<wsdl:operation>

Handler information

5. Implementation

5.1. *Test Environment*

As per our proposed architecture we have developed the following setup to implement our architecture:

- Two application servers (Application server 1 and Application server 2) having two web services (Web service 1and Web service 2) with different configuration of handlers.
- A UDDI on which the two web services have been published under the same business category.
- A DII client running on an application server, which would be dynamically configured and would call the different web services by querying the UDDI.

5.2. Activity Diagram

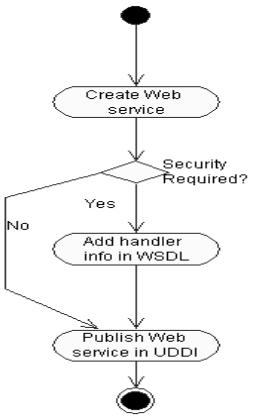


Fig. 7. Server-side

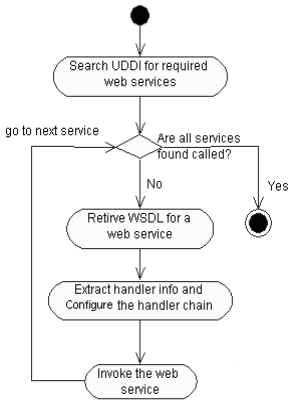


Fig. 8. Client-side

As per our proposed architecture we have implemented the following scenario:

- We have developed two web services one with handlers and another without handlers.
- As current version of WSDL does not define a standard tag for message handler chain, we have created a standard message called 'getHandlerInfo' and inserted handlers information as the 'parts' of the message in WSDL, as shown in figure 7 server side activity diagram.
  -<wsdl:message name="getHandlerInfo">
  <wsdl:part name='security.authentication.handler.
  ClientAuthenticationHandler" />
  <wsdl:part name='security.signature.handler.
  ClientSignatureHandler" />
  </wsdl:message>
- We have also configured handlers at operation level dynamically.
- We have deployed and published the web services with modified WSDL.
- We have developed a DII (Dynamic Invocation Interface) client that queries the UDDI registry and dynamically invokes the web services as per the query result.
- We have inserted a module in a DII client, which extracts handler information from WSDL and configures a handler chain at runtime, as shown in figure 8 client side activity diagram.

6. Result

We have captured SOAP envelope at different stages on the client side. As we have configured two handlers dynamically at the client side, we captured SOAP envelope at the following stages

- Between the client application and the first i.e. Authentication Handler.

- Between Authentication Handler and Digital Signature Handler
- After Digital Signature Handler execution

Each handler processes SOAP envelope modifies or adds some security information in it and forwards it to the next handler. Final SOAP envelope generated is a output of handler chain execution which is forwarded to the Web Service. It is shown in Section 6.1, 6.2 and 6.3.

6.1. SOAP envelope input to the authentication handler.

```
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
     <buyStock xmlns="">
      <String_1 xsi:type="xsd:string">IBM</String_1>
     </buyStock>
  </soapenv:Body>
</soapenv:Envelope>
```

6.2. SOAP envelope output after authentication handler execution

```
<soapenv: …….
  <soapenv:Header>
 <Authentication mlns:soapenc="http://schemas.xmlsoap.org/
 soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       <Username xmlns="">thinkers</Username>
       <Password xmlns="">1234</Password>
   </Authentication>
  </soapenv:Header>
  <soapenv:Body ……
  </soapenv:Body>
</soapenv:Envelope>
```

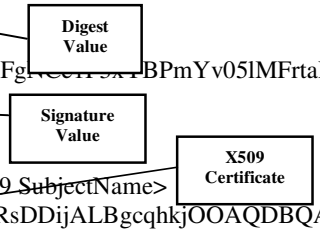
Adding Username and Password

6.3. Final SOAP envelope after digital signature handler execution

```
<soapenv:……
   <soapenv:Header>
     <Signature xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
     <CanonicalizationMethod Algorithm="……
      <SignatureMethod Algorithm="http://www.w3.org/ 2000/09/xmldsig#dsa-sha1"/>
```

Adding Signature Element

```
        <Reference URI="">
          <Transforms>
            <TransformAlgorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/ 2000/09/xmldsig#sha1"/>
<DigestValue>MlP7ZgINXxADnmZg+KGPZCtX78o=</DigestValue>
        </Reference>
       </SignedInfo>
<SignatureValue>gFQl7rmch/tc1vQtlj98t3/FsFg                BPmYv05lMFrtaKlWqwQ=
        </SignatureValue>
       <KeyInfo>
     <X509Data>
<X509SubjectName>CN=pubcertclient</X509 SubjectName>
<X509Certificate>MIICajCCAiigAwIBAgIERsDDijALBgcqhkjOOAQDBQAwGDEW
MBQGA1UEAxMNcHViY2VydGNsaWVu …….
gD/dKR8+hRd5Zrp8wCwYHKoZIzjgEAwUAAy8AMCwCFHSr9M4lJsTnuZPcRSfZ+
DJ0yv+sAhQCbrNKu2VrmlB4V7CCcNc675VvXg==
    </X509Certificate>
      </X509Data>
       </KeyInfo>
  </Signature>
    <Authentication xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="">
        <Username xmlns="">thinkers</Username>
        <Password xmlns="">1234</Password>
    </Authentication>
   </soapenv:Header>
   <soapenv: …….
   </soapenv:Body>
</soapenv:Envelope>
```



7. Interoperability

Now the concern is whether the concept of message interceptor can be implemented in other environment. Microsoft's Web Service Enhancements (WSE) pack for .NET framework presents "custom soap filters" to intercept SOAP messages, which can be used for securing the target web service. WSE architecture encapsulates two sets of filters, one for inbound messages and one for outbound messages (as shown in figure 9).

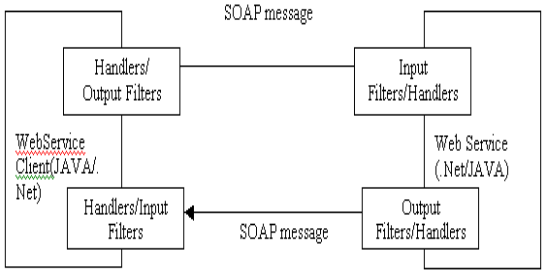


Fig. 9. Interoperatibility between filter(.net) and Handler(Java)

- Unique Identification*:* SOAP Interceptor (Filter/handler) information needs to be published in WSDL, which is used by client to configure them. We revisit the Global SOA security concerns for unique identification of handlers and propose identification based on the unique functionality so that JAVA and .NET SOAP interceptors have the same identifier if functionality is same.
- SOAP Envelope*:* As interceptors process and modify SOAP messages, SOAP envelope generated by them should be identical. Thus we give identification to interceptor, irrespective of platform, solely based on final SOAP envelope generated by interceptors.

7.1. *Test environment*

1. We have used .NET framework with WSE 2.0 support to develop web services.
2. We developed Authentication filter, which follows the same standard SOAP envelope.
3. Then we insert filter identifiers in WSDL in the same way as handler identifiers in JAVA web services as shown in following code sample.
4. We published this web service in IBM Unit Test UDDI.
   <Microsoft.web.services2>
   <diagnostics />
   <filters>
   <add type=”security.authentication.handler.serverAuthenticationHandler,
   security.authentication.handler” />
   </input>
   </filters>
   </Microsoft.web.services2>
5. JAVA client then retrieves filter identifier from WSDL, configures handlers and calls the .NET web service.

8. Performance Evaluation

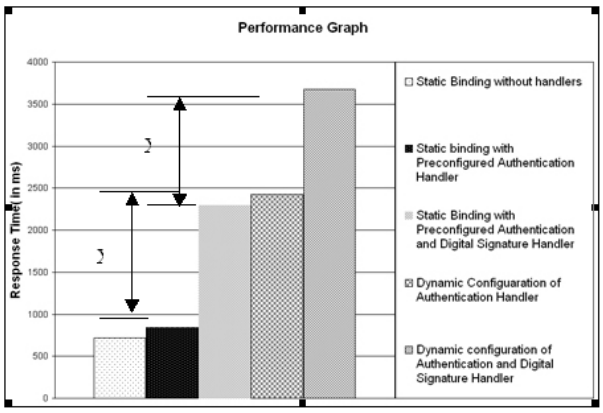


Fig 10. Performance Bar Graph

We have performed series of tests to find application response time and checked how it varies with different scenarios. Client and server were connected using cross cable to avoid any latency due to collision domain as handlers' configuration and execution time is independent of network traffic delay. Scenarios have been tested namely Static binding between web services without handlers, Static binding with one handler configured and with two handlers configured (best for EAI), dynamic invocation of web services with one handler and with two handler (For Global SOA).

For static binding without handlers the response time is minimum and it is considerably increased when we include handlers (as depicted in figure 10). It is further increased when we dynamically invoke web services with DS (Digital Signature) and Auth (Authentication) handlers. Above graph is an average graph on 20 test cases for each scenario. Our message interceptor use heavy xml processing for its security implementation for cryptographic calculations. Hence the fact remain that there is a need for performance enhancement of xml parser and light weight cryptographic primitives.

9. Conclusion

In this paper we have presented architecture proposal for security in Global SOA. We use SOAP interceptor as a mechanism to implement various security standards for SOA. When SOA migrates over web as global SOA, then it is necessary to implement security mechanism between client and server web services dynamically and ubiquitously. We have taken step-by-step approach to solve this problem. First we created different interceptors based on various SOA security requirements, and gave unique identification to them. We discussed how service requester dynamically configures interceptors at its side based on security requirement at service provider side, and implemented the solution for the same. We proposed and implemented the concept of key information publishing by service provider through WSDL. We incorporated flexibility in security requirement for a web service to security for operations within web service. Finally we worked on

interoperability issue where we implemented and tested the solution with JAVA and .NET with appropriate identification to interceptors based on standard SOAP envelope generated by them.

Overall effort is to create architecture framework for Global SOA using simple handlers that support dynamic security features with interoperability and flexibility.

References

Abdalla M.; Chevasut O.; Pointcheval D. (2005) : One –Time verifier based Encrypted key exchange, 8th International workshop on practice and theory in PKC, LNCS 3386, pp.47-64

Alonso G.; et.al.(2004): Review of Web Services, springer-verlog, ISBN: 30540-44008-9, pp. 86-87

Bertino E.; Martino L. (2006): Security in SOA and web services, IEEE SCC'06, ISBN: 0-7695-2670-5,pp.41

Cotroneo D.; Graziano A.; Russo S. (2004): Security requirements in Service Oriented architectures for ubiquitous computing, 2nd workshop on Middleware for Pervasive and ad-hoc computing, ISBN 1-588113-951-9, pp. 172-177

Panstasso C.; Zimmermann O.; Leymann F. (2008): RESTful web service vs "big" web services: making the right architectural decision; IWWWC, china, ISBN: 978-1-60558-085-2, pp. 805-814

Rahaman M.A.; Schaad A.; Rits M.(2006): Towads secure SOAP message exchange in a SOA; workshop on secure web services; ISBN 1-59593-546-0, pp. 77-84

Schroth C.; Christ O.(2007): Brave New web: Emerging Design principles and Technologies as Enablers of a Gloabal SOA, IEEE SCC'07,pp. 597-604

Shah D.; Patel D. (2008) Global SOA Architectural Style –SOA for the user, SEEC 2008, ISBN: 978-81-907337-0-0, pp. 172-175