A FRAMEWORK FOR ADAPTATION IN SECURE WEB SERVICES

Vassilakis, Costas, University of Peloponnese, Terma Karaiskaki, 22100, Tripoli, Greece, costas@uop.gr

Kareliotis, Chris, University of Athens, Panepistimiopoli, Tmima Plifororikis, 15784, Athens, Greece, ckar@di.uoa.gr

Abstract

In the context of service-oriented computing, the introduction of the Quality-of-Service (QoS) aspect leads to the need to adapt the execution of programs to the QoS requirements of the particular execution. This is typically achieved by finding alternate services that are functionally equivalent to the ones originally specified in the program and whose QoS characteristics closely match the requirements, and invoking the alternate services instead of the originally specified ones; the same approach can also be employed for tackling exceptions. The techniques proposed insofar, however, cannot be applied in a secure context, where data are encrypted and signed for the originally intended recipient. In this paper, we introduce a framework for facilitating adaptation in the context of secure SOA.

Keywords: Secure web services, Adaptive execution, Quality of Service

1. INTRODUCTION

In the context of service oriented architecture, service consumers invoke web services deployed by service providers. Choosing a particular service implementation to be invoked among a pool of functionally equivalent services is typically performed at the implementation phase of the application. However, such an *early binding* has a number of drawbacks, especially if we consider quality of service (QoS) aspects of services, and the fact that different application executions (possibly by different users) may have different requirements regarding the QoS parameters (Kareliotis, Vassilakis, Rouvas and Georgiadis, 2008). To this end, *adaptation techniques* have been proposed, according to which the choice of the actual service to be executed is deferred until the actual execution of the application, at which stage the concrete QoS requirements for the particular execution are known. Adaptation techniques typically employ a middleware layer, which undertakes the task of intercepting requests for service invocations, typically complemented with QoS specifications, and then extracting from a *service repository* those that are equivalent to the one being invoked; the information retrieved for each service from the repository includes at least the information needed to locate and invoke the service (e.g. endpoint address) *and* the QoS parameters for the service. Then, the middleware matches the QoS parameters of each service against the requirements included in the request, and the service providing the closest match is selected and invoked; the results are received by the middleware and forwarded back to the requesting client (Zeng, Lei, Jeng, Chung and Benatallah, 2005; Moser, Rosenberg and Dustdar, 2008). In some cases, the middleware may cater for bridging syntactic differences between the service originally specified in the invocation and the service actually invoked, by applying transformations to both the request (in order to align the received payload to the one expected by the selected service) and the reply (to align the reply of the selected service to the one expected by the client) (Moser, Rosenberg and Dustdar, 2008; Kareliotis, Vassilakis, Rouvas and Georgiadis, 2009).

This approach, however, cannot be used in the context of invocations placed in a *secure context* for the following reasons:

1. if SSL/TLS (Network Working Group, 2008) is used as a security provider (i.e. the communication between the client and the service provider is realized using secure socket layer connections), then the middleware cannot use the intercepted message, since this will have been

encrypted by the client and will be decryptable only by the service provider (or vice versa, for the reply).

2. if WS-Security (Oasis, 2006) is used as a security provider, the unencrypted portions of the message will be usable by the middleware, however (a) if a service different than the originally specified is chosen to be invoked, the *encrypted* portions of the message will be unusable by the chosen service, since they will be encrypted using the originally specified service's public key and (b) modifications to the message payload (such as changing the namespace to match that of the selected service and alterations to the message structure to bridge syntactic differences) will break the message integrity assertion mechanisms within the message [e.g. XML signatures (Oasis, 2006)].
3. the option of using unprotected messages in the *client-to-middleware* communication path and protecting only the *middleware-to-service* portion is unacceptable in the general case, since this enables the middleware to have access to sensitive data (e.g. credit card numbers, client health records etc). Additionally, in many cases the service provider will require message to be encrypted or signed using the client's private key, not the middleware's. This approach could be used in the case that the middleware is trusted *for all clients* and all clients place their requests from a specific address, as is the case of web services being invoked from within database stored procedures (Kurtvm, 2008).

In this paper, we present a framework for providing adaptation for invocations placed in a secure context. This framework again uses a middleware layer which undertakes the task of (a) determining which is the service that best matches the QoS criteria specified by the client in the context of the request and (b) compiling a list of transformation rules that need to be applied to the original payload so as to align it to the requirements of the chosen service (and similarly for the reply). However, contrary to the approaches dealing with adaptation in an unprotected context, the middleware does not either apply the transformations or place the invocation to the chosen service: instead, it returns this information to the client, delegating to it the tasks of payload preparation and invocation placement. In this manner, the client is able to encrypt relevant message portions and produce signatures using the appropriate keys for each particular action. The framework has been designed so as to not require any alteration to programming techniques (*at most* programmers need to declare certain objects as instances of framework-provided classes), allowing programmers to maintain their development environments and techniques; additionally, if certain system libraries are replaced by framework-provided libraries, the change in variable declarations above is *not* required, while the enhanced functionality is readily incorporated even to already developed programs.

The rest of this paper is organized as follows: section 2 surveys related work in the areas of adaptation techniques and secure service invocations. Section 3 overviews QoS parameters that are considered in the context of service adaptation and elaborates on security aspects. Section 4 presents the proposed framework and its operation, while section 5 concludes and outlines future work.

2. RELATED WORK

2.1 Service adaptation

AgFlow (Zeng, 2003; Zeng, Benatallah, Ngu, Dumas, Kalagnanam and Chang 2004) is an approach to adaptation according to which the execution plan is revised to ensure that it adheres to the QoS constraints specified by the user. AgFlow may consider either local optimization, where each web service invocation is considered individually, or global planning, in which case the whole execution plan is reviewed and modified. VieDAME (Moser, Rosenberg and Dustdar, 2008) adapts BPEL (Business Process Execution Language) scenarios taking into account QoS parameters; adaptation in VieDAME is implemented using extensions available only in the ActiveBPEL engine (Active Endpoints, 2007), being thus platform-dependent. In the work presented by Baligand, Rivierre and Ledoux (2007), end-user specified policies are introduced through QoSL4BP, while BPEL transformers arrange for incorporating policy interceptors and QoS monitors in the BPEL scenarios. Cao, Jin, Wu and Qi (2006) present a BPEL extension and a correspondingly extended BPEL engine, to deliver QoS-based adaptation.

An indispensible requirement for providing fully automated adaptation, is the ability to identify services which are functionally equivalent to a given service and retrieve the QoS parameters of each service. The METEOR-S project (Kochut, 1999; Verma, Sivashanmugam, Sheth, Patil, Oundhakar, and Miller, 2005), combined with WSMX (Web Services Execution Environment) (Cimpianet, Moran, Oren, Vitvar and Zaremba, 2005) is the most widely adopted underpinning for this functionality. The selection among functionally equivalent services can be performed in various ways, ranging from simple "always the first" to multi-criteria selection of variants (Feier, Roman, Polleres, Domingue, Stollberg and Fensel, 2005). Non-functional properties of services (QoS parameters) are usually represented through DAML+OIL or OWL-S (The OWL Services Coalition, 2003) shared ontologies.

2.2 Secure service invocation

Initially, the only method for performing securing service invocations was to employ a secure communication channel for all data exchanges between the client and the service provider; typically, communication in this channel is encrypted following the SSL/TLS standards (Network Working Group, 2008). This technique however has been identified to present performance problems, as it requires to encrypt the whole bulk of the transferred data, regardless of the secrecy/authenticity requirements of each portion of the message (e.g. in a forum registration service, it suffices to encrypt the user's password whereas the user's avatar image can be transmitted without encryption), while we must also take into account that in the web services environment peer connections are often short-lived, and under this scheme the cost of establishing and breaking SSL connections can be too high (Progress Actional, 2008). A more important consideration however is that SSL does not fit well in workflow environments, since it only protects the transport between two computers or devices; therefore, if in a workflow environment any participating node is compromised, the whole data is exposed (Kumar, 2004; Progress Actional, 2008).

To tackle these issues, the web service community has worked towards standards that will be oriented to *message securing* (as opposed to *communication channel securing*) and to allow for finer granularity in the application of securing mechanisms. The predominant approach is nowadays the WS-Security standard (Oasis, 2006), which includes provisions for ensuring message integrity and confidentiality through inserting and verifying *security claims* in the SOAP (Simple Object Access Protocol) message. The mechanisms provided by the WS-Security standard can deliver *end-to-end message level security*: this means that when a message traverses multiple applications (one or more SOAP intermediaries) within and between business entities (e.g. companies, divisions and business units) is secure over its full route through and between those business entities (as opposed to SSL, where the message is secured between communication peers only).

When applying provisions from the WS-Security standard, it is the responsibility of the application programmer to ensure that the appropriate techniques are suitably used; since this is an additional burden for the programmer, the software industry has produced appropriate software libraries that allow the programmer to easily insert and verify security claims in a SOAP message; the Metro Web Services (Sun Microsystems, 2008) in the Java world (a successor to Web Services Development Pack) and Windows Communication Foundation-WCF (Microsoft Corporation, 2008) in the .Net domain are the most widespread software libraries for developing secure web services. The programming paradigm for securing a web service invocation is similar in both environments, and follows the pattern illustrated in Figure 1 (the Java syntax is adopted in this figure).

As can be seen in the code, after building the SOAP message, the client requests that the message is secured by invoking the *secureOutboundMessage* method. This method reads a configuration file (associated with the *cprocessor* variable) which indicates which parts of the body need to be encrypted and/or signed. A sample of such a file is illustrated in Figure 2. Afterwards, the call is placed and the received reply is subsequently verified through the invocation of *verifyInboundMessage*. Analogous configuration files are used at the server side.

```
SOAPMessage  msg = createPayloadForService(); // create the SOAP message
// Message securing operations are performed by a Security Processor within a Processing context
ProcessingContext context = new ProcessingContext();
XWSSProcessorFactory factory = XWSSProcessorFactory.newInstance();
XWSSProcessor cprocessor = factory.createForSecurityConfiguration(clientConfig,
    new SecurityEnvironmentHandler("client"));
context.setSOAPMessage(msg); // contextualize the message
SOAPMessage secureMsg =  cprocessor.secureOutboundMessage(context); // Secure the message.
// establish the connection, place the call and receive the response
SOAPConnection connection = SOAPConnectionFactory.newInstance().createConnection();
SOAPMessage reply = connection.call(secureMsg, "http://app.server.com/service_server");
// verify the secured message
XWSSProcessor sprocessor = factory.createForSecurityConfiguration(serverConfig,
    new SecurityEnvironmentHandler("server"));
context = new ProcessingContext();
context.setSOAPMessage(reply);
SOAPMessage verifiedMsg= null;
try{
    verifiedMsg= sprocessor.verifyInboundMessage(context);
} catch (Exception e) { /* problem in verifying the response message security claims */ }
```

Figure 1. Performing a secured service invocation using Java Metro Web Services

```
<xwss:SecurityConfiguration xmlns:xwss="http://java.sun.com/xml/ns/xwss/config" dumpMessages="true">
    <xwss:Timestamp/>
    <xwss:RequireEncryption>
        <xwss:Target type="qname">SOAP-BODY</xwss:Target>
    </xwss:RequireEncryption>
    <xwss:Encrypt>
        <xwss:X509Token certificateAlias="s1as"/>
        <xwss:KeyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <xwss:Target type="qname">SOAP-BODY</xwss:Target>
    </xwss:Encrypt>
</xwss:SecurityConfiguration>
```

Figure 2. A client configuration file, specifying that the whole SOAP body should be encrypted

3. QUALITY OF SERVICE ASPECTS IN SERVICE ADAPTATION

Besides their functionality, services are characterized by qualitative attributes that describe various non-functional, yet important, aspects of their observed behaviour. These qualitative attributes are typically referred to as *QoS attributes* or *QoS characteristics*, and the most usually considered QoS attributes include response time, security, cost, availability etc. In the context of adaptation, it is essential to consider the services' QoS attributes, since (a) user policy is typically specified by means of QoS attribute values and (b) using a replacement service with very different QoS characteristics than the originally specified one may have undesirable side-effects, e.g. using a bank transaction service with low security may lead to the leaking of credit card numbers, or using a service with much higher cost will lead to excessive charging of the service consumer. The QoS aspects considered in this work are briefly presented in the following paragraphs.

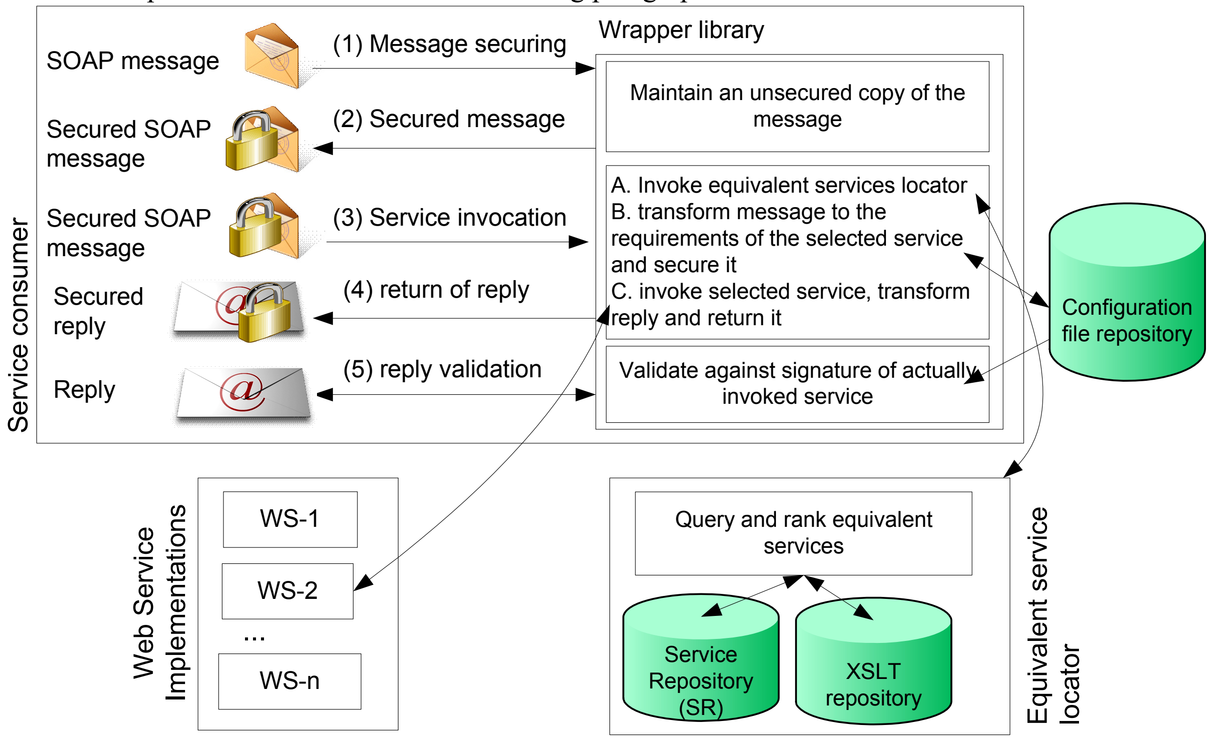
- **Cost,** q_c, which reflects the cost-related and charging-related aspects of a service (O'Sullivan, Edmond and Ter Hofstede, 2002).
- **Performance,** q_{perf}, expressing the speed at which the service executes. Speed may be expressed in terms of *response time*, *throughput* etc, while in some works performance is considered a composite attribute including multiple of these performance aspects.
- **Security**, q_{sec}. This is a composite attribute, expressing the service's ability to implement identification, authentication of messages, secrecy, non-repudiation, resistance against certain types of cryptanalytic attacks and other aspects pertaining to security. Further elaboration on the security aspects is beyond the scope of this paper.
- **Reputation,** q_{re}, reflecting a measure of its trustworthiness, mainly dependent on end users' experiences of using the service. This is an indication on how credible the service's advertised QoS parameters are.

- **Successful Execution Rate,** q_{suc}, depicting the probability that a request to the particular service is successfully concluded within the maximum expected time frame indicated in the Web service description.
- **Availability**, q_{av}**,** of a service s is the probability that the service is accessible to be invoked.

In the proposed framework, we consider that each QoS attribute has a value between 0 (minimum value) and 10 (maximum value), thus each service S_i is associated with a sextuple $QoS(S_i) = (q_c^{Si}, q_{perf}^{Si}, q_{sec}^{Si}, q_{re}^{Si}, q_{suc}^{Si}, q_{av}^{Si})$. Clients designate their QoS requirements by specifying minimum and maximum values for the different QoS attributes, while clients may additionally specify a *weight* for each attribute, indicating the attributes' perceived importance. Extension of the framework to include more QoS attributes is straightforward.

4. FRAMEWORK ARCHITECTURE AND OPERATION

The proposed framework introduces two additional components in the secure service delivery and invocation environment. The first component is the *equivalent service locator*, which is deployed and maintained independently of the services and the clients. The second component is a *wrapper library*, which is installed at client-side, and arranges for liaising with the equivalent service locator component to identify services that can be used as a replacement to the originally specified one, and placing invocations to the service considered more appropriate. The wrapper library can replace the respective libraries provided by the development environment; in cases that this is not possible (or desirable), programs needing to incorporate adaptation should be modified so as to use specific classes of the framework instead of the classes provided by the development environment. The framework classes are subclasses of the respective development environment classes, so substitution can be performed by simple textual replacement in the source files and recompilation, without any other modification. The framework operation is described in the following paragraphs.



Service consumer
SOAP message
(1) Message securing
Wrapper library
Maintain an unsecured copy of the message
Secured SOAP message
(2) Secured message
Secured SOAP message
(3) Service invocation
A. Invoke equivalent services locator
B. transform message to the requirements of the selected service and secure it
C. invoke selected service, transform reply and return it
Configuration file repository
Secured reply
(4) return of reply
Reply
(5) reply validation
Validate against signature of actually invoked service
Web Service Implementations
WS-1
WS-2
...
WS-n
Query and rank equivalent services
Service Repository (SR)
XSLT repository
Equivalent service locator

Figure 3. Overall architecture of the secure services adaptation framework

Recall from section 2.2 that when a secure invocation needs to be placed, the client initially creates the message payload (as it would in a non-secure environment) and subsequently *secures* the message, by invoking a suitable method in the software library; this method applies the appropriate techniques (encryption and/or signing) to portions of the message, as designated by a configuration file. Finally, the message can be sent to its destination, a reply is received and the security claims therein are verified, before the reply content is used by the client. In the proposed framework, the client follows exactly the same steps, but requests for *message securing*, *invocation placement* and *reply verification* are intercepted by a wrapper library, which arranges for:

1. determining which of the functionally equivalent to the originally specified service is the most suitable to be invoked.
2. transforming the message payload to the needs of the selected service.
3. securing the message payload as appropriate for the selected service.
4. verifying the reply and transforming the response to the needs of the client.

These steps are described in the following paragraphs.

4.1 Choosing the most suitable service

The choice of the most suitable service to be invoked, takes place when the client actually places the invocation. At this point, the wrapper library transmits to the *equivalent services locator* a request containing (a) the service that the client has asked to be invoked [in WSDL terms, the service *endpoint* (Newcomer and Lomow, 2005)] (b) the QoS constraints and selection policy and (c) the configuration file, specifying which security claims (encryption and/or signing) should be inserted in the SOAP message (c.f. Figure 2).

If the client program has been developed in a QoS-aware fashion (i.e. the programmer is knowledgeable about the QoS parameters, constraints and selection policy, and employs programming constructs to set them explicitly), then these parameters are used; if on the other hand, the client application has been developed in a QoS-unaware fashion, then constraints and selection policy can be set at a *program execution* granularity level (i.e. the same constraints and selection policy will apply to *all* service invocations placed within a particular execution of the program), using mechanisms such as setting environment variables or system properties. For instance, the command

```
java -DQoS_constraints=resp_time_max:20ms,enc_key_size_min:256
          -DQoS_weight=resp_time:-1,reputation:2 Application.class
```

specifies that within the scope of executing `Application.class`, all replacement services should have a response time of 20 ms (or less) and employ encryption using a key of length 256 bits (or more). For the services that have qualified, the *equivalent services locator* then computes a *suitability score* considering their values for the *response time* and *reputation* QoS attributes, as specified in the `QoS_weight` system property. At the particular computation, the *reputation* is considered twice as important as the response time, while the negative number in `resp_time` indicates that lower response times are preferred over higher ones (Kareliotis, Vassilakis, Rouvas and Georgiadis, 2009). The service with the highest score is chosen as the *most suitable replacement* for the particular invocation.

Note that the service chosen in this step may be *syntactically different* than the service originally specified (Moser, Rosenberg, and Dustdar, 2008; Kareliotis, Vassilakis, Rouvas and Georgiadis, 2009). Syntactic difference means practically that the same payload information can be wrapped using different XML tags or different tag nestings, as shown in the example in Figure 4. To tackle these cases, the *equivalent services locator* module maintains a repository of XSLT transformations, which cater for converting between different forms of payloads. The XSLT template that rearranges a payload crafted for the originally specified service to a payload suitable for the selected service is extracted from the repository and attached to the reply that will be returned by the equivalent services locator module; the XSLT file arranging for aligning the reply of the selected service to the one expected by the client is also extracted and attached to the reply.

```
<CreditCardCharge>
    <CreditCardNumber>12345678</CreditCardNumber>
    <PIN>1234</PIN>
    <Amount>100</Amount >
    <Currency>Euro</Currency>
</CreditCardCharge>
```

(a)

```
<CreditCardDebit>
    <CreditCardNo>12345678</CreditCardNo>
    <PIN>1234</PIN>
    <DebitAmount>
        <Currency>Euro</Currency>
        <Units>100</Units>
    </DebitAmount>
</CreditCardDebit >
```

(b)

Figure 4. Syntactically different payloads for a credit card charging service

Finally, the equivalent services locator module parses the configuration file contained in the request, to identify the element(s) of the original payload that need to be encrypted or signed. Since the elements may be different in the final payload (the one that will be sent to the selected service), the equivalent services locator module computes a new configuration file that matches the schema of the final payload; the new configuration file is attached to the reply, and the reply is returned to the client.

4.2 Transforming and securing the message payload

Having received the reply from the equivalent services locator module, the client application transforms the original payload to the one required by the selected service, by applying the XSLT transformation within the reply; after the XSLT transformation is applied, the resulting message is secured according to the configuration file that was attached to the reply, and at this stage the selected service is invoked and the reply is received.

One issue that is worth noting here is that in the secured service invocation programming pattern (Figure 1) message securing (execution of the `secureOutboundMessage` method) precedes the invocation of the service (`connection.call` method execution), whereas in the proposed framework, the SOAP message must remain in plaintext format until the service to be invoked has been determined, since (a) encryption should be performed using the key of the service to be invoked and not the key of the originally specified service and (b) if the payload were encrypted, certain XSLT transformations could fail as some tags needed for the transformations might be included in a subtree that was encrypted as a whole). In order not to disrupt the normal programming practice, the wrapper library addresses this issue as follows:

1. when the `secureOutboundMessage` method is invoked, this invocation is intercepted by the wrapper library. The wrapper library keeps a plaintext copy of the message to be secured. Programmatically, this is achieved by having the wrapper implementation of the `secureOutboundMessage` method to returned an instance of WrappedSOAPMessage (a subclass of SOAPMessage), which –besides the secured message– contains the plaintext copy of the message.
2. the `connection.call` method as implemented by the library, extracts from its first parameter (the instance of the WrappedSOAPMessage class) the plaintext version of the SOAP message; subsequently, it uses the XSLT template and the configuration file contained in the *equivalent service locator* module's reply to transform the plaintext message to the format expected by the selected service and subsequently secure it. Note that the final destination of the message (i.e. the service that will be invoked) is now known, so the message can be encrypted with the proper key.

4.3 Reply handling

Once the invoked service returns the reply, the wrapper library immediately proceeds to the verification of the security claims contained in the reply, i.e. the decryption of the encrypted portions and the checking of the digital signatures. In both decryption and digital signature verification, the public key of the service that has actually been invoked is used. The plaintext version of the reply and the result of the verification are stored in the reply returned as a result of the `connection.call` method invocation. Note that if the verification has failed, no exception is raised at this point; the exception is caught and stored, to be re-thrown when the client invokes the `verifyInboundMessage` method for the reply: this arrangement ensures that exceptions are raised when the client expects them according to the secure service invocation programming paradigm.

After the reply has been decrypted and verified, the XSLT transformation for aligning the reply of the selected service to the one expected by the client is applied; the result of this transformation is returned to the client. Finally, when the client invokes the `verifyInboundMessage`, the wrapper library checks the result of the verification performed upon the result received from the invoked service. If the verification had failed, the exception originally thrown at that point is retrieved and re-thrown; if the verification had succeeded, the plaintext version of the reply (as transformed using the reply alignment XSLT) is returned to the client.

5. CONCLUSIONS

In this paper we have presented a framework for introducing adaptation to web service invocations that are placed in a secure context. The framework includes two modules, one of them being an independent software entity that undertakes the tasks of locating the “most appropriate” service and providing the required elements for handling syntactic differences between the originally invoked service and the selected one. The proposed framework maintains the programming paradigm followed in the secure web service invocation context, and can be introduced to existing programs either seamlessly (if system libraries can be replaced) or with simple textual transformations (if such a replacement is not feasible).

Future work will focus on completing the implementation of the wrapper library, and evaluating the performance of the adaptation scheme. The incorporation of such techniques in BPEL processes (as opposed to *individual service invocations*) will also be examined.

References

Active Endpoints (2007). *ActiveBPEL Engine*. Retrieved April 22, 2009, from http://www.active-endpoints.com/.

Baligand, F., Rivierre, N. and Ledoux, T. (2007). *A Declarative Approach for QoS-Aware Web Service Compositions*. Proceedings of ICSOC 2007 (LNCS 4749), pp. 422–428.

Cao, H., Jin, H., Wu, S. and Qi, L. (2006) *ServiceFlow: QoS Based Service Composition in CGSP*. Proceedings of IEEE EDOC'06.

Cimpian, E., Moran, M., Oren, E., Vitvar, T. and Zaremba, M. (2005). *Overview and Scope of WSMX*. Technical report, WSMX Working Draft. Retrieved April 22, 2009, from http://www.wsmo.org/TR/d13/d13.0/v0.2/.

Feier, C., Roman, D., Polleres, A. Domingue, J., Stollberg, M. and Fensel, D. (2005). *Towards Intelligent Web Services: Web Service Modeling Ontology*. Proceedings of the International Conference on Intelligent Computing 2005.

Kareliotis, Ch., Vassilakis, C., Rouvas, E. and Georgiadis, P. (2008). Exception Resolution for BPEL Processes: a Middleware-based Framework and Performance Evaluation. *Proceedings of the tenth International Conference on Information Integration and Web-based Applications & Services* (iiWAS2008), pp. 248-256.

Kareliotis, Ch., Vassilakis, C., Rouvas, E. and Georgiadis, P. (2009). QoS-Driven Adaptation of BPEL Scenario Execution. *Proceedings of the IEEE International Conference on Web Services 2009.*

Kochut, K. J. (1999). *METEOR Model version 3.* Athens, GA, Large Scale Distributed Information Systems Lab, Department of Computer Science, University of Georgia.

Kumar, P. (2004). *The pros and cons of securing Web services with SSL*. Retrieved April 22, 2009, from http://searchsoa.techtarget.com/news/interview/0,289202,sid26_gci995388,00.html.

Kurtvm (2008). *Consuming WS-Security enabled web services in PL/SQL*. Retrieved April 22, 2009, from http://www.ora600.be/consuming-ws-security-webservices-in-plsql.

Microsoft Corporation (2008). *Getting Started with Windows Communication Foundation (WCF)*. Retrieved April 22, 2009, from http://msdn.microsoft.com/en-us/netframework/aa663324.aspx.

Moser, O., Rosenberg, F. and Dustdar, S. (2008). Non-Intrusive Monitoring and Service Adaptation for WS-BPEL, *Proceedings of the WWW 2008 Conference*, Beijing, China, pp. 815-824.

Network Working Group (2008). *The Transport Layer Security (TLS) Protocol Version 1.2 (IETF RFC 5246)*. Retrieved April 22, 2009, from http://tools.ietf.org/html/rfc5246.

Newcomer, E. and Lomow, G..(2005). *Understanding SOA with Web Services*. Pearson Education, Inc., Upper Saddle River, NJ, USA.

O'Sullivan, J., Edmond, D. and Ter Hofstede, A. (2002) What is a Service?: Towards Accurate Description of Non-Functional Properties. *Distributed and Parallel Databases*, 12, pp. 117-133.

Oasis (2006). *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*. Retrieved April 22, 2009, from http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

Progress Actional (2008). Web services and SSL. In *SOA Governance: Where the Rubber Meets the Runtime*, Retrieved April 22, 2009, from http://www.actional.com/resources/whitepapers/Web-Service-Risks/Web-Services-SSL.html.

Sun Microsystems (2008). *Metro Web Services*. Retrieved April 22, 2009, from http://java.sun.com/webservices/downloads/index.jsp.

The OWL Services Coalition (2003) *OWL-S: Semantic Markup for Web Services*, Retrieved April 22, 2009, from http://www.daml.org/services/owl-s/1.0/owl-s.html.

Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S. and Miller, J. (2005). METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web services. *Journal of Information Technology and Management*, Special Issue on Universal Global Integration, 6, 1, pp. 17-39.

Zeng, L. (2003). *Dynamic Web Services Composition*, PhD thesis, Univ. of New South Wales, 2003.

Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam J. and Chang H. (2004). QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5), 2004.

Zeng L., Hui L., Jeng, J.J., Chun, J.Y.and Benatallah, B. (2005). Policy-driven exception-management for composite Web services, E-Commerce Technology, Proceedings of CEC 05, 19-22 July 2005, pp. 355 – 363.