A Solution to Change Security Policies on-the-fly

Ricardo Ferraz Tomaz[1], M. Mehdi Ben Hmida[2], and Valérie Monfort[1,2]

[1] Université Paris 1 - Panthéon - Sorbonne
Centre de Recherche en Informatique , 90 rue de Tolbiac 75634 Paris cedex 13
{Ricardo.Ferraz-Tomaz,valerie.Monfort}@malix.univ-paris1.fr

[2] Université Paris IX Dauphine LAMSADE
Place du Maréchal de Lattre Tassigny, Paris Cedex 16
Benhmida.mehdi@gmail.com

ABSTRACT. *Although Web Services are not the only way to design the Service paradigm, they are likely to be one of the major technologies used to achieve both the interoperability and loose coupling required for Service Oriented Architecture (SOA). However, there is still much to be done in order to get a genuinely flawless Web Service. Our research work consists on finding new service based solutions to increase flexibility and adaptability to SOA-based applications. In this paper we introduce an engine named Aspect Service Weaver (ASW) that enables to weave and un-weave new behavior (such as security) to a SOA-based application at runtime. We explain our approach through a concrete scenario where we replace an existing Kerberos policy for a Digital Certificate policy.*

1. Introduction

Service-Oriented Architecture (SOA) is an architectural style whose goal is to achieve loose coupling among interacting software entities in order to facilitate the integration of software components into distributed applications.

SOA-based applications are usually composed of simple Web Services that are offered by different providers. This scenario gets worse because services are, usually, shared by multiple applications. Each application has its own requirements that can demand specialization of a service behavior. If the specialization is performed by modifying the service, then the change may conflict with the requirements of other application.

So, the services providers have a serious problem to change their Web Services behaviors, because a simple changing can affect many applications.

To solve this problem the system administrators must be able to selectively apply different specializations to the same service depending on the caller or other environmental condition.

Usually, security policies vary depending on the nature of the environment - internal or external network - and the nature of the application – e.g. financial applications require high security level.

Enterprise security policies can be changed depending on the types of data and applications that need to be accessed by various parties. Policy must be set for internal departments and users, extranet partners, remote-access users and customers.

In addition, new security policies are proposed overtime and the systems administrators may, simply, want to change their security policies to be up-to-date with technology innovations. Tearing down a service for reconfiguration is often undesirable.

In order to solve this problem, we propose an engine named Aspect Service Weaver (ASW). The ASW enables to weave and un-weave new behaviors to a service at runtime without affecting the other applications that share the service.

Our approach is based on Aspect Oriented Programming (AOP) [1,2]. The ASW interposes itself between the client caller and the service callee and uses the AOP weaving time (before or after the service invocation) to insert new behaviors to the original web service.

In this paper we present a concrete scenario where our approach can be used to change the security policy of a service-based system at runtime without side effect for the other applications that share the service.

In the next section, we present the basic ideas behind AOP. The section 3 presents our approach and its applicability to a concrete case study. The section 4 draws some conclusions.

2. AOP Foundations

AOP is about structure and problem decomposition. More specifically, AOP is about separation of concerns. Aspect Oriented Software Development (AOSD) advocates the separation of concerns and their implementation in separated modules or components that are, posteriorly, woven to compose an application.

According to AOSD, the problem should be decomposed into modules such that each module has one concern. However, some concerns cannot be easily separated, and we are forced to map such concerns over many modules. This is called *crosscutting***.** A *crosscutting* concern is a serious problem since it is harder to understand, reuse, extend, adapt and maintain the concern because it is spread over several places.

AOP allows encapsulating the crosscutting code into separate module - known as *advices* - and then applying the code where it is needed. We achieve application of the crosscutting code by defining specific places - known as *join point* - in our object model. The Crosscutting code is injected at the specified *join point* by a tool named *weaver*. Another important concept in AOP is the *pointcut*. A *pointcut* is a set of *join points* that enables the *weaver* to inject an *advice* in several places. The weaver inserts the *advices* "before", "after" or "around" the *join points*.

3. Aspect Service Weaver (ASW)

3.1Overview

The ASW intercepts the SOAP (Simple Object Access Protocol) [3] messages between the Web Services that compose a Service based application, and then redirects these messages for other Web Services (Aspect Services) that performs the new requirements.

We use the AOP weaving time to intercept and redirect the messages. The ASW intercepts a message "before" or "after" an Original Web Service invocation and then redirects the SOAP message for the appropriated Aspect Service (in charge of implementing the new behavior).

Our platform infrastructure is based on an XML language to register and deploy (XML file descriptor) both Original Web Services and Aspects Services. The engine invokes the Aspects Services at runtime to change the original Web Service behavior.

The ASW architecture is composed by two engines: XQuery Engine [5] and XQuery Module Generator (XMG).

The XQuery Engine can be anyone ready-to-use tool equipped with a transport layer to generate http/SOAP messages such as XQuery Stylus Studio [6], TigerLogic [7] and Liquid Data [8]. So, we can use any off-the-shelf software component.

The XMG receives as input the XML file descriptor and the WSDL file (to obtain the Original Method and Service Advice signatures) and then translates them to an XQuery script. Finally, the XQuery script is submitted to the XQuery engine that generates the http/SOAP messages to the appropriated Services Advices.

Since the file descriptor and the WSDL file are both XML documents, XPath [4] is the natural choice as the pointcut language. In an Aspect described in the XML file descriptor, the pointcut element is an XPath Expression selecting all Original Web Service methods where the execution of new crosscutting behavior will be introduced.

Let us now explain the ASW through a more realistic scenario taken from one of our industrial projects.

3.2 Concrete Scenario

A company aims to develop automatons to analyze blood plasma, which means patient data information has to be highly reliable and correct. In order to support consequent evolution and successive reuse of the machines, the company decided to define and to promote a flexible and adaptable architecture according to the new emerging requirements. We decided to use Web Services technology to implement this architecture.

The Application must display specific Human Machine Interface (HMI) according to profile and maturity level of the user. Access is allowed or denied according to user profile and protected from unauthenticated usage.

Consider the hypothetical scenario where an original Web Service has an authentication policy like Kerberos token and, that after a while, the company resolves to replace this authentication policy for a digital certificate security policy.

Using a classical approach, considering that a change is required in the authentication method signature, the programmer must identify all authenticated methods and rewrite the authentication invocation.

Using the ASW engine, the programmer can, for instance, develop a Service Advice called *digital_certificate* and specify, on the XML file descriptor, that before the invocation of the authenticated methods in the Original Web Service the engine must invoke the Service advice *digital_certificate*.

This way, on the next Original Web Service invocation the engine will be aware of the rules specified on the XML file descriptor and will generate an http/SOAP invocation to the *digital_certificate* Advice instead of to the Kerberos method in order to authenticate the user based on the new policy. The authentication policy is a Service Advice woven at run-time.

The XML file descriptor that contains the Aspects rules interactions with the Original Web Services for the above described example would be like this:

```
<Aspect name=" analyze_blood_ plasma ">
    <pointcut name="analyze" pattern=
     "//operation[starts-with(@name," SendResult")]"/>
     <Advice name="security" type="before">
        <pointcutName name=" analyze" />
        <content urn="urn_wsdl_service1"
        invoke="digital_certificate" status="true"/>
     </Advice>
</Aspect>
```

Figure1. XML file descriptor

This XML file descriptor indicates to the engine that always the methods that match with the XPath expression "//operation[starts-with(@name,"SendResult")]"(equivalent to the methods whose names match the regular expression "SendResult*") were invoked the engine must invoke before the Service *Advice* digital_certificate. The Original method invocation is conditioned to the answer got from *digital_certificate* Service Advice. Only if the authentication is succeeded (*status* is true) the *SendResult* method must be invoked. Based on this XML file descriptor the engine must generate the following XQuery script:

```
1. import service namespace Service1 = "urn:
urn_wsdl_service1" name "digital_certificate";
2. import service namespace Service2 = "urn:
urn_wsdl_service2 " name “SendResult";
3 let $i Service1: digital_certificate(UserId, Passwd);
4. if $i/status=true then
5  let $j:=  Service2:SendResult(PatientID);
```

Figure 2. Generated XQuery Module output

Depending on a well defined set of rules the engine can generate complex interactions among Original Web Services methods and Services Advices. But, the mechanism to accomplish the interactions is relatively simple

When a client invokes an original method, the engine intercepts the http/SOAP request and captures the original method signature, and then the engine tries to match the method signature with an entry in the XML file descriptor. After this step, if there is one or more matching, the engine performs the interactions schemas according to the kind of advices found in the XML file descriptor.

For example, whenever a ”before” advice is detected the engine extracts the Service Advice signature and redirects the request to the proper Service Advice before the request reaches the original method. Then, it captures the Service Advice answer and, if the Service Advice answer is positive (the user was successfully authenticated), it sends the answer to the original method. The original method performs its task and sends back the answer to the engine. Finally, the engine dispatches the final answer to the client caller.

The interactions performed when an ”after” or a ”replace” advice is detected fit to similar logic. Only the invocation sequence is changed.

4. Conclusion

This paper presents how to apply AOP paradigm to improve Web Services flexibility. There are many ongoing works in this direction, but we highlight the AO4BPEL [1] due to its similarity with our approach.

AO4BPEL is an aspect-oriented extension to BPEL4WS (Business Process Execution Language for Web Services) that allows the registration and deployment of both processes and aspects. Aspects can be deployed dynamically at runtime whilst BPEL processes are running.

The AO4BPEL authors suggest extending BPEL4WS standard. Conversely, no changes on the already existent Web Services standards are required to use our engine.

5. References

[1]Anis Charfi and Mira Mezini. Aspect-oriented web service composition with AO4BPELl. In ECOWS, volume 3250 of LNCS, pages 168,182. Springer, 2004.

[2] Tomaz, Ricardo Ferraz; Hamida, Mehdi Ben; Monfort, Valerie. Concrete Solutions for WEB Services Adaptability Using Policies and Aspects. IEEE Journal Of Digital Information Management, 2006.

[3] SOAP Version 1.2, W3C specification, Web site http ://www.w3.org/TR/ soap/.

[4] XML Path Language (XPath), W3C specification, www.w3.org/TR/xpath.

[5] Onose Nicola and Simeone Jerome. XQuery at Your Web Service. In *Proceedings International WWW Conference*, New York, USA, 2004.

[6] Data Direct Technology, Web site available at http ://www.stylusstudio.com.

[7] RainingData, Web site available at http ://www.rainingdata.com.

[8] BEA, Web site available at http ://www.bea.com.