SOA-aware Authorization Control

Christian Emig, Heiko Schandua, Sebastian Abeck
Cooperation & Management, Universität Karlsruhe (TH)
{emig | schandua | abeck}@cm-tm.uka.de

Abstract

The question how to handle authorization of digital identities in a service-oriented architecture (SOA) remains an open issue. In this paper we present a design pattern for the integration of legacy systems with SOA using out-of-the-box (unmodified) application servers and discuss how the architecture has to be extended by an Identity Management (IdM) infrastructure. We claim that the IdM infrastructure itself must be designed in a service-oriented way to fit into the overall SOA approach. We introduce a possibility how to decouple the policy enforcement point from the application server and propose an architectural design pattern to seamlessly integrate the SOA's business-related functionality and the IdM infrastructure. An implementation case study illustrates how to apply the invocation pattern for secured web services.

1. Introduction

Currently most companies use web service technologies as their first step towards SOA. However, SOA can not be built from scratch but rather the functionality of the existing legacy systems and their components are being wrapped to web service interfaces. This not only eases their integration but also enables business analysts to perform so-called Programming-in-the-Large, the orchestration of business-related services along business processes [1, 2].

The mass and complexity of the existing and upcoming specifications in the web service security area like WS-Security, SAML, WS-Trust, WS-Policy or the Liberty Alliance's stack proposal make software developers often neglect the web service security part at first. This is why, as yet, the existing web services in most cases have little or no identity management features. Hence web service invocations are simply "trusted" by default. Complication increases when composing several web services which provide functionality from different underlying applications. To benefit from the advantages of SOA, the composition of two or more web services is implemented using the Business Process Execution Language (BPEL, [3]). This is where missing identity management becomes an obstruction and that is the reason why web services should be secured by a sophisticated identity management solution [4, 5].

The question how to integrate the existing applications, especially their internal identity management, remains. The migration of existing applications to SOA is an important step for the protection of the companies' investments. These legacy systems have to be hooked into SOA. To achieve this goal, their (mainly) proprietary interfaces considering their communication protocols and interface description have to be wrapped to WSDL interfaces [6] and SOAP communication [7], being the lowest common denominator in SOA. Application servers are used for this adaption. In the context of SOA the application servers are just a means to an end so their internal parts should be rather transparent for both the software developer involved in the application integration and the upper part of the SOA itself. For the sake of flexibility, the application servers should be interchangeable, at least if staying in the same of the two hemispheres: the Java/J2EE world or the Microsoft .NET part. Ensuring flexibility and transparency at the level of the employed application servers regarding the identity management aspects is the key driver for the work in this paper.

In this paper we focus on the core web service layer [8] where the application servers reside and where the context change from SOA to application specific identity management has to take place. A central question is where to put the necessary IdM infrastructure elements and why. We suggest applying design patterns at the core web service layer to solve this challenge. In this paper we set up on the design patterns secure service proxy (SSP) and intercepting web agent (IWA) [9] that will be discussed in chapter 2 and enhance them for a better fit to SOA.

The succeeding chapters are organized as follows: Chapter 2 treats the related work in the combination of SOA with the authorization part of IdM. The two software design patterns intercepting web agent (IWA) and secure service proxy (SSP) are introduced and discussed. Chapter 3 builds up on these patterns and introduces our approach how to add security to web services that are implemented using existing (legacy) systems with out-of-the-box application servers. In chapter 4 the policy enforcement point is focused upon. In chapter 5 the application of our approach is illustrated using a concrete case study. In chapter 6 an evaluation comparing our solution to the existing approaches is given. A conclusion and an outlook on future work in this area close the body of the paper.

2. Related Work

In [9] security patterns and best practices for identity management in the context of web services are addressed. Among these there are two patterns introduced that have a strong relationship with the security layer in SOA: the secure service proxy (SSP) and the intercepting web agent (IWA).

The SSP pattern is suggested to be applied in scenarios when wrapping legacy systems as a whole for integration purposes and is derived from the generic proxy design pattern which is enriched for security by forming a policy enforcement point (PEP). There are two aspects that have to be thought about: First of all, the signatures of the methods in the existing system should not change as there is a very tight coupling between the proxy and the addressed functionality in the system. Whenever there is a change in signatures, the proxy itself has to be changed as well. Additionally the SSP has to be enhanced (or a new SSP has to be created) in case of new methods in the backend systems. But the great advantage of an SSP is the ability to transform protocols, for example a transformation from a HTTP/SOAP to a proprietary protocol. This is achieved by the complete decoupling of the requesting subject from the requested object. So what the SSP does is to cut off all the communication from the client, to enforce the corresponding security policies by calling its associated policy decision point (PDP), optionally to transform the protocol and format and then to call the destination method of the legacy system. Finally the way back goes as well through the SSP which can perform protocol transformation if necessary. An important point is that the service requestor (subject) does not get in touch with the destination service (object) itself at all. If a security context for the destination service is needed, this job is done by the SSP and masked for the service requestor.

The strong relationship between the SSP and the system to proxy leads to much work in software development as the SSP has always to be recoded when changes occur in the backend. This does not only comprise identity management related functionality, but the business-related one as well. That is what led to the design pattern intercepting web agent (IWA). The idea of the IWA is to act like a “door steward” that has never to be replaced when changes at the protected services occur. The IWA acts as a PEP which is technically speaking realized as a component that is hooked into a container like a web server or application server as an “entrance” module. There it is added to the communication queue and listens to the incoming requests. For every request it verifies the authorization by asking the corresponding PDP and either lets the request pass through or rejects it without letting the requestor get in touch with the desired resource. Typically this is implemented as a module – like *mod_access* in the Apache web server, or a handler within the Apache Axis SOAP engine. Though this approach is highly flexible to changes with the protected backend services as it does not rely on method signatures, the problem is that the IWA is strongly bound to a specific application server. For each combination of application server and PDP (represented quite often in an IdM-Suite), an individual IWA has to be implemented considering the architecture of the application server which often differs strongly.

In [10], a typical authorization architecture with the following logical actors is discussed: The subject (e.g. a user) wants to access an object (e.g. an service). The authorization architecture (which is part of the identity management architecture) involves a policy enforcement point (PEP) that takes care of requesting authorization decisions and enforcing them. It has to intercept the requests of the users and ask the policy decision point (PDP) if the user is authorized. The PDP evaluates the applicable policies and builds the authorization decision (deny / permit) upon that. The paper focuses on the realization of the combination of PDP and PEP and applies the IWA design pattern. There a JAX-RPC message handler for the Apache Axis SOAP engine is used, which results in a very tight coupling. The major weakness of this approach is that for each application server (like JBoss, BEA, …) further handlers have to be developed. A strong binding to the technology of the application server is introduced.

3. The Secure Service Agent (SSA)

As discussed in the previous chapters, an appropriate solution for identity management is a prerequisite to establishing service-oriented architectures. The central question is where to place the necessary IdM infrastructure elements as PEP and PDP and why. [9] introduces the SSP design pattern which is hard to apply in SOA as it is necessary to generate an individual proxy for each method (operation) in the existing systems. This is both laborious and unnecessary. The IWA design pattern on the other hand solves this problem but introduces a very tight coupling to the application server where the service is deployed. This sets up unnecessary constraints and reduces flexibility at the service provider, which is fixed to application servers that are supported by the developer of the IWA.

For that reason we decided to combine and enhance the existing approaches and to create a design pattern that we call "Secure Service Agent" (SSA). It is based on the SSP and IWA pattern but the PEP is moved out of the internal part of the web or application server and both the proprietary and the close binding of the PEP to the server itself is reduced. So a central feature of the SSA is to achieve the same flexibility as with the web service deployment: web services should be deployable without change apart from the specific deployment descriptor at any application server of the same kind (Java/J2EE vs. MS .NET).

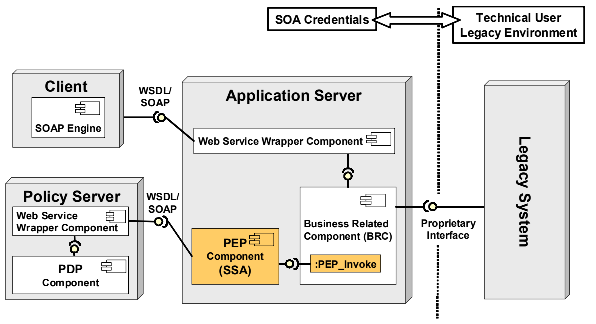

Figure 1: Secure Service Agent – Component Diagram

Figure 1 describes the relevant elements of this approach:

1. On the right hand side is the legacy system, whose functionality is to be exposed via web services for easiness of integration and reuse in SOA.

2. In the middle there is the application server that catches the proprietary interface. This is handled by a business-related component (BRC) which is deployed at the application server and wrapped to a web service using the application server's web service wrapper component.

3. The PEP is developed as a stand-alone component that is deployed at the application server. As a major difference to the IWA design pattern, the PEP is not hooked into the web service wrapper component which leads to much work considering the adaption to exactly suit for the concrete web service wrapper – with hard coded support for the combination of security protocols that have to be applied. To give an example what problems might arise by doing so: Even if staying with the same web service wrapper, like Apache Axis for instance, there is the need for adoption if the next version is released. Not even to think of changing to JBoss or BEA etc. As application servers are just a means to an end this is the reason why we decided to put the PEP both out of the web service wrapper as well as out of the application server internal part. This enables flexible deployment at application servers within the same basis framework (Java/J2EE vs. MS .NET).

4. The PDP is implemented at a policy server which is usually external to the application server and should be equipped with a SOA compatible WSDL / SOAP interface. Loose coupling and static interfaces enable the service-oriented usage with different application servers.

5. The client uses its SOAP engine to communicate with the web service.

We focus on the second and third point, the definition of the PEP component. We suggest splitting up the PEP into two elements. First, a single PEP component, that is not fixed to one specific application server – this can be achieved by using standard design like a stateless session bean (EJB) in the Java/J2EE context. Secondly a simple PEP_Invoke, that is implemented in each component deployed at the application server. As this invocation has a standardized and fixed signature, the extension of the business-related code to the IdM-related part can be done automatically using toolsets in software developer's integrated development environment (IDE), to ensure that he is not to be asked for as regards content. Putting the IdM relevant parameters to the SOAP body offers the chance to bypass the proprietary internals of the application server and do the IdM handling in the applications server's "user space". This is a promising achievement as most security standards require that all servers in integration scenarios have PEP modules which are capable of handling the defined combination of security standards. We call this design pattern with the split-up PEP with small PEP_Invoke and a flexibly deployable PEP component “Secure Service Agent” (SSA) as an enhancement of SSP and IWA to web services.

The following chapter focuses on the combination of the PEP component (SSA) and the corresponding PEP_Invoke that has to be inserted to the BRCs.

4. The SSA-compliant PEP Component

As discussed in chapter 3, we suggest keeping the signature of the PEP_Invoke as generic as possible to avoid going back for adapting the security part if changes occur and to ensure not to burden the software developer but to handle this automatically using tools.

Considering the BRC component, there is no further security shield maintained by the application server. If the BRC wants to use the IdM infrastructure, instantiated in the PEP and PDP components, it is invited to simply add the generic PEP_Invoke into its program code.

The signature of the PEP's service interface offered towards the BRCs has to comprise the following items:

1. The SOA security token of the subject (usually the user). This is the (temporary) session token issued to the user at the SOA's portal after successful authentication which is completed in a separate process. The token could be a XML-style document like a SAML token.

2. The identification of the object (e.g. the service) that is to be invoked. In the web service context this has to be at a granularity of operations. The identification of the object is needed so that the PDP can match it against the policies, if authorization exists. To ensure the uniqueness of this identifier, it is reasonable to build upon the identifier that is used in the service registry (e.g. UDDI). This enhances the useful coupling between the registry and the policy decision point.

3. The set of parameters that has been attached to the service invocation. This might be necessary for the PDP as it is possible that a subject can perform action on an object dependant of the applied parameters. This enables the evaluation of fine-grained security policies.

There is no need that the business-related component itself is capable of identifying, authenticating or authorizing the requesting subject. It simply has to transparently pass through the given information to the PEP component. This ensures that the security-related functionality is minimal in the business-related component. Furthermore it is important to notice that at the BRC the realm of the SOA credential ends. If the PDP confirms authorization, the BRC is meant to connect to the backend system using a technical user's credentials (might be equipped with a higher access level). So the authorization part of the backend system is externalized to the PDP.

The following shows an OMG IDL [11] description of the interface of the PEP towards the BRCs:

```
module SSA
{
interface PEP2BRC
{
  void PEP_Invoke
    (
    in char[] subject_soa_security_token,
    in string object_id,
    in string[] parameters,
    out boolean authorization_decision
    )
};
```

Figure 2 depicts the dynamic aspects of the SSA pattern using a sequence diagram:

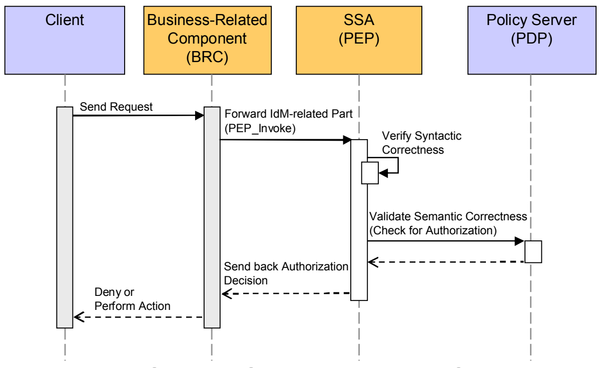
Client
Business-Related Component (BRC)
SSA (PEP)
Policy Server (PDP)
Send Request
Forward IdM-related Part (PEP_Invoke)
Verify Syntactic Correctness
Validate Semantic Correctness (Check for Authorization)
Send back Authorization Decision
Deny or Perform Action

Figure 2: Secure Service Agent – Sequence Diagram

The client sends a request to the BRC. Both the business-related parameters, as well as the subject's SOA security token (e.g. temporary session ticket) are transmitted from the client to the BRC which forwards them, as described above, to the PEP (PEP_Invoke) together with its object identification. The SSA then verifies the syntactic correctness of the token, first of its bare existence then if it has the appropriate structure, for instance a SAML assertion or simply an array of char / byte. After confirming the syntactic correctness, the SSA starts communication with the external PDP. The complexity of this web service communication was one of the reasons for splitting of the PEP into two elements.

The SSA forwards the subject's session token, the objects identification and the subject's parameters within an XACML statement to the PDP. There the actual policy decision takes place, resulting in an

XACML answer to the SSA. This answer is then transformed by the SSA to a yes/no reply and sent back to the BRC. Upon the Boolean answer it either denies the request or performs the desired action.

From the view of the BRC, the PEP_Invoke is quite similar to a remote procedure call, with the following pseudo code:

```
if ( PEP_Invoke(
subject_soa_security_token,
object_id, parameter[]) == FALSE )
{
  end all operations and exit;
}
// business-related functionality
follows here
```

As both the BRC and the PEP component reside on the same application server, there is no need for complex and expensive web service style invocations, but internal access structures like RMI can be used, if it is a Java-based application server like JBoss.

5. Implementation Case Study

In this chapter the application of the SSA pattern within an integration project being pursued at our university is presented. We focus on two different applications needed in the administration process for generating certificates after students have passed an exam. One of these two systems is SAP Campus Management [12], the higher education component for the SAP R/3 enterprise resource planning system. SAP CM is internally built up with SAP's ABAP code stack and is not shipped with built-in web services interfaces. To get SAP R/3 web service compatible, there is the Java Connector (JCo) which is an adapter for Java to call SAP's proprietary interfaces. With SAP only deploying JCo, there is the intrinsic limitation to the Java framework. This is why the application server in our scenario is favored to be Java-based. Because established decentralized IT structures are in use at the university, the centralized usage of a fixed application server for the complete university is not feasible. We decided to use JBoss 4.0.3 SP1, a J2EE-conforming, open-source application server for deploying the business-related component, providing the service for the *getExamResult()* method of SAP CM that was deployed as an Enterprise Java Bean (EJB). Combined with JBoss comes an adapted version of Apache Axis (WS4EE), a web service wrapper component. The second application which had to be integrated is a HR System of HIS GmbH, which is quite popular at German universities but will not be discussed here further. It is used to obtain the relevant personal core data of the students (*getPersonalCoreData*).

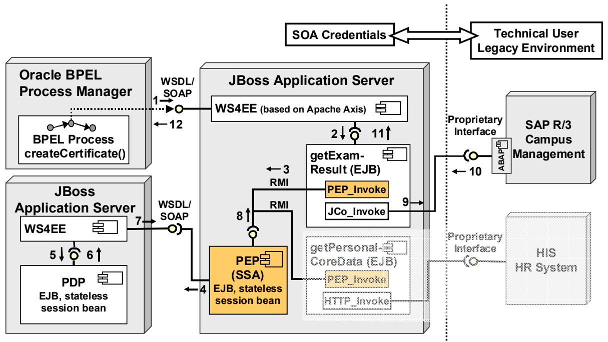

Figure 3: Case Study – Implementation Architecture

Figure 3 shows the implementation architecture. The IT-support for the business process of certificate creation starts with the Oracle BPEL Process Manager, where the BPEL Process provides the *createCertificate()* composed web service. The *getExamResult* web service is called using SOAP communication (1). The parameters of the web service include the matriculation number of the student whose certificate is to be created as well as the SOA security token of the invoking subject, which is further passed through all SOA layers. These parameters are caught by the WS4EE component in JBoss. To keep the integration process free of unnecessary bindings to the application server, both JBoss and WS4EE are used in the out-of-the-box version. The business-related code for the *getExamResult* component was developed in advance and finally the PEP_Invoke was added before deploying the EJB. Though not being complex for the software developer, the last step could be simplified using automated deployment tools. When registering the *getExamResult* web service to the UDDI, a UUID was assigned that we use as the object_id in the PEP_Invoke. After deserialization by the WS4EE component, the *getExamResult* EJB is locally called using RMI/JNDI passing through all parameters (2). The first step inside the *getExamResult* EJB is to do the PEP_Invoke to the SSA. This call is done using RMI as well and passes by the subject's parameters as well as the service's object_id (3). After a syntactical check, it is SSA's job to marshal this data and to handle the web service communication with the external PDP (4). In a secondary JBoss, a centralized PDP was deployed as an EJB, which is not in the focus of this publication (5). The Boolean return value is propagated back via the SSA to the business-related EJB *getExamResult* (6) (7) (8). In case of failure, the EJB simply stops and sends back an error message, in case of success the functionality of the backend SAP system is called.

It is important to notice that here the change of the user contexts takes place. The SAP system is called using a technical user with (almost) full rights to the SAP system. This has three major advantages:

1. The policy decision is evaluated at only one place: centrally in the SOA.

2. If backend systems have a pay-per-user concept, it would be necessary to pay for a license for every single user that might access the system – in the university context this can easily be more than 10,000 people.

3. There is no need for password synchronization to ensure that the BRC has the correct credentials to call the backend system.

Finally the XML-based data is sent back to the calling BPEL process (10) (11) (12) that carries on with further calls until the final certificate is created.

6. Comparison and Evaluation

With the transparency of the IdM-related security information that is simply passed through like business-related parameters, there is a high flexibility if changes in both application servers as well as applied security standards occur. This is why the integration project can start without the predefined decision which application server is to be used for integration and which security standards to apply.

When comparing the three design patterns SSA, IWA and SSP, the first aspect is the flexibility and the dependencies of the patterns. Focusing on the application server, the IWA hooks into its internal processing handlers which leads to a tight coupling and in IT environments with different application servers, there has to be an individual IWA for each different server. The SSA solves this problem by moving out of the internal part to the application server's containers. This enables the developing of an SSA component that can be deployed in different servers of the same basis framework (Java/J2EE vs. MS .NET). The SSP is not meant to touch the application server's internals and should be therefore transparent in this respect.

Taking a look at the applied security standards, it is the case that specific IWAs have to be developed for the combination of specific application servers with specific security standards. Having the need for the knowledge of the interns of the application server, it is not easy to develop a matching IWA on one's own compared to the SSA where it is sufficient to know how to apply the security standards. In this respect the SSP is comparable with the SSA.

The security part of this evaluation is examined by looking at the shielding of the business-related component. Whilst in the IWA scenario the 'door steward' is positioned directly at the application servers internal queues preventing communication to the BRC, in both the SSA and the SSP environment communication is meant to pass-through from the calling party to the BRC which itself takes care of the policy enforcement. It is important to recognize that the SSA and the SSP are only concerned with policy enforcement - and attacks like denial of service can still be detected and rejected by general filters at the application server or even before at systems in the communication path.

Having a look at reusability the SSA has great advantages over the IWA as it avoids the tight coupling to the application server. When staying in the same framework like Java/J2EE, it is only needed to change the application-specific deployment descriptor to deploy the SSA component on a different application server. Considering the complexity for the software developer of the business-related part, the SSP highly burdens as there is a tight coupling between the BRC and the legacy application. Each operation is meant to be published individually. Both the SSA and the IWA do not focus on this.

When considering performance aspects it has to be said that the most expensive operations are not inside the three patterns but in the communication to the PDP.

7. Conclusion and Further Work

In this paper an approach how to enforce access control in integration scenarios especially when migrating existing systems into SOA has been presented. Based on the Secure Service Proxy (SSP) and the Intercepting Web Agent (IWA) the design pattern "Secure Service Agent" (SSA) that is more focused to SOA has been introduced and its practicability in a case study has been demonstrated. The SSA was very useful in a scenario where different organizations with heterogeneous IT systems tried to integrate their applications using an SOA approach. In a divide-and-conquer like process, each organization unit tried to make their existing applications web service compatible which is a highly individual approach. The flexibility in respect to the application server on the one hand and the security protocols on the other hand allow a great decoupling of the business-related aspects and the security ones. Software developers of the business-related part think about the change of their software artifacts to serviceorientation without the fear of blocking points at security. With the standardized way of interaction with the PEP, services can be combined with IdM at a later stage.

The next steps necessary when considering an identity management infrastructure for service-oriented architectures are to think about the upper layers of SOA. On the integration layer and on the process layer technologies like BPEL are applied. The question is how to secure them or if it is even necessary to secure them if the underlying web services are protected? Which security specifications fit best at which layers? Another area to be investigated is the architecture of an SOA conforming PDP – how should it be constructed and how can the policies of the underlying systems be integrated with the overall policies of the SOA?

8. References

[1] Frank Leymann: Web Services - Distributed Applications without Limits, Business, Technology and Web, Leipzig, 2003.

[2] Christian Emig, Jochen Weisser, Sebastian Abeck: Development of SOA-Based Software Systems – an Evolutionary Programming Approach, International Conference on Internet and Web Applications and Services ICIW'06, Guadeloupe / French Caribbean, ISBN 0-7695-2522-9, February 2006.

[3] Organization for the Advancement of Structured Information Standards (OASIS): Web Services Business Process Execution Language (WS-BPEL), Version 1.1.
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[4] Jason Bloomberg: Enterprise IdM: Essential SOA Prerequisite, Zapthink Zapflash, September 2003.

[5] Ulrike Ostler: Safety first in Service-Orientierted Architectures, silicon.de Portal, September 2005.
http://www.silicon.de/enid/druckfunktion/15607,4

[6] W3C: Web Services Description Language (WSDL), version 1.1, March 2001.
http://www.w3.org/TR/wsdl

[7] W3C: Simple Object Access Protocol (SOAP) 1.1, May 2000.
http://www.w3.org/TR/soap/

[8] Ali Arsanjani: Service-Oriented Modeling and Architecture, IBM developer works, 2004.

[9] Christopher Steel, Ramesh Naggapan, Ray Lai: Core Security Patterns, Prentice Hall, ISBN 0-131-46307-1, Oktober 2005.

[10] Eric Yuan, Jin Tong: Attribute Based Access Control (ABAC) for Web Services, IEEE International Conference on Web Services (ICWS 2005), Orlando Florida, July 2005.

[11] Object Management Group (OMG): Interface Definition Language (IDL).
http://www.omg.org/gettingstarted/omg_idl.htm

[12] SAP for Higher Education & Research, Brochures & White Papers: SAP Campus Management.
http://www.sap.com/industries/highered/pdf/BWP_Campus_Mgt.pdf