A Web Service Architecture for Enforcing Access Control Policies

Claudio Agostino Ardagna [1], Ernesto Damiani [2],
Sabrina De Capitani di Vimercati [3], Pierangela Samarati [4]

Dipartimento di Tecnologie dell'Informazione
Università degli Studi di Milano
26013 Crema, Italy

Abstract

Web services represent a challenge and an opportunity for organizations wishing to expose product and services offerings through the Internet. The Web service technology provides an environment in which service providers and consumers can discover each other and conduct business transactions through the exchange of XML-based documents. However, any organization using XML and Web Services must ensure that only the right users, sending the appropriate XML content, can access their Web Services. Access control policy specification for controlling access to Web services is then becoming an emergent research area due to the rapid development of Web services in modern economy.

This paper is an effort to understand the basic concepts for securing Web services and the requirements for implementing secure Web services. We describe the design and implementation of a Web service architecture for enforcing access control policies, the overall rationale and some specific choices of our design are discussed.

Key words: Web Services, Security, Interoperability, Distributed System, XML.

1 Introduction

Accessing information on the global Internet has become an essential requirement of the modern economy. Web services, which are developed for this purpose, represent a new technology that permits the exchange of information through the network, using standard protocols and allowing communica-

[1] Email: ardagna@dti.unimi.it
[2] Email: damiani@dti.unimi.it
[3] Email: decapita@dti.unimi.it
[4] Email: samarati@dti.unimi.it

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

tion between heterogeneous architectures. Today Web services are essentially based on four standards [8]: the eXtensible Markup Language (XML) [14], the Simple Object Access Protocol (SOAP) [17], the Web Services Description Language [18] (WSDL), and the Universal Discovery, Description and Integration (UDDI) [19]. While XML Web services are an increasingly successful paradigm for the development of complex Web-based applications, the original specifications of their underlying technologies did not even mention security. It is therefore easy to understand why security is currently one of the biggest concerns about future development of XML Web services. Specifically, two main issues need to be addressed:

- restricting access to an XML Web service to authorized users;
- protecting the integrity and confidentiality of XML messages exchanged in a Web service environment.

At first sight, it may seem that both these issues can be addressed straightforwardly by relying on the security techniques already used for Web sites. For instance, HTTPS (i.e., HTTP over the Secure Sockets Layer protocol) is typically used as a tool for encrypting private information. It can also provide authentication, but it is only used to authenticate the identity of the Web server to the client. It is capable of authenticating the client identity to the server, but most Web servers are not set up for that. However, HTTPS cannot provide, for example, authorization, that is, it cannot regulate what a user is attempting to do and selectively allowing/disallowing the operation. The result is that HTTPS is a good solution for providing strong encryption and server identity authentication to the client and vice versa. Also, HTTPS is a "point-to-point" security, which does not allow intermediaries to act on the data, and requires trust between the HTTPS end-point and the location of the application being secured.

Recently, the technology industry has been working on various XML-based security languages to provide comprehensive and unified security solutions for Web services. These languages include the Security Assertion Markup Language (SAML) [28] and the Web Services Security specification (WS-Security) [20]. SAML is an XML-based framework for exchanging security information developed by the OASIS organization. The SAML specification defines how to represent security credentials (*assertions* in SAML) using XML. SAML is designed to enable secure single-sign-on to applications within organizations and across companies and supports many different authentication mechanisms such as the combination of username and password, SSL client-side certificate, X.509 certificate, and so on. After the subject authentication, the server SAML returns a particular security token to the client that makes the initial request. WS-Security specification has been developed by IBM, Microsoft, and Verisign. WS-Security is a means of using XML to encrypt and digitally sign SOAP messages. It also provides a mechanism for passing security tokens for authentication and authorization for the SOAP messages.

A typical example of security token is a user name and password token, in which a user name and password are included as text.

Another important aspect to be considered for securing Web services is the *access control* whose solution requires investigating policies for specifying access control rules together with a language for expressing them, and an architecture for their enforcement. Several proposals have been introduced for access control to distributed heterogeneous resources from multiple sources [3,4,5,6]. Two relevant access control languages using XML are WS-Policy [21] and XACML [25]. Based on the WS-Security, WS-Policy includes a set of general messaging related assertions defined in WS-PolicyAssertions [23] and a set of security policy assertions related to supporting the WS-Security specification defined in WS-SecurityPolicy [22]. In addition to the WS-Policy, WS-PolicyAttachment [24] defines how to attach these policies to Web services or other subjects such as service locators. The eXtensible Access Control Markup Language (XACML) [25] is the result of a recent OASIS standardization effort proposing an XML-based language to express and interchange access control policies. XACML is designed to express authorization policies in XML against objects that are themselves identified in XML. The language can represent the functionalities of most policy representation mechanisms.

In this paper we provide an overview of the Web Service technology, and illustrate the basic concepts for securing Web services. We then describe the design of a Web service architecture for enforcing access control policies and provide an example of implementation based on the WS-Policy as access control language [2]. Note, however, that our proposal is completely independent from the specific access control language and can be therefore used with any other solution.

The remainder of this paper is organized as follow. Section 2 illustrates the basic characteristics of WS-Policy. Section 3 presents our architecture. Section 4 describes how the access control is enforced. Finally, Section 5 presents our conclusions.

2 WS-Policy overview

Web Service Policy framework (WS-Policy) provides a generic model and a flexible and extensible grammar for describing and communicating the policies of a Web service [21]. Other specifications, such as *WS-PolicyAssertions* [23] and *WS-SecurityPolicy* [22], provide specific applications of this grammar for their domains. A policy is a collection of one or more *policy assertions* that represent an individual preference, requirement, capability, or other properties that have to be satisfied to access the *policy subject* associated with the assertion. The XML representation of a policy assertion is called *policy expres-*

Value	Meaning
wsp:Required	The assertion must be applied to the subject. If the assertion is not satisfy, a fault or error will occur.
wsp:Rejected	The assertion is not supported and if present will cause failure.
wsp:Optional	The assertion may be applied but it is not required.
wsp:Observed	The assertion will be applied and requestors of the service are informed that the policy will be applied.
wsp:Ignored	The assertion is processed, but ignored; no action will be taken as a result of it being specified. Requestors are informed that the policy will be ignored.

Fig. 1. Values for attribute Usage

sion.[5] Element wsp:Policy is the container for a policy expression. Policy assertions are typed and can be *simple* or *complex*. A simple assertion can be compared to other assertions of the same type without any special consideration about their semantics. A complex assertion requires an assertion type-specific means of comparison. The assertion type can be defined in such a way that the assertion is parametrized. For instance, an assertion describing the maximum acceptable password size (number of chars) would likely accept an integer parameter indicating the maximum char count. In contrast, an assertion that simply indicates that a password is required does not need parameters; its presence is enough to convey the assertion. Every assertion is associated with a mandatory attribute called Usage that specifies how the assertion should be processed. Figure 1 illustrates the five possible values for attribute Usage together with their meaning.

In cases where there are multiple choices for granting a given access (e.g., different authentication mechanisms), attribute wsp:preference can be used to establish an order among the different choices. Possible values for attribute wsp:preference are integers, where a higher number represents a higher preference. WS-Policy also provides an element, called wsp:PolicyReference, that can be used for sharing policy expressions between different policies. Conceptually, when a reference is present, it is replaced by the content of the referenced policy expression. Policy assertions are combined by using the following *policy operators*:

- wsp:All requires that all of its child elements be satisfied;
- wsp:ExactlyOne requires that exactly one of its child elements be satisfied;
- wsp:OneOrMore requires that at least one of its child elements be satisfied.

Lack to specify a policy operator is equivalent to specify the wsp:All operator. Figure 2(a) illustrates a simple example of policy stating that the

[5] Note that using XML to represent policies facilitates interoperability between heterogeneous platforms and Web service infrastructures.

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
  <wsp:ExactlyOne>
    <wsp:All wsp:Preference="100">
      <wsse:SecurityToken>
        <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
      </wsse:SecurityToken>
      <wsse:SecurityToken>
        <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
        <wsse:Username>Alice</wsse:Username>
      </wsse:SecurityToken>
    </wsp:All>
    <wsp:All wsp:Preference="1">
      <wsse:SecurityToken>
        <wsse:TokenType>wsse:X509v3</wsse:TokenType>
      </wsse:SecurityToken>
      <wsse:SecurityToken>
        <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
        <wsse:Username>Alice</wsse:Username>
      </wsse:SecurityToken>
    </wsp:All>
    <wsp:PolicyReference URI="#opts" />
  </wsp:ExactlyOne>
</wsp:Policy>
```

(a)

```
<wsp:Policy xmlns:wsse="..." xmlns:ns="...">
  <wsp:All wsu:Id="opts">
    <wsse:SecurityToken>
      <wsse:TokenType>wsse:X509v3</wsse:TokenType>
    </wsse:SecurityToken>
    <wsse:SecurityToken>
      <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
      <wsse:Username>Bob</wsse:Username>
    </wsse:SecurityToken>
  </wsp:All>
</wsp:Policy>
```

(b)

Fig. 2. A simple example of policy (a) and the corresponding referred policy (b)

access is granted if exactly one security token among the following is provided: *i)* a Kerberos certificate and a UsernameToken with Username `Alice`, *ii)* an X509 certificate and a UsernameToken with Username `Alice`, or *iii)* an X509 certificate and a UsernameToken with Username `Bob`. The third option corresponds to the referred policy, called `opts`, illustrated in Figure 2(b).

3 System architecture

We describe our architecture for enforcing access control policies [3]. The proposed architecture satisfies the following requirements.

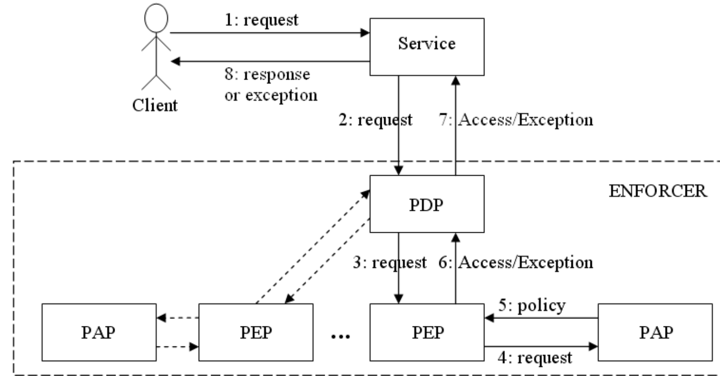

Fig. 3. System Architecture

- *Modularity:* it includes different modules (i.e., the PAP, PEP, and PAP modules described in the following) that can be realized by different parties thus reducing, for example, the development time.
- *Policies language independency:* it is independent from the specific access control language used for specifying restrictions. Therefore, different access control languages can be adopted such as WS-Policy (used in our implementation) or XACML.
- *Extensibility:* it can be easily extended by adding modules thus resulting in additional levels of control (see Figure 4).
- *Re-usability:* the architecture's modules can be re-used and shared among different entities making the architecture generic and scalable.
- *High performance:* it provides a low time response to eventually support real-time applications.
- *Hardware and software independency:* it allows the distribution of different modules on different machines and with different operating systems. This independency is related to the programming language used for the implementation of the architecture.
- *Programming language independency:* the modules' developers can use the preferred programming language without any restriction. In other words, the developers can select the preferred programming language for the implementation of our architecture.

As shown in Figure 3, the architecture includes an *Enforcer* module that wraps up entirely the computation of access permissions to individual Web services, returning, for each request, the decision of whether the access should be granted or denied. Internally, the Enforcer is composed of three main modules implemented as Web services [10]:

- The *Policy Decision Point* (PDP) module receives an access request and returns a "yes" or "no" decision.

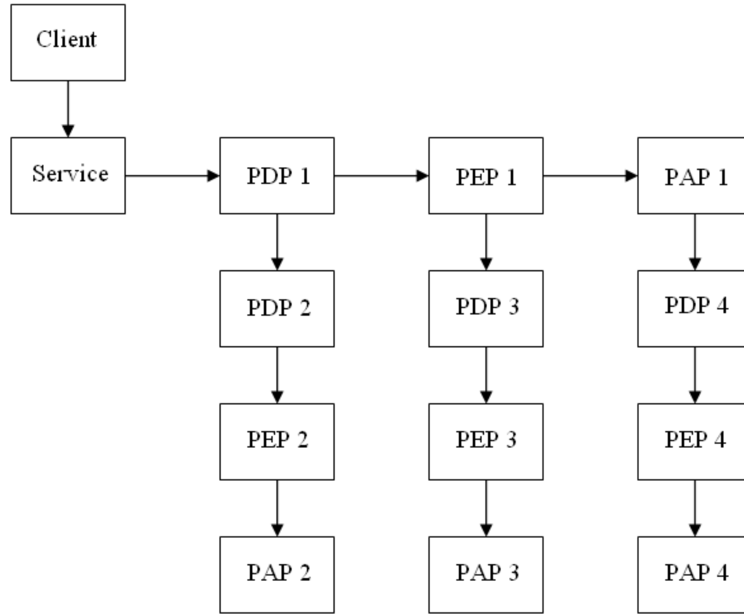

Fig. 4. Waterfall Authentication

- The *Policy Evaluation Point* (PEP) module interacts with the PAP that encapsulates the information needed to identify the applicable policies. It then evaluates the request against the applicable policies and returns the final decision to the PDP module.
- The *Policy Administration Point* (PAP) module retries the policies applicable to a given access request and returns them to the PEP module.

The use of these modules can be regulated by means of an instance of PDP-PEP-PAP. For instance, if the PAP module requires the authentication of the PEP module, the PAP module has to create a new PDP and at least a pair of PEP-PAP (see Figure 4). Client and service represent the entity submitting an access request and the target service, respectively. A service contains a set of publicly available functions that clients can invoke. To fix ideas and make concrete examples, in the following we consider the simple Web service illustrated in Figure 5. This Web service contains two functions, called `TemptureFAut` and `TemptureCAut`, which convert temperatures from Fahrenheit to Celsius and vice versa.

In the remainder of this section, we describe the system components. We then describe how access requests submitted by clients are processed.

3.1 Policy Administration Point (PAP)

The PAP module is a policy repository that provides an administrative interface for inserting, updating, and deleting policies. One of the most important features of this interface is that it can prevent from inserting invalid poli-

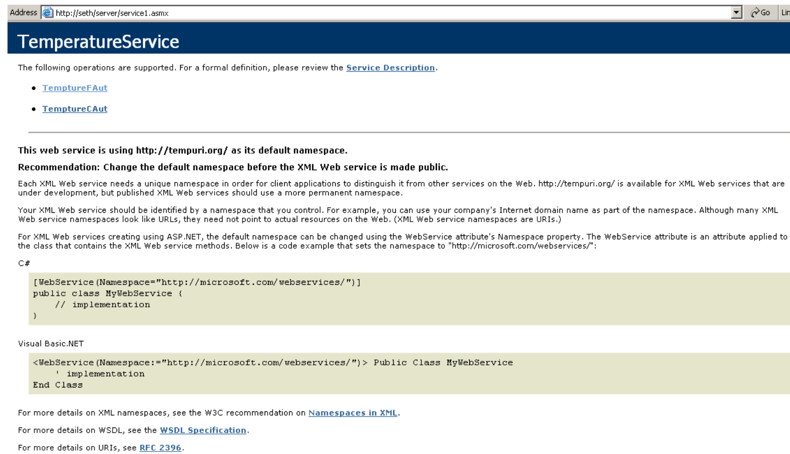

Fig. 5. A simple example of Web service

```
Public Class PAP : Inherits WebService
Public Function Insert( ByVal service As String, ByVal method As String,
ByVal policy As String)
...
End Function
Public Function Modify( ByVal service As String, ByVal method As String,
ByVal policy As String)
...
End Function
Public Function Delete(ByVal policy As String)
...
End Function
Public Function DeleteByService(ByVal service As String)
...
End Function
Public Function DeleteByMethod(ByVal method As String)
...
End Function
Public Function DeleteByServiceMethod(ByVal service As String, ByVal
method As String)
...
End Function
End Class
```

Fig. 6. Administrative interface of the PAP module

cies and from executing update operations that generate invalid policies: a *validation function* ensures that the policies are well-formed. The PAP administrative interface is depicted in Figure 6.

The PAP module invocation is realized with the following function:

Service	Method	PolicyLocation
TemperatureService	TemptureFAut	/Policy/policy1.xml
TemperatureService	TemptureCAut	/Policy/policy2.xml

Fig. 7. An example of policy repository

```
Public Class PAP : Inherits WebService
  <WebMethod()> Public Function PolicyMatch(ByVal service As String,
  ByVal method As String) As String()
  ...
  End Function
End Class
```

The main purpose of this module is to retrieve the policies applicable to a given access request. The PAP module performs a search in the repository based on the received parameters (e.g., the service name and optionally the method name). In our implementation the repository is a relational database that includes a table, called `PolicyRepository`, with three attributes: `Service`, `Method`, and `PolicyLocation`. A tuple $\langle s_1, m_1, path_policy \rangle$ in table `PolicyRepository` states that policy *path_policy* is a policy applicable to method m_1 (if any) of service s_1. The policies applicable to a given access request are stored in an array that is then returned to the PEP module.

Example 3.1 Consider the Web service illustrated in Figure 5. The corresponding `PolicyRepository` is illustrated in Figure 7. It states that the policies applicable to method `TemptureFAut` of service `TemperatureService` are stored in the `/Policy/policy1.xml` file. Analogously, policies applicable to method `TemptureCAut` of service `TemperatureService` are stored in the `/Policy/policy2.xml` file.

3.2 Policy Evaluation Point (PEP)

The PEP module realizes the enforcement of the policies returned by the PAP module. The access request is granted if at least one policy is satisfied; the access is denied otherwise. In this latter case, the PEP module returns to the client an exception string indicating the error (see Section 4). The PEP module invocation is realized with the following interface:

```
<WebMethod()> Public Function Evaluation(ByVal MethodName As String,
ByVal theCallHeader As theHeaderClass) As Boolean
...
End Function
```

More precisely, the PEP module works as follows. The PEP module creates a SAXParser [15] for analyzing the policies and enforces them iteratively. The enforcement phase can generate two possible events: *i)* a policy is satisfied, the enforcement process terminates, and the access request is granted; *ii)* a policy is not satisfied, an exception is raised and stored in a vector. To verify

whether a policy is satisfied or not, all assertions in the policy are evaluated. This evaluation depends on the assertion type and on the subelements of the assertions. After that all policies have been evaluated and none of them is satisfied, the PEP module selects the exception with the highest priority and sends it to the service.

Example 3.2 The PEP module evaluates the following assertions' types as follows.

- *SecurityToken.* The PEP module extracts from the access request all elements whose type is that specified in the attribute `TokenType` (e.g., `UsernameToken`, `X509SecurityToken`, and `Kerberos`) and compares them with the `SecurityToken` element specified in the policy. If the `SecurityToken` has not subelements, the assertion is evaluated to true if the access request includes a token of the same type of that specified in the policy. Otherwise, if the `SecurityToken` has some subelements (e.g., `SubjectName`, `UsePassword`, `Password`), the assertion is evaluated to true if there is a (partial/exact) match between the information specified in the access request and the information contained in these subelements.
- *MessageAge.* The SAXParse compares the timestamp (or, more precisely, the creation date) associated with the access request and the current date and time. If the difference between these two values is less than the value specified in the attribute `Age`, then the assertion is evaluated to true.
- *Language.* The SAXParse verifies whether the access request contains a certificate that specifies the requested language.

3.3 Policy Decision Point (PDP)

The PDP module is the interface between the service and the Enforcer. The service creates an instance of PDP and call the following web method `Response` by using a SOAP message.

```
<WebMethod()> Public Function Response(ByVal theHeader As theHeaderClass,
ByVal ServiceName As String, ByVal MethodName As String) As Boolean

Return MyPEP.Evaluation(ServiceName, MethodName, theHeader)
End Function
```

The body of the SOAP message is used for communicating the target service name and/or method name and the header can be used for specifying additional information (e.g., information for authenticating the service to the PDP). The PDP module instantiates one or more PEP modules that can be based on different policy repositories (PAPs). The interaction between each pair PEP-PAP can return a different decision; the PDP module defines a policy for establishing how to compute a final decision based on the responses of each PEP. Different decision criteria could be adopted, each applicable in specific situations. A natural and straightforward decision policy is the one

```xml
<?xml version="1.0" ?>
<xs:schema xmlns:targetNamespace="http://seth/errors"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="All" type="Compositor"/>
  <xs:element name="ExactlyOne" type="Compositor"/>
  <xs:element name="OneOrMore" type="Compositor"/>
  <xs:element name="Error" type="xs:string"/>
  <xs:complexType name="Compositor">
    <xs:group ref="CompositorContent" maxOccurs="unbounded"/>
  </xs:complexType>
  <xs:group name="CompositorContent">
    <xs:choice>
      <xs:element ref="All"/>
      <xs:element ref="ExactlyOne"/>
      <xs:element ref="OneOrMore"/>
      <xs:element ref="Error" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:group>
  <xs:group name="root">
    <xs:choice>
      <xs:element ref="All"/>
      <xs:element ref="ExactlyOne"/>
      <xs:element ref="OneOrMore"/>
    </xs:choice>
  </xs:group>
  <xs:element name="ErrorsReport" type="ErrorsReportExpression"/>
  <xs:complexType name="ErrorsReportExpression">
    <xs:group ref="root" minOccurs="0" maxOccurs="unbounded"/>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>
</xs:schema>
```

Fig. 8. XSD Schema of message errors

stating that the exception raised by the PEP module with the highest priority wins, or a majority policy can be adopted.

3.4 Exception handling

When there is an exception during the evaluation of an access request submitted by a client, the system should not only capture the exceptions, but also communicate the exception to the client. Such a communication should be performed in a platform-independent way. To accomplish this, we exploit a tree structure based on DOM [16]. Intuitively, the leaves of the tree are the assertions and the internal nodes are the boolean operators (`All`, `OneOrMore`, `ExactlyOne` corresponding to and, or, and xor) used for combining the assertions. The PEP module simplifies the tree evaluating its leaves to true or false. Then, the evaluation tree is simplifies using the usual boolean laws for true and false. If the access request is denied, the Enforcer has to report an error indicating the reason for which the access request has been rejected. Such a error message can specify, for example, that there is no a particular

```
POST /QuoteService HTTP/1.1
SOAP-Action="http://www.acme.com/TemperatureService"
Content-Type: text/xml; charset="UTF-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
  <SOAP-ENV:Header>
    <wsse:Security xmlns:wsse= "http://schemas.xmlsoap.org/ws/2002/04/secext">
      <wsse:BinarySecurityToken ValueType="wsse:X509v3"
          EncodingType="wsse:Base64Binary">
        MIIEZzCCA9CgAwIBAgIQEmtJZc0...
      </wsse:BinarySecurityToken>
      <wsse:UsernameToken >
        <wsse:Username>Alice</wsse:Username>
      </wsse:UsernameToken>
    </wsse:Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <TemperatureService>
      <TemptureFAut>32</TemptureFAut>
    </TemperatureService>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig. 9. An example of access request

```
<wsp:Policy xmlns:wsse="..." xmlns:wssx="...">
  <wsp:All wsp:Preference="1" wsp:Usage="wsp:Required">
    <wsse:SecurityToken>
      <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
      <wsse:Username>Alice</wsse:Username>
    </wsse:SecurityToken>
    <wsse:SecurityToken>
      <wsse:TokenType>wsse:X509</wsse:TokenType>
      <Claims>
        <SubjectName>SubscriptionDate="31/01/2004"</SubjectName>
      </Claims>
    </wsse:SecurityToken>
  </wsp:All>
</wsp:Policy>
```

Fig. 10. An example of policy

certificate or there is no a pair username-password. In our system, the error messages are in XML format and are well-formed with respect to the XSD schema shown in Figure 8.

4 Access request enforcement

We now describe the enforcement in more details. Suppose that user `Alice` issues an access request for the method `TemptureFaut` of the service `TemperatureService` illustrated in Figure 5. The access request evaluation proceeds as follows.

- The access request issues by the client is passed to the requested service. In our example, `Alice` issues the SOAP request illustrated in Figure 9, where, for the sake of simplicity, the namespaces are omitted.
- The access request is then passed to the PDP module that instantiates one or more PEP modules. The request is redirected to the PEP module by using the `Response` method illustrated in Section 3.3.
- The PEP module sends the access request to the PAP module that returns all the applicable policies. In our example, according to the `PolicyRepository` table illustrated in Figure 7, the PAP module returns the policies specified in `/Policy/policy1.xml` and shown in Figure 10. It states that the access is granted if the access request includes a `UsernameToken` with `Username Alice` and an X509 certificate declaring a specific subscription date.
- The PEP module then evaluates the policies returned by the PAP module. If the access is granted, the decision is sent to the service that in turn returns the response to the client. Otherwise, an exception is returned. In our example, the policy in Figure 10 evaluates to false because the X509 certificate contained in the access request does not include any information about the subscription date. The error message returned to the client is as follows.

```
<ErrorsReport>
  <All>
    <Error>Needed subscription date</Error>
  </All>
</ErrorsReport>
```

5 Conclusions

XML-based Web services represent a challenge and an opportunity for organizations wishing to expose product and service offerings through the Internet. In such a context, security is currently one of the main concerns and several initiatives are currently ongoing aimed at achieving a standardized way for supporting integrity, confidentiality, and access control for XML Web services. In this paper, we have presented a Web service architecture for enforcing access control policies. We conclude by mentioning some interesting future directions for extending our work.

- *UDDI integration.* Since the modules of the Enforcer have been implemented as Web services, it is possible to integrate our architecture with an UDDI register where the available PEP, PAP, and PDP can be registered. A trusted PAP module can, for example, use the UDDI register for searching a particular PEP satisfying given requirements.
- *Certificate validation.* In our system, the validation of a certificate is performed by checking the creation date an the expiry date contained in the certificate itself. The next evolution of our architecture could include a certification authority (CA).

- *Other policy language specification.* Our architecture is completely independent from the particular access control language adopted. We can then use the architecture to realize an access control policy enforcement based on other languages such as XACML.
- *Java and open source.* Our architecture has been implemented in Visual Basic and can be installed on Windows machines. Note, however, that as discussed in Section 3, the architecture design is platform-independent. An interesting alternative consists in implementing an open source, platform-independent architecture based on Java services.

6 Acknowledgments

This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591 and by the Italian MIUR within the KIWI and MAPS projects.

References

[1] Damiani, E., S. De Capitani di Vimercati and P. Samarati, *Towards Security XML Web Services*, in Proc. of the 2002 ACM Workshop on XML Security, Washington, DC, USA, November 2002.

[2] Ardagna, C.A., and S. De Capitani di Vimercati, *A comparison of modeling strategies in defining XML-based access control language*, Computer Systems Science & Engineering Journal, 2004 (to appear).

[3] Bonatti, P., and P. Samarati, *A unified framework for regulating access and information release on the web*, Journal of Computer Security, **10**, 241-272.

[4] Damiani, E., S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati, *Securing SOAP E-services*, International Journal of Information Security (IJIS), **1**, 100-115, February 2002.

[5] Bacon, J., J.A. Hine, K. Moody, and W. Yao, *An architecture for distributed OASIS services*, In *Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms nd Open Distributed Processing*, Hudson River Valley, New York, USA, April 2000.

[6] Koshutanski, H., and F. Massacci, *An access control framework for business processes for web services*, In *Proc. of the 2003 ACM workshop on XML security*, Fairfax, Virginia, November 2003.

[7] De Capitani di Vimercati, S., and P. Samarati, *Access control: Policies, models, and mechanisms*, Foundations of Security Analysis and Design, 2001.

[8] Newcomer, E., "Understanding Web Services : XML, WSDL, SOAP, and UDDI", Addison Wesley, 2002.

[9] Galbraith, B., W. Hankinson, A. Hiotis, M. Janakiraman, D. V. Prasad, R. Trivedi, and D. Whitney, "Professional Web Services Security", Wrox Press Ltd., December 2002.

[10] "Web Services Architecture", http://www.w3.org/TR/ws-arch/.

[11] Hoque, R., "CORBA 3", IDG Books Worldwide, Inc., 1998.

[12] Bray, T., E. Maker, J. Paoli, and C. M. Spenberg-McQueen , "Extesible Markup Language (XML) 1.0 (Second Edition)", World Wide Web Consortium (W3C), http://www.w3.org/TR/REC-xml, October 2000.

[13] "Apache XML Project", http://xml.apache.org/.

[14] Bradley, N., "The XML Companion", Addison Wesley, 2002, 3rd.

[15] "Official website for SAX", http://www.saxproject.org/.

[16] "Document Object Model (DOM)", http://www.w3.org/DOM/.

[17] Box, D., "Simple Object Access Protocol (SOAP) 1.1", http://www.w3.org/TR/SOAP, May 2000.

[18] Chinnici, R., M. Gudgin, J. Moreau, and S. Weerawarana, "Web Services Description Language (WSDL) version 1.2. World", Wide Web Consortium (W3C), http://www.w3.org/TR/wsdl12, July 2002.

[19] "UDDI Technical White Paper", http://www.uddi.org.

[20] Atkinson, B., and G. Della-Libera et al., "Web services security (WS-Security)", http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-security.asp, April 2002.

[21] Box, D., "Web Services Policy Framework (WS-Policy) version 1.1", http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policy.asp, May 2003.

[22] Della-Libera, G., and other, "Web Services Security Policy Language (WS-SecurityPolicy) version 1.0", http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-securitypolicy.asp, December 2002.

[23] Box, D., "Web Services Policy Assertions Language (WS-PolicyAssertions) version 1.1", http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyassertions.asp, May 2003.

[24] Box, D., "Web Services Policy Attachment (WS-PolicyAttachment) version 1.1", http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-policyattachment.asp, May 2003.

[25] Box, D., "OASIS eXtensible Access Control Markup Language TC", http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[26] Ambler, S., T. Jewell, and E. Roman, "Mastering Enterprise JavaBeans", Wiley Computer Publishing, 2002.

[27] Brown, N., and C. Kindel, "Distributed Component Object Model Protocol", http://www.globecom.net/ietf/draft/draft-brown-dcom-v1-spec-03.html.

[28] "OASIS Security Services TC (SAML)", http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.

[29] "XML Schema", http://www.w3.org/XML/Schema.