Fine Grained Access Control for SOAP E-Services

Ernesto Damiani
Dip. Tecnologie dell'Informazione
Università di Milano
Via Bramante 65
26013 Crema, Italy
edamiani@crema.unimi.it

Sabrina De Capitani di Vimercati
Dip. Elettronica
Università di Brescia
Via Branze 38
25123 Brescia, Italy
decapita@ing.unibs.it

Stefano Paraboschi
Dip. Elettronica e Informazione
Politecnico di Milano
Piazza L. da Vinci 32
20133 Milano, Italy
parabosc@elet.polimi.it

Pierangela Samarati
Dip. Tecnologie dell'Informazione
Università di Milano
Via Bramante 65
26013 Crema, Italy
samarati@dsi.unimi.it

ABSTRACT

Lightweight protocols for remote service invocation via HTTP and XML, such as SOAP, are rapidly gaining acceptance among developers of Internet-based e-services, especially because of their firewall-traversal capabilities. However, no standard technique for access control security is currently defined for either HTTP or SOAP itself. Concerns have been raised about the possibility that different SOAP applications will deal with embedded security in different ways, leading to application-dependent security holes. In this paper, we propose an approach that relies on the XML structure of SOAP requests to support fine-grained authorizations at the level of individual XML elements and attributes that compose a SOAP call. The result is a simple, yet powerful and general, technique to enforce access restrictions to SOAP invocations.

Categories and Subject Descriptors

C.2.0 [**Computers-Communication Networks**]: General—*Security and protection*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Security*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Design

Keywords

Access control, SOAP, XML, Certificates, Roles

1. INTRODUCTION

Accessing information on the global Internet has become an essential requirement of the modern economy. Recently, the focus has shifted from access to traditional information stored in WWW sites to access to large *e-services* such as e-government services, remote banking, or airline reservation systems [12]. *Global e-services* are also coming of age, designed as custom applications exploiting single e-services already available on the Internet and integrating the results. The Hypertext Transfer Protocol (HTTP), the most common protocol used on the Web today, has proved to be an effective, scalable technology for transferring multimedia information, but it was not designed for accessing distributed e-services. Indeed, calls to e-services are more easily modeled by *distributed object protocols* as *Remote Method Calls (RMCs)*, such as *CORBA* [4], *DCOM* [7], and *Java-RMI* [1], in which clients pass parameters to remote components and get some kind of result in return. Many RMC-based protocols support object invocation access policies that govern whether a client, acting on behalf of its current user, can invoke the requested operation on a target object. The policy is normally enforced by the software implementing the protocol. For instance, *CORBA Security Service* access policies [23] are expressed in terms of the user's privilege attributes, which are encapsulated in a `Credentials` object, and the target object's control attributes, which are encapsulated in an `AccessDecision` object. The control attributes, which are associated with every object that accepts invocations, describe the authorizations that a user must have in order to be allowed to invoke the requested operation on the specific object implementation. Most object invocation access control services are implemented by intercepting all invocations, possibly on both client and server sides. Having intercepted the object invocation, the `Credentials` object is consulted to obtain the client's authorizations, which are then passed as a parameter to the `AccessDecision` object. Finally, `AccessDecision` either grants or denies permission to continue the object invocation. Access control provisions like those provided by CORBA Security Services would be particularly useful on the public Internet; on the other hand, other features of RMC-based distributed object protocols have proved to be rather unsuitable for Internet use. These protocols exhibit two main problems that prevent their large-scale use on the Net.

- *Verbosity*: Many RMC-based protocols require considerable bandwidth due to their high service to data packets ratio. For instance, DCOM's *ping-based* lifecycle management requires continuous conversation between client and server to keep the interaction alive.

- *Firewall traversal and user authentication*: Many organizations are reluctant to enable RMC-based protocols such as CORBA-IIOP or DCOM through their security firewalls. Moreover, current authentication

WWW10, May 1-5, 2001, Hong Kong.
ACM 1-58113-348-0/01/0005.

techniques for RMC protocols were not designed for Internet use. For instance, in the case of DCOM, it seems unlikely that generic Internet-based clients will ever be able to perform domain-based authentication with DCOM application servers.

Several vendors provide patches based on *tunneling firewall-traversal*, encapsulating packets sent to RMC servers into a single packet stream to a *bastion host*. However, these products tend to be very sensitive to configuration mistakes and are not interoperable. To foster a clean solution to this problem, the Internet and Web communities have provided several proposals for the use of XML in *lightweight* network protocols and distributed applications: XML-RPC [3], SOAP [6], XMI [2], WebDAV [27], ICE [22], and IOTP [8] are only a few examples. In this paper, we focus on SOAP [6], seen as an attempt to codify the main concepts behind lightweight techniques into a simple and generic protocol that can serve as an industry standard. SOAP is a clean and elegant solution to the verbosity and firewall traversal problems; however, no standard technique for access control security has been yet defined for either HTTP or SOAP itself. Concerns have been raised about the possibility that different SOAP applications will deal with embedded security in different ways, leading to application-dependent security holes. We propose a simple yet general technique to specify and enforce *fine-grained access control* for SOAP-like invocations, leveraging the XML encoding used by SOAP for both service invocation and responses. Our approach allows the specification of *fine-grained usage policies* for e-services using XML. We believe the application of access control techniques to XML-based service invocations to be crucial for the development of reliable and secure e-services.

The paper is structured as follows. Section 2 briefly outlines firewall-related issues for RMC-based protocol security. Section 3 presents the features of the SOAP protocol that are relevant to our approach and discusses the limitation of firewall-based security policies applied to it. Section 4 describes our proposal for a fine-grained flexible access control technique for SOAP invocations, which overcomes dependence from firewalls. Section 5 discusses the design and implementation of an authorization filter enforcing our approach. Finally, Section 6 draws our conclusions.

2. FIREWALL IMPACT ON OBJECT PROTOCOLS

Most organizations insert *firewalls* between their publicly accessible Web servers and the remote clients who can access those servers, blocking incoming traffic according to various criteria. Firewalls partition the Internet in *security cells*, whose boundaries are difficult to penetrate for remote service invocations. As a simple example, consider the TCP/IP architecture, where each *well-known service* is assigned a *port number* and each service request carries that number. While blocking all ports except the standard port 80 used for HTTP connections is a common practice, it prevents using distributed object protocols like CORBA or DCOM, which rely on dynamically assigned ports for remote method invocations. Enabling access to an e-service through a firewall requires manual intervention for the firewall configuration, discouraging free development of applications using e-services as building blocks. Furthermore, clients of distributed applications that lie behind another corporate firewall suffer a similar problem. Requiring clients to reconfigure their firewalls to access a remote e-service is a rather unrealistic assumption.

```
POST /QuoteService HTTP/1.1
SOAP-Action="http://www.acme.com/Quote"
Content-Type:  text/xml; charset="UTF-8"
Content-Length:  nnnn
<!-- XML tree encoding the invocation goes here -->
```

Figure 1: A HTTP header carrying a SOAP request

Several palliative remedies to these problems have been proposed in recent years, such as *COM Internet Services* (CIS) [18] and *Remote Data Services* (RDS) [9]. CIS makes it possible to use DCOM over HTTP on port 80, thanks to a special HTTP-based handshake used to establish the initial connection between client and server. From that point on, CIS relies on DCOM over TCP. While CIS does solve the firewall traversal problem, it is a platform-specific solution for Windows-based systems and retains all the verbosity of the original DCOM protocol. Remote Data Services (RDS) allows for instantiating remote DCOM objects and invoking their methods over HTTP. Again, however, RDS is platform-specific, as it relies on a proprietary Dynamic Link Library running on Microsoft Internet Information Service.

In the next section, we describe SOAP a standard, platform-independent solution that was recently proposed to solve these problems.

3. SOAP IN A NUTSHELL

We give a brief and informal overview of the SOAP protocol, including the format of SOAP HTTP requests and responses. For the sake of conciseness, we only deal with SOAP features that are relevant to our approach, and therefore our description does not attempt to be exhaustive.

3.1 SOAP Requests

SOAP requests carry remote method invocations over HTTP. They are fully declarative, inasmuch they do not dictate how the target component should handle the request. Many application scenarios can be envisioned for this kind of requests; for instance, a courier service could offer to its customers an Internet-based e-service to get quotes and place orders for international deliveries. Client applications could freely integrate this service with others, using HTTP as the common transport protocol. For instance, the HTTP `POST` request in Figure 1 encodes the invocation of a quote service using the SOAP protocol.

It should be noted that while the HTTP request in Figure 1 points to a valid *Uniform Resource Identifier* (URI) of `http://www.acme.com/Quote`, it leaves it entirely to the *SOAP software gateway* behind the URI to decide how to activate the corresponding local component and invoke locally the specified method. Note that the `Content-Type` header contains the generic value `text/xml`, used by all XML-based HTTP traffic. In principle, the `SOAP-Action` field could be to used by firewalls that, by looking at the URI value of the field, could filter out all HTTP requests carrying SOAP traffic, or, conceivably, allow only certain interfaces to pass through. However, this would require the firewall to be SOAP-aware to some extent.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
 "http://schemas.xmlsoap.org/soap/Envelope"
 xmlns:ACME="http://www.acme.com/soap"
 SOAP-ENV:EncodingStyle =
 "http://schemas.xmlsoap.org/soap/encoding"
 ACME:headers="#ref-0" ACME:main="#ref-1">
  <SOAP-ENV:Header ACME:id="ref-0">
    <!-- header entries go here -->
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
  <!-- method element -->
    <ACME:GetQuote ACME:id="ref-1">
      <!-- parameter elements go here -->
    </ACME:GetQuote>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 2: The SOAP payload

3.2 SOAP XML Payload

The SOAP XML payload contains the encoded invocation; its lexicon is defined by a standard XML namespace SOAP-ENV. In SOAP, the XML payload is used mainly to encode parameters' datatypes in a platform independent way, much as CORBA's *Common Data Representation* [4] or DCOM's *Network Data Representation* [7]. The XML payload includes a root `Envelope` element and a child `Body` element, the latter having an optional `Header` sibling. The SOAP payload's root element, `Envelope`, provides the serialization context for the method calls that follow. The `Envelope` element can contain additional attributes (qualified by a suitable XML namespace).

Elements encoding methods and parameters must also be namespace-qualified. To illustrate this point in the example of Figure 2, we added an attribute named `main`, whose value is a URI fragment identifier that points to the method element. Assuming a `Header` tag to be present, our sample `Envelope` also contains another additional attribute (named `headers`) whose value is again a URI fragment identifier, which references the `Header` element. The `headers` and `main` custom attributes we added make it possible to access the method or headers elements simply by following the `ID-IDREF` implicit link.

The `Body` element contains a first sub-element whose name is the method name. This element should contain all the information that the software gateway needs to perform the corresponding local invocation. Namely, it contains a child element for each parameter, as follows:

```
<ACME:GetQuote ACME:id="ref-1">
 <ACME:OriginZIP> 90070 </ACME:OriginZIP>
 <ACME:DestZIP> 16804 </ACME:DestZIP>
 <ACME:Weight> .500 </ACME:Weight>
 <ACME:ServiceType> Overnight </ACME:ServiceType>
</ACME:GetQuote>
```

The `Header` element contains auxiliary information (called *header entries*) not functionally related to the method invocation, such as transaction management and payment. SOAP headers may contain the standard `Actor` and `MustUnderstand` attributes (as well as other optional, namespace-qualified ones), respectively stating the URI of the final destination of the message and whether header processing capability on the part of the recipient is mandatory (1) or not (0).

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="UTF-8"
Content-Length: nnnn
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
 "http://schemas.xmlsoap.org/soap/Envelope"
 xmlns:ACME="http://www.acme.com/soap"
 SOAP-ENV:EncodingStyle =
 "http://schemas.xmlsoap.org/soap/encoding">
  <sOAP-ENV:Body>
    <ACME:GetQuoteResponse>
      <ACME:Amount> 18 </ACME:Amount>
    </ACME:GetQuoteResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 3: A SOAP response

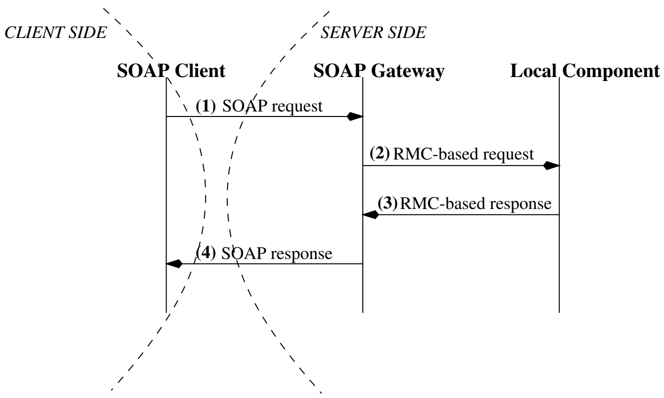

Figure 4: Execution Sequence of a SOAP Call

In the remainder of the paper, we shall use SOAP header entries to support client identification and a variety of security arrangements in the framework of a complete access control technique.

3.3 SOAP Response

A SOAP response is similar to a request, apart from the fact that it adds a `Response` suffix to the element name used for the method. For instance, for the method `GetQuote`, the response element is `GetQuoteResponse`. A sample response to our `GetQuote` call of Figure 2 is reported in Figure 3.

3.4 Firewalls and SOAP

The SOAP messaging protocol piggybacks a lightweight distributed object protocol on top of HTTP, using HTTP connections to carry messages formatted with XML. In other words, SOAP defines a mechanism to pass commands and parameters between HTTP clients and servers that uses XML for data encoding and is therefore independent from the operating system, programming language, or object model used on either the server or the client side. SOAP solves the firewall traversal problem, as it relies on HTTP as the transport mechanism. Moreover, custom HTTP headers have been proposed as a technique for identifying SOAP invocations [6]. As illustrated in Figure 4, when a client sends a SOAP invocation over HTTP, the header triggers the execution of a software gateway on the server side, which forwards the invocation to the target component using (locally) a suitable heavyweight RMC-based protocol. This technique addresses the verbosity problem, as it uses a simple XML structure for both invocation and response. Moreover, relying on custom HTTP headers makes it possible for system administrators to configure firewalls to selectively block out SOAP requests using SOAP-specific HTTP headers. Besides the firewall security benefits of designing SOAP using extended HTTP headers, the SOAP specification does not define any protocol-specific security features. SOAP implementations may well utilize any standard HTTP security feature, taking advantage of HTTP authentication mechanisms as well as *SSL* for secure channel communications (using secure HTTP connections via *HTTPS*). Also, *secure cookies* have been proposed [24] to provide user authentication, integrity and confidentiality when interacting with WWW sites. However, these techniques are not a valid substitute for a fully-fledged security model, and several security issues related to SOAP are still to be solved. Indeed, both firewall filtering of HTTP headers and the secure cookies approach have been conceived for simple document retrieval on the WWW and cannot be considered satisfactory for remote invocations. Access to e-services requires a more sophisticated security model than the one normally applied to HTTP traffic. This is the focus of the next section.

4. ADDING FINE-GRAINED AUTHORIZATIONS TO SOAP

While the invocation of Figure 1 does not contain any provision for access control, the organization managing a remote interface to an international courier service is likely to impose some constraints. For instance, getting quotes could be restricted to retailers, or allowed only for customers connecting from some a trusted domain. Of course, checking the first constraint could be done by writing application code on the server, while the second constraint could be enforced via the SOAP specific firewall configuration just discussed. Both these ad-hoc solutions, however, potentially restore the dependence on firewall configuration and operation, whose elimination was the reason why a lightweight, XML-based protocol like SOAP was selected in the first place. Our research line is to avoid firewall dependence while providing organizations managing e-service with full control on how their SOAP servers are used. To this end, we propose to employ *fine-grained XML access control techniques* to specify usage policies for SOAP based e-services, in order to obtain the full functionality of an object invocation access control service.

As illustrated in Figure 5, our access control system is based on an authorization filter intercepts every invocation addressed to the SOAP gateway and evaluates it against authorizations specifying restrictions to service accessibility. Based on the authorizations, the request may: 1) be rejected; 2) be allowed as is; or 3) be filtered and executed in a modified form, where filtering of a request may involve elimination of some of its parameters that the current invoker is not allowed to specify. Once filtered, requests are passed to the SOAP gateway, which will produce a response to be returned to the client. The response also, is sent through the access control system and may be subject to some filtering. In this paper, we focus on the specification and enforcement of restrictions applicable to requests (i.e., request filtering). Response filtering can be performed in a similar way.

4.1 Authorization Objects

In traditional authorization systems, an object characterizes an entity *on* which access is being requested and consequently authorizations define whether access to the entity should be granted or not [26]. In the SOAP context, full interfaces may not be readily available. However, clients submit requests, whose XML structure is modeled after the interface offered by the remote e-service. Therefore, in our opinion, it makes more sense to consider the *requests* themselves as objects of our authorization system. Fine-grained authorizations are supported by allowing reference to specific elements/attributes within a request (such as the method name or the value of any of its parameters). Reference to individual elements and attributes can be used to:

- evaluate conditions on requests (e.g., in the case where a request can pass only if its parameters have certain values), and
- explicitly refer access restrictions to specific portions of a request (e.g., in the case where specific parameters within a request should be filtered out).

In our model, authorization objects therefore coincide with XML elements and attributes in the SOAP request (or response), identified via *path expressions* expressed using a simple XPath-like syntax [28]. An *XPath expression* on an XML document tree is a sequence of element names or predefined functions separated by the character `/` (slash): $l_1/l_2/\ldots/l_n$. Path expressions may terminate with an attribute name, syntactically distinguished prefixing it with the special character `@`. For instance, path expression `/SOAP-ENV:Envelope/SOAP-ENV:Body/ACME:GetQuote` denotes the nodes of the `GetQuote` element (the method's name), which are children of `Body` elements, which in turn are children of `Envelope` elements. *Absolute* path expressions, prefixed by a slash character, start from the root of the document while *relative* ones, starting with an element, describe a path whose initial point is any element in the document.

The XPath syntax allows to specify conditions on any element of a path expression. Conditions are enclosed within square brackets and may operate on the "text" of elements (i.e., the character data in the elements) or on names and values of attributes. For instance, path expression `SOAP-ENV:Envelope/SOAP-ENV:Body/ACME:PlaceOrder/[./ServiceType="48-hours"]` identifies all orders of type "48-hours".

4.2 Authorization Subjects

Authorization subjects characterize entities whose requests have to be evaluated and to which authorizations (permissions or denials) can be granted. In our model we allow the identification of subjects on the basis of:

- the identity of users (on behalf the which the client is executing);
- the location (either numerical or symbolic) from which the request originates;
- the membership of the user in user groups (stored at the server);
- the role/s in virtue of which a user is presenting his request (e.g., `premier_member`).

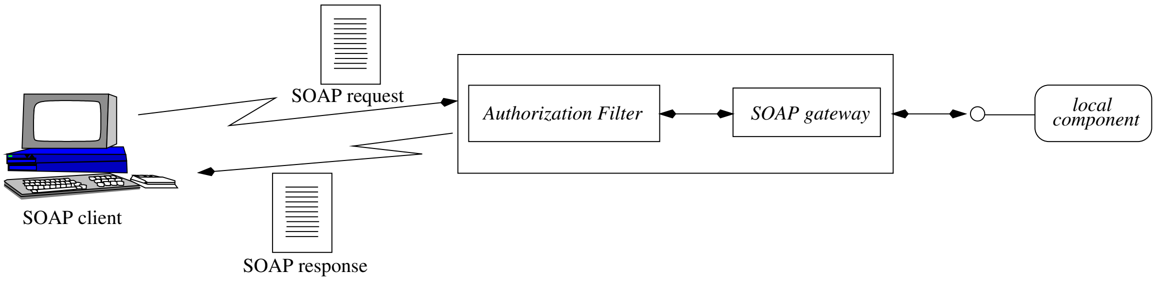

Figure 5: The System Operation

The support of roles appears crucial in an open context in which SOAP can operate, where not all *individuals* may be registered at a service and requests may arrive from previously unknown parties. In such a context the ability to invoke a service often depends on the capacity in which a user can exercise [5, 16] (which we characterize with a *role*) rather than on who the user actually is. Note that support of both groups and roles is not redundant, as the two exhibit different behaviors. In particular, unlike groups, roles carry a dynamic behavior and can be activated and deactivated by users at their will. This behavior is reflected in our system where groups define named sets of users (whose composition is defined and known at the SOAP server) and roles activation is enabled by attaching credentials with a request. Our model therefore need not to control role assignment and consequent activation/deactivation (which, as a matter of fact, are not competence of the SOAP server but of the authority managing the role) but only needs to know the name of the roles.

A subject presenting an invocation request is characterized by a 4-tuple ⟨`user-id`, `IP-address`, `sym-address`, {`role-id`}⟩ identifying a user connected from a given machine and operating in a given capacity (stated by the set of roles for which he/she presents certificates). Note that no explicit mention of group is made in the definition of the subject as the group membership cannot be discretionary released and all the groups to which the user belongs will be considered. Only roles need to be explicitely stated.

Authorizations stating permissions or denials can be specified for subjects with reference to any of the characteristics with which they can be distinguished. In particular, a subject of an authorization can be either a *user*, a *group*, or a *role*, possibly restricted with respect to the *location* from which requests can originate. The location can be also specified with classical patterns identifying all the machines belonging to a given sub-network (e.g., `131.100.*` or `*.it`). We support also the definition of abstractions for roles, and assume a role abstraction hierarchy is defined which can group set of roles and refer to them with a single name. (Non-minimal elements of such a hierarchy are not roles that can be enabled by invokers but simply abstractions defined on them [5].)

4.2.1 Identifying Authorization Subjects via Custom SOAP Headers

Our approach exploits the features of the SOAP protocol, and represents all the information characterizing the subject by means of a custom header included in each SOAP call. Of course, the use of this feature it not mandatory: the system may also rely on usual HTTP authentication techniques as in [11]. Custom headers provide an elegant and model-neutral way to specify security arrangements. The structure of our header entry (written on a custom namespace SOAP-AC) is closely related to the work currently in progress in XML encoding for SPKI certificates (XML-SPKI [24]). For the sake of conciseness, we shall not explain its structure and operation in detail; rather, Figure 6 contains a simplified, commented DTD.

```
<!ELEMENT credential (user,role*) >
<!-- user element is mandatory, possibly followed by roles -->
<!ELEMENT user (userid,passwdhash,symname?,netaddr?)>
<!ELEMENT role (issuer?,subject,validity?)>
<!-- role element supports certificate handling -->
<!ELEMENT issuer ((public-key|hash-of-key),uri*)>
<!ELEMENT subject ((public-key|hash-of-key|object-hash?|
         name|keyholder?|threshold?),uri*)>
<!ELEMENT validity (notbefore?,notafter?,online*,new-cert?)>
<!ELEMENT name ((public-key|hash-of-key)?,roleid)>
<!ELEMENT symname #PCDATA>
<!ELEMENT netaddress #PCDATA>
<!ELEMENT roleid #PCDATA>
```

Figure 6: DTD for a custom header entry for SOAP subject credentials.

An example of a header entry for a SOAP subject credential is illustrated in Figure 7.[1] With respect to the work in progress for XML-SPKI certificate encoding, our `credential` header includes one additional *mandatory* field (`user`) carrying the user identity and the location from which the connection originates, and zero or more `role` elements. The main difference between our `role` element and an XML-encoded certificate is the `roleid` subelement. In order to foster compatibility with future standards for XML-SPKI, we defined `roleid` to be a child of the `name` element, where the XML-SPKI work in progress allows for any `#PCDATA` content [24].[2] The `user` tag identifies the client with respect to `userid`, `symname` and `netaddr` fields [10]. Note that systems supporting role-based authentication only may still use this element (with a dummy `Anonymous` content, and a dummy password hash) to identify the machine

[1] This figure contains a slight abuse of notation as technically the name space declaration should be given in an ancestor element in order to allow `credential` to be part of its scope.

[2] Another difference is that, unlike XML-SPKI, our DTD leaves both `issuer` and `validity` to be optional in the `role` element. This choice ensures compatibility with standard XML encoding of certificates while retaining flexibility, as role-based security may not always require full certificate handling.

```
<SOAP-AC:credential xmlns SOAP-AC=
  "http://www.xmlsec.org/AC">
  <SOAP-AC:user>
    <SOAP-AC:userid> Alice </SOAP-AC:userid>
    <SOAP-AC:passwdhash SOAP-AC:hash-alg="none">
      DUMMY
    </SOAP-AC:passwdhash>
  </SOAP-AC:user><!-- roles and certificates follow -->
  <SOAP-AC:role>
  <!-- certificate issuer info follows -->
    <XMLSPKI:issuer xmlns XMLSPKI =
    "http://www.xmlsec.org/SPKI">
      <XMLSPKI:public-key>
        <XMLSPKI:dsa-pubkey>
          ...
        </XMLSPKI:dsa-pubkey>
      </XMLSPKI:public-key>
    </XMLSPKI:issuer>
  <!-- end of certificate issuer info, begins subject info --!>
    <XMLSPKI:subject>
      <XMLSPKI:name>
        <XMLSPKI:hash-of-key>
          <XMLSPKI:hash XMLSPKI:hash-alg="sha1">
            ...
          </XMLSPKI:hash>
          <XMLSPKI:uri>
            urn:spki:issuer
          </XMLSPKI:uri>
        </XMLSPKI:hash-of-key>
        <SOAP-AC:roleid>
          FidelityProgram
        </SOAP-AC:roleid>
      </XMLSPKI:name>
    </XMLSPKI:subject>
  <!-- end of subject info (object hash and
  threshold are omitted), begins validity info -->
    <XMLSPKI:validity>
      <XMLSPKI:notbefore>
        2000-12-21_12:00:00
      </XMLSPKI:notbefore>
      <XMLSPKI:notafter>
        2000-12-31_24:00:00
      </XMLSPKI:notafter>
    </XMLSPKI:validity>
  </SOAP-AC:role>
</SOAP-AC:credential>
```

Figure 7: Sample Header Entry for SOAP subject credential

where the client is running.

Client authentication by means of our XML credential can be performed exactly as it is done for standard certificates, taking advantage of challenge-response [14] and secure channel technology like SSL [13] when needed. Also, location addresses stated in the header could be trusted only on secure channels or checked using HTTP services. In the remainder of the paper we shall focus on authorization rather than on authentication issues.

4.3 Authorization Syntax and Semantics

The XML syntax of authorizations supported by our system is illustrated in Figure 8. Basically, an authorization is characterized by a triple (*subject, object, sign*) where:

- *subject* identifies the requestors for which the authorization is intended. It can be specified with reference to their identity, group memberships, location, location patterns, or roles (Section 4.2).

```
<!DOCTYPE set_of_authorizations [
  <!ELEMENT set_of_authorizations (authorization+)>
  <!ELEMENT authorization (subject,object,sign)>
  <!ELEMENT subject ((userid | groupid | roleid)?,
            symname?,netaddr?)>
  <!ELEMENT object (#PCDATA)>
  <!ELEMENT sign EMPTY>
  <!ELEMENT userid (#PCDATA)>
  <!ELEMENT groupid (#PCDATA)>
  <!ELEMENT roleid (#PCDATA)>
  <!ELEMENT symname (#PCDATA)>
  <!ELEMENT netaddr (#PCDATA)>
  <!ATTLIST set_of_authorizations
            about CDATA #REQUIRED>
  <!ATTLIST sign value (+ | −) #REQUIRED>
]>
```

Figure 8: Authorization syntax

- *object* is the element of the SOAP message to which the authorization refers, and is expressed by means of an XPath (Section 4.1).
- *sign* can take values "+" or "−" and defines whether the authorization states a permission or a denial.

Note that there is no explicit mention of the action within authorizations; the action being always the submission of the SOAP request to the SOAP Gateway.

If the object of an authorization is `Envelope`, the authorization refers to the request as a whole; and if the authorization is negative, its enforcement implies a complete rejection of the request. As already discussed, the object can also be any element of the SOAP request tree (e.g., to identify, within the `Body`'s tree, specific parameters that requestor cannot specify).[3]

One might object that negative authorizations assigned to roles are unreliable, as users may simply not present their role certificate. However, this is not a problem in our context where negative authorizations for a role will be primarily used to specify exceptions to positive authorizations granted to the role itself (which the user is obviously presenting).

4.4 Access control

Our access control filter intercepts every SOAP request (and its response) and evaluates it against the specified authorizations. Depending on the authorizations to be enforced, each request can: 1) be rejected completely (request denied); 2) pass unaltered (request fully authorized); or 3) pass modified (a "filtered" request authorized), where modifications to a request usually involve the deletion of some parameters. The semantics of authorization enforcement is simple. All authorizations whose subject matches the subject headers of the SOAP request are applicable to the request. Intuitively, the application of an authorization to a request imposes a permission (or denial) on each node of the SOAP request tree identified by the object term in the authorization. It may happen that the access control policy includes multiple authorizations referred to the same node of

[3]Headers can also appear as objects of authorizations; in particular a negative authorizations can be applied on headers corresponding to specific certificates. The enforcement of such an authorization will remove the certificate from the SOAP request before passing it to the SOAP gateway.

the SOAP request tree (e.g., a positive authorization specified for a user's group and a negative authorization for a specific individual belonging to the group). In this case, the (unique) sign that it is considered to hold for the node is determined according to priorities established on the authorizations. In principle, different priority policies can be applied, and the system can be parametric with them (as in [15] and [17]). In our system, we consider a specific priority, which we consider to be natural for the specific context under consideration. The reason for focusing on a specific policy is to make management simple and intuitive (additional policies could easily be added and the access control made parametric with respect to them). The priority policy we apply is as follows:

- If a request has multiple roles enabled, the union of their privileges is taken; as it is customary in corresponding paper-world situations. Consequently, in the case where some roles hold contradicting authorizations, the positive authorization takes precedence.
- Authorizations specified for a role take precedence over authorizations specified for role abstractions (specificity in the role-abstraction hierarchy).
- Authorizations specified for a user (or a sub-group) take precedence over authorizations specified for a group (specificity of the user-group hierarchy).
- Authorizations specified for a user (either directly or as a member of a group) take precedence over authorizations specified for any of the roles he has activated (priority of the "individual" over the roles he is playing).

The evaluation of a request against the authorizations with respect to the priority policy described above produces a labeling of the SOAP request tree where each node may be assigned either "+" or a "-", stating the access control outcome with respect to the node (or more precisely to the subtree rooted at it). The enforcement of such labels produces a "filtered" request obtained by eliminating, from the original request, all the subtrees whose root is labeled "-". Intuitively, this enforcement can be seen as the one obtained by propagating the permissions stated by "+" down the request tree until a node labeled "-" is found (note that the "-" on a node applies to the whole subtree rooted at the node, regardless of the presence of other labels in the subtree). The reason for completely eliminating the whole subtree rooted at an element to which a negation is applied is that enforcing the negation would partition the element' descendants (which independently taken would not make sense), and preserving the element for the sake of linking its descendants may not correctly reflect the specifications to be enforced.[4]

4.5 An Example

We now illustrate an application of our authorization model to regulate access to the international courier service introduced in Section 3.

[4]Note the different approach with respect to the case where authorization enforcement (and the consequent tree labeling and pruning) [10] was executed for producing view of documents to be returned to specific requestors. In that case an element labeled "-" could have been maintained for the sake of preserving the document structure and reachability of descendant nodes the requestor was entitled to see.

One service offered by the courier is the ability of getting quotes. Let us assume that the company policy is to allow quote querying only to requestors that are customers, where customers can be either individual users registered at ACMEor corporations subscribing to a nation-wide association of courier users, called ACU, which ACMEsupports. Individual users are registered with the system, and therefore ACMEcan recognize them by checking membership in group `Registered_users`. ACU subscribers are not maintained at ACME, and therefore handled with credentials. Namely, requests from them will include a credential stating the ability of the requestor to play the role of `ACU_subscribers`. The policy above is then stated by the following two authorizations (here and in the following we omit the `<authorization>` tag).

```
<subject><groupid>Registered_users</groupid>
</subject>
<object>
/SOAP-ENV:Envelope/[SOAP-ENV:Body/ACME:GetQuote]
</object>
<sign value = "+" />
```

```
<subject><roleid>ACU_subscribers</roleid>
</subject>
<object>
/SOAP-ENV:Envelope/[SOAP-ENV:Body/ACME:GetQuote]
</object>
<sign value = "+" />
```

Consider now the service of automatically placing delivery orders via Internet. Here the policy is the following:

Individual users can only place order for 48-hour service (quicker deliveries are not accepted through this interface)

```
<subject><groupid>Registered_users</groupid>
</subject>
<object>
/SOAP-ENV:Envelope/[SOAP-ENV:Body/ACME:PlaceOrder/
ServiceType="48-hours"]
</object>
<sign value = "+" />
```

Individual users members of the `Retailers` group can place any order if connected from subnetwork 131.175.*.

```
<subject><groupid>Retailers</groupid>
<netaddr>131.175.*</netaddr>
</subject>
<object>
/SOAP-ENV:Envelope/[SOAP-ENV:Body/ACME:PlaceOrder]
</object>
<sign value = "+" />
```

ACU subscribers can place any order *but* a corporate discount code can be specified only if a valid fidelity card (e.g., a credential) is attached to the request.

```
<subject><roleid>ACU_subscribers</roleid>
</subject>
<object>
/SOAP-ENV:Envelope/[SOAP-ENV:Body/ACME:PlaceOrder]
</object>
<sign value = "+" />

<subject><roleid>ACU_subscribers</roleid>
</subject>
<object>
/SOAP-ENV:Envelope/SOAP-ENV:Body/ACME:PlaceOrder/
```

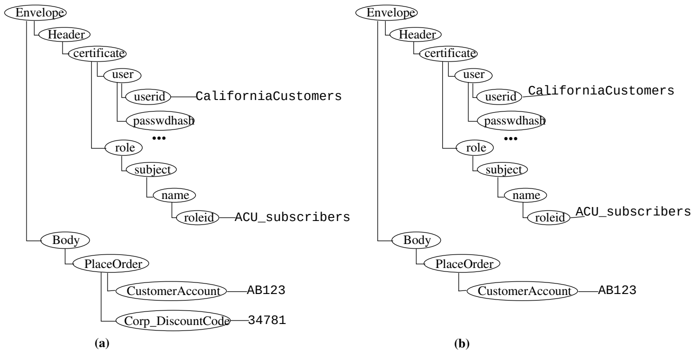

Figure 9: An example of Request Filtering

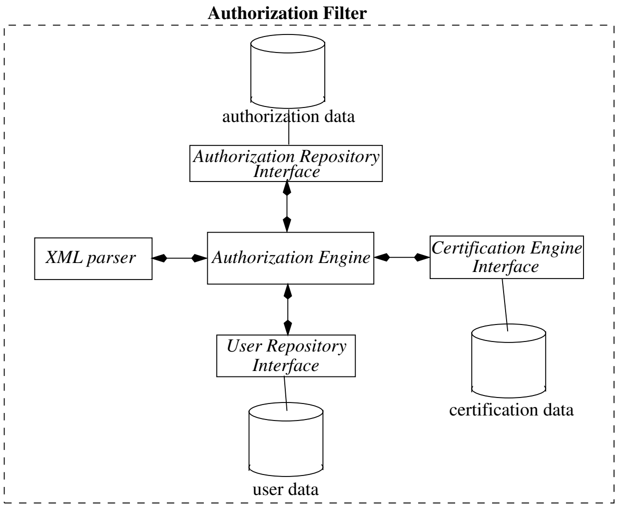

Figure 10: Architecture of the Authorization Filter

ACME:Corp_DiscountCode
</object>
<sign value = "−" />

<subject><roleid>ACMEFidelitySubscribers</roleid>
</subject>
<object>
/SOAP-ENV:Envelope/SOAP-ENV:Body/ACME:PlaceOrder/
ACME:Corp_DiscountCode
</object>
<sign value = "+" />

Note that the negative authorization specified for ACU_subscribers will cause, for requests lacking the fidelity certificate, the pruning of the `Corp_DiscountCode` element. As an example, Figure 9 illustrates the original (a) and pruned (b) XML trees of a SOAP request presented by a ACU subscribers not holding a valid fidelity certificate (i.e., either the certificate is missing or could not be validated).

5. DESIGN AND IMPLEMENTATION

The authorization filter discussed in this paper is currently under development. In this section we discuss the main features of our design. Overall, the architecture of a SOAP system consists of a client, a SOAP gateway and a communication channel. The client produces the SOAP request and receives a SOAP response; the SOAP gateway translates the SOAP request to a call to a local or remote server; the answer of the server is then translated back to the format of the SOAP response; the communication between the client and the gateway uses the HTTP protocol. In this architecture, shown in Figure 5, we introduce authorization services with an Authorization Filter.

5.1 Architecture of the Authorization Filter

The Authorization Filter is the core of the system. It is located on the communication channel between the client and the SOAP gateway and analyzes all the requests. Its internal architecture, illustrated in Figure 10 comprises a *User Repository* that describes the users, groups and roles on which authorizations are defined, an *Authorization Repository* that describes all the privileges that are granted to users/groups/roles, a *Certification Engine* that evaluates the correctness of the certificates provided with the request, and an *Authorization Engine* that applies the model we presented to a given request instance to determine if the request must be restricted by the filter or can pass unaltered through it. The conversion from the textual XML payload to an internal representation is the responsibility of an *XML parser*. Depending on the degree of integration, the request can be forwarded to the SOAP gateway after it has been again serialized by the XML parser, or it can be directly passed in an internal format.

5.1.1 User Repository

The *User Repository* maintains the description of the users, groups, and roles that have been defined on the system. This information is stored persistently in XML using a simple DTD. The User Repository offers an interface that permits to access the properties of every component (like the hashed password, required for user authentication), and the service it most typically offers permits to determine if a given authorization is applicable to a certain user instance, characterized by a userid, an IP address, possibly a symbolic name, and a list of certified roles. This evaluation may need to use service *isSubjectMoreSpecific* to enforce the priority policy illustrated in Section 4.4.

There are two main implementation strategies for the User Repository: 1) it can be designed as a system that requires initialization, possibly running in a separate thread; 2) it can be a stateless set of services available to the other components. Our current implementation choice opted for the first solution, which although more difficult to manage and requiring a relative expensive initialization, also offers better performance. As a matter of fact it analyzes the content of the stored repository only at startup and it can also use optimized data structures that permit to offer a low service time. The second solution instead would require to load the content of the Repository every time a request arrives.

5.1.2 Authorization Repository

The *Authorization Repository* maintains the collection of authorizations that describe the security policy. The authorizations are persistently stored in XML format. The services offered by the Authorization Repository permit the retrieval of the authorizations applicable to a given request. A problem to be considered in this context is the organization to use for the data. A trivial solution consists in keeping all the authorizations in a single document, as a flat list. We selected a more sophisticated solution that stores the authorizations of each interface (characterized by the URI of the HTTP `POST` action) in separate documents. We also plan to investigate authorization indexing techniques, organizing the authorizations depending on their content, e.g., object (i.e., ACLs) or subject (i.e., capabilities).

5.1.3 Certification Engine

The goal of the *Certification Engine* is to evaluate the correctness of the certificates that a subject may present to substantiate his ability to play a certain role. We assume that this service can be realized by the integration into our architecture of existing solutions [21] to the management of certificates, some of which are currently available under the Java 2 platform. Such components support the integration of our service into a public-key infrastructure [20], realize sophisticated protocols for the exchange of a set of messages, and manage challenge-response authentications.

5.1.4 Authorization Engine

The *Authorization Engine* is the main component of the Filter and it coordinates the use of the other subsystems. It reacts to the receipt of a request and parses its `<Header>` element to determine the subject. If the subject presents a userid and a password, it asks to the User Repository the hashed password of the user; if the comparison is successful, the user is authenticated. If the subject presents credentials, each one of them is verified with the Certification Engine. For every verified certificate, the corresponding role is associated with the subject. Then, the Authorization Engine retrieves all the authorizations whose object is in the request and for each of them asks the User Repository to verify if its subject corresponds to the actual subject producing the request. For all the applicable authorizations, the Authorization Engine produces a labeling of the internal DOM representation of the SOAP request. Finally, the request that remains after the removal of the nodes with a negative label is passed to the SOAP gateway.

5.1.5 XML parser

The XML parser is the component responsible of the conversion from the textual representation of XML to its equivalent memory description. A technical aspect that should be considered is the type of parser to use. Building on our previous experience,[5] our design exploits the DOM representation of the SOAP call. The reason for using the DOM representation is that, in our model, authorization objects are defined via generic expression of the XPath language, and the evaluation of these expressions may require to navigate the XML structure in arbitrary ways.

We note however that the construction of the DOM tree involves building a complete memory representation of the XML information and in some contexts (where the XML data require a considerable area of memory) this may become a bottleneck on the workload of the authorization system.

The alternative to the construction of a DOM representation is the use of a SAX parser, which analyzes the textual representation of the XML information and produces events for every component of the structure which emerges from the analysis. The SAX parser is indeed the mechanism upon which the construction of the DOM representation is based, and it is an efficient mechanism for the analysis of the XML content. The Authorization Filter can use a SAX parser to analyze an incoming stream of data, triggering *events* representing only the nodes that are authorized to be part of the request. This mechanism stores in memory only the status of the parser, consisting of the path connecting the current node with the root. In order to be evaluable in this context, the path expressions appearing as authorization object must satisfy a set of restrictions. The main restriction is that the path expression should be restricted to descending terms and to conditions at the local level. This approach also benefits from a careful preprocessing of the authorizations.

The use of a SAX parser is particularly relevant for future management of authorizations on the response, which can often require the retrieval of a large amount of XML information; instead, it can be considered as less critical for the management of SOAP requests, where typically the size of the message is limited. Also, the SOAP gateway, for all the open source prototypes that we analyzed, manages the requests creating a DOM representation. If the Authorization Filter is realized with a strict integration with the SOAP gateway, passing to it directly the DOM representation of the request, the cost of the DOM conversion can be factored out.

5.2 Impact on current SOAP components

The realization of the authorization services has a limited impact on the current components of the SOAP infrastructure. Indeed, the SOAP gateway does not have to be modified in a significant way because of the presence of the Authorization Filter. A solution, which strictly integrates the Authorization Filter within the SOAP gateway, simply delegates to the Authorization Filter the task of acquiring the request, in place of the DOM parser. The Filter parses the request and returns it to the gateway after the authorizations have been applied. A solution with no modification to the SOAP gateway can also be realized, where the Authorization Filter is separated from the SOAP gateway and communicates with it using the HTTP protocol.

The client must be enriched to submit a SOAP request with the identification of the user and its certificates. The addition of a header containing the userid and the hashed password requires a trivial extension of the client. Moreover, clients unware of the access control facility can easily be handled by adding a default header to their calls. The management of certificates instead requires a more complex service, for which it is convenient to reuse already available solutions, for reasons analogous to those presented for the implementation of the Certification Engine.

Finally, the communication channel needs to be secured. This is the only constraint that we set, but the satisfaction of this requirement is also relatively easy, as there are available many robust implementation of secure transport protocols, like SSL, that can be easily integrated into the components of the architecture.

6. CONCLUSIONS

SOAP is a clean and elegant solution to the problems raised by the Internet use of RMC-based protocols. However, lack of standardization could lead to application-dependent security holes in the enforcement of access control

[5] We have already developed a related authorization system for selective access to XML data (http://131.175.16.43:8080/XML-AC).

policies for SOAP. In this paper we have presented a general fine-grained authorization model for controlling SOAP requests and sketched the architecture of the system implementing this approach. The approach provides flexibility as it is able to support a variety of protection requirements concisely. We are currently addressing some specific features of the SOAP standard, such as the exception responses that may be generated by the SOAP gateway. We plan to exploit this characteristic in order to produce a `<FAULT>` header with a description of the security violation detected by the Authorization Filter, offering to the client an explanation of the reason why a request was not fulfilled as it was expected. In our design, this mechanism to notify authorization violations is a configuration parameter of the system.

7. ACKNOWLEDGMENTS

This work was supported in part by the European Community within the FASTER Project in the Fifth (EC) Framework Programme under contract IST-1999-11791 and by the Italian MURST within the DATA-X project.

8. REFERENCES

[1] Java Remote Method Invocation (RMI). http://java.sun.com/j2se/1.3/docs/guide/rmi/index.html.

[2] XML Metadata Interchange (XMI) specification. http://www.omg.org/cgi-bin/doc?ad/98-10-05.

[3] XML-RPC Home Page. http://www.xmlrpc.com/.

[4] The Common Object Request Broker: Architecture and Specification, Revision 2.1, August 1997. ftp://ftp.omg.org/pub/docs/formal/97-09-01.pdf.

[5] P. Bonatti and P. Samarati. Regulating Service Access and Information Release on the Web. In *Proc. of the 7th ACM Conference on Computer and Communication Security*, Athens, Greece, November 2000.

[6] D. Box. *Simple Object Access Protocol (SOAP) 1.1.* World Wide Web Consortium (W3C), May 2000. http://www.w3.org/TR/SOAP.

[7] N. Brown and C. Kindel. Distributed Component Object Model Protocol – DCOM/1.0, January 1998. http://www.globecom.net/ietf/draft/draft-brown-dcom-v1-spec-03.html.

[8] D. Burdett. Internet Open Trading Protocol - IOTP, Version 1.0, April 2000. http://www.landfield.com/rfcs/rfc2801.html.

[9] Microsoft Corp. Remote Data Service: A Web Data Access Feature, 2000. http://www.microsoft.com/data/ado/rds.

[10] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing XML Documents. In *Proc. of the 2000 International Conference on Extending Database Technology (EDBT2000)*, Konstanz, Germany, March 2000.

[11] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. XML Access Control Systems: A Component-Based Approach. In *Fourteenth Annual IFIP WG 11.3 Working Conference on Database Security*, Schoorl, The Netherlands, August 2000.

[12] S. Feldman. The Changing Face of E-Commerce. *IEEE Internet Computing*, 4(3):82–84.

[13] A.O. Freier, P. Karlton, and P.C. Kocher. The SSL Protocol - Version 3.0, March 1996. http://ftp.nectec.or.th/CIE/Topics/ssl-draft/INDEX.HTM.

[14] B. Gladman, C. Ellison, and N. Bohm. Digital Signatures, Certificates and Electronic Commerce. http://jya.com/bg/digsig.pdf, 1999.

[15] S. Jajodia, P. Samarati, V.S. Subramanian, and E. Bertino. A Unified Framework for Enforcing Multiple Access Control Policies. In *Proc. of the 1997 ACM Internationa SIGMOD Conference on Management of Data*, Tucson, AZ, May 1997.

[16] J. Kahan. WDAI: A Simple World-Wide Web Distributed Authorization Infrastructure. *Computer Networks*, 33(1-6), 2000.

[17] M. Kudo and S. Hada. XML Document Security and e-Business applications. In *Proc. of the 7th ACM Conference on Computer and Communication Security*, Athens, Greece, November 2000.

[18] M. Levy. COM Internet Services, April 1999. http://msdn.microsoft.com/library/backgrnd/html/CIS.htm.

[19] S. Lewontin and M.E. Zurko. The DCE Project: Providing Authorizations and other Distributed Services to the World-Wide Web. In *Proc. of the 2nd World Wide Web Conference*, October 1994. http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Security/lewontin/Web_DCE_Conf_94.html.

[20] U. Maurer. Modeling a Public Key Infrastructure, booktitle = Proc. of the Fourth European Symposium on Research in Security and Privacy. volume LNCS 1146, pages 325–350, Rome, Italy, September 1996.

[21] P. Nikander and A. Karila. A Java Beans Component Architecture for Cryptographic Protocols. In *Proc. of the 7th Usenix Security Symposium*, San Antonio, Texas, January 1998. http://www.tml.hut.fi/Research/TeSSA/Papers/Nikander-Karila/nikander-karila-98.html.

[22] W3C Note. The information content exchange protocol. http://www.w3.org/TR/note-ICE.

[23] Object Management Group. *The CORBA Security Service Specification.* ftp://ftp.omg.org/pub/docs/ptc.

[24] J. Paajarvi. *XML Encoding of SPKI Certificates.* Internet Draft.

[25] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST Model for Role-Based Access Control: Towards A Unified Standard. In *Proc. of 5th ACM Workshop on Role-Based Access Control*, Technical University of Berlin, Berlin, Germany, July 2000.

[26] R. Sandhu and P. Samarati. Authentication, Access Control and Intrusion Detection. In A. Tucker, editor, *Database Security VII: Status and Prospects*, pages 1929–1948. CRC Press Inc., 1997.

[27] E.J. Whitehead. World Wide Web Distributed Authoring and Versioning (WebDAV): An Introduction. *ACM StandardView*, 5(1):3–8, 1997.

[28] World Wide Web Consortium (W3C). *XML Path Language (XPath) Version 1.0*, November 1999. http://www.w3.org/TR/xpath.