Authorization Service for Web Services and Its Application in a Health Care Domain

Sarath Indrakanti, Macquarie University, Australia
Vijay Varadharajan, Macquarie University, Australia
Michael Hitchens, Macquarie University, Australia

ABSTRACT

In this paper, we discuss the design issues for an authorization framework for Web Services. In particular, we describe the features required for an authorization policy language for Web Services. We briefly introduce the authorization service provided by Microsoft .NET MyServices and describe our extended authorization model that proposes extensions to the .NET MyServices authorization service to support a range of authorization policies required in commercial systems. We discuss the application of the extended authorization model to a health care system built using Web Services. We use the XML Access Control Language (XACL) in our implementation to demonstrate our extended authorization model. This also enables us to evaluate the range of authorization policies that XACL supports.

Keywords: *access control; authorization; policy language; security; Web Services*

INTRODUCTION

Security plays a vital role in the design and practical deployment of distributed applications, as greater availability and access to information, in turn, imply that systems are more prone to attacks. In a networked computing system, when one principal requests a service from another, the receiving principal needs to address at least two fundamental questions: Is the requesting principal the one it claims to be? Does the requesting principal have appropriate privileges for the requested service? These two questions relate to the issues of authentication and authorization**.** The authorization requirements in networked applications are much richer than authentication both in terms of the types of privileges required and the nature and degree of interactions. There also are other security concerns, such as auditing, secure communication (confidentiality and integrity), availability, and accountability. All of these security issues apply in general to the Web Services architecture.

In general, security for Web Services is a broad and complex area covering a range of technologies. At present, there are several efforts underway that are striving for the provision of security services, such as authentication between participating entities, confidentiality, and integrity of communications. A variety of existing technologies can contribute to this area, such as TLS/SSL (Rescorla, 2001) and IPSec (Kent & Atkinson, 1998). We also have security functionality based on XML Signature (Bartel et al., 2002) and XML Encryption (Imamura et al., 2002) standards. There also are natural extensions of these to integrate security features in technologies such as SOAP (Gudgin et al., 2003) and WSDL (Christensen et al., 2001). WS-Security (Atkinson et al., 2002) specification describes enhancements to SOAP messaging to provide message integrity, confidentiality, and authentication. There also is work on XKMS (Ash et al., 2004) defining interfaces to key management and trust services based on SOAP and WSDL. There is a draft proposal on Web Services security policy language (Della-Libera et al., 2002) that describes the capabilities and constraints of the security policies on intermediary Web Services and end points. Web Services trust draft specification (Anderson et al., 2005) considers secure interoperation of Web Services by outlining a framework of trust models. There also is some work on Web Services privacy issues (O'Keefe & Greenfield, 2003). However, while there is a large amount of work on general access control and more recently on distributed authorization (Herzberg et al., 2000; Jajodia et al., 1997; Varadharajan et al., 1998), research in the area of authorization for Web Services is still at an early stage (Kraft, 2002; Yagüe & Troya, 2002). There is not yet a specification or a standard for Web Services authorization. Currently, most Web-service-based applications, having gone through the authentication process, make authorization decisions using application-specific access control functions, which result in the practice of frequently reinventing the wheel. This motivates us to have a closer look at the issues involved in designing an authorization framework for Web Services.

DESIGN CHOICES FOR AUTHORIZATION

In some sense, there is a lot of similarities in the design of authorization services and frameworks for distributed applications and Web Services. Perhaps one aspect that makes it somewhat different in the case of Web Services is the need to take into account multiple domains and jurisdictions in a federated structure as a default. Although this is true in the case of distributed systems, as well, in the past the approach has been one of developing the service on a per-domain basis and then extrapolating it to multiple domains.

In designing a distributed authorization service for Web Services, there are some fundamental design issues that we need to address. First, we need to consider the types of authorization information to be used in the decision-making process. These range from static and generic information (e.g., identities) to specific information (e.g., roles) to dynamic and specific information, which are dependent on system state. Then, we need to address the class of authorization policies that need to be supported in the Web Services architecture. These can range from identity-based to role-based to delegation to joint actions to dynamic separation of duty. Associated with the different types of information, there can be different places at which checks need to be performed by different authorities. This, in turn, will lead to what authorization information (e.g., privileges or credentials) can be pushed to the decision-making authority and what information needs to be pulled, or a combination of both. Related to these is the replication strategy and who updates and manages what information and policies. Different design choices to these questions lead to different authorization frameworks that, in turn, need to be positioned within the distributed Web Services architecture. A detailed discussion of these issues can be found in Varadharajan (2002). In this paper, we address the class of authorization policies that needs to be supported by an authorization policy language used by the Web Services architecture.

In this paper, we discuss the features required for an authorization policy language designed for the Web Services architecture. We describe our authorization model (Indrakanti et al., 2003), which extends the authorization service used by .NET MyServices. Note that .NET MyServices is used as an example system for illustration purposes. The authorization techniques proposed in this paper apply in general to Web Services environments. In particular, we extend the authorization policy language to support a range of access control policies required in a commercial environment. We then discuss the demonstration of our extended authorization model to an application in the health care domain. We made use of the XML Access Control Language (XACL) (Kudo & Hada, 2000) policy language in our implementation in order to demonstrate our extended authorization model. We describe our system architecture, explaining our design, the policy specifications and associations, and their practical evaluation. We then discuss our experience with XACL. In particular, we evaluate the range of access control policies supported by XACL. We propose extensions to XACL to support a range of access control policies required in commercial systems. We discuss some related work in the area of authorization policy languages based on XML before concluding the paper with some final remarks.

AUTHORIZATION POLICY LANGUAGE FEATURES

Languages long have been recognized in computing as ideal vehicles for dealing with expression and structuring of complex and dynamic relationships. Over recent years, a language-based approach to specifying access control policies has (rightly) gained prominence, which is helpful not only for supporting a range of access control policies but also for separating policy representation from policy enforcement.

A standard access control policy language can replace several application-specific languages and is useful for interoperability between different systems and applications. Administrators save time and money, because they are not required to rewrite their policies in many different languages. Developers save time and money, because they do not have to invent new policy languages and write code to support them. They may reuse existing code. Good tools for writing and managing policies for a policy language are likely to emerge, if the policy language is standardized. We believe policy languages in which one can specify policies using XML have an advantage over other policy languages such as Ponder (Damianou et al., 2001) and Tower (Hitchens & Varadharajan, 2000). Web-services-based applications using an XML-based language can leverage on the benefits of XML like interoperability over multiple platforms in a heterogeneous environment. A policy based on XML technology with its own namespaces and schemas is necessary in a heterogeneous environment of Web Services. Standard specifications, such as XML Encryption and XML Signature, then can be used to secure and sign those policies, where needed. Such a policy language's policies can be specified and associated with any service-based application, whether it is running on a Java-based platform or the .NET platform.

We also need to consider the range of authorization policies that likely are to be required for Web Services deployed in a commercial environment. Support for a range of policies, including dynamic separation of duty policies (Simon & Zurko, 1997), Role Based Access Control (RBAC) (Sandhu et al., 1996) policies, conditional authorization policies (discussed in this paper), joint-action based policies (Varadharajan & Allen, 1996), and Chinese-wall policies (Brewer & Nash, 1989), must be provided by the policy language.

EXTENDED AUTHORIZATION MODEL

In this section, we briefly introduce .NET MyServices, its authorization model, and our evaluation of the model, along with our proposed extensions.

Motivation for Evaluation. We evaluated the policy language used by the authorization service for .NET MyServices for the following reasons:

Figure 1. Example of role-map

```
<roleMap>
    <!-- List of scopes are defined here –>
    <scope id="1"> <!– a portion of an XML document –>
      <name xml:lang="en">allElements</name>
      <shape base="t"/>  <!-- "t" implies full document –>
    </scope>
    <!-- List of role-templates are defined here -->
    <!-- Role-template gives access to methods on a scope –>
    <roleTemplate name="rt0">
      <method name="query" scopeRef="1"/>
      <method name="insert" scopeRef="1"/>
      <method name="replace" scopeRef="1"/>
      <method name="delete" scopeRef="1"/>
      <method name="update" scopeRef="1"/>
    </roleTemplate>
</roleMap>
```

1. It is an XML-based language;
2. It supports Role Based Access Control (RBAC) model for access control. The privileges of an organization can be modeled effectively using RBAC. Advantages of the RBAC model are widely understood; and
3. Unlike other policy languages (Damianou et al., 2001) designed for distributed systems, the policy language used by .NET MyServices was designed specifically for Web Services.

Microsoft .NET MyServices

Microsoft .NET MyServices (Microsoft Corporation, 2001b) is the name for a set of XML message interfaces implemented as a set of Web Services. It is a platform for building user-centric Internet applications, where users can place their private personal data. .NET Calendar, .NET Inbox, and .NET Contacts are examples of .NET MyServices. They serve as a single central location, where users can store, update, and control all of their personal data. .NET MyServices aims to provide a fine granularity of control to the user (owner of the data). Users may choose to share that information with friends, family, groups with which they have an association, and businesses. Additionally, users can sign up to receive alerts on both desktop and mobile devices.

Authorization Service in .NET My Services

The authorization service used by .NET MyServices uses a policy language comprised of the following items. The definition of a role in terms of the permissions that it has on different objects (a set of elements in an XML document) is specified using a role-template. A role-template defines the set of methods allowed by the role-template and the scope visible to each method. A role-template has a name (e.g., rt1). A scope is nothing but a group of elements and attributes within an XML document. The mapping between a role-template and a user is defined by role. Then, users have a list of roles that are authorized to access the resources (XML documents) associated with those users. This is defined in role-lis*t*, which describes what information to share, who to share it with, and how to share it. A role-map describes what information is to be shared and how that sharing is to occur. It defines for each role (from the role-list) what the allowable Web Service methods are and what scope of data is visible while using this method. The role-map is the same for all instances of a particular service and is authored by the implementer of the service.

An example of role-map is shown in Figure 1. The role-map is defined for all elements of an XML document or, in other words, for an entire XML document. The role-template rt0 gives query, insert, replace, delete, and up-

Figure 2. Example of role-list

```
<roleList>                                  <!-- for patient Bob -->
    <!-- list of scopes specified here -->
    <scope id="1">                          <!-- scope for writing medical report –>
      <shape base="nil">
        <include select="/medical_record/medical_report/content"]/>
      </shape>
    </scope>
    <scope id="2">
      <shape base="nil">                    <!-- scope for certifying medical report –>
        <include select="/medical_record/medical_report/certify"]/>
      </shape>
    </scope>
    <!-- list of roles specified here -->
    <role roleTemplateRef="rt0">
      <subject userId="Bob's ID" appAndPlatformId="healthcare-app@hospital.org"/>
    </role>
    <role roleTemplateRef="rt0">
      <subject userId="Alice's ID" appAndPlatformId="healthcare-app@hospital.org "/>
    </role>
    <role roleTemplateRef="rt0">
      <subject userId="Joe's ID" appAndPlatformId="healthcare-app@hospital.org"/>
    </role>
</roleList>
```

date privileges on all elements of the XML document.

An example of role-list is given in Figure 2. The role-list defines the privileges for different roles to Bob's medical record. The roles include Bob, Alice, and Joe. All the three roles in this case have privileges given by role-template rt0 on Bob's medical record (stored as an XML document). We refer the reader to Microsoft Corporation (2001a) for XML schemas and detailed descriptions of role-map, role-list, role, and role-template.

Proposed Extensions

A key architectural aspect of any authorization policy language is its ability to support a range of access control policies. After evaluating the authorization model used by .NET MyServices, we discovered that the policy language used does not support the range of access control policies required in a commercial environment. We proposed extensions to the policy language to be able to capture policy requirements, such as identity-based discretionary access control, role-based access control, static and dynamic separation of duty, Chinese-wall policy, delegation, and joint and collaborative action-based access control.

In this paper, we briefly discuss our proposed extensions to the policy language along with examples to specify conditional authorization policies, delegation policies, joint-action policies, and dynamic separation of duty policies. We give some examples of the kinds of policies one can specify using the extended authorization model in a health care application scenario. In each case, we first will describe the access policy requirements to be supported by the Web Service, discuss the existing limitations, and propose extensions to the policy specification mechanism provided by the authorization service. In particular, we provide extensions to the XML schemas of role-map and role-list.

In describing our extensions to the authorization service, we will use the term *provider* to denote an entity that owns the content document (or Web Service data) and the term *user* to denote an entity that accesses the provider's document. For a more comprehensive discussion on this work, we refer the reader to Indrakanti et al. (2003).

Throughout this paper, when we use the element *role* in extended role-list schema, we mean that it is either a normal role, a delegated role, or a dynamic role. Delegated-role and dy-

Figure 3. Role-map example with minor extension

```
<roleMap>
    <scope … />
    <roleTemplate name="rt0">
      <method name="query" scopeRef="1" id = "1"/>
      <method name="insert" scopeRef="1" id = "2"/>
      <method name="update" scopeRef="1" id = "3"/>
      <method name="replace" scopeRef="1 id = "4""/>
      <method name="delete" scopeRef="1 id = "5""/>
    </roleTemplate>
</roleMap>
```

namic-role are defined in the sections to follow. For simplicity, we only show the schema skeleton for role element.

Every .Net MyServices provider has a content document that actually stores its data and an access control list (ACL) document that stores authorization information. We suggest that every provider provisioned for a .NET My Service also should be provisioned a log document along with the content document and ACL (role-list) for auditing purposes. We will not be discussing the auditing aspects in this paper.

To begin with, we propose a small modification to the role-map schema that helps to uniquely identify a method-scope pair in the role-template. This is achieved by including an ID attribute. This new attribute is introduced so that this pair can be referred to in a role element in a provider's role-list document. An example with this minor extension given in bold is shown in Figure 3.

Conditional Authorization

The existing authorization service in .NET MyServices does not have the ability to specify conditions to methods when performing access control checks. Each role-template provides access to a number of methods on various scopes to a user. To further restrict access on each of those methods based on arbitrary conditions, we introduce a new element: *methodRef*. This construct enables us to associate conditions and actions with a method. A method can have a condition and one or more actions associated with it. The modified schema skeleton for a role-list document is shown in Figure 4. The elements and attributes introduced to role-list to support conditions and actions based policies are described.

/roleList/role/methodRef element refers to the ID of the method for this role in its role-template. Multiple method IDs can be referenced here (e.g., methodRef = "1,2"). In this case, the conditions and actions are performed for the methods with IDs 1 and 2. */methodRef/@idRef* attribute gives the reference to the method ID in the role template referred to by this role.

/roleList/role/methodRef/condition element is a Boolean formula. If the formula evaluates to true, then the access is granted. In general, the condition expression can involve several predicates involving standard logic operators, such as *and*, *or*, and *not*. Defining a condition element within another condition element can specify recursion. That is, a condition also can be a predicate. Recursion is useful to specify policies such as (A OR B) AND (C OR D), where A, B, C, and D are different conditions to be evaluated before authorizing access to a method. Each predicate may have one or more parameters.

/roleList/role/methodRef/action specifies actions that need to be performed when an access request is made to the method(s) through methodRef. Each action has an optional ID attribute. The type attribute is used to define whether the action only is performed when the access is granted (on_access) or always is performed whether or not access is granted (always). The predicates in action expressions are similar to those defined in conditions, except that there should be at least one predicate element.

An example conditional authorization policy is shown in Figure 5. In the example, a patient (Bob) can access only his own patient

Figure 4. Modified schema skeleton for role-list to support conditional authorization

```
<roleList ... >1..1
        <scope ... />0..unbounded
        <role ... >0..unbounded
          ...
          <subject ... />1..1
          <methodRef idRef = "... " >0..unbounded
              <condition operation = "... ">0..1
                  {any}
                  <predicate name = "... ">0..unbounded
                      <parameter name= "... " value = "... ">0..unbounded
                          {any}
                      </parameter>
                  </predicate>
                  <condition operation = "... " ... />0..unbounded
              </condition>
              <action id = "... " type = "... ">0..unbounded
                   <predicate name = "... ">0..unbounded
                       <parameter name= "... " value = "... ">0..unbounded
                           {any}
                       </parameter>
                   </predicate>
               </action>
          </methodRef>
          <expiresAt ... />0..1
          {any}
        </role>
</roleList>
```

Figure 5. Example of conditional authorization policy

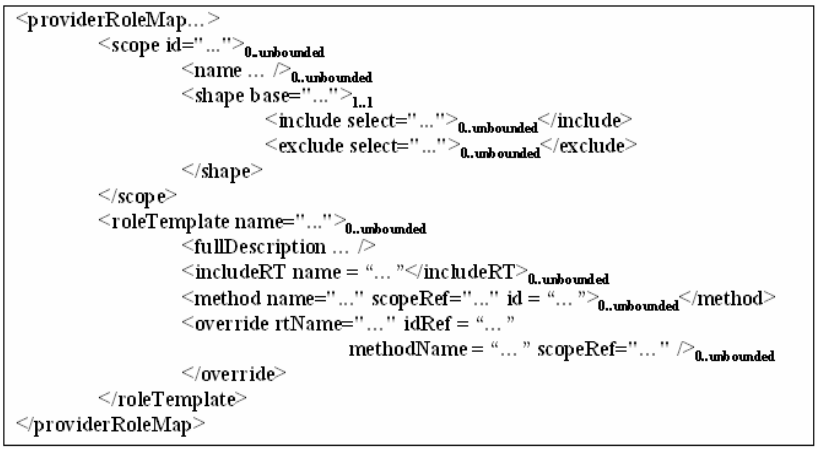
```
<roleList>
        <role roleTemplateRef="rt0
                <subject userId= "ID of Bob" appAndPlatformId=
                                "healthcare-app@hospital.org"/>
                <methodRef id = "1, 2, 3, 4, 5">
                        <condition operation = "and">
                                <predicate name = "checkUserID"/>
                                <predicate name = "checkUserLocation"/>
                        </condition>
                        <action type = "always">
                                <predicate name = "log" />
                        </action>
                </method>
        </role>
</roleList>
```

record, if he is located within the hospital. We do this by using a predicate *checkUserID* that matches Bob's identity (ID) (after authentication) with the ID in his record. The predicate *checkUserLocation* is used to find the current location of Bob. Both the predicates used do not take any parameters as input. In the example policy, the condition is evaluated, and the action is always performed, whether access is granted or not. That is, each time an access to a patient record is requested, the action is logged for auditing purposes.

Role Hierarchy

One of the advantages of the role-based approach is that it can model the privileges in an organization more effectively. A simplistic view of an organization structure is a hierarchical ordering of responsibilities, with the senior positions encompassing all the privileges of the junior positions with some extra privileges. Such a role hierarchy can be achieved by providing hierarchy in role-template definition. At present, most of the services in .NET MyServices only

Figure 6. Schema skeleton for provider-role-map



use about three to five role templates. Customized roles and role hierarchy policies cannot be specified using the authorization policy language. We propose the following extensions to .NET MyServices authorization policy language in order to enable role-based access control and role hierarchy. Each service should be able to associate itself with a dynamic role-map at the user level. The role-map must be static at the Web Service level, but the provider should be able to change it, when needed. Therefore, every provider will now have his or her role-map along with the Web Service role-map. We name the provider's role-map as the provider-role-map (XML document name = providerRoleMap). New scopes now can be defined in this provider-role-map instead of in the provider's role-list. We also remove all scope definitions from the role-list for a provider and move them to the provider-role-map for consistency. The role-template names in provider-role-map must be anything other than rT0, rT1, rT2, rT3, or rT99, as these are the default role-template names. Every method in the default Web Service role-map and provider role-map now must have an ID number attribute to uniquely identify the method.

When a provider is provisioned to a service, the provider has his or her own provider role-map document along with the content, system, log, and acl (role-list) documents. Our definition of XML schema skeleton for providerRoleMap is shown in Figure 6.

The schemas for scope, role-template, and their child elements and attributes are the same as those defined for the Web Service role-map. Other elements and attributes introduced to specify role hierarchy policies are described as follows.

providerRoleMap/roleTemplate/includeRT element is introduced to specify hierarchy. A role-template can inherit another role-template's privileges with this element. */includeRT/@name* attribute is introduced to specify the name of the parent role-template.

providerRoleMap/roleTemplate/override element is provided to override either a scope or method name for a role-template's method. */override/@rtName* attribute refers to the role-template (inherited) in which the overriding is to be done. If the mentioned role-template is not inherited with an includeRT element, override element is ignored. */override/@idRef* attribute is provided to refer to the method element (using its ID) from the inherited role-template that is to be overridden. */override/@methodName* is an optional attribute. It should be used only when the method in the inherited role-template is to be overridden. */override/@scopeRef* is an optional attribute. It should be used only when the scope in the inherited role-template is to be overridden.

Figure 7. Example role hierarchy policy

```
<roleTemplate name="rt10">
        <method name="query" scopeRef ="1" id = "1"/>
        <method name="insert" scopeRef="3" id = "2"/>
        <method name="replace" scopeRef="3" id = "3"/>
        <method name="delete" scopeRef="3" id = "4"/>
</roleTemplate>
<roleTemplate name="rt20">
        <method name="query" scopeRef ="5" id = "1"/>
        <method name="insert" scopeRef="5" id = "2"/>
</roleTemplate>
<roleTemplate name="rt30">
        <includeRT name = "rt10">
        <includeRT name = "rt20">
        <override rtName="rt10" idRef= "4" methodName = "create" />
        <override rtName = "rt20" idRef = "2" scopeRef ="2" />
        <method name="query" scopeRef="4" id = "1"/>
        <method name="insert" scopeRef="4" id = "2"/>
        <method name="replace" scopeRef="4" id = "3"/>
</roleTemplate>
```

Note that the role-templates inherited either can be the default role-templates from the Web Service role-map or from the provider-role-map. To achieve role hierarchy, a suitable new role-template (with appropriate hierarchy definition) in the provider role-map can be created. Then, the role-template can be associated with suitable role element(s) in the provider's role-list.

For illustration purposes, a few scenarios are shown in Figure 7. Role-templates rt10, rt20, and rt30 are defined in the example. Role-template rt30 inherits privileges from both rt10 and rt20. Method with ID "4" from rt10 is overridden in rt30 to create. Scope for method with ID 2 (insert) from rt20 is overridden to 2 in rt30.

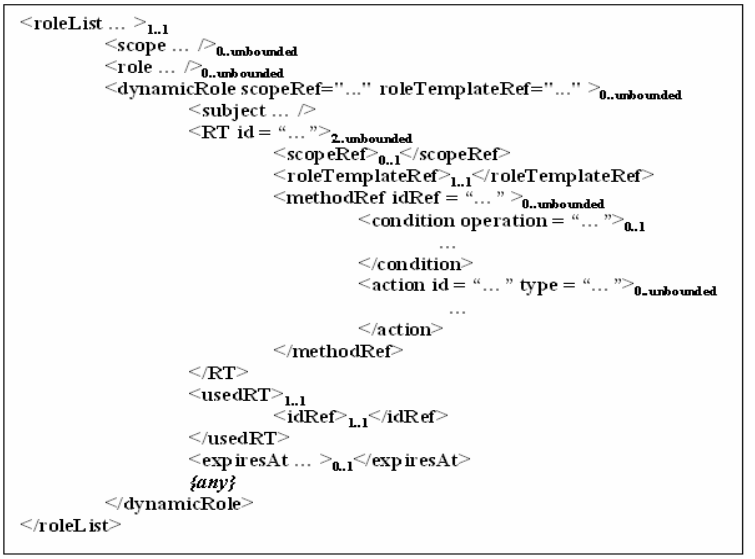
Dynamic Separation of Duty

In the case of dynamic separation of duty, a user having chosen a specific action or role at runtime is not authorized to perform another action or assume another role. To begin with, the user is allowed to choose either of the actions or assume either of the roles. A user may have two privileges (p1 and p2) but cannot have both on the same object. Compliance with these requirements only can be determined at runtime. We introduce a new element called *dynamicRole* to the role-list's schema to support specification of dynamic separation of duty policies. The modified schema skeleton for role-list document is shown in Figure 8. Extensions to the role-list schema are described in subsequent paragraphs.

/roleList/dynamicRole element is introduced to support dynamic separation of duty policies. Most of the child elements and attributes for this element are exactly the same as defined for the role element. Only the new elements and attributes are described here. */dynamicRole/RT* — a minimum of two (or more) RT elements — must be used in each dynamic-role. RT encapsulates both role-template and the scope on which it is used, along with the conditions and actions on the methods. The user with a dynamic-role has access to any of the role-templates and on any of the scopes associated with a dynamic-role. The role-template scope pairs are mutually exclusive. Once the user accesses a method in a role-template on a scope (privilege provided by RT), he or she is not allowed to access any other privileges provided by his or her other RT elements. */dynamicRole/RT/scopeRef* stores the scope to which the user has access. */dynamicRole/RT/roleTemplateRef* stores reference to the role-templates IDs to which the user has access.

/roleList/dynamicRole/usedRT/ is introduced to store information to enable dynamic separation of duty. */usedRT/idRef* is defined to store the reference (ID) to one of the RT ele-

Figure 8. Modified schema skeleton for role-list to support dynamic separation of duty policy



ments (in this dynamic-role). Until the first time a user accesses any of the privileges in his or her set of RT elements, usedRT is null. Once the user accesses privileges provided by any of the RT elements, the ID of that RT element is written here by the authorization service. Every method in an RT element uses the *checkAccess* predicate. This predicate implements the logic to enforce the dynamic separation of duty policy using information in usedRT.

In the example of dynamic separation of duty policy shown in Figure 9, a doctor (Alice) both can write a medical report after a medical procedure and certify it but cannot write and certify the same medical report. In our example, scope 1 denotes the part (scope) of the patient record document used to write the medical report, and scope 2 denotes the part of the document to certify the medical report. For a patient (Bob), if Alice chooses to write a medical report, the idRef value in usedRT element for Alice in that patient's role-list document is set to 1. In a subsequent access to Bob's record, Alice cannot certify the medical report, as the *checkAccess* predicate uses the usedRT value passed as a parameter and enforces the dynamic separation of duty at runtime.

Delegation

Our extended authorization model supports two forms of delegation policies: Forwardable and Proxiable. Forwardable delegation is used when privileges are being delegated to a client application that acts on behalf of the user. Proxiable delegation is used where delegation is between real-world users. A detailed analysis of delegation in distributed systems is made in Varadharajan, et al. (1991).

Our delegation scheme only allows transfer of privileges less than or equal to privileges owned by the user delegating his or her privileges. If more privileges are delegated, then access to the delegate will be denied at runtime. Whenever delegation of privileges is made, the access details must be logged in to the provider's log document, clearly mentioning who delegated what privileges to whom. We suggest that such dynamic delegation be specified by extending the authentication tickets and protocols in order to carry privileges and authorization information. As such, this needs to be achieved in the Passport (Microsoft Corporation, 1999) protocol used by .NET MyServices.

In the case of static delegation, such as a manager delegating part of his or her privileges to a personal assistant, one can create a del-

Figure 9. Example of dynamic separation of duty policy

```
<roleList>
  <dynamicRole>
        <subject userId="ID of Alice" appAndPlatformId=
                                    "healthcare-app@hospital.org"/>
        <RT id = "1">
              <scopeRef>1</scopeRef>          <!-- scope to write a medical report -->
              <roleTemplateRef>rT0</roleTemplateRef>
              <methodRef id = "1,2,3,4,5">  <!-- for all methods of rT0 -->
                    <condition operation = "and">
                          <predicate name = "checkAccess"/>
                            <parameter>./usedRT</parameter>
                          </predicate>
                    </condition>
                    <action type = "always">
                          <predicate name = "log" />
                    </action>
              </methodRef>
        </RT>
        <RT id = "2">
              <scopeRef>2</scopeRef>      <!-- scope to certify the medical report -->
              <roleTemplateRef>rT0</roleTemplateRef>
              <methodRef id = "1,2,3,4,5">              <!-- for all methods of rT0 -->
                    <condition operation = "and">
                          <predicate name = "checkAccess"/>
                            <parameter>./usedRT</parameter>
                          </predicate>
                    </condition>
                    <action type = "always">
                          <predicate name = "log" />
                    </action>
              </methodRef>
        </RT>
        <usedRT>
              <idRef>1</id_ref>
        </usedRT>
  </dynamicRole>
</roleList>
```

egated-role (delegatedRole element) in the provider's role-list document using our extensions to the policy language. This saves processing overhead, as each time the personal assistant accesses his or her manager's privileges, his or her privileges need not be computed from the authorization data field in the authentication ticket. Hence, a user requesting his or her privileges to be delegated must send the type of delegation information — either *temporary* or *permanent* — along with the request. If it is temporary (i.e., one-time delegation), then at runtime, the privileges are verified with the user's role-template, and access is denied or granted according to the authorization algorithm. If it is permanent, then a delegated-role is created in the provider's role-list document for the delegate. The schema for the delegatedRole element is defined next. Note that for a permanent delegation type request, if the granted privileges are a subset of the sender's privileges, then a new role-template element is created for those privileges in the provider role-map, and the delegated-role refers to it. Modified XML schema skeleton for role-list document to enable static delegation is shown in Figure 10. Schema modifications for a role-list document are described in subsequent paragraphs.

/roleList/delegatedRole element represents a user with delegated privileges. This element is created dynamically when a delegation request comes in. The role-template and scope for this delegated-role are set per the request. Once this is set up, the user with these delegated privileges has similar access privileges to those of a normal user. Other child elements and attributes for this element are similar to a normal role element. They are not redefined here. In */delegatedRole/expiresAt*, unlike in the normal role element, the expiresAt element is mandatory here. If the user delegating his or her privileges does not mention when

Figure 10. Modified schema skeleton for role-list document to support delegation policy

```
<roleList ... >1..1
        ...
        <delegatedRole scopeRef="..." roleTemplateRef="...">0..unbounded
                <subject userId="..." credType="..."
                                        appAndPlatformId="...">1..1
                </subject>
                <methodRef idRef = "... " >0..unbounded ...</methodRef>
                <expiresAt>1..1</hs:expiresAt>
                {any}
        </delegatedRole>
</roleList>
```

Figure 11. An example of delegated-role

```
<providerRoleMap>
        <!-- other role-templates defined here -->
        <roleTemplate name = "rt11"
                <method name = "query", scopeRef = "1" id = "1"/>
        </roleTemplate>
</providerRoleMap>

<roleList>
        <role roleTemplateRef="rt10">
                <subject userId="Alice's ID" appAndPlatformId="puid-
                        of:healthcare-app@hospital.org"/>
        </role>
        <delegatedRole roleTemplateRef="rt11">
                <subject userId="Jane's ID" appAndPlatformId="puid-
                        of:healthcare-app@hospital.org"/>
        </delegatedRole>
</roleList>
```

the delegated-role should expire in the request, then the Web Service default is stored here when the delegated-role is created. Every service must define a default expiry time for a delegated-role. Once this delegated-role expires, then the user with these privileges no longer has access. To regain this access, an explicit delegate request should be made again.

An illustration of the delegation policy is given in Figure 11. In the example, the doctor Alice delegates (proxiable form of delegation) permanently certain privileges (roleTemplate rt10 from Figure 7) that she gives to Jane, her personal assistant. In particular, Alice delegates privilege to query on scope 1. Our extended authorization service automatically generates a delegated-role for Jane, once the delegation request is approved. Also, a new role-template (e.g., rt11) is created in the provider-role-map to reflect the privileges given to Jane. rt11 is shown in Figure 11. The delegated-role for Jane is associated with rt11.

Joint Action-Based Policies

Joint action-based policies (Varadharajan et al., 1996) are used in situations where trust in individuals needs to be dispersed. Often this arises because individuals are trusted according to their expertise, which, in turn, maps the concept of trust to a specific set of actions. In delegation, there is a partial or complete granting of privileges; whereas, in joint actions, users may acquire privileges by working together in tandem, which none possesses in isolation. We provide an example of a joint action-based policy after describing XML schema extensions. A modified XML schema skeleton for provider-role-map is shown in Figure 12 and for role-list in Figure 13.

Figure 12. Modified schema for provider role-map to support joint actions

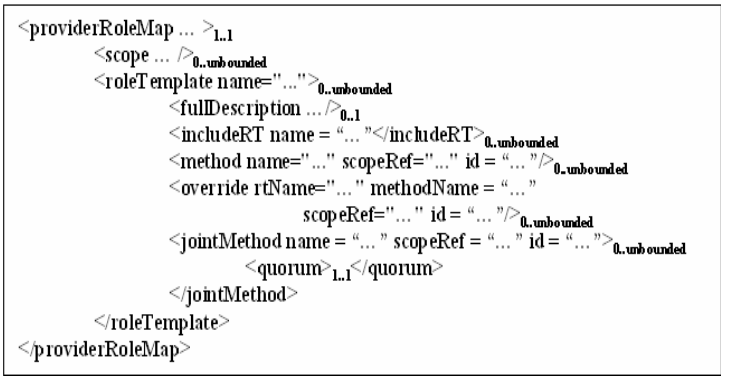
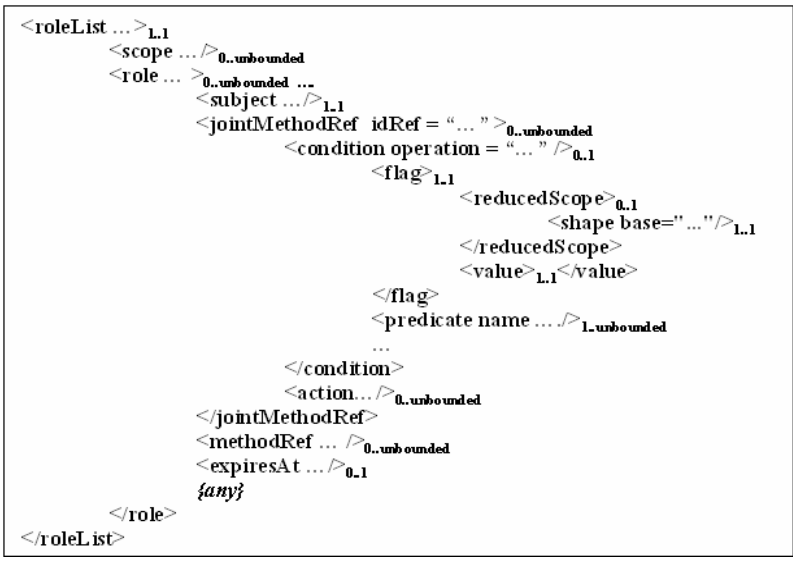


Figure 13. Modified schema for role-list to support joint actions



A description of schema extensions to provider-role-map document follows.

providerRoleMap/roleTemplate/jointMethod element is added to a role-template. It defines a joint method and its quorum number. A joint-method's name, scopeRef, and ID attribute schemas are the same as the schema for a normal method. They are not redefined here. */jointMethod/quorum* element gives the authorization service the quorum number for this joint method. Quorum is the minimum number of times users (who have access) must invoke the joint-method in order for it actually to be committed. A temporary Boolean flag is used to store the invocation information until the quorum is reached.

A description of schema extensions to role-list document follows.

/roleList/role/jointMethodRef: There are as many jointMethodRef elements in a role, as there are joint-methods in the role-template with which it is associating itself. It is similar to a methodRef element, except that it has a Boolean flag element. All other elements in jointMethodRef schema are exactly the same as those defined for methodRef schema. /

jointMethodRef/@idRef attribute refers to the joint method's ID. It is used to uniquely refer to a joint-method.

/roleList/role/jointMethodRef/condition/flag element is a Boolean value that encapsulates information about invocation of a joint-method. */condition/flag/reducedScope* is an optional element. It may so happen that a user should gain authorization only to perform a joint action on a further reduced scope than that permitted by his or her role-template. This scope element solves such a purpose. Shape element schema is the same as the one defined for scope element in role-map. */condition/flag/value* by default is set to false. When the request for the joint-method comes in from the user represented by a role, the flag value in that user's role element is set to true. Until the quorum is reached, only this flag in the participating users' role elements is set to true. Once the quorum is reached, the actual action is committed on the content document of the provider.

A new predicate called "checkQuorum" is defined for all services that need to implement joint action based policy. Every jointMethodRef element in a role element must call this predicate in its condition. checkQuorum predicate does the following:

- When a call to a joint-method is made, the predicate checks the number of flags that are set to true in the roles that have access to this joint-method.
- If the number of flags set to true is more than or equal to the quorum defined in the joint-method, then the actual action is committed or else the flag in the caller's jointMethodRef element is set to true.

An illustration of joint action-based policy is given in Figure 14. In the example, the policy states that two or more doctors are required to jointly certify a patient's medical report after a medical procedure. Role-template rt20 defines joint-methods for write, update, and delete methods. This means that two or more doctors must invoke write method (i.e., certify the medical report) on the medical report in order for it to be committed. The same applies for update and delete methods. However, the doctors can read the medical report at any time, as query is defined as a normal method.

In the example, Alice and Bob are doctors, and both of them certified the medical report. This is indicated by the current status of the flag values. The flag values are currently true. This means that the status of the joint-action, certifying a medical report, is committed. This demonstrates how different users can work together in tandem and perform an action or acquire a privilege, which none possesses in isolation.

Other Policies Supported by the Extended Model

We cannot specify only conditional authorization policies, separation of duty policies, role hierarchy policies, delegation policies, and joint action-based policies, but also other policies, such as the Chinese-wall policy and user and session constraints using our extended authorization policy language. We can specify various constraints on subjects such as mutual exclusion set on roles, cardinality constraints on roles, and users using the extended authorization policy language. For example, a subject cannot be assigned to both Doctor and Patient roles at the same time (in this case, Doctor and Patient roles are mutually exclusive). Another example of a constraint is that the total number of users that are authorized as nurses can be limited to five. Session constraints such as "a role or user only can activate two sessions at any one time" also can be specified. Role activation constraints such as "a subject assigned to both Doctor and Doctor-in-charge roles cannot activate both the roles in a session at any one time" can be specified. We will give an illustration of the Chinese-wall policy in the system implementation section.

Authorization Algorithm

The .NET MyServices authorization algorithm is redefined here to take into account the extensions proposed to the authorization model. We refer the reader to Microsoft Corporation (2001a) for original authorization algorithm for .NET MyServices.

Figure 14. An example for joint-action policy

```
<providerRoleMap>
        <scope id = "1"/>              <!– scope to certify a medical report –>
        <roleTemplate name="rt20">
                <method name="query" scopeRef="1" id = "1"/>
                <jointMethod name="write" scopeRef="1" id = "2">
                        <quorum> 2</quorum> <!-- minimum 2 doctors –>
                </jointMethod>
                <jointMethod name="delete" scopeRef="1" id = "3">
                        <quorum> "2"</quorum>
                </jointMethod>
                <jointMethod name="update" scopeRef="1" id = "4">
                        <quorum> "2"</quorum>
                </jointMethod>
        </roleTemplate>
</providerRoleMap>

<roleList>
        <role roleTemplateRef = "rT20">
                <subject userId = "Alice's ID" <!-- Alice is a doctor -->
                        appAndPlatformId = "healthcare-app@hospital.org">
                <jointMethodRef id = "2" >    <!– write method -->
                        <condition operation = "and">
                                <flag>
                                        <value>true</value>
                                </flag>
                                <predicate name = "checkQuorum"/>
                        </condition>
                </jointMethodRef>
        </role>
        <role roleTemplateRef = "rT20">
                <subject userId = "Bob's ID" <!-- Bob is a doctor -->
                        appAndPlatformId = "healthcare-app@hospital.org">
                <jointMethodRef id = "2" >    <!– write method -->
                        <condition operation = "and">
                                <flag>
                                        <value>true</value>
                                </flag>
                                <predicate name = "checkQuorum"/>
                        </condition>
                </jointMethodRef>
        </role>
</roleList>
```

1. The user, application, and platform identities as well as the credential type are determined from a user request.
2. The matching role from the Web Service provider's role-list document then is located. If a matching role, delegated-role, or dynamic-role is not found, then the user request is not authorized.
3. If a matching role or delegated-role is found, then:

    - If the matching role contains expiry information (in expiresAt element), it is checked to see if the role has expired. If so, the user request is not authorized (expiresAt element use is mandatory for delegated-roles).
    - The role-template (referenced by the role) is located in the respective role-map or provider-role-map. This template is checked to see if the method or joint-method requested is allowed by it. If it is not allowed, then the user request is not authorized.
    - Using the scope from the role-map or provider-role-map (referenced by role-template), combined with the optional scope referenced by the role, the node set that is visible to this message is determined.

4. If a matching dynamic-role is found, then:

    - If the matching dynamic-role contains expiry information, it is checked to see if it has expired.

- If the dynamic-role has not expired, usedRT's idRef value in the dynamic-role element is verified. If the value is null, then:
    - The role-template and scope referenced by first RT element are located from the role-map or provider-role-map. This template is then used to determine whether the method or joint-method requested is allowed. The scope is used to determine if the operation requested is within that scope. If either the method cannot be invoked or the operation is not within the scope, then the role-template and scope referenced by the next RT element are located. It then is determined if the access to the method or joint-method requested is allowed. Similarly all other RT elements in the dynamic-role are verified to determine if the operation requested is within the specified scope. If none of the RT elements provide the requested privileges, then the user request is not authorized.
    - The usedRT's idRef is set to the value of the ID attribute of the RT element that gives the requested privileges.

- If idRef value in usedRT element refers to an RT's ID, and if either the method is not allowed by the role-template in that RT or if the operation requested is not within the scope specified, then the user request is not authorized.
- Using the scope from the role-map or provider role-map referenced by the role-template, combined with the optional scope referenced by the RT in the dynamic-role, the XML node set visible to the user is computed.

5. If the method requested is a normal method, then:
    - All conditions are evaluated, and if any one returns false, then the user request is not authorized.
    - All the actions with type set to *always* are performed.
    - All the actions with type set to *on-access* only are performed, if condition is true.

6. If the method requested is a joint-method, then there exists at least one predicate (checkQuorum) in the condition element. In this case:
    - The condition is evaluated. If it returns false, the user request is not authorized.
    - Otherwise, all the actions with type set to *always* are performed.
    - All the actions with type set to *on-access* only are performed, if condition is true.

The requested operation is then authorized, ensuring that the user has no access to information outside the scope computed in step 3.

IMPLEMENTATION OF EXTENDED AUTHORIZATION MODEL

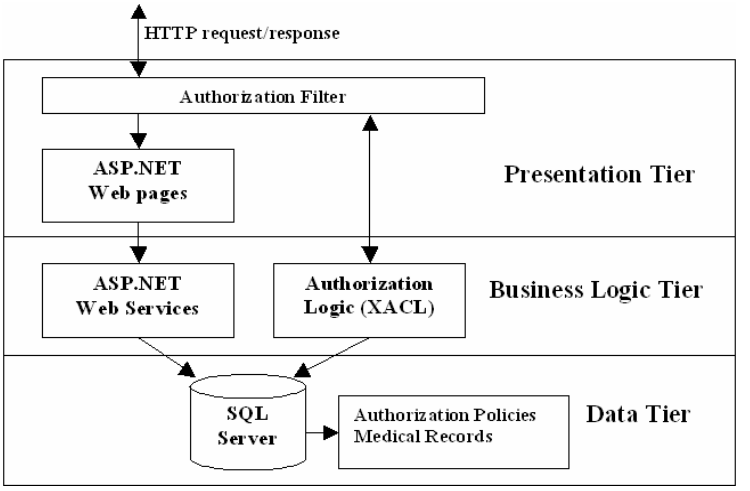
In this section, we first introduce XACL and then describe the system architecture.

Introduction to XACL

Our choice to use XACL access control language in our demonstration to specify authorization policies and its policy evaluation engine to enforce them is based on the following reasons.

1. XACL is an XML-based access control policy language.
2. It provides for an implementation of a processor that provides for provisional authorization (Kudo et al., 2000). This closely matches our requirement for conditional authorization policies.
3. XACL supports RBAC policies, which is an important requirement, as discussed earlier

Figure 15. System architecture: Overview



in the authorization policy language features section.

4. XACML (Godik & Moses, 2003) specification for the expression of authorization policies is based strongly on the XACL language and in the future could become a standard for specifying authorization policies for Web Services.

XACL provides fine-grained provisional access control to XML documents. It is implemented using the Java language. XACL is an access control policy specification language and is oriented around triples of subject, object, and action. Every subject has a user ID and either can be part of a role or a group. An object can be a single XML element or attribute in an XML document or the entire document. The actions can be read, written, created, or deleted. XACL provides the notion of *provisional actions* that are attached to the access decision. For example, if a provisional action log is specified in the access control rule of "Alice is allowed to read the HIV status field of a patient," it means that "Alice is allowed to read the HIV status field of a patient, provided the access is logged."

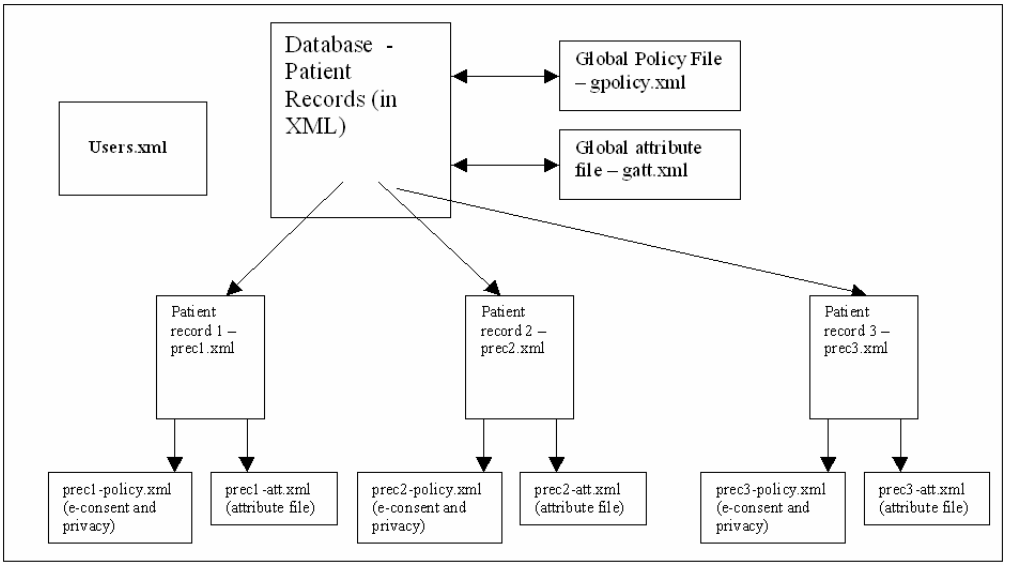
Health Care Application: System Architecture

We used the access control requirements for medical practice given by the specification in Sim (2002) to implement the health care application. We created the health care application as a .NET Web application. Per the specification, the roles required for the medical practice are Patient, Receptionist, Pathology Collector, Nurse, Doctor, Doctor-in-charge, and Practice Manager. Each of these roles has a specific set of privileges. We mapped those privileges into specific XACL policies. We specified role, session, and user constraints and also a range of access control policies such as those discussed in the extended authorization model section using XACL in our demonstration. In this section, we outline the high-level, three-tier architecture overview (shown in Figure 15) of our implementation.

Presentation Tier. The health care application is exposed to clients such as doctors, patients, and nurses using Web pages built with ASP.NET Web Forms technology.

Incoming HTTP requests are filtered using the Authorization Filter, and access is controlled appropriately. The Authorization Filter is a .NET component. All the Web Service requests pass through the Authorization Filter and are sent to the XACL processor for access control evaluation. XACL processor is an authorization policy evaluation engine that gives out a decision after evaluating the request us-

Figure 16. Data and policy view



ing the policies specified. It generates a decision list with *permit* or *deny* permission to each element and attribute in the set of XML nodes passed in the request. The Authorization Filter receives these decisions as output and gives appropriate read, write, or modify permissions to each of the XML elements and attributes in the patient record. The text boxes in the Web interface, for instance, are disabled for writing if there is no write access for a nurse on a field in the medical record. However, when there is read access, the data is visible to the nurse. If both read and write operations are not allowed, the text box is grayed out, and no data are visible.

The Authorization filter makes use of .NET Remoting technology (Microsoft Corporation, 2001c) and JNBridgePro (JNBridge Corporation, 2002) proxy tool to send the authorization requests to XACL (running) on a Java platform from the .NET platform and receive the access control decisions back on the .NET platform.

Business Logic Tier. The application's logic is implemented as Web Services. EssentialDetailsWS, ClinicDetailsWS, and HealthDetailsWS are the Web Services used to read, write, create, or delete the essential, non-essential, clinic, and health details of a patient in his or her record. The fourth Web Services, E-ConsentWS, is used to set the consent of the patient on his or her record.

The XACL processor is a Java class that implements the authorization logic for policy evaluation on the lines of our extended authorization algorithm. It takes the target, request, policy, status, role hierarchy, and group hierarchy files as input parameters in the form of XML documents. It gives the access decision on each XML element and attribute as output.

Data Tier. We use an SQL Server to store both the medical records and the authorization policies. The medical records in the system are stored in XML format using the SQLXML (Microsoft Corporation, 2004) technology in a Microsoft SQL Server. The security policies are associated with them, as shown in Figure 16.

Each patient has an ID, and the name of the XML file is ID.xml. For example, if a patient's ID is 1234, then the patient's corresponding record is 1234.xml. The patient records consist of four main XML sections holding the patient's essential details, non-essential details, clinic details, and health details.

The system also maintains a users.xml file that consists of each user's user ID and

password along with the roles to which he or she is provisioned and other role-session constraints details. The security administrator maintains this XML file. When a user logs in (using .NET passport authentication), he or she is provided the choice of choosing which roles he or she wants to activate in the current session.

The global security policy file gpolicy.xml contains the XACL access control policies applicable to all patient records (resources) and to all users (subjects) in the system. The global security attribute file gatt.xml contains the global attributes details required to enforce access control policies. For example, an access policy could state that a doctor only can access patient records from a location in Sydney. As this policy is generic (i.e., applies to all patient records), the attribute — *location* — for a doctor is stored in this file when he or she is authenticated.

The e-consent file pid-policy.xml is used to record the individual patient's consent when he or she creates a record with the medical practice. E-consent policies also can be seen as protecting a client's privacy. When a receptionist creates the record for the first time, the patient consent is asked and recorded in his or her file. The consent given at the record creation time is not static and can be changed at any time by the patient.

We allow for four types of e-consent:

1. **General Consent.** Global policy applies to this patient's record.
2. **General Consent With Specific Denial.** Global policy applies to this patient's record. However, the patient is allowed to explicitly deny access to any subject.
3. **General Denial.** Global policy does not apply here. Patient explicitly creates the policy for himself or herself, which is stored in pid-policy.xml.
4. **General Denial With Specific Consent.** Patients are allowed to explicitly give specific consent to subjects. This consent information either can be stored in the global policy and/or the personal policy in pid-policy.xml.

The patient record security attribute file pid-att.xml stores the security attributes relevant only to this patient's record. For example, if a patient (Bob) with patient ID = 9876 wishes to limit the number of accesses to his file to five times to a nurse (Alice), the current number of accesses by Alice to his record will be stored in the file 9876-att.xml.

EXTENSIONS TO XACL

As mentioned earlier, we used the XACL processor to demonstrate our extended authorization model. In our model, authorization policies used in commercial environments such as conditional authorization, role-hierarchy, dynamic separation of duty, delegation, Chinese-wall policy, joint-action-based policies, and limited number of accesses policy can be specified. Using the concept of provisional authorization (Kudo et al., 2000), XACL can be used to specify conditional access control policies, which we can specify in our model. Similarly, role-based access control policies can be specified using XACL. However, dynamic separation of duty, delegation, Chinese-wall policy, joint-action based policies, and limited number of accesses policies cannot be specified using XACL. We extended XACL to be able to specify and enforce such policies. We were able to do this by implementing the generic ProvisionalActionInterface interface provided by the XACL Application Programming Interface (API). We implemented the provisional action checkDynamicAccess to enforce dynamic separation of duty policies, checkDelegation to enforce delegation policies, checkCWPSet to enforce Chinese-wall policy, checkJointAction to enforce joint action-based policies, and checkLimit to enforce a limited number of access policies. In the next section, we discuss how we are able to specify Chinese-wall policy (Brewer et al., 1989) using extended XACL.

Chinese-Wall Policy

In Chinese-wall policy (Brewer et al., 1989), objects are grouped together into different conflict of interest sets. Let us assume that every patient record belongs to a hospital and to only one department within the hospital. So, the patient records are grouped into:

Figure 17. Global attribute and XACL policy files

```
<!-- Global Attribute File -->
<global_attributes>
  <users>
        <id> 1234 </uid>
              <user_name> bsmith <user_name/>
              <password> pwd </password>
              <first_name> Bob </first_name>
              <last_name> Smith </last_name>
              <role> Intern </role>
              <attributes>
                    <CWP_Set>
                          <SetA> Cardiology </SetA>
                          <SetB> XYZ Hospital </SetB>
                    </CWP_Set>
                    <!-- other attributes stored here -->
              </attributes>
        </id>
        <!--other users' details and attributes stored here        -->
  </users>
  <CWP_Set_Definitions>
        <CWP_Sets>
              <SetA> Cancer research, Cardiology, Neurology </SetA>
              <SetB> ABC Hospital, PQR Hospital, XYZ Hospital </SetB>
        </CWP_Sets>
  </CWP_Set_Definitions>
</global_attributes>

<!—XACL Policy File -->
<policy xmlns: xsi = http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation = http://www.trl.ibm.com/projects/xml/xacl/xacl.xsd"
        xmlns = "http://www.trl.ibm.com/projects/xml/xacl">
  <xacl>
    <object href = "/medical_record" />
    <rule>
      <acl>
        <subject>
          <role>
        </subject>
        <action name = "read" permission = "grant">
            <provisional_action timing = "before" name = "checkCWPSet">
              <parameter>gatt.xml</parameter>
              <parameter>
                <function name = "getUid"/>
              </parameter>
            </provisional_action>
        </action>
      </acl>
    </rule>
  </xacl>
</policy>
```

Set A. Departments (Cancer Research, Cardiology, Neurology)

Set B. Hospitals (ABC Hospital, PQR Hospital, XYZ Hospital)

The global policy may wish to implement Chinese-wall policy on interns when they access the patient records. The interns may be allowed to do research by accessing patient records from only one department in set A and from only one hospital in set B. According to the Chinese-wall policy, an intern (Bob) initially may access any of the documents from Set A and Set B. Once Bob chooses a medical record from a category (Cardiology) from Set A, he is not allowed to access any other category from that set. Similarly, if the record chosen is from XYZ Hospital, Bob is not allowed to access patient records from any other hospital in Set B.

In Figure 17, we show how to specify the required Chinese-wall policy in XACL. We also

show the attributes required for Bob in the global attribute file. The global attribute file contains the definition for the Chinese-wall policy sets. The CWPSet attribute records the accessed categories information needed to enforce the Chinese-wall policy. The checkCWPSet provisional action implements the logic necessary to enforce the Chinese-wall policy on interns. It does the following:

- It takes two parameters — the global attribute file and the user ID. It searches for the user's CWPSet attribute in the attribute file and gives suitable access as per Chinese-wall policy.
- It writes to the CWPSet of the user the first time the user accesses a category in a set.

RELATED WORK

WS-Policy (IBM et al., 2004) specification is used to specify policy assertions associated with a Web Service. The WS-Trust (Anderson et al., 2005) language uses the secure messaging mechanisms of WS-Security (Atkinson et al., 2002) specification to define additional primitives and extensions for the issuance, exchange, and validation of security tokens. WS-Trust also enables the issuance and dissemination of credentials within different trust domains. However, because WS-Security was published prior to WS-Policy, WS-SecurityPolicy (Della-Libera et al., 2002) language was created to specify security-related assertions required for a Web Service.

A security policy specified using WS-SecurityPolicy language is a set of domain-specific policy statements. A policy statement is a group of policy assertions. A policy assertion represents an individual preference, requirement, capability, or other property. Security Token assertion informs a Web Service's client what kind of security token it needs to send before invoking a Web Service. A security token assertion either can be a X.509 or a Kerberos certificate, a SAML (OASIS, 2005) assertion or an XrML (OASIS, 2003) assertion. Similarly, integrity and confidentiality assertions specified in a WS-SecurityPolicy statement inform a client what integrity and confidentiality mechanisms a client needs to use. If our extended XACL authorization policy language or any other XML-based authorization policy language is standardized for Web Services, there is a requirement to extend WS-SecurityPolicy language's XML schema to add a new authorization assertion token. A Web Service will use this assertion to inform its clients about what privilege attributes are required to get authorized to the Web Service.

XACML (Godik et al., 2003) defines a core schema and corresponding namespace for the expression of authorization policies in XML against objects that are identified in XML. This language in the future could become a standard to specify authorization policies for Web Service. XACL, on which XACML specification is based, has a Java implementation. As mentioned earlier, this is another reason for us to use XACL to implement our extended authorization model.

Damiani, et al (2002) propose a model for fine-grained access control policies for XML documents. Gabillon, et al (2002) define a security model for regulating access to XML documents. However, these models are designed only to control access to XML documents and not to any generic resource. They also do not support features such as provisional authorization supported by XACL.

Damiani, et al (2002) also propose an approach that relies on the XML structure of SOAP requests and supports fine-grained authorizations at the level of individual XML elements and attributes that comprise a SOAP call. This work complements our work, as it looks into fine-grained authorization on XML elements and attributes when they are transported within SOAP requests and responses over the network.

SAML (OASIS, 2005) is an XML-based security specification for exchanging authentication and authorization information about a user or subject. It defines XML schema and definition for security assertions. The assertions are of three types—authentication, any security related attributes for the subject, and finally, the authorization decisions given based on the security/privilege attributes. SAML also

can be extended to send any arbitrary security assertions. This shows that SAML is used primarily to carry security/privilege attributes related to a subject from one entity to another in a network. The only requirement is that both entities comply with the SAML standard. This specification complements our work, as we need a standard mechanism to carry authentication and authorization-related attributes on behalf of a subject to the entity evaluating the authorization policies.

XrML (OASIS, 2003) is another specification for XML-based policy language for controlling access to digital resources. However, it is designed in specifics for specifying and managing rights and conditions for digital resources, such as audio and video. For instance, if a digital media player has an XrML-compliant rights enforcer, the media played on it can have XrML-based policies associated with it to restrict the usage of the media only to licensed users. XrML tries to solve the problem of digital rights management and is not designed to specify authorization policies streamlined for stand-alone application systems or Web Service.

CONCLUDING REMARKS

At present, there are several efforts underway that are striving for the provision of security services, such as authentication between participating entities, confidentiality, and integrity of communications. Currently, however, most applications, having gone through the authentication process, make authorization decisions using application-specific access control functions that are inflexible and inadequate. There remains a significant amount of research that needs to be done in the area of authorization in large-scale deployment of Web Services in heterogeneous environments. There are a lot of similarities in the design of authorization service and architecture between distributed applications and Web Services. One aspect that makes it somewhat different is the need to take into account multiple domains and jurisdictions in a federated structure as a default. We discussed the fundamental design issues that need to be taken into consideration when building an authorization framework for Web Services. In this paper, we specifically addressed the features that an authorization policy language designed for Web Services must have.

We introduced the authorization model used by .NET MyServices. We then discussed our proposed modifications and extensions to the authorization model. In particular, we extended the authorization policy language to support a range of access control policies, including conditional access control policies, dynamic separation of duty policies, Role Based Access Control (RBAC) policies, delegation policies, joint-action-based policies, and Chinese-wall policies required in commercial environments. We redefined the .NET MyServices authorization algorithm to reflect the extensions that we proposed to the authorization model. We then discussed the application of the extended authorization model to a health care application. We gave an overview of the high-level three-tier system architecture of our implementation. We used the XML Access Control Language (XACL) in our implementation to demonstrate our extended authorization model. This is because XACL is an XML policy language that supports RBAC policies and provisional authorization. This enabled us to evaluate the range of access control policies supported by XACL and propose extensions to the XACL policy language to be able to specify a range of access control policies, including role-hierarchy, dynamic separation of duty, delegation, the Chinese-wall policy, joint-action-based policies, and limited number of accesses. For illustration purposes, we provided an example that shows the specification of the Chinese-wall policy using the extended XACL policy language.

ACKNOWLEDGMENTS

The authors would like to thank the Australian Research Council and Microsoft Pty. Australia for their support.

REFERENCES

Anderson, S. et al. (2005). Web Services trust language (WS-Trust). Retrieved from *http://www-106.ibm.com/developerworks/li-*

brary/specification/ws-trust/

Ash, D. et al. (2004, April 5). XML key management specification (XKMS 2.0). Retrieved from *http://www.w3.org/TR/xkms2/*

Atkinson, B. et al. (2002). Web Services security (WS-Security) specification. Retrieved from *http://www-106.ibm.com/developerworks/webservices/library/ws-secure/*

Bartel, M., Boyer, J., Fox, B., LaMacchia, B., & Simon, E. (2002, February 12). XML-signature syntax and processing. Retrieved from *http://www.w3.org/TR/xmldsig-core/*

Brewer, D.F.C., & Nash, M.J. (1989). The Chinese wall security policy. In *Paper Presented at the IEEE Symposium on Security and Privacy*.

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). Web Services description language (WSDL) version 1.1. Retrieved from *http://www.w3.org/TR/wsdl*

Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., & Samarati, P. (2002). Securing SOAP e-services. *International Journal of Information Security*, 100-115.

Damiani, E., Vimercati, S.D.C.D., Paraboschi, S., & Samarati, P. (2002). A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security, 5*(2), 169-202.

Damianou, N., Dulay, N., Lupu, E., & Sloman, M. (2001). The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, (pp. 18-38). Springer-Verlag.

Della-Libera, G. et al. (2002). Web Services security policy language (WS-SecurityPolicy). Retrieved from *http://www-106.ibm.com/developerworks/library/ws-secpol/*

Gabillon, A., & Bruno, E. (2002). Regulating access to XML documents. In *Proceedings of the 15th Annual Working Conference on Database and Application Security*.

Godik, S., & Moses, T. (2003, August 7). eXtensible access control markup language version 1.1. Retrieved from *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml*

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., & Nielsen, H.F. (2003, June 24). SOAP version 1.2. Retrieved from *http://www.w3.org/TR/soap12-part1/*

Herzberg, A., Mass, Y., Michaeli, J., Ravid, Y., & Naor, D. (2000). Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*.

Hitchens, M., & Varadharajan, V. (2000). Design and specification of role based access control policies. *IEE proceedings, Software*.

IBM, BEA, Microsoft, SAP, Sonic Software, & VeriSign. (2004). Web Services policy framework (WS-Policy). Retrieved from *http://www-128.ibm.com/developerworks/library/specification/ws-polfram/*

Imamura, T., Dillaway, B., & Simon, E. (2002, December 10). XML encryption syntax and processing. Retrieved from *http://www.w3.org/TR/xmlenc-core*

Indrakanti, S., Varadharajan, V., Hitchens, M., & Kumar, R. (2003, August 4-6). Secure authorization for Web Services. In *Paper Presented at the 17th IFIP Conference on Data and Applications Security*, Estes Park, Colorado.

Jajodia, S., Samarati, P., Subrahmanian, V.S., & Bertino, E. (1997). A unified framework for enforcing multiple access control policies. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data* (pp. 474-485). ACM Press.

JNBridge Corporation. (2002). JNBridgepro users' guide version 1.2. Retrieved from *http://www.jnbridge.com/docs.htm*

Kent, S., & Atkinson, R. (1998, November). Security architecture for the Internet protocol, RFC 2401. Retrieved from *http://www.ietf.org/rfc/rfc2401.txt*

Kraft, R. (2002, November 22). Designing a distributed access control processor for network services on the Web. In *Paper Presented at the ACM Workshop on XML Security*, Fairfax, Virginia.

Kudo, M., & Hada, S. (2000). XML document security based on provisional authorization. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Greece.

Microsoft Corporation. (1999). .NET passport Web

site. Retrieved from *http://www.passport.net*

Microsoft Corporation. (2001a). *.NET MyServices Specification*. Microsoft Press.

Microsoft Corporation. (2001b). *Microsoft .NET MyServices specification*. Microsoft Press.

Microsoft Corporation. (2001c). Microsoft .NET remoting: A technical overview. Retrieved from *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp*

Microsoft Corporation. (2004). SQLXML. Retrieved from *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/anch_SQLXML.asp*

OASIS. (2003). Extensible rights markup language (XrML) version 2.0. Retrieved from *http://xml.coverpages.org/xrml.html*

OASIS. (2005, January 17). Security assertion markup language version 2.0. Retrieved from *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security*

O'Keefe, C.M., & Greenfield, P. (2003). Role and expression of consent in Web Services. In *Paper Presented at the International Conference on Web Services*, Las Vegas, Nevada.

Rescorla, E. (2001). *SSL and TLS: Designing and building secure systems*. Addison Wesley.

Sandhu, R., Coyne, E.J., Feinstein, H.L., & Youman, C.E. (1996). Role-based access control models. *IEEE Computer, 29*(2), 38-47.

Sim, P. (2002). *Access control requirements for a medical practice and hospital environment and a secure access control architecture*. Sydney, Australia: Macquarie University.

Simon, R., & Zurko, M.E. (1997). Separation of duty in role-based environments. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW '97)*, (pp. 183). IEEE Computer Society.

Varadharajan, V. (2002). Distributed authorization: Principles and practice. In *Coding theory and cryptology, lecture notes series, Institute for Mathematical Sciences, National University of Singapore*. Singapore University Press.

Varadharajan, V., & Allen, P. (1996). Joint action based authorization schemes. *ACM Operating Systems Review Journal*.

Varadharajan, V., Allen, P., & Black, S. (1991). An analysis of the proxy problem in distributed systems. In *Paper Presented at the IEEE Symposium on Security and Privacy*, Oakland, California, May 20-22.

Varadharajan, V., Crall, C., & Pato, J. (1998). Authorization in enterprise-wide distributed system: A practical design and application. In *Proceedings of the 14th Annual Computer Security Applications Conference*.

Yagüe, M.I., & Troya, J.M. (2002). A semantic approach for access control in Web Services. In *Paper Presented at the Euroweb 2002 Conference: The Web and the GRID: From E-Science to E-Business*, Oxford, UK.

Sarath Indrakanti received his bachelor's of applied science (major in computing) from the University of Western Sydney, Australia in 2001. He is currently a doctoral candidate in the Department of Computing at Macquarie University, Australia. His area of research is in the context of security for Web Services. In particular, his work focuses on providing an authorization framework for Web Services. His research work is sponsored by the Australian Research Council and Microsoft Pty. Australia.

Vijay Varadharajan received his BS in electronic engineering from Sussex University, UK, in 1981 and his PhD in computer and communication security in the UK in 1984, which was sponsored by BT Research Labs. He is the Microsoft chair and professor of computing at the Macquarie University and the director of information and networked systems security research. He is also the technical board director of computer science, Australian Computer Society. He has been a member of the board of advisors in the Trusted Computing Platform Association (TCPA) and is on the Microsoft Trustworthy Computing Academic Advisory Board (TCAAB). He was responsible for worldwide security research at corporate Hewlett-Packard Labs based

in Europe at HP Labs, UK. Prior to HP, he was a research manager at British Telecom Research Labs, UK. He is on the editorial boards of several international journals, including the ACM Transactions on Information Systems Security *and the* International Journal of Information Security, *Springer Verlag. His current areas of research interest include security in high speed networks, security for large distributed systems, security policies and management in distributed applications, Internet security, secure electronic commerce and payment models, secure mobile agents, wireless security, security models and architectures, and security protocols.*

Michael Hitchens received his bachelor's of math. from Newcastle University, Australia, in 1986 and his PhD in computer science from the same university in 1991. He is a senior lecturer in the Department of Computing at Macquarie University, Australia. He has also worked at the universities of Sydney and Western Sydney. His current areas of research interest include security for large distributed systems, models and languages for access control, security for wireless networks, security for distributed file systems, authorization management, and security protocols.