CERIAS Tech Report 2006-03

A POLICY-BASED AUTHORIZATION SYSTEM FOR WEB SERVICES: INTEGRATING X-GTRBAC AND WS-POLICY

by Rafae Bhatti, Daniel Sanz, Elisa Bertino, Arif Ghafoor

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

A Policy-Based Authorization System for Web Services: Integrating X-GTRBAC and WS-Policy

Rafae Bhatti, Daniel Sanz, Elisa Bertino, Arif Ghafoor

Authorization and access control in Web services is complicated by the unique requirements of the dynamic Web services paradigm. Amongst them is the requirement for a context-aware access control specification and a processing model to apply fine-grained access control on various components of a Web service. In this paper, we address these two requirements and present a policy-based authorization system that leverages an emerging Web service policy processing model, WS-Policy, and integrates it with X-GTRBAC, an XML-based access control model to allow specification and processing of fine-grained, context-aware authorization policies in dynamic Web services environments. The architecture is designed to support the WS-Policy Attachment specification, which allows attaching, retrieving and combining policies associated with various components of a Web service in the WSDL document. Consequently, we present an algorithm to compute the effective access control policy of a Web service based on its description. The effective policy, represented as a normalized WS-Policy document, is then used by the X-GTRBAC system to evaluate an incoming access request. We have prototyped our architecture, and implemented it as a loosely coupled Web service, with logically distinct, heterogeneous modules acting as Policy Enforcement Point (PEP) and Policy Decision Point (PDP). Our prototype demonstrates the true promise of the decentralized Web services architecture, and incorporates SAML-based single sign-on communication between multiple system modules.

1. Introduction

Access control in Web services is a neglected frontier that has not seen the development and adoption of many standards, as opposed to the number of current and emerging specifications for authentication aspects of Web services security [13, 14, 16]. These specifications allow one to express preferences for use of security attributes to establish trusted and authenticated connections between multiple service providers or end users. While authentication and privacy can ensure the security of connections and privacy of user information, respectively, the security of the information content provided by the service is controlled by the authorization policies. However, not many specifications exist that are primarily designed to provide support for authorization policies for Web services. Additionally, no truly Web-service oriented architectures and implementations have been reported that identify and address important issues related to processing of authorization policies in Web services. In this paper, we attempt to close the gap between authentication and authorization support in Web services by presenting an authorization system that is designed specifically to work in a Web services environment.

At the very onset, we would like to motivate the problem addressed in this paper with a typical Web services scenario. A Web application usually requires the invocation of one or more Web services in order to provide its functionality. An individual Web service in turn provides several operations to carry out a specific function that it offers. The primary functional specification of a Web service is described by the Web Service Description Language (WSDL) [10]. A WSDL document includes the definitions of port types, bindings, and ports used to make up a Web service. Operations and interfaces grouping those operations provided by the service are also defined using the WSDL. A service instance typically comprises of a unique port of a given type with an associated binding, and a set of operations included with that port type. In this way, multiple service instances can be exposed using the same WSDL file by associating the service instances with different types of ports, each exposing a different set of operations.

One such scenario is depicted in Figure 1. It illustrates a Health Information System Web application that uses multiple Web services to offer a variety of services to its clients. It offers a top level service called Patient Track Service (PTService) which allows physicians to track all patients in the system based on the authorization of the physician. This service returns a list of patients and the location of their records. Subsequently, the physician can choose to view a specific patient record from a given location (USA and Spain in this example),

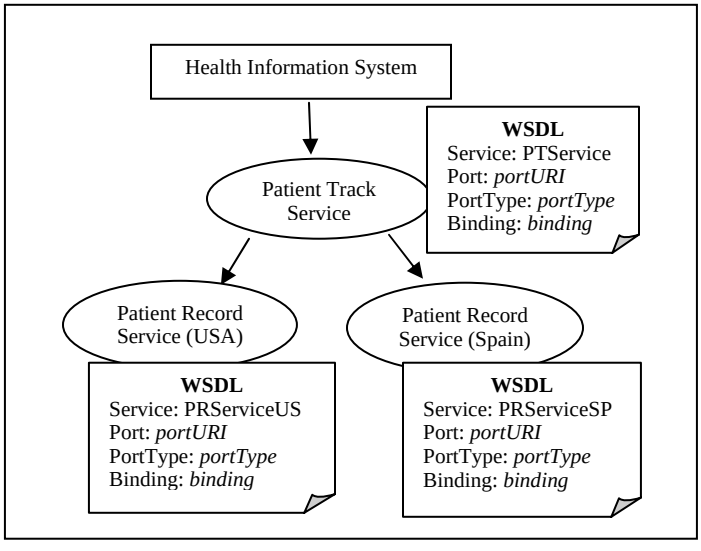

Figure 1: A Web-service based application: services are described using WSDL documents.

for which the system will invoke the appropriate Patient Record Service (PRService). The authorization credentials of the requesting physician will again be required to obtain this new service. The level of access that the physician is allowed will depend on his/her authorization credentials, and several service instances with a different set of operations corresponding to various levels of authorization may be defined to accommodate this requirement. The specific service instance to invoke can be predefined or dynamically discovered (using for instance UDDI), but in both cases WSDL is required to initiate the interaction. Clearly then, the task of interaction between Web services requires a fine grain of control on what components of a service may be allowed to be invoked and under what circumstances.

This example serves to illustrate the following two requirements for Web service access control: (i) the granularity of service access must be based on the level of user authorization, and (ii) the expressiveness of the service access policy must support context-aware access control. The first requirement is a service design issue, which can be addressed by creating differentiated service instances (with different set of operations) at distinct ports and associating an access control policy with those instances, and with the components that make up those instances (ports, port types and bindings). The second requirement is a policy design issue, and can be addressed by a using a policy specification language that can adequately capture the context-aware access control requirements in a dynamic Web service environment. Both these aspects need to be tied together in a seamless manner and included with the service definition so that the WSDL file can be a self-contained description of the service and its applicable access control policies. The focus of our current work is to present an authorization architecture that specifically achieves this objective.

1.1. Contributions and Organization

In this paper, we propose a policy-based authorization system that leverages an existing Web service policy processing model, WS-Policy, [11] and integrates it with X-GTRBAC, an XML-based access control model [3] to allow specification and processing of fine-grained context-aware authorization policies in dynamic Web services environments. The architecture is designed to support the WS-Policy Attachment model [12], which allows attaching, retrieving and combining policies associated with various components of a Web service description. Consequently, we present an algorithm to compute the *effective access control policy* of a Web service based on its description document. The effective policy, represented as a normalized WS-Policy document, is then used by the X-GTRBAC system to evaluate an incoming access request. We have prototyped our architecture, and implemented it in a Web services environment. In fact, the authorization system itself is also implemented as a loosely coupled Web service, with logically distinct, heterogeneous modules acting as Policy Enforcement Point (PEP) and Policy Decision Point (PDP). The system has been tested with a PHP-based PEP deployed at Carlos III University of Madrid in Spain and a Java-based PDP at Purdue University in USA. Our prototype demonstrates the true promise of the decentralized Web services architecture, and incorporates SAML-based single sign-on communication between multiple system modules.

The remainder of this paper is organized as follows. The next section discuses related work in Web services access control, and identifies the shortcomings of the existing approaches with respect to the challenges outlined above. Section 3 presents the details of the policy specification in our authorization system. It first provides an overview of X-GTRBAC and WS-Policy specification languages, and then presents a WS-Policy profile of X-GTRBAC that is used in our system to express Web service authorization policies. Section 4 discusses the mechanisms needed for policy processing in a Web services environment. It provides an overview of the WS-Policy Attachment specification, and discusses in detail the mechanism of attaching, retrieving, and combining policies associated with multiple components of a Web service description, and computing the effective service access policy. Section 5 presents the proposed architecture and implementation of our authorization system. Section 6 concludes the paper.

2. Related Work

There has been an effort in the research community to highlight the challenges associated with Web-based access control. Many of these mechanisms provide specification of context-aware access control languages [1, 2, 5, 6, 9]. They, however, do not provide a mechanism for policy processing in a Web services environment to allow fine-grained access control on individual Web service components defined in WSDL.

A fair amount of related research in the area of Web services security is due to the industry, with standards such as Security Assertion Markup Language (SAML) [16] and eXtensible Access Control Markup Language (XACML) [17] having been recently adopted. SAML defines an XML framework for exchanging authentication and authorization information for securing Web services, and relies on third-party authorities for provision of "assertions" containing such information. However, SAML itself is not designed to provide support for specifying authorization policies; it is in fact a complementary specification, and we use it in our work. XACML is an XML framework for specifying context-aware access control policies for Web-based resources. The Web Service Policy Language (WSPL) [18] is an XACML profile for Web services that can be used to publish the access control requirements of a Web service using XACML. Being derived from XACML, WSPL can be used for policy specification to satisfy one of the requirements for Web service access control identified above. It, however, does not support a generalized policy processing mechanism and must be bound to an XACML target to be used in a Web service. Unlike WS-Policy, XACML does not provide a formal mechanism to associate policies with components of a Web service definition.

The most notable set of emerging specifications are the ones outlined in WS security roadmap [15]. The roadmap consists of a number of component specifications, the core amongst them are WS-Security [13], WS-Policy [11], and WS-Trust [14]. WS-Security is a specification for securing whole or parts of an XML message using XML encryption and digital signature technology, and attaching security credentials thereto. WS-Policy is used to describe the security policies in terms of their characteristics and supported features (such as required "security tokens", encryption algorithms, privacy rules, etc.). In fact, WS-Policy is a meta-language which can be used to create various policy languages for different purposes, and can indeed be used to define an access control policy. WS-Trust defines a

trust model that allows for exchange of such security tokens (using mechanisms provided by WS-Security, and according to the requirements supplied by WS-Policy) in order to enable the issuance and dissemination of credentials within different trust domains, and establish online trust relationships.

The models proposed in the roadmap have been directed primarily at the authentication aspect of Web services security, with an emphasis on designing secure messaging protocols to communicate the security-relevant information, such as security tokens and characteristics of security policy. The specification leaves room for custom authorization models to be tied into the architecture at the appropriate (i.e. WS-Policy) level. This is exactly where our current work fits in; we use X-GTRBAC to provide support for expressing authorization policies within the WS-Policy model. The choice for the use of X-GTRBAC as the authorization model in our system is motivated by prior work [1] that highlights the various advantages of X-GTRBAC specification language, such as simplified role-based administration and expressive yet flexible constraint specification, which make it suitable for context-aware access control in dynamic Web services. That work does not address the issues specific to policy processing identified in the paper. To provide this support, we integrate X-GTRBAC with a Web service specific policy processing model comprising of the WS-Policy and WS-Policy Attachment specifications. To the best of our knowledge, addressing this aspect of Web service access control remains a novel contribution.

3. Policy Specification

This section presents the details of policy specification in our system.

3.1 X-GTRBAC

Our authorization system is based on X-GTRBAC model [3]. X-GTRBAC is an XML-based extension of the role-based access control (RBAC) model [8]. RBAC uses the notion of roles to embody a collection of permissions; permissions are associated with roles through a permission-to-role assignment, and users are granted access to resources through a user-to-role assignment. X-GTRBAC extends RBAC to provide a generalized mechanism to express a diverse set of constraints on user-to-role and permission-to-role assignments. It is this constraint specification mechanism which is of specific relevance to us for this paper, and we shall discuss that below.

3.1.1. Constraint Specification

X-GTRBAC allows an expressive and flexible constraint specification mechanism to define temporal and non-temporal contextual constraints. An (user-to-role or permission-to-role) assignment constraint in X-GTRBAC comprises of a set of assignment conditions, where each condition has associated with it an optional temporal constraint expression and an optional set of non-temporal logical expressions. The syntax of an assignment constraint is described in Table 1.

Table 1: X-GTRBAC assignment constraint expression

<AssignConstraint op="AND|OR"> [<AssignCondition [cred_type=""] [pt_expr_id=""]> [<!--<Logical Expression>-->]* </AssignCondition>]+ </AssignConstraint> <!--<Logical Expression>-->::= <LogicalExpression op=" AND|OR"> [<!--<Predicate Block>-->]+ <LogicalExpression> <!--<Predicate Block>--> ::= <!--<Logical Expression>--> | <!--<Predicate>--> <!--<Predicate>--> ::= <Predicate> <Operator/> <FuncName/> <ParamName/> <RetValue/> </Predicate>	AssignConstraint: represents a set of constraints to apply to the assignment. The attribute op defines the evaluation mode of the included conditions. AssignConstraint/AssignCondition: represents a contextual condition. It may specify a credential type and a periodic time expression. The former indicates that the subject of the constraint (user or role) must present a credential, the attributes of which must satisfy the rules defined in this condition. The latter represents a temporal constraint expression (See [3]). AssignConstraint/AssignCondition/LogicalExpression: represents a logical expression. It contains one or more predicates. The attribute op defines the evaluation mode of the predicates. AssignConstraint/AssignCondition/LogicalExpression/{PredicateBlock}: represents either another logical expression or a simple predicate. AssignConstraint/AssignCondition/LogicalExpression/Predicate: A simple predicate defines rules on credential attributes of the constraint subject (user or role). It includes a comparison using an (Operator) between the value of the credential attribute computed using a function (FuncName) having one or more arguments (ParamName) and the expected (RetValue).

An assignment constraint is satisfied if all included assignment conditions are satisfied (according to the supplied operator, which defaults to AND if none is supplied). An assignment condition is satisfied if (i) the associated temporal constraint expression, if any, is satisfied; a temporal constraint expression checks for time-based conditions, such as periodicity, interval or duration, and (ii) the associated set of logical expressions, if any, is satisfied; a logical expression is satisfied if all included predicates are satisfied (according to the supplied operator). A logical expression defines rules on the credential attribute of the constraint subject (user or role). As an example, an assignment constraint can state that "role r can access resource o if (a) the access occurs between 9AM and 5PM during the month of

`January` in year `2006`, and (b) the `location` is "London" and the `system load` is "low". Here, (a) is an example of a temporal constraint, represented as a temporal constraint expression in X-GTRBAC, and (b) is an example of a non-temporal constraint represented as a set of logical expressions. To evaluate this logical expression, the role `r` must supply a credential having the attributes `location` and `system load`.

For our current work, we are particularly interested in expressing contextual constraints on service usage, which can be modeled in X-GTRBAC as a permission-to-role assignment policy. The only assumption we need to make is that the service access policy is designed based on the RBAC model. Therefore, user authorization levels are modeled as roles, and services are modeled as permissions associated with roles. Note that a permission in this model then refers to the service instance represented by the "service" element in the WSDL. Therefore, fine-grained service access control policies can be composed by associating multiple roles with multiple differentiated instances of a Web service defined by the same WSDL (To recall, this will be done by associating multiple service instances with different types of ports, each exposing a different set of operations). Each such service instance will be a permission which is assigned to a given role subject to the permission-to-role assignment policy in X-GTRBAC[1]. To keep our exposition clear, the assignment policies we use shall only include the non-temporal constraints modeled by logical expressions, since the treatment of temporal constraint expressions requires more detail than can be provided in this paper.

3.2. WS-Policy

WS-Policy defines an abstract model for expressing the capabilities, requirements, and general characteristics of entities in XML Web service-based systems. These properties are expressed as policies. WS-Policy does not specify how policies are discovered or attached to a Web service, only focuses on defining them. A *policy* is a collection of *policy alternatives*, where each policy alternative is a collection of *policy assertions*. An assertion can express requirements or capabilities that will manifest in the wire, some others will refer to service usage or selection. For the purposes of this work, we will use the term assertion to indicate assertions used in the authorization policies. A set of constructs is provided by the specification to indicate how choices and/or combination of policy assertions apply in a Web services environment.

A policy is represented by its corresponding policy expression. While many policy expressions are possible according to the model, the *normal form policy expression* is the canonical form, and is described in Table 2.

Table 2: WS-Policy normal form policy expression

<wsp:Policy> <wsp:ExactlyOne> [<wsp:All> [<assertion …> … </assertion>]* </wsp:All>]* </wsp:ExactlyOne> </wsp:Policy>	wsp:Policy: represents a policy wsp:Policy/wsp:ExactlyOne: represents a collection of policy alternatives wsp:Policy/wsp:ExactlyOne/wsp:All: represents a policy alternative wsp:Policy/wsp:ExactlyOne/wsp:All/*: XML expressions for assertions, all of which must be satisfied

For the purposes of this work, we will use normal form policy expression to interface with the X-GTRBAC system. The specification does not impose any restriction on the kind of XML policy expressions that may be used for assertions. Therefore, the normalized policy expression can be used to convey assertions related to any domain specific policy. It is this flexibility of the specification that will allow us to integrate WS-Policy with X-GTRBAC.

3.3. WS-Policy Profile of X-GTRBAC

To allow the expression of X-GTRBAC permission-to-role assignment policies in WS-Policy, we have developed a WS-Policy profile for X-GTRBAC. The profile has been designed to be used in a scenario as depicted in Figure 1, where Web applications need to invoke (potentially unknown) Web services. Each service publishes its usage policy, in the form of WS-Policy, attached to its component definitions in its WSDL[2]. These policies include contextual constraints that must be satisfied to invoke the service, such as user location or system load.

The profile defines the nature and semantics of permission, the representation of WS-Policy Assertions as X-GTRBAC constraints, and proposes a loosely coupled architecture comprising of a Policy Decision Point (PDP) and a Policy Enforcement Point (PEP), such that Web applications acting as PEP can obtain an access control decision from a logically (and even geographically) distinct PDP using standard Web-based protocols, such as SAML [16].

3.3.1. Representing a Service as a Permission

A permission in the WS-Policy profile of X-GTRBAC represents access to a service instance. It is identified by the "name" attribute of the <service> element in the WSDL. Since a conventional permission in RBAC (and X-GTRBAC) is a combination of an object and an associated operation, we now need a special interpretation for the purposes of the profile. We interpret the object and operation of a permission defined in the profile as follows:

- *An object*: Since a given service instance is identified in a WSDL by a unique port, the object the permission refers to is the value of "portURI" attribute of the <service> element in the WSDL. Each service instance implements an interface via this unique port, and access to the service at this portURI implies access to all the operations provided by that interface.

[1] Note that a user-to-role assignment policy will also be used to assign authenticated users into roles, but that is orthogonal to our current discussion. It will be discussed in Section 5.1.
[2] Attachment mechanisms will be discussed in Section 4.

- *An operation*: It is currently fixed to be HTTP:GET; it indicates that the access to the service (via its defined operations) will be through an HTTP binding and will use the GET verb[3].

3.3.2. Representing WS-Policy Assertions as X-GTRBAC Constraints

We now define a mechanism to represent WS-Policy assertions as X-GTRBAC constraints. As already indicated, we will only consider normal form WS-Policy expressions, and non-temporal X-GTRBAC constraints modeled by logical expressions. Our analysis proceeds as follows.

Normal Form Expression Element	Contextual Constraint Element	Analysis
<wsp:ExactlyOne>	AssignConstraint/ AssignCondition cred_type="" /LogicalExpression op="OR"	<wsp:ExactlyOne> indicates a collection of alternatives for a policy specific to a particular service component; this implies that the corresponding constraint in X-GTRBAC will comprise of one condition having a top-level logical expression with opcode ="OR"; logical expressions with opcode ="AND" will be nested inside this top level expression, and each of them will represent an alternative; all logical expressions included in this condition contain a set of predicates defining rules on the attributes of a credential provided by the role requesting access to the service.
<wsp:All>	LogicalExpression op="AND"	<wsp:All> indicates a collection of assertions for a policy alternative; this will be represented in X-GTRBAC as a collection of predicates in a logical expression with opcode ="AND"; each predicate included in this logical expression represents an assertion which must be satisfied.
<assertion …>	Predicate	<assertion> represents a system-specific assertion; this will be represented in both policies using the Predicate element of X-GTRBAC.

The table on the right is an example of the mapping between a normal form WS-Policy and X-GTRBAC constraint. It may be observed that the credential type associated with an assignment condition is predefined based on the role that is accessing the service. This credential type must be registered with the PDP in order to evaluate the condition. In practice, this can be done by either a prior arrangement between the PDP and the Web application invoking the PDP, or a dynamic registration using a SAML-based protocol.

WS-Policy	X-GTRBAC Constraint
<wsp:ExactlyOne> <wsp:all> <Predicate>predicate 1</Predicate> <Predicate>predicate 2</Predicate> </wsp:All> </wsp:All> <Predicate>predicate 3</Predicate> <Predicate>predicate 4</Predicate> </wsp:All> </wsp:ExactlyOne>	<AssignConstraint> <AssignCondition cred_type="role_cred_type"> <LogicalExpression op="OR"> <LogicalExpression op="AND"> <Predicate>predicate 1</Predicate> <Predicate>predicate 2</Predicate> </LogicalExpression> <LogicalExpression op="AND"> <Predicate>predicate 3</Predicate> <Predicate>predicate 4</Predicate> </LogicalExpression> </LogicalExpression> </AssignCondition> </AssignConstraint>

4. Policy Processing

We now discuss the mechanisms needed for policy processing in a Web services environment using the WS-Policy profile for X-GTRBAC. We begin by providing an overview of the WS-Policy Attachment specification, and discuss in detail the mechanism of attaching, retrieving, and combining policies associated with multiple components of a Web service description.

4.1 WS-Policy Attachment

WS-Policy Attachment [12] defines a general purpose mechanism for associating policies with the subjects to which they apply, as well as the mechanism to attach policies to WSDL 1.1 descriptions. A *policy subject* is an entity with which the policy is associated, which in our case is a Web service component defined in the WSDL document. A given service may have associated policies by means of multiple attachments associated with the various components defined in the WSDL file. The WS-Policy Attachment specification states that these multiple policy attachments must be combined to obtain the effective policy for the service. We will focus our analysis on WSDL 1.1 metamodel [10], because this is the target of current WS-Policy Attachment specification.

An important notion in computing the effective policy is that of *policy scope*. A policy scope is a collection of policy subjects to which a policy may apply, and a *policy attachment* is the mechanism to associate a policy with a policy scope. WS-Policy Attachment defines four types of policy subjects in WSDL 1.1: *Service Policy Subject*, *Endpoint Policy Subject*, *Operation Policy Subject* and *Message Policy Subject*. The *effective policy* for a given subject is defined to be the combination of all policies attached to policy scopes that contain that subject. The subject types must be considered nested, due to the hierarchical nature of WSDL. Table 3 relates policy scopes with their corresponding subject types in WSDL 1.1.

[3] For the sake of simplicity, this paper does not address the SOAP binding.

This information has an important consequence in our framework; it tells us which policies need to be merged to compute the effective policy of a service, for which there are multiple policies attached to its various components. The *merge* operation takes all relevant policy expressions, replaces their <wsp:Policy> with a <wsp:All> element, and places them as children of a wrapper <wsp:Policy>. The resulting policy expression is the combined policy of all attachments of the subject. The result is equivalent to normalize all policies and do the cross product among all alternatives of each policy, yielding alternatives that consider all possibilities. Using the policy specification from Section 3, we discuss the computation of effective policy for a Web service in the next sub-section.

Table 3: WSDL 1.1 policy scopes and subject types

Policy scope	Policy Subject Type
wsdl:service	Service policy subject
wsdl:port wsdl:binding wsdl:portType	Endpoint policy subject
wsdl:binding/wsdl:operation wsdl:portType/wsdl:operation	Operation policy subject
wsdl:message wsdl:binding/wsdl:operation/wsdl:input wsdl:portType/wsdl:operation/wsdl:input	Message policy subject

4.2 Computing Effective Service Access Policies

The computation of effective policy in our system uses the *merge* operation defined in WS-Policy since our policies are expressed in WS-Policy. Since access to a service is equivalent to the existence of a permission for that service, we define the *effective permission policy* as the policy that must be enforced in order to invoke a given service provided at a given port. The WS-Policy Attachment specification defines several ways of attaching policies to WSDL elements, as well as the policy semantics regarding the hierarchical nature of WSDL definitions. These semantics define which policies need to be merged to compute the effective policy of a service. Currently, we allow policy specification at the level of service, port, port type and binding elements of a service definition, and support the XML attachment mechanism. As a consequence, our architecture supports processing of policy attachments for the Service and Endpoint policy subjects of a WSDL (See Table 3). We define and illustrate in Table 4 the three different levels where merging occurs in our system.

Table 4: Merging process at different levels in WSDL

Type of Merge	Example	Explanation
XML Element Merges individual policies	<wsdl:service PolicyURI= uriPol > <wsp:Policy> inlinePol </wsp:Policy> </wsdl:service> XMLPol = uriPol + inlinePol	WS-Policy Attachment allows a policy to be attached either by using a URI or including it inline. Merge is needed to compute the XML element policy from different fragments attached to it, when more than one attachment mechanism is used. This applies to XML elements representing different WSDL components such as services or port types.
Policy Subject Merges XML Element Subject policies	<wsdl:portType/> → XMLPTPol + <wsdl:port/> → XMLPPol + <wsdl:binding/> → XMLBPol = endpSubjectPol	WS-Policy Attachment defines an Endpoint Subject as combination of port, port type and binding elements, each of which may have an attached policy. In turn, the Service Subject is defined by the WSDL service element. Merge is needed to compute the effective policy for an Endpoint Subject and for a Service Subject.
Permission (Service usage) Merges WSDL Subject policies	Endpoint Subject → endpSubjectPol + Service Subject → serviceSubjectPol = PermissionPol	WS-Policy Attachment specifies that Service Subject includes the Endpoint subject (due to hierarchical nature of WSDL). Merge is needed to compute the *effective permission policy* from effective policies of Endpoint and Service subjects. This *effective permission policy* represents the overall policy for the Web service.

Note that the + symbol denotes the use of merge as described in the WS-Policy specifications [7, 11], which allows us to compute the composed access policy of the service. The processing is independent of the semantics of the assertions and alternatives, and results in a *normal form expression*, where different assertions (<Predicate> tags) will appear grouped within in <All> tags as alternatives. In turn, all alternatives are enclosed within one top-level <ExactlyOne> tag. The normalized policy expression covers all alternatives of each individual policy, and yields

alternatives that consider all possibilities. The normalized policy produced this way therefore has a large number of alternatives (since it is equivalent to a cross product of alternatives). Note that a normalized policy may contain conflicting alternatives that cannot be simultaneously satisfied. This requires the notion of *conflict resolution*. We do not consider this issue in this paper, i.e. we assume non-conflicting policy alternatives which yield a conflict-free normalized policy.

Based on the preceding discussion, we now provide a formal algorithm for computing the effective permission policy.

Algorithm (ComputeEffectivePermissionPolicy):

1. Let *perm* be a permission in the WSPolicy profile of XGTRBAC. *perm* is related to a service *s* provided by a WSDL port, whose effective policy should be calculated from the two types of policy subjects involved, namely endpoint subject and service subject. "
2. Let $PS_{perm} = \{e, s\}$ be the set of *policy subjects* involved in the invocation of any operation in *s*, as defined in the WS Policy Attachment specification (where elements *e, s* represent the endpoint and service respectively). For every possible PS_{perm} in a WSDL file, we have that $e \subset s$ (here $\subset$ can be seen as "defined in", as follows from the WSDL 1.1 schema). Note that all PS_{perm} sets must have these two elements, because every WSDL service is implemented at a given port, from which the rest of WSDL elements in the hierrachy are accesible. Therefore, from <wsdl:service name="s" > tag in the WSDL file one can construct PS_{perm}.
3. Let $EP_{perm} = \{p_e, p_s\}$ be the set of normalized *effective policies* associated with PS_{perm}, whose elements represent the endpoint and service effective policy respectively, computed as stated in WS Policy Attachment. Note that any element of EP_{perm} can be undefined (eg. you can have policies attached only at the service level), but we assume that there are no empty or null policies. Computing EP_{perm} and PS_{perm} involves miminal XML processing: fetching and merging the corresponding policy fragments (see Table 4) and transforming the resulting policy to the normal form expression.
4. Let EPP_{perm} be the *effective permission policy* for a permission *perm*. We define EPP_{perm} as the merging of all policies in EP_{perm}:

$$EPP_{perm} = \text{merge } (p_i) \mid p_i \in EP_{perm}$$

5. Architecture and Implementation

This section presents the architecture and implementation of our authorization system.

5.1 System Architecture

Figure 2 shows the system architecture. We use the well-known UML notation to show the various components of the architecture and their interactions. The UML-based architecture has three main actors:

- **Web Services** (right side): These are the PatientTrack, PatientRecordUS and PatientRecordSP services depicted and explained in Figure 1. All three of them publish their usage policies using WS-Policy, which are processed using WS-Policy Attachment model. According to our requirement for context-aware and fine-grained access control policies for individual service components, the access policy for port, port type, binding and/or service component is individually specified for each service. The latter two services are semantically equivalent, and share the same port type policy (the policy applicable to the abstract service interface).
- **Web application (PEP)** (bottom left): This is the Web application hosted at a Web server, and is the policy enforcement point (PEP) in the architecture. It provides secure access to healthcare services to its clients and issues Web service invocations to determine their authorizations to access the services. The Web application can either dynamically discover service providers, or have a prior agreement with them. In this architecture we assume the service agreement has been defined, and the PEP has the service descriptions in the form of WSDL documents. The Web application also has its own access control policy for user authentication and role assignment. The role and credential definitions used by the Web application and the service provider must match, and this is ensured by the existence of a prior service agreement.
- **X-GTRBAC PDP** (top left): This is the X-GTRBAC system which acts as the Policy Decision Point (PDP). It implements the WS-Policy profile for X-GTRBAC, accepting access control requests from the Web application and returning authorization decisions. It provides a SAML-based Web service interface for message exchange.

We now describe how the architecture allows communication between service providers, Web applications (PEP) and X-GTRBAC system (PDP). The communication uses the SAML profile for X-GTRBAC described in an earlier work [4]. The sequence of steps is as follows:

1. The Web application needs to register a service, so it accesses the WSDL URI (see UML <<use>> stereotype). Then, using the Permission Manager, it creates a permission corresponding to the service based on the WSDL description (Recall that a service corresponds to a permission in the WS-Policy profile of X-GTRBAC).
2. From the permission, the Policy Processor retrieves the service usage policies via attachments (see the UML "attachment" association), merges them and computes EPP_{perm}.

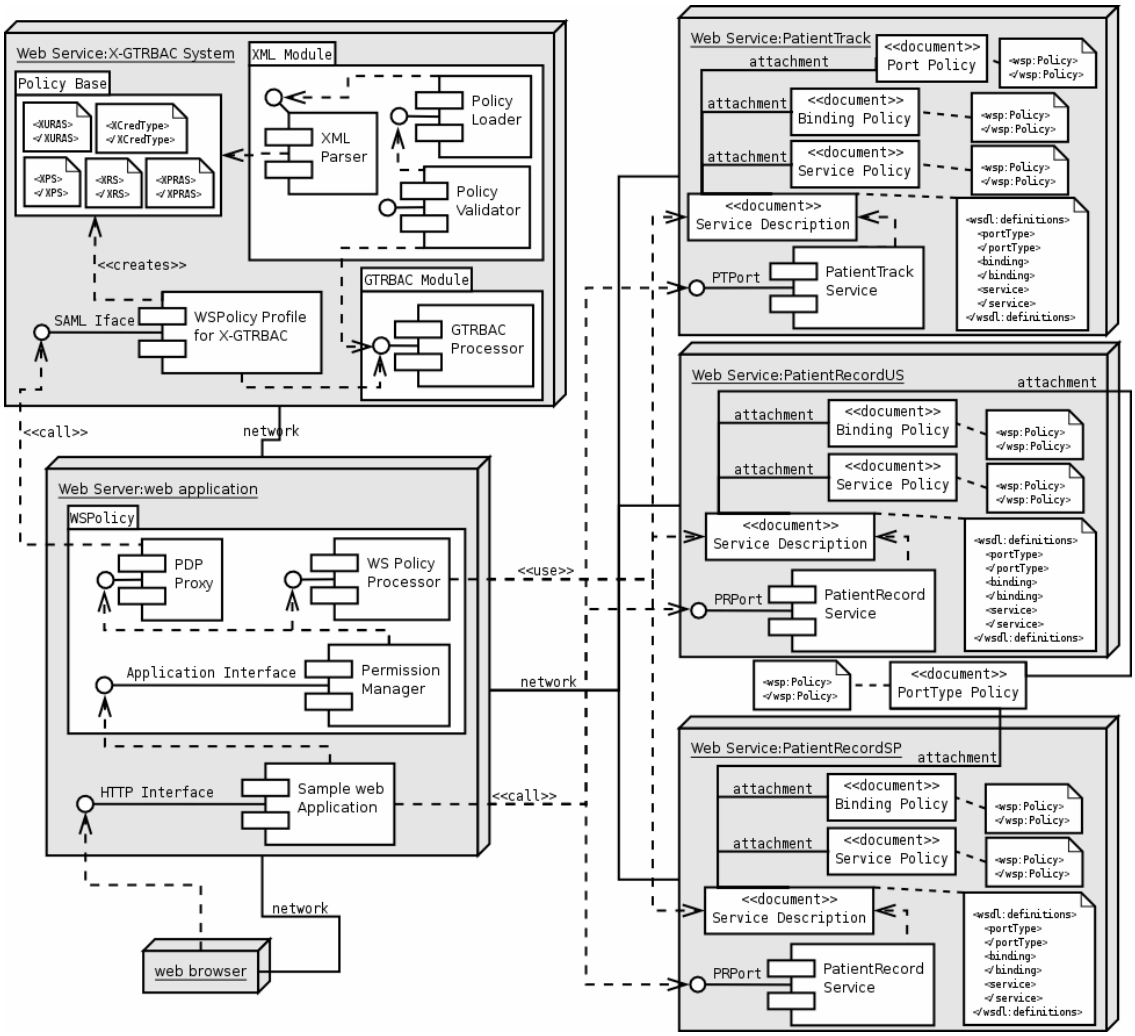

Figure 2: Architecture for the authorization system

3. When a user issues a request to access a service, the PDP proxy within the Web application prepares an access request to be sent to the actual PDP, encoding it as a SAML Authorization Decision Query (see UML <<call>> stereotype). The request asks for access to a service defined at a given port. It includes the port URI as the value of the <resource> attribute, HTTP "get" as the <Action> element, the user id as the <Subject> element, and a set of assertions including the user credential. The user credential includes the attributes that the user must possess in order to access the service. These assertions are included within the <Evidence> element of the SAML query. Along with the SAML query, the PDP proxy also sends a URI pointing to the location of the effective normalized WS-Policy file (i.e. EPP_{perm}).
4. Upon receiving the access request as a SAML query, the X-GTRBAC system consults a policy base to make the authorization decision (see UML <<creates>> stereotype). The policy base comprises of a set of XML files representing the RBAC policy. This set includes the permission definition as an XML Permission Sheet (XPS), a role definition as an XML Role Sheet (XRS), a credential definition as an XML Credential Definition Sheet (XCredTypeDef), a user-to-role assignment policy as an XML User-to-Role Assignment Sheet (XURAS), and a permission-to-role assignment policy as an XML Permission-to-Role Assignment Sheet (XPRAS). The syntax of the policy base is best visible by looking at the policy files for our example supplied in Appendix A[4]. We briefly describe below how the policy files are used in relation to the WS-Policy profile for X-GTRBAC:
    - A permission *P* is created in the XPS corresponding to the requested service. *P* represents access to the service at the defined port using a given HTTP verb, as defined in Section 3.3.1, and is built from the port URI and the <Action> element indicated in the SAML query.

[4] A detailed description of these policy files appears in [3].

- A credential type *CTu* is added to the XCredTypeDef. *CTu* is a user credential, and includes a set of attributes used by the X-GTRBAC system for a user-to-role assignment.
- A credential type *CTr* is added to XCredTypeDef. *CTr* is a role credential, and includes a set of attributes belonging to the role *R* which are used to define rules on permission-to-role assignment, as discussed in Section 3.3.2.
- A role *R* is created in the XRS. *R* is an internal role created specifically to access a specific service, and has an associated credential type *CTr* with a set of attributes.
- A user-to-role assignment policy is added to the XURAS. To assign a user to a role *R*, a credential of type *CTu* must be presented and evaluated against the set of rules included in the user assignment policy. In our system, this occurs at two stages: (i) the user is initially authenticated into a role by the Web application to access PatientTrack service, and (ii) the user is subsequently assigned another role by the PatientTrack service to access either of PatientRecordUS or PatientRecordSP service. This is actually a single sign-on scenario, where the authorization by the secondary service depends on the authorization provided by the first service. In case (i), the user provides the credentials at the time of login to the Web application, and in case (ii), the credentials are forwarded by the PatientTrack service, and includes an attribute that specifies the role currently assigned to the user. The information from primary user-to-role assignment then becomes the criterion of secondary user-to-role assignment. Overall, the assignment of a user to a role *R* (whether primary or secondary) is done based on the credentials of a user supplied as an assertion in the SAML query. The PDP is aware of the role and credential definitions, and uses them to automate the user-to-role assignment.
- A permission-to-role assignment policy is added to the XPRAS. To assign *P* to the role *R*, a credential of type *CTr* must be presented and evaluated against the set of rules included in the EPP_{perm}. Thus, *P* is automatically assigned to *R* if the evaluation of EPP_{perm} succeeds. Generating the service-to-role assignment policy using the constraints imposed on the service usage requires an XML transformation from the WS-Policy syntax to the assignment constraint syntax of X-GTRBAC according to the WS-Policy profile for X-GTRBAC, and we accomplish this using an XSL transformation.

5. The GTRBAC Processor evaluates the XPRAS (generated from the attribute assertions included in the EPP_{perm}), and, subject to a successful evaluation, assigns the permission *P* to *R*.
6. A response in the form of a SAML Authorization Decision Statement is prepared, including either the “permit” or the “deny” value, and sent to the Web application.
7. The PDP proxy inside the Web application reads the authorization decision. If it is “permit”, returns true, otherwise returns false.
8. The Web application enforces the policy: if the PDP returned true, the service invocation is performed and the requested resource is accessed using an HTTP:GET operation.

5.2. Implementation

In this section, we demonstrate the use of our authorization system in a real Web service environment. We have prototyped our system architecture using Web services model. The WS-Policy profile for X-GTRBAC has been implemented as a Java-based Web service, whereas the policy processing model of WS-Policy supported by WS-Policy Attachment has been implemented as a PHP-based class library. The most important classes of the WS-Policy package are depicted in Figure 3, and described below.

- **WSDL**: This class encapsulates access to Web service descriptions. Given the URI of the WSDL file, it provides methods to retrieve XML nodes representing services, ports, port types and bindings. All these elements are Policy Scopes that contain Policy Subjects (Service, Endpoint, Operation and Message).
- **WSPolicy**: This class represents policies. It is able to load a policy from a URI or from an XML document containing it, perform merge between two WSPolicy objects, and return the resulting normalized policy expression.
- **WSAttachment**: This class works with all attachments specified in a service description. It can extract policies by using attachment mechanisms (currently it supports the XML attachment). From the URI of a WSDL description, it is able to compute the Effective Policy for a given Policy Subject, returning a WSPolicy object. Policy Subjects are identified by name, and the corresponding XML elements are retrieved from the service description using a WSDL object.
- **XGTRBACPermission**: This class represents permissions. A permission provides access to the operations defined on a given port. This class stores all information required to actually invoke the service, and computes its own effective permission policy (EPP_{perm}) from the policies attached to the different service components according to the algorithm given in Section 4.2.
- **PermissionPool**: This class manages all permissions used by an application. It serves as front end to Web applications that use the pool as a factory to create new permissions (corresponding to services) by specifying the service invocation data, and ask the pool for permission usage.
- **PDPManager**: This class is responsible to instantiate the PDP that will take a decision about the usage of a given permission (i.e. a service). All PDPs implementations (whether local or remote) have to implement the IPDPProxy interface, thus providing code for the checkAccess method.
- **SAMLPDP**: This class acts as PDP proxy with a remote X-GTRBAC system. The checkAccess method generates a SAML Authorization Decision Query as discussed in Section 5.1, sends it to the real PDP, and parses the received SAML Authorization Decision Statement containing the access control decision.

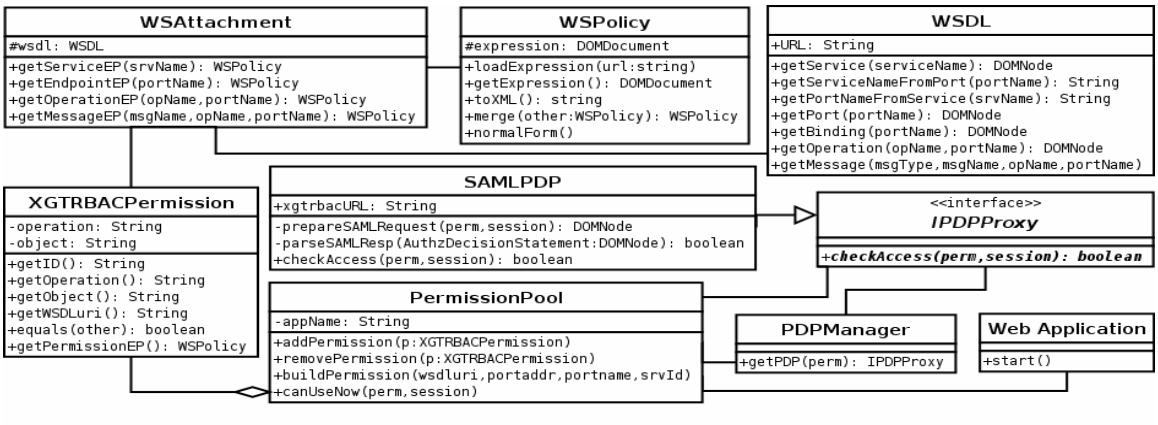

Figure 3: Class diagram for the WS-Policy package.

Our prototype implements the example scenario depicted in Figure 1. The Web application described in the example has been developed in PHP. In the current prototype, a basic underlying Web system interface is assumed, and is used as a mechanism to glue together contents provided by different services, perform some application-dependent computations, and present the information to the user. Our current implementation employs a PHP-based PEP (implementing the WS-Policy package) deployed at Carlos III University of Madrid in Spain and a Java-based PDP (implementing the WS-Policy profile for X-GTRBAC) at Purdue University in USA.

As already described in Section 1, the Web application allows physicians to track and view records for patients in the system based on the authorization of the physician using the PatientTrack service, and the two PatientRecord services, respectively. We now provide a discussion on their implementation.

- **PatientTrack**: This service provides a patient list to the authorized physician. For each entry in the list, it includes a patient id, together with the name of institution that has created a medical record for the patient. For each institution, the service also maintains a port URI for the PatientRecord service from where the medical record may be obtained (see Figure 4). This service defines a parameter-less operation, and the authorization is based on the requesting physician's credentials encoded in the SAML query itself.

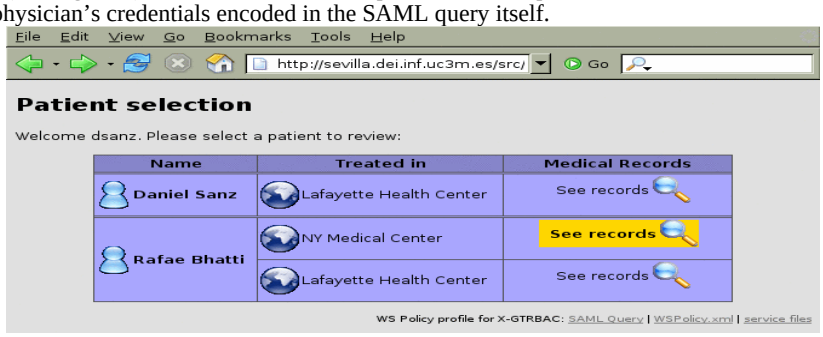

Figure 4: View of the patient list as result of Patient Track service invocation.

- **PatientRecord**: This set of services (PatientRecordUS and PatientRecordSP) provides medical records for a patient given the patient id. The records have a very simple structure. We assume that both services in this set share the same PatientRecord interface, though each one will provide at least one concrete implementation at a given port for that interface using an HTTP binding. The records are accessed through an HTTP:GET operation and displayed in the client browser (See Figure 5).

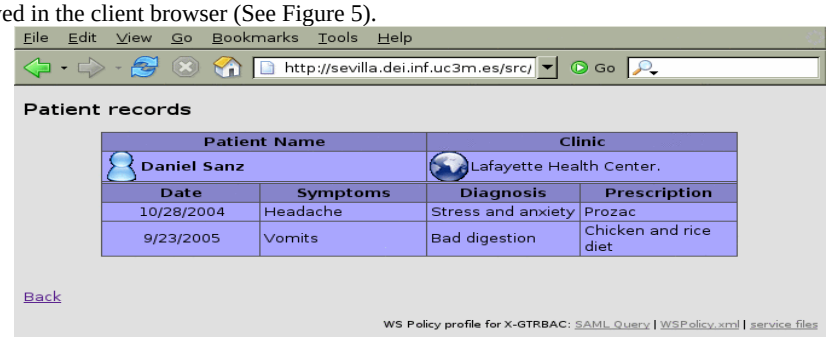

Figure 5: View of the patient record as result of Patient Record service invocation.

In the prototype, the tasks of the Web system interface are the following:

- **Service discovery**: The Web system discovers the WSDL URI of all services required to provide the functionality of the Web application. (In our current prototype, we do not address service discovery.)
- **Authentication**: The Web system provides a simple login page to access the top-level PatientTrack service. Users are authenticated by providing their authenticating credentials in the form of a set of (attribute, value) pairs.
- **Role assignment**: Each user is assigned a role within the system based on the supplied attributes. (As pointed out earlier, the credential and role definitions are shared between Web application and PDP.) The assigned role will be used to determine the authorization of the user to access the service.
- **Policy preparation**: With the help of WSPolicy package, the Web system gathers all policies from the corresponding attachments and prepares a SAML query including the user credentials. It then submits the query and the link to the merged policy URI to the X-GTRBAC PDP.
- **Policy enforcement**: The Web system enforces the policy according to the decision returned by the X-GTRBAC PDP. Thus, the Web service invocation only occurs if the access control decision so allows.
- **Content presentation**: The role of the Web system is limited to glue together all information pieces provided by different service invocations. This involves some basic mechanisms to create the policy base, compose the effective service policy, maintain the information state across service accesses and display the requested content using HTTP: GET, as has been discussed in the preceding sections.

Appendix A shows the policy files and the XML policy based created by the Web system on behalf of the Web application to provide access to the PatientTrack service (PatientRecordUS and PatientRecordSP are similar). It includes the service WSDL files (Figure A.1), the policies attached to the service components (Figure A.2), and the corresponding XML files that comprise the policy base for the X-GTRBAC system (Figure A.3). The overall Web application and the associated application and Web service files can be accessed at http://sevilla.dei.inf.uc3m.es/src/websystem/index.php. The X-GTRBAC PDP and associated policy files used by the system can be accessed at http://mmpc3.ecn.purdue.edu:8090/index-wspolicy.html.

6. Conclusions and Future Work

In this paper, we have proposed a policy-based authorization system that leverages an existing Web service policy processing model, WS-Policy, and integrates it with X-GTRBAC, an XML-based access control model to allow specification and processing of fine-grained context-aware authorization policies in dynamic Web services environment. The architecture is designed to support the WS-Policy Attachment specification, which allows attaching, retrieving and combining policies associated with various components of a Web service description. We presented an algorithm to compute the *effective access control policy* of a Web service based on its description document. The effective policy, represented as a normalized WS-Policy document, is then used by the X-GTRBAC system to evaluate an incoming access request. We presented an architecture of the authorization system, and also discussed our implementation prototype. Our authorization system has been implemented as a loosely coupled Web service, with logically distinct, heterogeneous PEP and PDP modules. The system incorporates SAML-based single-sign-on communication between multiple system modules, and thereby demonstrates the true promise of decentralized Web services architecture.

Our work can be extended in several directions. We indicated that our architecture supports policy processing for dynamically discovered services, but our current prototype assumes the existence of a prior agreement between the service provider and the PEP, and the PEP and the PDP. We would in our next prototype like to remove this assumption and implemented the suggested approach of using a SAML-based protocol to exchange policy definitions before the transaction. There are a few limitation of our authorization system that can be overcome in future work. Our policy processing algorithm does not currently consider the case of having conflicting policy alternatives in the normalized policy. As we indicated in the paper, this requires the notion of *conflict resolution*, which we would like to incorporate in our model in future. Finally, our existing Web system interface maps a single result page with a single Web service invocation, but certain situations may require complex pages where several invocations take place to provide more advanced computations and navigational structures. This has important applications in dynamic service composition. We would like to extend our basic Web system interface into a full-fledge hypermedia model to provide authorization support in such advanced Web service invocation scenarios. Such an extenson would allow us to adopt a similar approach to the one suggested in [19], thus facilitating the authorization policy design and integration with the rest of the Web system components.

References

[1] R. Bhatti, E. Bertino, A. Ghafoor, “A Trust-based Context-Aware Access Control Model for Web Services”, Distributed and Parallel Databases, Special Issue on Web Services, Vol. 18, No. 1, July 2005

[2] R. Bhatti, J. B. D. Joshi, E. Bertino, A. Ghafoor, “XML-Based Specification for Web Services”, IEEE Computer, Vol. 37, No. 4, April 2004

[3] R Bhatti, J. B. D. Joshi, E. Bertino, A. Ghafoor, "X-GTRBAC: An XML-based Policy Specification Framework and Architecture for Enterprise-Wide Access Control”, ACM Transactions on Information and System Security (TISSEC), Vol. 8, No. 2.

[4] R Bhatti, E. Bertino, A. Ghafoor, “An Integrated Approach to Federated Identity and Privilege Management in Open Systems”, Accepted for publication in Communications of the ACM. An earlier version is available online as CERIAS tech. report 2004-32.

[5] J. Hu, A. C. Weaver, “Dynamic, Context-aware Security Infrastructure for Distributed Healthcare Applications”, Proceedings of First Workshop on Pervasive Security, Privacy and Trust (PSPT), August 26, 2004.

[6] D. Kaminsky, “An Introduction to Policy for Autonomic Computing”, March 2005. http://www-128.ibm.com/developerworks/autonomic/library/ac-policy.html (Accessed: November 11, 2005)

[7] P. Nolan, “Understand WS-Policy Processing”, December, 2004. http://www-128.ibm.com/developerworks/webservices/library/ws-policy.html (Accessed: November 11, 2005)

[8] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, “Role Based Access Control Models”, IEEE Computer Vol. 29, No 2, February 1996

[9] E. Sirer, K. Wang, “An access control language for web services”, Proceedings of the seventh ACM symposium on Access control models and technologies, June 03 - 04, 2002, Monterey, CA.

[10] Web Services Description Language (WSDL 1.1), March 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315 (Accessed: November 11, 2005)

[11] Web Services Policy Framework (WS-Policy), September, 2004. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polfram/ (Accessed: November 11, 2005)

[12] Web Services Policy Attachment (WS-PolicyAttachment), September, 2004. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polatt/ (Accessed: November 11, 2005)

[13] Web Services Security (WS Security), April 2002. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-secure/ (Accessed: November 11, 2005)

[14] Web Services Trust Language (WS Trust), May, 2004. http://www-128.ibm.com/developerworks/library/specification/ws-trust/ (Accessed: November 11, 2005)

[15] Security in a Web Services World: A Proposed Architecture and Roadmap, April 2002. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-secmap/ (Accessed: November 11, 2005)

[16] Security Assertions Markup Language (SAML), August, 2004. http://xml.coverpages.org/saml.html (Accessed: November 11, 2005)

[17] Extensible Access Control Markup Language (XACML), February, 2005. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml (Accessed: November 11, 2005)

[18] XACML Profile for Web Services (WSPL), September 2003. http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf (Accessed: November 11, 2005)

[19] I. Aedo, P. Díaz, S. Montero, “A methodological approach for hypermedia security modelling”, Information and Software Technology, 2003, 45(5). 229–239.

Appendix A

```
<?xml version="1.0"?>
<definitions
    name="PatientTrack"
    targetNamespace="http://sevilla.dei.inf.uc3m.es/src/services/PTService/PTService.wsdl.xml"
    xmlns:tns="http://sevilla.dei.inf.uc3m.es/src/services/PTService/PTService.wsdl.xml"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    >
  <wsp:UsingPolicy Required="true"/>
  <types>
    <schema targetNamespace="http://sevilla.dei.inf.uc3m.es/src/services/PTService/PTService.wsdl.xml"
xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="patients">
         <complexType>
            <element name="patient" minOccurs="1" maxOccurs="unbounded">
            <complexType>
             <sequence>
                <element name="name" type="string"/>
                <element name="recordSet" minOccurs="1" maxOccurs="unbounded">
                   <complexType>
                     <element name="clinic" type="string"/>
                     <element name="provider">
                       <complexType>
                         <sequence>
                           <element name="wsdlUri" type="anyURI"/>
                           <element name="service" type="string"/>
                            <element name="portUri" type="anyURI"/>
                          </sequence>
                       </complexType>
                     </element> <!-- end provider -->
                   </complexType>
                 </element> <!-- end recordSet -->
              </sequence>
            </complexType>
          </element> <!-- end patient -->
        </complexType>
      </element> <!-- end patients -->
    </schema>
  </types>
  <message name="GetPatientListResponse">
    <part name="body" element="patients"/>
  </message>
  <portType name="PTPortType"
wsp:PolicyURIs="http://sevilla.dei.inf.uc3m.es/src/services/PTService/PTPortTypePolicy.xml">
    <operation name="GetPatientList">
      <output message="GetPatientListResponse"/>
    </operation>
  </portType>
  <binding name="PTHttpBinding" type="PTPortType">
    <http:binding verb="GET"/>
    <operation name="GetPatientList">
      <!-- location is the operation relative URI, which base uri is port uri -->
      <http:operation location="PTService.php"/>
      <output>
        <!-- this should be a MIME type representing XML documents -->
        <mime:content type="text/xml"/>
      </output>
    </operation>
  </binding>
  <service name="PTService"
wsp:PolicyURIs="http://sevilla.dei.inf.uc3m.es/src/services/PTService/PTServicePolicy.xml">
    <port name="PTPort" binding="PTHttpBinding"
wsp:PolicyURIs="http://sevilla.dei.inf.uc3m.es/src/services/PTService/PTPortPolicy.xml">
      <!-- port base addres, all operations are provided from this URI -->
      <http:address location="http://sevilla.dei.inf.uc3m.es/src/services/PTService/"/>
    </port>
  </service>
</definitions>
```

Figure A.1: WSDL for the PatientTrack service

Policy attached to the <port>

```
<wsp:Policy>
 <wsp:ExactlyOne>
  <wsp:All>
   <Predicate>
    <Operator>eq</Operator>
    <ParamName>system_load</ParamName>
    <FuncName>hasCredAttributeValue</FuncName>
    <RetValue>low</RetValue>
   </Predicate>
  </wsp:All>
 </wsp:ExactlyOne>
</wsp:Policy>
```

Policy attached to the <portType>

```
<wsp:Policy>
 <wsp:ExactlyOne>
  <wsp:All>
   <Predicate>
    <Operator>eq</Operator>
    <ParamName>location</ParamName>
    <FuncName>hasCredAttributeValue</FuncName>
    <RetValue>NewYork</RetValue>
   </Predicate>
  </wsp:All>
 </wsp:ExactlyOne>
</wsp:Policy>
```

Policy attached to the <service> tag

```
<wsp:Policy>
 <wsp:ExactlyOne>
  <wsp:All>
   <Predicate>
    <Operator>eq</Operator>
    <ParamName>priority</ParamName>
    <FuncName>hasCredAttributeValue</FuncName>
    <RetValue>high</RetValue>
   </Predicate>
  </wsp:All>
  <wsp:All/>
 </wsp:ExactlyOne>
</wsp:Policy>
```

Merged policy: EPP_{perm}

```
<wsp:Policy>
 <wsp:ExactlyOne>
  <wsp:All>
   <Predicate>
    <Operator>eq</Operator>
    <ParamName>system_load</ParamName>
    <FuncName>hasCredAttributeValue</FuncName>
    <RetValue>low</RetValue>
   </Predicate>
   <Predicate>
    <Operator>eq</Operator>
    <ParamName>location</ParamName>
    <FuncName>hasCredAttributeValue</FuncName>
    <RetValue>NewYork</RetValue>
   </Predicate>
   <Predicate>
    <Operator>eq</Operator>
    <ParamName>priority</ParamName>
    <FuncName>hasCredAttributeValue</FuncName>
    <RetValue>high</RetValue>
   </Predicate>
  </wsp:All>
  <wsp:All>
   <Predicate>
    <Operator>eq</Operator>
    <ParamName>system_load</ParamName>
    <FuncName>hasCredAttributeValue</FuncName>
    <RetValue>low</RetValue>
   </Predicate>
   <Predicate>
    <Operator>eq</Operator>
    <ParamName>location</ParamName>
    <FuncName>hasCredAttributeValue</FuncName>
    <RetValue>NewYork</RetValue>
   </Predicate>
  </wsp:All>
 </wsp:ExactlyOne>
</wsp:Policy>
```

Figure A.2: Individual policies attached to PatientTrack service definition (left) are merged to produce the normalized WS-Policy (EPP_{perm}) for the service (right)

```
<?xml version="1.0" encoding="UTF-8" ?>
<XCredTypeDef xctd_id="LibElseXCTD">
 <CredentialType cred_type_id="LEResPT" type_name="LibElseResPT">
  <AttributeList>
   <Attribute name="role" type="string" usage="mand" />
 </AttributeList>
</CredentialType>
 <CredentialType cred_type_id="LERolePTP" type_name="LibElseRolePTP">
  <AttributeList>
  <Attribute name="system_load" type="string" usage="mand" />
  <Attribute name="location" type="string" usage="mand" />
  <Attribute name="priority" type="string" usage="opt" />
 </AttributeList>
 </CredentialType>
</XCredTypeDef>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<XRS xrs_id="LibElseXRS">
 <Role role_id="rPTP" role_name="PTPhysician">
  <CredType cred_type_id="LERolePTP"
            type_name="LibElseRolePTP">
   <CredExpr>
    <Attribute
            name="location">NewYork</Attribute>
    <Attribute
            name="system_load">low</Attribute>
    <Attribute name="priority">low</Attribute>
   </CredExpr>
  </CredType>
 </Role>
</XRS>
```

```
 <?xml version="1.0" encoding="UTF-8" ?>
 <XPS xps_id="LibElseXPS">
  <Permission perm_id="LEPTService">
   <Object object_type="port"
object_id="http://sevilla.dei.inf.uc3m.es/src/services/PTService/" />
   <Operation context="saml:ghpp">GET</Operation>
  </Permission>
 </XPS>
```

```
 <?xml version="1.0" encoding="UTF-8" ?>
 <XURAS xuras_id="LibElseXURAS">
 <URA ura_id="uraPTP" role_name="PTPhysician">
  <AssignUsers>
   <AssignUser user_id="any">
    <AssignConstraint>
     <AssignCondition cred_type="LibElseResPT">
      <LogicalExpr>
       <Predicate>
        <Operator>eq</Operator>
        <FuncName>
              hasCredAttributeValue</FuncName>
        <ParamName order="1">degree</ParamName>
        <RetValue>MD</RetValue>
       </Predicate>
       <Predicate>
        <Operator>eq</Operator>
        <FuncName>
           hasCredAttributeValue</FuncName>
        <ParamName
          order="1">affiliation</ParamName>
        <RetValue>USAMed</RetValue>
       </Predicate>
      </LogicalExpr>
     </AssignCondition>
    </AssignConstraint>
   </AssignUser>
  </AssignUsers>
 </URA>
</XURAS>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<XPRAS xpras_id="LibElseXPRAS">
<PRA role_name="PTServiceCustomer"
pra_id="praPTServiceCustomer">
 <AssignPermissions>
  <AssignPermission perm_id="LEPTService">
   <AssignConstraint>
    <AssignCondition cred_type="LibElseRolePT">
     <LogicalExpr op="OR">
      <Predicate>
       <LogicalExpr op="AND">
        <Predicate>
         <Operator>eq</Operator>
         <FuncName>hasCredAttributeValue</FuncName>
         <ParamName order="1">priority</ParamName>
         <RetValue>high</RetValue>
        </Predicate>
        <Predicate>
         <Operator>eq</Operator>
         <FuncName>hasCredAttributeValue</FuncName>
         <ParamName order="1">system_load</ParamName>
         <RetValue>low</RetValue>
        </Predicate>
        <Predicate>
         <Operator>eq</Operator>
         <FuncName>hasCredAttributeValue</FuncName>
         <ParamName order="1">location</ParamName>
         <RetValue>NewYork</RetValue>
        </Predicate>
       </LogicalExpr>
      </Predicate>
      <Predicate>
       <LogicalExpr op="AND">
        <Predicate>
         <Operator>eq</Operator>
         <FuncName>hasCredAttributeValue</FuncName>
         <ParamName order="1">system_load</ParamName>
         <RetValue>low</RetValue>
        </Predicate>
        <Predicate>
         <Operator>eq</Operator>
         <FuncName>hasCredAttributeValue</FuncName>
         <ParamName order="1">location</ParamName>
         <RetValue>NewYork</RetValue>
         </Predicate>
        </LogicalExpr>
      </Predicate>
     </LogicalExpr>
    </AssignCondition>
   </AssignConstraint>
  </AssignPermission>
 </AssignPermissions>
</PRA>
</XPRAS>
```

Figure A.3: X-GTRBAC policies for the Patient Track service
(Note that XPRAS is a result of XSL transformation on EPP_{perm} in Figure A.2)