An Authorization Architecture for Web Services

Sarath Indrakanti and Vijay Varadharajan

Information and Networked Systems Security Research,
Department of Computing, Macquarie University,
Sydney, NSW, 2109, Australia
{sindraka, vijay}@ics.mq.edu.au

Abstract. This paper considers the authorization service requirements for the service oriented architecture and proposes an authorization architecture for Web services. It describes the architectural framework, the administration and runtime aspects of our architecture and its components for secure authorization of Web services as well as the support for the management of authorization information. The proposed architecture has several benefits. It is able to support legacy applications exposed as Web services as well as new Web service based applications built to leverage the benefits offered by the service oriented architecture; it can support multiple access control models and mechanisms and is decentralized and distributed and provides flexible management and administration of Web services and related authorization information. The proposed architecture can be integrated into existing middleware platforms to provide enhanced security to exposed Web services. The architecture is currently being implemented within the .NET framework.

1 Introduction

In general, security for the Service Oriented Architecture (SOA) [1] is a broad and complex area covering a range of technologies. At present, there are several efforts underway that are striving to provide 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 and IPSec. There are also related security functionalities such as XML Signature and XML Encryption and their natural extensions to integrate these security features into technologies such as SOAP and WSDL.

WS-Security specification [2] describes enhancements to SOAP messaging to provide message integrity, confidentiality and authentication. There is also work on XKMS defining interfaces to key management and trust services based on SOAP and WSDL. However, while there is a large amount of work on general access control and more recently on distributed authorization [3][4] research in the area of authorization for Web services is still at an early stage. There is not yet a specification or a standard for Web services authorization. There are attempts by different research groups to define authorization frameworks and policies for Web services [5][6][7][8][9]. Currently most Web service based applications, having gone through the authentication process, make authorization

S. Jajodia and D. Wijesekera (Eds.): Data and Applications Security 2005, LNCS 3654, pp. 222–236, 2005.

decisions using application specific access control functions. This results in the practice of frequently re-inventing the wheel and motivates us to have a closer look at authorization requirements for the SOA.

1.1 Authorization Requirements for the SOA

Broadly speaking, the SOA is made up of Web services and business workflows built using Web services. These workflows are called business processes [10]. Figure 1 shows the layers comprising the SOA. In general, Web services and business processes have different authorization requirements. Authorization services for business processes must provide orchestration services to coordinate the authorization decisions from individual partner's authorization policy evaluators. Each partner must be allowed to control its own authorization policies and also not require disclosing them to all the partners. Even in cases where the binding to actual end-points of partner services happens dynamically at runtime, the authorization architecture must be able to orchestrate the partners' authorization policy evaluators and arrive at an authorization decision.

Authorization services for the Web services layer have different design requirements as Web services present a complex layered system. For instance, a service could be a front-end to an enterprise system and the enterprise system accesses information stored in databases and files. Web services may be used by enterprises to expose the functionality of legacy applications to users in a heterogeneous environment. Or new business applications could be written to leverage benefits offered by the SOA.

A Web service's method may invoke one or more abstract operations, each operation having its own responsible Authorization Policy Evaluator (APE). For instance, a purchase order service may have three methods – submit order, cancel order and confirm order as shown in Figure 2. Submit order and cancel order methods perform two operations say a Web operation and a mail operation,

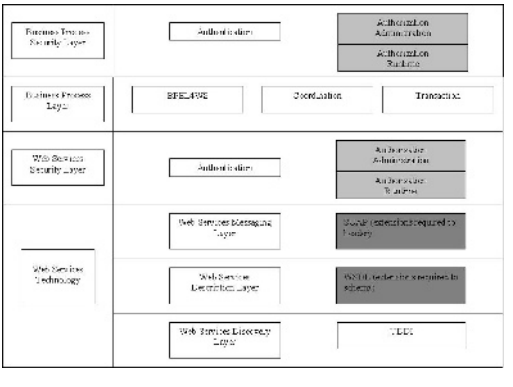

Fig. 1. Layers in the Service Oriented Architecture

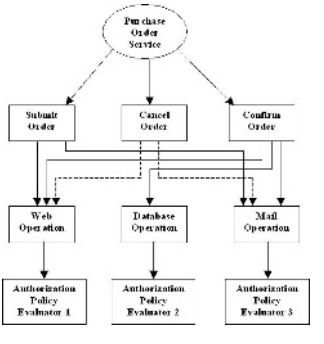

Fig. 2. WS operations and Authorization Policy Evaluators

and confirm order performs say three operations a Web operation, a database operation and a mail operation. Each operation may have its own responsible APE to control access to the operation.

We envisage an authorization architecture for the SOA to provide extensions to both the security layers of Web services as well as business processes as indicated by the grey colored boxes in Figure 1. In this paper, we propose our Web Services Authorization Architecture (WSAA). WSAA provides authorization services for Web services. It extends the Web services security layer in the SOA. We also extend the Web services description and messaging layers (indicated by dark-grey colored boxes in Figure 1) to provide authorization support for Web services. We will describe our authorization services for the business process layer in a separate paper.

In section 2, we outline our design principles and goals underlying the design of WSAA. In Section 3, we give an overview of WSAA and discuss the design of our architecture. In Section 4, we briefly describe the extensions required to the Web service Description and Messaging Layers. We briefly describe the authorization algorithms used by WSAA in Section 5 and give a brief introduction to our implementation work in Section 6. We highlight the benefits of our architecture in Section 7 and discuss some related work in Section 8. Finally, we give some concluding remarks in Section 9.

2 Design Principles

In this section, we outline some of the key design principles and goals behind our proposed architecture.

(a) Different Access Control (AC) Models: WSAA should be able to support a range of AC models. This is necessary as it is not realistic to expect every Web service based application to use the same AC model. In fact, where Web services are used to expose the functionality of legacy enterprise applications, it is likely that organizations will prefer to use their currently existing AC mechanisms that they have been using before exposing them as Web services. Therefore, we believe an authorization architecture must be generic enough to support different AC models including traditional Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role Based Access Control (RBAC), and Certificate based AC models.

(b) Authorization Architecture Design: Conceptually, there are two stages for authorization [3] namely the administration phase and the runtime or the evaluation phase. The administration phase involves facilities and services for the specification of authorization policies, updating and deleting of policies and their administration. The runtime phase is concerned with the use of these authorization policies in the evaluation of the access requests.

(c)Authentication: In WSAA, we assume that authentication is a prerequisite to authorization and that a principal (client) and its request has undergone some reliable authentication service before being subjected to the authorization service.

(d) Authorization Policy Evaluation: Every AC mechanism that is supported by WSAA defines an interface or end-point (defines the input parameters as

well as the output result) to the Authorization Policy Evaluator (APE). APE is responsible for achieving end-point decisions on access control. An APE also defines a set of abstract operations such as Web operations, database operations or file operations to which it provides access control.

Note: A Web service method is a high-level task that the Web service exposes to its clients. WSAA provides access control indirectly to operations performed by a Web service method. We map each Web service method to a set of abstract operations. One or more of these operations are then mapped to an APE, which is responsible for controlling access to these operations. These abstract operations help security administrators and Web service developers have a common ground to map the resources (Web services themselves and any other resources such as databases, files, applications, etc.) to APEs and therefore to authorization policies.

e) Authorization Policies: Languages have long been recognized in computing as ideal vehicles for dealing with expression and structuring of complex and dynamic relationships. A language-based approach is helpful for not only supporting a range of AC policies but also in separating out the policy representation from policy enforcement. Hence one of our design principles is to enable the support of a range of policy languages for specifying AC policies. The policy language(s) used may support both fine-grained as well as coarse-grained policies depending on the requirement. The respective authorization policy administrators manage these policies.

(f) Authorization Credentials: WSAA provides support for defining what AC related credentials are required and how to collect them. Some AC mechanisms may pull the credentials from the respective authorities and send them to the responsible APEs. Other AC mechanisms may expect the principal to collect the credentials from the respective authorities and send them to the responsible APEs. WSAA supports both the push and pull model approaches to credentials collection and decision-making.

3 Design of the Proposed Architecture

3.1 Overview

Let us now first briefly describe an overview of the proposed architecture (refer to Figure 3). WSAA comprises of an administrative domain and a runtime domain. We manage Web services in the administration domain by arranging them into collections and the collections themselves into a hierarchy. We provide administration support to manage a collection of Web services. We also provide support for the arrangement (adding, removing) of Web services within the collections and the movement of Web services within collections. Authorization related components can be managed in the administration domain. Also security administrators can assign a set of APEs to authorize requests to Web services. To make the authorization process efficient, we have a runtime domain where the authorization related information such as what credentials are required to

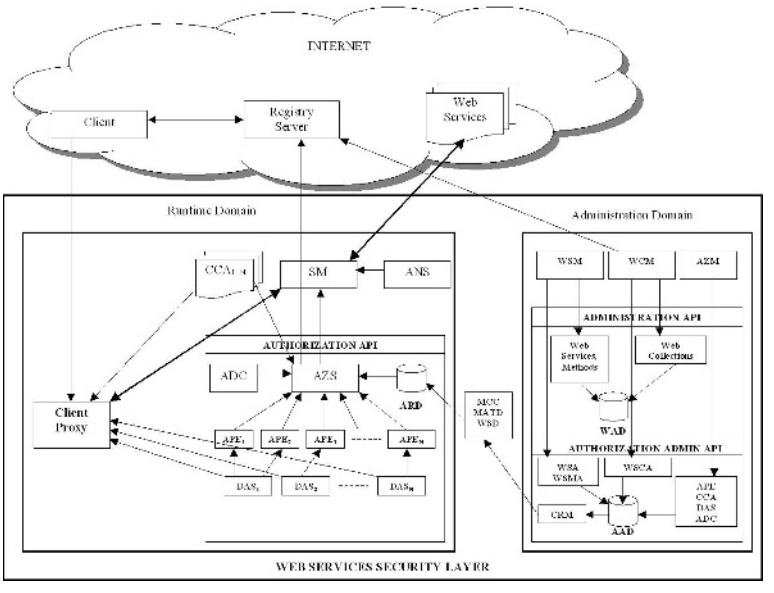

Fig. 3. Web Services Authorization Architecture (WSAA)

invoke a particular Web service and how to collect those credentials, is compiled and stored. This information is automatically compiled from time to time when necessary using the information from the administration domain and it can be readily used by components in the runtime domain.

The Registry Server located anywhere in the Internet is responsible for maintaining relations between services and their service providers. When a client requests the Registry Server (UDDI directory for instance) for a specific service, the latter responds with a list of Web services that implement the requested service.

3.2 System Components

We define the set of *Certificate and Credential Authorities*, *Dynamic Attribute Services*, *Authorization Policy Evaluators* and *Authorization Decision Composers* as objects in our system. The *Authorization Manager* (AZM) for an organization is responsible to manage these components. S/he uses the Authorization Administration API (AA-API) to manage them and the related data is stored in the *Authorization Administration Database* (AAD). These objects are formally defined in definitions 1–4.

Certificate and Credential Authority (CCA) is responsible to provide authentication certificates and/or authorization credentials required to authenticate and/or authorize a client.

Dynamic Attribute Service (DAS) provides system and/or network attributes such as bandwidth usage and time of the day. A dynamic attribute

may also express properties of a subject that are not administered by security administrators. For example, a nurse may only access a patient's record if s/he is located within the hospital's boundary. A DAS may provide the nurse's "location status" attribute at the time of access control. Dynamic attributes' values change more frequently than traditional "static" authorization credentials. Unlike authorization credentials, dynamic attributes must be obtained at the time an access decision is required and their values may change within a session.

Authorization Policy Evaluator (APE) is responsible for making authorization decision on one or more abstract system operations. Every APE may use a different access control mechanism and a different policy language. However, it defines an interface for the set of input parameters it expects (such as subject (client) identification, object information, and the authorization credentials) and the output authorization result.

Authorization Decision Composer (ADC) combines the authorization decisions from APEs using an algorithm that resolves authorization decision conflicts and combines them into a final decision.

Definition 1. Certificate and Credential Authority (CCA)
We define Certificate and Credential Authority (CCA) as a tuple cca = {i, l, CR, pa, ra(pa)}, where i is a URN, l is a string over an alphabet Σ^* representing a network location of the CCA such as a URL, CR is the set of credentials cca provides, pa is an input parameter representing a subject, ra uses pa and gives out an output (result) that is the set of credentials for the subject.

Definition 2. Dynamic Attribute Service
We define Dynamic Attribute Service as a tuple das = {i, l, AT, pd, rd(pd)}, where i is a URN, l is a string over an alphabet Σ^* representing a network location of the DAS such as a URL, AT is the set of attributes that das provides, pd is input parameter(s) representing attribute(s) name, rd uses pd and gives out an output (result) that is the value of the attribute(s).

Definition 3. Authorization Policy Evaluator
We define Authorization Policy Evaluator as a tuple ape = {i, l, pe, re(pe), OP, DAS, CCA}, where i is a URN, l is a string over an alphabet Σ^* representing a network location of the APE such as a URL, pe is the set of input parameters such as subject and object details, re is a function that uses pe and gives out an output (result) of authorization decision. OP is the set of abstract system operations for which ape is responsible. DAS is the set of dynamic attribute services responsible for providing dynamic runtime attributes to ape. ape uses these attributes to make authorization decisions. CCA is the set of certificate and credential authorities that provide the credentials required by ape.

Definition 4. Authorization Decision Composer
We define Authorization Decision Composer as a tuple adc = {i, l, a, pc, rc(pc)}, where i is a URN, l is a string over an alphabet Σ^* representing a network location of the ADC such as a URL, a is the name of a pre-defined algorithm adc uses to combine the decisions from the individual authorization policy evaluators. pc is an input parameter representing the decisions from individual APEs, rc uses

pc and authorization decision composition algorithm a to combine the decisions and gives out an output (result) that is the value of the final authorization decision.

The runtime domain consists of the *Client Proxy*, *Security Manager*, *Authentication Server* and the *Authorization Server* components.

Client Proxy (CP) collects the required authentication and authorization credentials from the respective authorities on behalf of the client before sending a Web service request and handles the session on behalf of the client with a Web service's Security Manager component.
Security Manager (SM) is a runtime component responsible for both authentication and authorization of the client. A client's CP sends the necessary authentication and authorization credentials to the SM. It is responsible for managing all the interactions with a client's CP. It uses the Authorization API to invoke the Authorization Server.
Authentication Server (ANS) receives the authentication credentials from SM and uses some mechanism to authenticate the client. We treat ANS as a black box in our architecture as our focus in this paper is on authorization of the client. We included this component in the Web services security layer for completeness.
Authorization Server (AZS) decouples the authorization logic from application logic. It is responsible for locating all the APEs involved, sending the credentials to them and receiving the authorization decisions. Once all the decisions come back, it uses the responsible ADCs to combine the authorization decisions. Where required, AZS also collects the credentials and attributes on behalf of clients from the respective CCAs and DASs.

3.3 Web Services Model

We consider a Web service model based on the model discussed in [11] where Web Service, Web Service Method and Web Service Collection are viewed as objects (definitions 5–7). Web service collections are used to group together a set of possibly related Web service objects.

Definition 5. Web Service
We define a Web Service as a tuple ws = {i, b, l, S, OP_{ws}, M, MD, wsm, sm}, where i is a non-empty string over an alphabet Σ^* representing a globally unique identifier such as a URN, b is a string over an alphabet Σ^* representing a network protocol binding such as SOAP over HTTP, l is a string over an alphabet Σ^* representing a network location such as a URL, S is a finite set of states representing the internal state of the object at a given time, OP_{ws} is the set of abstract operations performed by the methods of the ws object. M is the set of supported Web service methods, MD is the set of metadata providing additional description for ws, wsm is the Web Service Manager responsible for managing ws object. sm is the Security Manager responsible for ws object. S, M, OP_{ws} or MD can be the empty set ϕ.

Definition 6. Web Service Method
We define a Web Service Method as a tuple m = {i, ws, OP_{wsm}, pm, rm(pm), MD}, where i is a URN, ws is the Web service object the method belongs to, OP_{wsm} is the set of abstract operations wsm performs. OP_{wsm} is a subset of the set OP_{ws} defined in the ws object. pm is the set of input parameters, string over an alphabet Σ^*, rm is a function $\Sigma^* \rightarrow \Sigma^*$ that maps pm onto a result string over an alphabet Σ^* representing the output (result) or return value(s) of a computation. pm and rm(pm) may be the empty string ϵ. MD is a set of metadata providing additional description for wsm. OP_{wsm} or MD can be the empty set ϕ. A wsm has to be a member of exactly one ws.

Definition 7. Web Service Collection
We define a Web Service Collection (WSC) as a tuple wsc = {i, WS, $WSC_{CHILDREN}$, p, MD, wcm, sm}, where i is a URN, WS is a finite set of (possibly related) Web service objects in wsc, $WSC_{CHILDREN}$ is a finite set of Web service collections that are children of wsc, p is the parent WSC (a WSC can have only one parent collection), MD is a finite set of metadata providing additional description and semantics for wsc. sm is the security manager responsible for wsc. wcm is the Web service Collection Manager responsible for wsc. sm is null for all Web service collections in a hierarchy except for the root Web service collection, or the one without a parent p. A root wsc object's sm is responsible for authentication and authorization of requests to all the ws objects under its descendant collections. Figure 4 shows an example of a hierarchy of Web service collections.

3.4 Authorization Administration and Policy Evaluation

A *Web Service Manager* (WSM) is responsible to manage the authorization information for the Web services s/he is responsible for. We consider a Web service

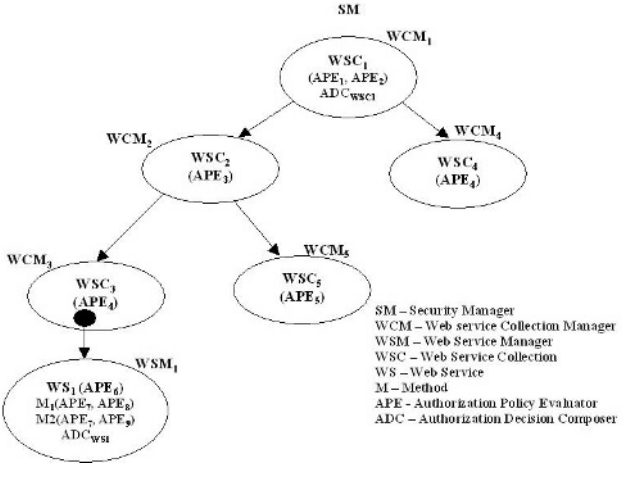

Fig. 4. Web Service Collection Hierarchy

method to be a high-level task that is exposed to clients. Each task (method) is made up of a number of system operations. These operations can be of different abstract types as shown in the example in Figure 2. It is reasonable to assume a WSM knows the set of tasks a Web service under his/her control performs. Similarly a WSM knows the set of operations each of these tasks (methods) perform. Using the APE definitions from AAD (database), WSM associates APEs to Web services and their methods. This association is made in the *Web Service Authorization* and the *Web Service Method Authorization* objects. WSM uses the AA-API to create and manage these objects. Similarly, a *Web service Collection Manager* (WCM) manages APE and authorization decision composer (ADC) information (using AA-API) in a separate tuple called *Web Services Collection Authorization* (WSCA) for all the collections s/he manages. We formally define these objects in definitions 8–10. These objects are stored in AAD.

Similar to Web service methods, a Web service can also have one or more APEs responsible for the Web service itself. Web service level policies are first evaluated before its method level authorization policies are evaluated. A Web service's APEs evaluate Web service level authorization policies. These policies will typically not be as fine-grained as method level policies. A WSM may choose to create a new ADC for one or more Web services s/he manages or may decide to use one from the set of existing ADCs from AAD if it serves the purpose.

Similar to Web services and their methods, a Web service collection can also have one or more APEs responsible for authorizing access to the collection itself. Collection level policies are first evaluated before a Web service's policies are evaluated. A Web service collection's APEs evaluate collection level policies. These policies will typically be course-grained when compared to the Web service and Web service method level policies. Every root Web service collection has an ADC associated with it responsible for combining the decisions from all APEs involved. The coarse-grained authorization policies for all the relevant ancestor Web service collections (of an invoked Web service) are first evaluated, followed by the Web service level policies and finally the fine-grained Web service method level policies are evaluated. For example (refer to Figure 4), when a client invokes WS_1's method M_1, WSC_1's authorization policies are first evaluated by APE_1 and APE_2, followed by WSC_2 (APE_3) and then WSC_3 (APE_4) policies. If APE_1, APE_2, APE_3 and APE_4 give out a positive decision, WS_1's authorization policies are evaluated by APE_6. If APE_6 gives out a positive decision, then finally M_1's authorization policies are evaluated by APE_7 and APE_8. WS_1's ADC, ADC_{WS_1} combines the decisions from APE_6, APE_7 and APE_8 and if the final decision is positive, WSC_1's ADC, ADC_{WSC_1} combines the decisions from APE_1, APE_2, APE_3, APE_4 and ADC_{WS_1}. If the final decision from ADC_{WSC_1} is positive, the client will be able to successfully invoke WS_1's method M_1.

Definition 8. Web Service Method Authorization
We define Web Service Method Authorization as a tuple wsma = {i, wsm, APE_{wsm}}, where i is a URN, wsm is the method to which wsma object is

defined. APE_{wsm} is the set of APEs responsible for authorizing requests from a client to wsm.

Definition 9. Web Service Authorization
We define Web Service Authorization as a tuple wsa = {i, ws, APE_{ws}, adc_{ws}}, where i is a URN, ws is the Web service to which wsa is defined. APE_{ws} is the set of APEs responsible for authorizing requests from a client to ws. adc_{ws} is an ADC for ws. It is responsible to combine the decisions from APEs (in the set APE_{ws}).

Definition 10. Web Service Collection Authorization
We define Web Service Collection Authorization as a tuple wsca = {i, wsc, APE_{wsc}, adc_{root}}, where i is a URN, wsc is the Web service collection for which wsca object is defined. APE_{wsc} is the set of APEs responsible for wsc. adc_{root} is an ADC for wsc. If wsc is not a root Web service collection, then adc_{root} is null. In other words, adc_{root} exists only for a root wsc.

3.5 Runtime Authorization Data

We addressed who assigns (and how) APEs and ADCs for Web services and Web service collections. The next question is, at runtime, how does a client know (where necessary) how to obtain the required authorization credentials and dynamic runtime attributes before invoking a Web service? What are the responsible APEs (and the credentials and attributes they require), CCAs (the credentials they provide) and DASs (the attributes they provide)? How does the Authorization Server (AZS) know what the set of responsible ADCs (adc_{ws} and adc_{root}) for a particular client request is?

To answer these questions, we have an Authorization Runtime Database (ARD) in the runtime domain. ARD consists of the runtime authorization related information required by clients and the Authorization Server. *Credential Manager* (CRM) is an automated component that creates and stores the authorization runtime information (using CRM algorithm) in ARD using the information from WAD (defined in section 3.6) and AAD. The runtime authorization information consists of three tuples defined in definitions 11–13. CRM is invoked from time to time, when a Web service object is added to or removed from a collection, moved within a hierarchy of collections or when the shape of the tree itself changes, to update these tuples in ARD. Refer to [12] for the CRM algorithm.

Definition 11. Method-Credential-CCA tuple
We define the Method-Credential-CCA tuple as mcc = {i, wsm, CR, cca, ape}, where i is a URN, wsm is a Web service method to which the tuple is defined, CR is the set of credentials to be obtained from the CCA cca to get authorized to invoke wsm. This means each wsm object can have one or more of these (tuple) entries in ARD. ape is the APE that requires these credentials.

Definition 12. Method-Attribute-DAS tuple
We define Method-Attribute-DAS tuple as matd = {i, wsm, AT, das, ape}, where i is a URN, wsm is a Web service method to which the tuple is defined, AT is

the set of attributes to be obtained from the DAS das. This means each wsm object can have one or more of these (tuple) entries in ARD. ape is the APE that requires these attributes.

Definition 13. WS-ADC tuple
We define WS-ADC tuple as wsd = {i, ws, adc_{ws}, adc_{root}}, where i is a URN, ws is a Web service, adc_{ws} is the ADC for ws. adc_{root} is the ADC for the root Web service collection in which ws is located.

3.6 Web Services Administration

A Web Service Manager (WSM) manages Web services and Web service methods and a Web service Collection Manager (WCM) manages Web service collections using the Administration API (see Figure 3). These objects are stored in the Web service Administration Database (WAD).

To effectively manage the collections, we arrange a set of related Web Service Collection (WSC) objects in a tree-shaped hierarchy as shown in Figure 4. Each WSC in the hierarchy has a responsible Web service Collection Manager (WCM). There is only one Security Manager for a hierarchy of WSCs. In a WSC hierarchy tree, the root WSC's manager is called the Root Web service Collection Manager (RWCM). A RWCM is responsible for providing the Security Manager details (such as its location) in the WSDL statement of every Web service located under the collections s/he manages.

Let us consider an organization with a single hierarchy (such as the one shown in Figure 4) of Web service collections. In Figure 4, the root WSC is WSC_1 and the RWCM is WCM_1. We can consider a newly initiated system to simply consist of the root WSC, WSC_1 and a few Web Service (WS) objects under it managed by WCM_1. WCM_1 can add new WS objects from WAD into WSC_1. S/he can delete or move WS objects within the collections s/he is responsible for. There are other issues to consider such as 1) Who decides the location of a WS object (and how is the location changed)? 2) Who decides the shape of the tree itself? There are various design choices to consider to answer these questions.

Due to space limitations, we have not included the discussion on such design choices in this paper. We refer the reader to [12] for a detailed discussion on Web services administration features provided by our architecture.

4 Extensions to the Description and Messaging Layers

WS-AuthorizationPolicy statement: We extend WSDL (description layer) to include a Web service's Authorization Policy as well as the location of its Security Manager. WS-SecurityPolicy [13] statement consists of a group of security policy "assertions", that represent a Web Service's security preference, requirement, capability or other property. Similarly, we define *WS-AuthorizationPolicy* as a statement that contains a list of authorization assertions. The assertions include which credentials (and from which CCA) and attributes (and from

which DAS) a client's CP has to collect before invoking a Web Service. WS-PolicyAttachment [14] standard can be used to link the WS-AuthorizationPolicy to a Web Service's WSDL statement.

Security Manager Location: When a client wants to invoke a Web service WS_1, its Client Proxy requires its Security Manager's location. Therefore, we need to give this information in WS1's WSDL statement. We introduce a new element *SecurityManager* to the WSDL document that encapsulates the Security Manager location information required by the Client Proxies.

SOAP Header Extension: We provide extensions to the SOAP header (messaging layer) to carry authorization related credentials and attributes. WS-Security [2] enhancements for confidentiality, integrity and authentication of messages have extended SOAP header (SOAP-SEC element) to carry related information. Similarly we suggest extending SOAP header to carry authorization credentials and attributes to carry authorization related information. When a client wants to invoke a Web service object, its client proxy creates an authorization header object and adds it to SOAP Header before making a SOAP request.

Refer to [12] for XML schema skeletons for WS-AuthorizationPolicy, extended WSDL statement and extended SOAP Header. We have not included the schemas in this paper due to space restrictions.

5 Authorization Algorithms

WSAA supports three algorithms. The first, push-model algorithm supports authorizations where a client's Client Proxy, using WS-AuthorizationPolicy, collects and sends the required credentials (from CCAs) and attributes (from DASs) to a Web service's Security Manager. The second, pull-model algorithm supports authorizations where the AZS itself collects the required credentials and APEs collect the required attributes. The third, combination-model supports both the push and pull models of collecting the required credentials and attributes.

An organization must deploy one of these algorithms depending on the access control mechanisms used. If all the access control mechanisms used by the set of APEs are based on a pull model, then the organization must deploy the pull-model algorithm. If all the access control mechanisms used are based on a push model, then the organization must deploy the push-model algorithm. However, when some of an organization's APEs use the pull-model and others use the push-model, the combination-model algorithm must be deployed. The authorization algorithms along with their respective system sequence diagrams can be found in [12].

6 Implementation

We are currently implementing WSAA as a middleware layer within the .NET framework [15]. We have developed UML design specifications and specifed a case

study in the healthcare domain to demonstrate the features proposed in this paper with an implementation. We will describe our design specifications and the implementation of WSAA along with the case study in detail in a separate paper.

7 Benefits of the Proposed Architecture

Some of the key advantages of the proposed architecture are as follows:

(a) Support for Various Access Control (AC) models: WSAA supports multiple AC models. The access policy requirements for each model can be specified using its own policy language. The policies used for authorization can be fine-grained or coarse-grained depending on the requirements. AC mechanisms may either use the push model or pull model or even a combination of both for collecting client credentials.

(b) Support for Legacy and New Web Service Based Applications: Existing legacy application systems can still function and use their current AC mechanisms when they are exposed as Web services to enable an interoperable heterogeneous environment. At the same time WSAA supports new Web service based applications built to leverage the benefits offered by the SOA. New AC mechanisms can be implemented and used by Web service applications. A new AC mechanism can itself be implemented as a Web service. All WSAA requires is an end-point URL and interface for the mechanism's APE.

(c) Decentralized and Distributed Architecture: A Web service can have one or more responsible APEs involved in making the authorization decision. The APEs themselves can be Web services specializing in authorization. This feature allows WSAA to be decentralized and distributed. Distributed authorization architecture such as ours provides many advantages such as fault tolerance and better scalability and outweighs its disadvantages such as more complexity and communication overhead.

(d) Flexibility in Management and Administration: Using the hierarchy approach of managing Web services and collections of Web services, authorization policies can be specified at each level making it convenient for Web service collection managers (WCM) and Web service managers (WSM) to manage their objects as well as their authorization related information.

(f) Ease of Integration into Platforms: Each of the entities involved both in administration and runtime domains is fairly generic and can be implemented in any middleware including the .NET platform as well as Java based platforms. The administration and runtime domain related APIs can be implemented in any of the available middleware.

(g) Enhanced Security: In our architecture, every client principal request passes through the Web service's security manager and then gets authenticated and authorized. The security manager can be placed in a firewall zone, which enhances security of collections of Web service objects placed behind an organization's firewall. This enables organizations to protect their Web service based applications from outside traffic. A firewall could be configured to accept and send only SOAP request messages with appropriate header and body to the responsible security manager to get authenticated and authorized.

8 Related Work

We briefly compare the related work in the area of design of authorization architecture for Web service layer (of the SOA) to WSAA.

Kraft [5] proposes an AC model based on a "distributed access control processor" for Web services. The model is generic enough to support different models of access control. This model however, does not provide support for administration of authorization related information. It also does not provide support to manage Web service collections and their authorization related information using standard APIs, which our architecture provides.

Yague and Troya [6] present a semantic approach for access control for Web services that is based on a Privilege Management Infrastructure (PMI). The authorization policies can only be written in the Semantic Policy Language (SPL). What is interesting in this model is that the authorization policies can be attached dynamically based on the metadata of the resource being accessed.

Agarwal et al [7] define an access control model that combines DAML-S , an ontology specification for describing Web services and SPKI/SDSI , used to specify access control policies and to produce name and authorization certificates for users. This is a certificate based AC model. The Access Control Lists (ACLs) in this model are simple and one cannot specify fine-grained and complex authorization policies using this model.

Ziebermayr and Probst propose an authorization framework [8] for "simple Web services". Their framework does not consider distributed authorization and assumes that Web services provide access to data or sensitive information located on one server and not distributed over the Web. This framework uses a simple rule based access control model. A disadvantage with this framework is that it cannot support authorizations for distributed Web services, which have access to data and/or information over a number of Web servers.

Unlike WSAA, the models [6][7][8] only support one model of AC and therefore legacy applications exposed as Web services cannot use different models of AC they have already been using. These models also do not provide management and administration support for Web service objects. There is also no abstraction of each Web service method's task into a set of operations in all these models [5][6][7][8]. This abstraction makes it easy to perform authorization administration as discussed earlier.

9 Concluding Remarks

We have proposed an authorization architecture for Web services that extends the Web services security layer in the Service Oriented Architecture (SOA). We have also provided extensions to the messaging and description layers to support the proposed architecture. We have described the architectural framework, the administration and runtime aspects of our architecture and its components for secure authorization of Web services as well as the support for the management of Web services as well as authorization related information. We are currently implementing the proposed architecture within the .NET framework.

The architecture supports legacy applications exposed as Web services as well as new Web service based applications built to leverage the benefits offered by the SOA; it supports old and new access control models and mechanisms; it is decentralized and distributed and provides flexible management and administration of Web service objects and authorization information. We believe that the proposed architecture is easy to integrate into existing platforms and provides enhanced security by protecting exposed Web services from outside traffic.

References

1. S. Wilkes and J. Harby. SOA Blueprints Concepts Draft v0.5. Technical report, The Middleware Research Company, June 2004.
2. B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, et al. Web Services Security (WS-Security) Specification, `http://www-106.ibm.com/developerworks/webservices/library/ws-secure/`, 2002.
3. V. Varadharajan. Distributed Authorization: Principles and Practice. In *Coding Theory and Cryptology, Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore*. Singapore University Press, 2002.
4. K. Beznosov, Y. Deng, B. Blakley, and J. Barkley. A Resource Access Decision Service for CORBA-based Distributed Systems. In *Proceedings of the 15th Annual Computer Security Applications Conference*, page 310. IEEE Computer Society, 1999.
5. R. Kraft. Designing a Distributed Access Control Processor for Network Services on the Web. In *ACM Workshop on XML Security*, Fairfax, VA, USA, 2002.
6. M.I. Yague and J.M. Troya. A Semantic Approach for Access Control in Web Services. In *Euroweb 2002 Conference. The Web and the GRID: from e-science to e-business*, pages 483–494, Oxford, UK, 2002.
7. S. Agarwal, B. Sprick, and S. Wortmann. Credential Based Access Control for Semantic Web Services. *American Association for Artificial Intelligence*, 2004.
8. T. Ziebermayr and S. Probst. Web Service Authorization Framework. In *International Conference on Web Services (ICWS)*, San Diego, CA, USA, 2004.
9. S. Godik and T. Moses. eXtensible Access Control Markup Language v1.1 (XACML), 07 August, 2003.
10. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, et al. Business Process Execution Language for Web Services v1.1 (BPEL4WS), http://www-128.ibm.com/developerworks/library/ws-bpel/, 2003.
11. R. Kraft. A Model for Network Services on the Web. In *The 3rd International Conference on Internet Computing (IC 2002)*, volume 3, pages 536–541, 2002.
12. S. Indrakanti. On the Design of an Authorization Architecture for Web Services. Technical report, Macquarie University, Sydney, Australia, January 2005.
13. G. Della-Libera, P. Hallam-Baker, M. Hondo, T. Janczuk, et al. Web Services Security Policy Language (WS-SecurityPolicy), `http://www-106.ibm.com/developerworks/library/ws-secpol/`. 2002.
14. S. Bajaj, D. Box, D. Chappell, F. Curbera, et al. Web Services Policy Attachment (WS-PolicyAttachment), `http://www-106.ibm.com/developerworks/library/specification/ws-polatt/`, September 2004.
15. Microsoft Corporation. .NET Framework, `http://msdn.microsoft.com/netframework/`. 2005.