

# Designing a Distributed Access Control Processor for Network Services on the Web

Reiner Kraft  
IBM Almaden Research Center  
650 Harry Road  
San Jose, CA 95120, USA  
rekraft@almaden.ibm.com

## ABSTRACT

The service oriented architecture (SOA) is gaining more momentum with the advent of network services on the Web. A programmable and machine accessible Web is the vision of many, and might represent a step towards the semantic Web. However, security is a crucial requirement for the serious usage and adoption of the Web services technology. This paper enumerates design goals for an access control model for Web services. It then introduces an abstract general model for Web services components, along with formal definitions and notation that can be used as a basis to design an access control processor independent of a particular Web service implementation. It follows the design of a distributed access control processor built upon this general model for Web services, along with implementation guidelines and examples. Main goals for a general authorization framework are identified, and design spaces enumerated.

## Categories and Subject Descriptors

D.4.6 [Security and Protection]: Access controls

## Keywords

XML, Web services, security, access control

## 1. INTRODUCTION

Web services promise to promote the vision of a machine accessible Web, which can be used as a platform to conduct e-commerce and provide interoperability across organizations leveraging from a global Web infrastructure. The idea behind Web services is that they provide a common protocol that Web applications can use to connect to each other over the Internet.

The main advantage of this service oriented architecture (SOA) is that it is based on industry and Internet standards. Hence, it provides an Internet friendly, standards-based mechanism for cross-organization (as well as within organization) control integration.

Web services are described in XML and are communicated over existing HTTP [17] infrastructure using the *Simple Object Access*

*Protocol* (SOAP)[22]. Publicizing Web services is done either using Universal Description, Discovery, and Integration (UDDI) [43], or WSIL (file based discovery mechanism) [33]. One of the major benefits of the Web services technology is its ease of integration, which enables tighter business relationships and more efficient business processes. As an example, a car rental agency could build a Web Services-based solution that translates reservation requests and data between the company's mainframe-based reservation system and an airline partner's UNIX servers. Or, consumers could use Web services that link applications, services, and devices together to act on information any time, any place, and from any smart device.

Recent informal polls [18] showed that security was the top issue among those considering Web services. When decision makers were asked what are the biggest obstacles to implement Web Services, 45.5% identified security and authentication issues. A common fear among IT decision makers is that it is not clear what level of exposure opening up an organization's Intranet to deploy Web services will have.

There is clearly a difference between a Web site and a Web service: A Web site can be set up in a secure environment (e.g., behind a company's firewall) serving static pages, or dynamically created ones based on user input from forms. This is a common scenario and people know how to make it relatively secure. However, a Web service may expose a company's secure back office and business logic for transactions to the public, potentially opening up a large security hole for hackers. Web services have to be reliable. There must be assurances regarding the identity of the systems and principals that interact (authentication), messages are delivered once and only once, and all business processes are completed. The Web service architecture will not be broadly adopted as long as there are no standard ways of securing Web services.

Security efforts in the area of Web services concern data integrity, non-repudiation, and encryption. For instance, *XML Key Management (XKMS)* [19] defines protocols for distributing and registering public keys. There are established standards on how to send and encrypt messages or documents, and digitally sign those, using an underlying secure communication channel (e.g., SSL [48], IPsec [15]). *XML Encryption* [46] and *XML Digital Signature* [50] handle more complex situations, where different parts of the same document need different treatment (e.g., parts of a document need to be signed, perhaps by different people, and this may need to be done in conjunction with selective encryption). The *SOAP Security Extensions* [7] propose a standard way to use the XML Digital Signature syntax to sign SOAP [22] messages. Furthermore, the *Organization for the Advancement of Structured Information Standards (OASIS)*[36] proposed a *Business Transaction Protocol (BTP)* [9]

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM Workshop on XML Security, Nov. 22, 2002, Fairfax VA, USA  
Copyright 2002 ACM 1-58113-632-3 ...\$5.00.

that is designed to allow transactional coordination of participants, which are part of services offered by multiple autonomous organizations (as well as within a single organization) in a Web Service environment. In addition, Microsoft is working on a *Global XML Web Services Architecture (GXA)* [31], which includes a proposal called the Web service security language (WS-Security) [16].

Efforts that solve problems regarding the authorization and access control for Web services are just in their beginning. For instance, the Extensible Access Control Markup Language (XACML) [34] and Security Assertions Markup Language (SAML) [37] are two visible efforts. In addition, AuthXML [11] and Security Services Markup Language (S2ML) [12] were two approaches to address the problem of adding authentication features to XML, now subsumed into SAML. SAML represents an XML-based security standard for exchanging authentication and authorization information, whereas XACML is an XML specification for expressing policies for information access over the Internet. XACML and SAML are not robust enough yet and overlap, which might cause interoperability or integration problems. Kraft [27] discusses research and design issues of access control for network services on the Web. His conclusion is that at present there is no comprehensive and integrated design of an access control or authorization architecture that addresses many problems that arise in a world of network services.

The contributions of this paper are two-fold: First, it enumerates design goals for a comprehensive, extensible and integrated access control architecture for Web services. Second, the paper presents a design for a distributed access control processor for Web services along with design justifications and rationale, based on a general model for Web services. The model incorporates basic Web services, Web service collections, as well as Web service composition, specialization, and operations on Web services.

Towards the end the paper critically reviews recent work related to security with a focus on access control in Internet information systems (e.g., WebDAV [30] [10], SOAP [7], and LDAPv3 [41]). The paper describes how some of these efforts are related and could be applied in the context of Web Services.

Since there is no existing integrated solution and architecture that adequately addresses access control for Web services, the proposed ideas represent a first step towards a comprehensive solution, and should also stimulate more research in the area of authorization issues in a distributed Web service environment. Such a solution is required as a foundation towards a broad deployment and usage of Web services to build the vision of a programmable Web.

## 2. DESIGN GOALS FOR AN ACCESS CONTROL ARCHITECTURE FOR WEB SERVICES

This section enumerates desirable goals for a general access control architecture for web services. Some of the goals do require trade-off decisions in the design, which will be explained later. Sandu and Samarati [38] provide an overview of access control principles and practice. They discuss security goals, the traditional access control matrix, along with some implementation approaches (e.g., access control lists, capabilities, authorization relations), which provide the foundation for the design of a distributed access control processor.

### 2.1 Basic access control capabilities

Kraft [26] presents a hierarchical model for Web services that we adopt and extend in this paper. It comprises Web service methods, Web service objects (container for Web service methods), and Web

service collections (container for Web service objects and other Web service collections).

We want to provide basic access control capabilities for each of these levels, and allow propagation of authorizations down the hierarchy. The overall design should be a simple, secure, and highly efficient access control model for Web services, while providing enough flexibility that addresses organizational, as well as Internet environments policies.

There should be support for the following basic operations on Web services:

**execute** Access and execute a Web service operation, and possibly change the state of a Web service component

**update** Update a Web service component

**replace** Replace the content of a Web service component

**find** Provide search capabilities for properties and metadata associated with a Web service component

The default operation on a Web service is to *execute* it. More interesting types of operations on Web service components are *update* and *replace*, which would update or replace the content (e.g., executable code, metadata) of the Web service component. This might be useful for a developer who needs to update or replace the current programming logic behind a Web service component. Furthermore, a search operation might be useful to query a Web service component for metadata and properties (e.g., for discovery of its interface, authorization requirements, or semantics).

To make the access control model flexible enough, it should be possible to define both *positive* and *negative* authorizations. That requires a conflict resolution policy, in case an authorization conflict arises. A conservative approach would be to not allow the requested access in case of a conflict.

### 2.2 Decentralized and distributed architecture

An access control model and the correspondent access control processor should be decentralized and distributed. Although a centralized access control processor (e.g., MS Passport [32]) also has advantages (e.g., convenience, single-sign on), the security risk in a centralized approach is higher. If an user id or passport is stolen, an attacker can do harm at all sites that accept the passport.

The Web's success is due to its decentralized architecture. Therefore it is reasonable to demand that an access control processor for Web services should embrace the same decentralized architecture. The need for distribution then follows as a logical consequence. As an example, a company typically comprises a hierarchical internal structure. The decentralized approach allows to specify authorizations for Web services on an organizational unit level for different components in the Web service hierarchy. A distributed architecture provides many advantages (e.g., fault tolerance, better scalability) and outweighs its disadvantages (e.g., more complexity, communication overhead).

### 2.3 Flexible identification requirements

Different Web service providers may have different security needs regarding principal identification and authentication. An access control model should therefore support different identification requirements for principals. This ties into privacy issues, since any form of revealed identity (e.g., online id, email address, session id) can be tracked and misused (e.g., for marketing purposes).

The paper distinguishes between the amount of identity that is being revealed during a transaction:

- no identity revealed (anonymous)
- partially revealed identity
- fully revealed identity

For instance, a Web service provider might not be interested in access control at all and allow unrestricted access to resources. In this case there is no need for principal identification and authentication. No identity of a principal is revealed. We refer to this as *anonymous access* to a Web service.

A Web service provider may need more restricted access, which requires a stronger form of subject identification. There are many examples found in common Internet protocols and technologies where access is generally allowed to the public using some weak subject identification scheme (e.g. anonymous FTP requiring an email address). Similarly, we can envision an anonymous access to Web services, where the Web service provider is interested in some sort of basic identity check (e.g., a valid email address, a symbolic network address, or some other form of online id). The paper refers to this as a *partially revealed identity*.

Another special case of partially revealed identity is when a principal represents a compound principal or role. There are some subtle differences between roles and groups that are discussed by Barkley et al. [23]. If a principal identifies itself with a group or role membership, then there is less identity information revealed compared to when the principal reveals its full identity. For instance, if the Web service provider knows that a principal belongs to the group, over 18 years old, less identity information is revealed than knowing an email address of that principal. At some point the principal has to reveal its identity to show evidence for an asserted role or group membership. However, an independent trusted authority or certification authority (CA) could assert that role or group membership by issuing a digital certificate in form of a X509 [2] certificate. During the authentication process only that certificate or token would need to be presented that asserts that role or group membership without revealing the full identity of the principal.

Alternatively, a more decentralized approach could provide an assertion and certificate based on a chain of trust relationships [24]. Once a principal is able to convince the Web service provider that it indeed belongs to a particular authorized group or is allowed to assume a particular role, access will be granted.

A financial Web service provider that transfers money from a client's account needs to know the identity of the principal who initiates the transfer. The paper refers to this as *fully revealed identity*. The Web service provider might maintain a user database of registered clients that are allowed to access that service, or an authentication service (e.g., MS Passport [32], Kerberos [1]) could be used to authenticate subjects.

There might be a potential conflict regarding the identification a Web service provider requires to provide the service, and the identification a Web service client is willing to reveal. The access control model should take this into consideration.

## 2.4 Dynamic discovery of authorization policies

A client of a Web service should be able to discover dynamically what type of identification is required to access a particular resource. For instance, a client wants to invoke a method of a Web service. A response for an invocation could contain what authorization information or credentials are needed to access that particular resource. This is similar to the HTTP [17] authorization mechanism. Based on that a client then is able to either present the required authorization credentials or not. However, a Web service

provider needs to be careful to not expose unnecessary information that represents a security risk. For instance, if an attacker knows that a certain principal has root access to a system, the attacker could exploit that knowledge.

## 2.5 Automatic user account creation

As a result of a discovery a client may need to establish a user account with a Web service provider to gain access to a resource (e.g., for accounting purposes). More generally, a Web service provider may want to setup a user account for a client, which requires principal identification. There has to be some standard way for a client or a Web service provider to do this automatically, such that both do not have to use an off-line channel (e.g., send email, phone call, fill out electronic form on a Web page) to do this.

As an example, in the person-to-business (P2B) world many service providers on the Web require users to fill out electronic forms to establish an user account. Typical data collected comprises a user name, password, email address, postal address and optional attributes. For verification of the email address a simple protocol that involves the generation of a test email and sending it to the user's mailbox including a token is performed. A user then has to reply, and include the token in the reply to prove that the email address is valid. Once the account is established a user can start using the services that are being offered. We can see that there is potential and need to automate this algorithmic process in the Web services P2B scenario, since a Web service does not know by itself how to fill out forms yet. To fully automate account creation, a Web service client and a Web service provider both need to rely on a standardized protocol.

For a business client (B2B) the setup of a user account might also require contract arrangements between parties. Typically a contract is signed using some off-line channel (e.g., signing some paper work). Then a user account will be created, and the business client gains access to a provider's services. It would be desirable to have also the contract signed electronically without having to use traditional off-line channels. It seems to be difficult to fully automate an account creation process, where a more complex contract agreement that might include negotiations is required.

## 2.6 Single Sign-on

Once a principal is authenticated and authorized to access a particular resource of a Web service provider, the requested resource itself may have to invoke other Web services (service aggregation). The call chain might be sequential, hierarchical, or a combination of both. In any case there might be third party Web service providers involved, which are not in the same organizational security domain.

This would require that each additional Web service invokes a separate authentication, discovery, and authentication step, which might be very inefficient. Worse, a principal would have to authenticate itself to possible unknown service providers. A single sign-on is desirable, because it would make that process transparent to the client. It is desirable for a principal to provide the proper authentication once to a special Web authentication service. Subsequent calls from this resource would pass credentials to related Web systems where cross authentication is required.

SAML [37] supports this functionality by having a principal to log on to a SAML-enabled Web system using ordinary HTTP. It enables passing of user credentials to related Web systems for authentication as required. It needs to be further investigated how SAML could be used in the Web services context. Similarly, Kerberos [1] represents a distributed access control system that originated at MIT. Authentication needs to be done only once using an authentication server that issues a ticket (session key) with a timestamp and

lifetime.

A general concern and vulnerability with single sign-on is that principals may want to limit the spread of their credentials to decrease the likelihood of replay attacks. This is related to a subtle problem in the Needham-Schroeder protocol [2], where the shared session key is assumed to be fresh. In general, key revocation is a problem. Kerberos fixes this problem by introducing timestamps instead of relying on nonces, but requires a synchronized time.

Single sign-on across organizations and across application vendors might also require trust between different parties. For instance, Alice forwards a credential of a client Ginger to Bob for further processing. Why should Bob trust Alice at all? There are several ways of establishing trust, for instance by introducing a trusted third party, that introduces Alice to Bob. Overall we would like to have a mechanism for two resources to establish and manage a trust relationship, which can be later used to facilitate single sign-on and dynamic authorization within aggregated Web service invocation chains. The notion of trust is quite complex and sometimes fuzzy. Khare and Rifkin [24] present more thoughts on this topic.

## 2.7 Fine grained level of authorization specification

It can be seen, that we can specify authorizations for every resource starting from a coarse level (e.g., a collection of Web services within one organizational unit) down to a finer level (e.g., a particular Web service method). However, it would be desirable to specify fine-grained access control authorizations below the method level. For instance, a method  $m$  takes as input a parameter  $p$ . We might want to be able to specify authorizations based on the content of that parameter for an incoming request. A client "Alice" might want to invoke  $m$  with a value of 10. Another client "Bob" might want to invoke  $m$  with a value of 20. We may want to have the ability to specify authorizations based on the content or type of some properties of a resource. However, not everyone might need this level of granularity. The design should be able to provide flexibility for Web service providers to decide what level of authorization specification is desirable.

## 2.8 Extensibility

The design of a general access control architecture for Web service resources has to be extensible enough to accommodate future changes of user behavior. For instance, there might be a new authentication mechanism that is superior and therefore could be used to replace an existing one. New approaches for authentication or new levels for authorization specifications might be of interest. Furthermore, the Web service architecture itself might mature and introduce changes, which will affect the underlying security model. In general, the access control model should be designed to be flexible enough to accommodate these changes based on user behavior.

## 2.9 Ease of integration

A distributed access control processor design should be built upon open standards to facilitate integration into existing Internet information systems infrastructure. One goal should be to reuse existing work and not reinvent from scratch.

## 2.10 Usability

Usability in the security design represents a central requirement that is very important for a system to be secure and successful. For instance, if the specification and management of authorization policies is too difficult and its usage cumbersome, errors are inevitably introduced, or users are simply trying to circumvent the security system. A good example can be found in operating sys-

tems, where fast and convenient user switching (usability aspect) would discourage users from logging on as superuser (root) while performing regular or non administrative tasks. In general, if a user is not able to understand the system, it will be almost impossible for him/her to define appropriate and secure policies. Therefore simplicity represents an important goal in the overall design process of a distributed access control processor for Web services.

Furthermore, administration efforts to manage authorization policies and specifications should be minimized. We want to use aggregation and hierarchical containment to specify authorizations on composite objects. Propagation helps to exploit this hierarchical structure to facilitate administration based on the idea that the most specific authorization takes precedence.

## 2.11 Performance and scalability

The design of a general distributed access control processor should be scalable to work efficiently with a large amount of Web services. Network traffic should be minimized, and additional computational efforts should be optimized.

## 2.12 Other goals and considerations

There are more desirable goals. For instance, auditing is a core requirement for access control according to Sandhu et al. [38]. They argue that access control must be coupled with auditing and does not represent a complete solution for the security of a system by itself. Auditing requires logging of all requests. That log data then can be used for further analysis. For instance, it can be checked whether particular users are misusing their privileges, flaws in the system can be detected, and possibly attempted violations can be prevented, since a user knows that all requests are being logged.

# 3. A MODEL FOR NETWORK SERVICES ON THE WEB

The design of a distributed access control processor is based on a general abstract Web services model. First, an informal high level overview of the components in this Web services model is presented along with some basic definitions. It follows a more formal definition of these components, and a notation is introduced to work conveniently with the presented model.

## 3.1 Objects, methods, and collections

Kraft [26] presents a model and terminology for components of the service oriented architecture (SOA), which we adopt and extend in this paper to handle Web service composition, specialization, as well as operations on Web service objects (e.g., union). It comprises Web service methods, Web service objects, and Web service collections. The proposed model already has support for access control processors, which act as a guard for Web service components and perform authorization decisions for these.

Using the model as a foundation for the design of an distributed access control processor introduces many benefits. First, it provides an abstract model that is not dependent on a particular Web service implementation. This helps to focus on access control aspects by not having to deal with different implementation platforms. For instance, the particular implementation of a distributed access control processor for Microsoft's .NET XML Web services [20] will likely be different than one targeted for Apache SOAP [40]. Second, using an abstract formal model allows to more precisely define and design an access control architecture, and avoid ambiguity.

## 3.2 Web service object

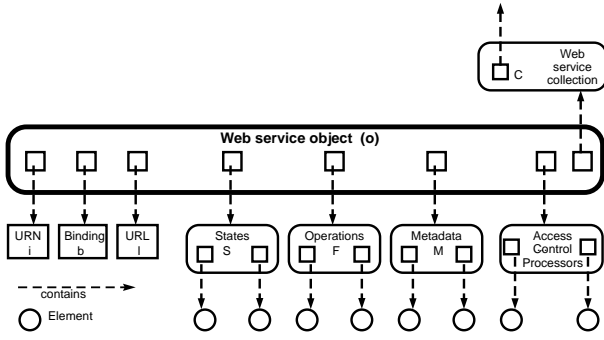


Figure 1: Web service object

The paper prefers the term “*Web service object*” over *Web service*, since the term “object” intuitively relates more to concepts in object-oriented programming, where an object encapsulates data and behavior. A *Web service object* (see figure 1) is accessible over a network by a network endpoint (port). It represents a component, with which applications can programmatically interact by exchanging messages.

Note that a *Web service object* has an internal set of states during execution. In case subsequent messages are being sent to an object they might result in different responses, i.e. there is no guarantee that methods can have the property of “idempotence”. That means that the side-effects of  $n > 0$  identical requests are the same as for a single request. Therefore a request message to an object should not be considered *safe*. Clients have to be aware of any actions they might take, which may have an unexpected significance to themselves or others. More formally, we define a *Web service object* as follows:

**DEFINITION 1. *Web service object***

A *Web service object* is a tuple  $o = (i, b, l, \Sigma, F, C, M, A)$ , where  $i$  is a *URN*,  $b$  is a network protocol binding (e.g., SOAP [5] over HTTP [17]),  $l$  is a network location (*URL*).  $\Sigma$  is a finite set of states to represent the internal state of the object at a given time.  $F$  is the set of supported operations (*Web service methods*).  $C$  is a (singleton) *Web service collection* where  $o$  belongs to,  $M$  is a set of metadata providing additional description for  $o$ .  $A$  is a finite set of access control processors that perform authorization decisions for  $o$ .  $\Sigma$ ,  $F$ ,  $C$ ,  $M$  or  $A$  can be the empty set  $\emptyset$ . A *Web service object* is a resource.

A *Web service object* has an internal set of states  $\Sigma$ . Since  $\Sigma$  can be the empty set  $\emptyset$ , stateless operations on objects are also supported. In this case a request method may be considered safe or idempotent. Using  $\sigma$  to range over states in  $\Sigma$  the invocation of a *Web service method* then maps  $\sigma$  to another state  $\sigma'$ , i.e.  $\sigma : \Sigma \rightarrow \Sigma$ .

If  $F$  is the empty set  $\emptyset$ , then the *Web service object* does not have any associated operations yet. However, these can be added at a later time.

Since  $o$  is a resource, we can use either an operation defined in  $F$  to manipulate the resource, or other general operations that may be defined to access or manipulate  $o$ . For example, an *update* or *replace* operation could be used to update or change the content of the resource at a specific point in time. There might be more plausible operations on the *Web service object* resource that might be useful.

The interface can be automatically derived from  $F$ , the set of supported *Web service methods*. It can be forwarded in a serialized

form (e.g., XML) to a client that wants to discover how to use the *Web service object*. The interface description may be expressed in any interface definition language (e.g., WSDL). A client only needs to know a description of an object’s interface, the protocol-binding, and its network location to interact. The interaction is then performed by exchanging standardized messages depending on the selected protocol binding  $b$  (e.g., using the SOAP messaging framework [22] over HTTP). How a *Web service object* itself is implemented is not relevant and should be hidden to the client. Hiding implementation issues and providing a well-defined interface is the key concept of the *Web services architecture*.

A *Web service object*  $o$  can be a member of at most one *Web service collection* (see definition 3). Therefore the set  $C$  may contain at most one element.

An access control processor  $a \in A$  is as a special kind of *Web service*, which makes authorization decisions for a different *Web service component* (see definition 8). If  $A$  is the empty set  $\emptyset$ , then there is no access control processor that makes authorization decisions for  $o$ .

### 3.3 Web service method

A *Web service method* (see figure 2) represents an operation on a *Web service object*. The paper adopts the term “method” since it is widely used in object-oriented languages to represent behavior of an object in form of a function.

**DEFINITION 2. *Web service method***

A *method* is a tuple  $m = (i, o, p, r(p), M)$ , where  $i$  is a *URN*,  $o$  represents the *Web service object*  $m$  belongs to,  $p$  is a string over an alphabet  $\Sigma^*$  representing a set of input parameters,  $r$  is a function  $r : \Sigma^* \rightarrow \Sigma^*$  that maps  $p$  onto a result string over an alphabet  $\Sigma^*$  representing the output (result) or return value(s) of a computation.  $p$  and  $r(p)$  may be the empty string  $\epsilon$ .  $M$  is a set of metadata providing additional description for  $m$ .  $M$  can be the empty set  $\emptyset$ . A *Web service method*  $m$  has to be a member of exactly one *Web service object*  $o$ , so  $\exists o | m \in o.F$ . It is a resource.

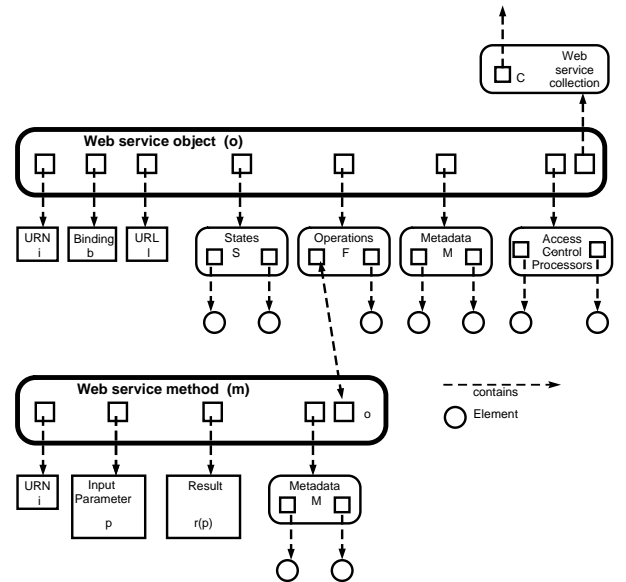


Figure 2: Web service method

A *Web service method* represented through an abstract interface can be accessed to perform some computation. In the proposed

model a Web service method is also a resource, which is quite different than the resource concept in HTTP [3], where methods are not considered resources. Since the definition of a resource allows a mapping to any concept with a conceivable identity, there is no reason to treat a Web service method differently than a Web service object. Basically we can treat a Web service method as a piece of software code in a programming language, which was written in an editor, and has an interface that can be used to activate that code. For instance, we can save that code to a file, and send it via email over the Internet. We can also read, debug, and analyze the code. Since the usefulness of a resource increases when there are some interesting methods to perform operations on them, we can easily see that there are many plausible operations that read, compile, update, replace or move the software code of such a Web service method. The definition even allows us to pass a Web service method as an argument to another Web service method, or a Web service method may return another Web service method as a result. Again, the general description of  $p$  and  $r(p)$  as a string would allow this. Working with methods in this way is similar to concepts in the functional programming area, and it needs to be investigated further what novel applications can be build by having this functionality. At this time we do not want to limit ourselves by not allowing a Web service method to be a resource to be extensible in the future.

On an abstract level given some input string  $p$  over some alphabet  $\Sigma^*$  a Web service method will perform a computation based on that input string and may return a result string  $r(p)$  over some alphabet  $\Sigma^*$  as a result of that computation. The computation may change current state  $\sigma \in \Sigma$  in  $o$  to  $\sigma'$ . It may also be the case that the computation diverges and loops forever, i.e. there's no state  $\sigma'$ . Thus a Web service method may not return a value at all.

Furthermore,  $p$  and  $r(p)$  may be the empty string  $\epsilon$ . For instance, if  $p = \epsilon$  then the Web service method does not require any input (e.g., a Web service method that returns a random number). If  $r(p) = \epsilon$  then the Web service method does not return any result. However, the invocation could have changed the internal state  $\Sigma$ .

Note that input parameter  $p$  and the result of the function  $r(p)$  in programming languages typically have an associated data type of a type system. We have chosen the more general representation using a string over an alphabet to be more flexible. This definition therefore allows that any arbitrary string can be used to encode any plausible type system. For instance,  $p, r(p)$  could be encoded using XML Schemas [47] to associate data types to parameters.

### 3.4 Web service collection

A *Web service collection* (see figure 3) represents an aggregation, and is used to bundle and organize (possibly related) Web service objects. This provides a convenient way to group Web service objects, which is useful to facilitate management of authorization specifications and metadata by using data propagation to exploit the hierarchical containment structure of Web service collections.

Organizing Web service objects with the help of Web service collection hierarchies also allows easier browsing and manipulation compared to what a flat namespace would provide. Hierarchical containment is widely used in current Internet information systems (e.g., Web servers, file systems, URLs) and broadly understood.

More formally, we define a Web service collection as follows:

#### DEFINITION 3. *Web service collection*

A *Web Service collection* is a tuple  $c = (i, O, C_{CHILDREN}, P, M, A)$ , where  $i$  is a URN,  $O$  is a finite set of (possibly related) Web services objects  $O = \{o | c \in o.C\}$ ,  $C_{CHILDREN}$  is a finite set of Web service collections that are children of  $c$ , i.e.  $C_{CHILDREN} = \{d | c \in d.P\}$ ,  $P$  is the parent Web

*service collection*,  $M$  is a finite set of metadata providing additional description and semantics for  $c$ .  $A$  is a finite set of access control processors that perform authorization decisions for  $c$ .  $O, C_{CHILDREN}, P, M$ , or  $A$  can be the empty set  $\emptyset$ . A Web service collection is a resource.

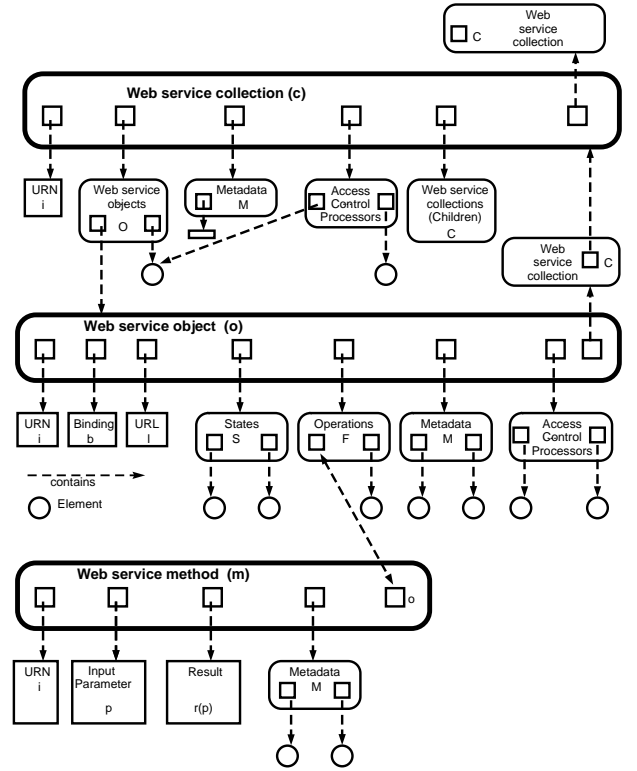


Figure 3: Web service collection

A Web service collection contains a finite set of Web service objects and other Web service collections, and therefore forms a tree (upside down), where the root is always a Web service object.  $c$  has at most one parent  $p$ , which is also a Web service collection. Cycles are not allowed, as well as self-containment, to prevent problems in the tree structure.

### 3.5 Notation

In the previous sections we already used informally some notation to refer to components and elements of components in the model. We will introduce some abbreviations in this section to work more conveniently with the notation.

Let the tuple  $o = (i, b, l, \Sigma, F, C, M, A)$  be a Web service object. To access an element in the tuple we introduce the “.” operator to refer to it. For instance, to refer to the set of operations on  $o$  we can write  $o.F$ . To refer to the set of access control processors that make authorization decisions for  $o$  we can write  $o.A$ . If  $c$  is a Web service collection, the set of access control processors for  $c$  would be  $c.A$ .

The “.” operator is left associative and can be used to refer also to elements of a particular selected set. As an example, let  $f \in F$  be a Web service method that belongs to  $o$ . To access  $f$  we can write  $o.F.f$ , which is identical to  $(o.F).f$ .

We can use an alias  $f() = o.F.f$  to refer to the Web service method  $o.F.f$  more conveniently. In this case we refer implicitly to the Web service object  $o$ , where  $f$  belongs to. The parenthe-

sis emphasize that  $f()$  is a function that might expect a parameter. Then  $f(p)$  represents the invocation of  $f()$ , where we pass the the formal input parameter  $p = o.F.f.p$  to the Web service method, and  $r = f(p)$  represents the result of that computation.

### 3.6 Combining Web services

A single Web service can be combined with other Web services to form a new Web service. Web service composition helps to model business processes. For instance, the Web Services Flow Language (WSFL) [29] models workflow between Web services. A Web service within this flow is called an *activity*. WSFL casts Web services into the role of implementing activities within an overall business process, or “flow” model. In the big picture, this means that individual types of Web services (e.g., calendar, address book, stock quotes) are of less importance than the overall workflow that is being implemented. Therefore WSFL can be used to model aggregation of Web services, when treating the resulted graph or flow model as a new Web service object.

The proposed distributed access control processor (see section 4) needs to know for a given Web service object, what the associated set of access control processors is. Per definition (see definitions 1, 3) we know what the access control processors for the basic Web service components (e.g., Web service object, Web service collection) are. However, Web service aggregation, composition, specialization, and operations on Web service objects result in new Web service components, for which we need to extend the model [26] to be able to determine the set of access control processors.

#### 3.6.1 Composition of Web service methods

We present additional definitions and notations that formalize the composition of Web service methods, and help to identify the set of access control processors that result from such a composition.

There is a difference between a Web service collection and a Web service composition. The former is used as a container for Web service objects, whereas the composition represents an operation between Web service methods, that results in a new Web service object. As a consequence we could use a Web service collection to group composite Web service objects.

Informally, we define the composition of two Web service methods as executing one Web service method, and passing the result of this computation as an input parameter into the second Web service method. The result of this computation then represents the result of the composition. More formally,

##### DEFINITION 4. *Web service composition*

Let  $f() = o_1.F.f$  be a Web service method that belongs to the Web service object  $o_1$ . Further, let  $g() = o_2.F.g$  be a Web service method that belongs to the Web service object  $o_2$ . Per definition of a Web service method  $f$  and  $g$  are functions from  $\Sigma^* \rightarrow \Sigma^*$ .

Further let  $g()$  be the Web service method from the set  $A \in \Sigma^*$  to the set  $B \in \Sigma^*$ , and let  $f()$  be a Web service method from the set  $B \in \Sigma^*$  to the set  $C \in \Sigma^*$ .

The Web service composition of the Web service methods  $f()$  and  $g()$ , denoted by  $f \circ g$  is defined by  $(f \circ g)(a) = f(g(a))$ .

In other words,  $f \circ g$  is the Web service method that assigns to the element  $a$  of  $A$  the element assigned by  $f$  to  $g(a)$ . Note that the Web service composition  $f \circ g$  cannot not be defined unless the range of  $g$  is a subset of the domain of  $f$ . Per definition of the Web service method this is always the case, since  $A$ ,  $B$ , and  $C$  are all subsets of  $\Sigma^*$ . However, in practical applications Web services typically use a type system. In this case this becomes a problem and needs to be addressed properly so that the composition works. As an example, consider  $g(a)$  returns a type `integer`, and  $f()$

expects a type `String` as an input. This will lead to trapped or untrapped errors, and results in a non-predictable behavior of computation. In the subsequent examples we assume that the composition is properly formed to avoid this type of problems.

Note that even if  $f \circ g$  and  $g \circ f$  is defined for the Web service methods  $f$  and  $g$ ,  $f \circ g$  and  $g \circ f$  are not equal. In other words, the commutative law does not hold for the composition of Web service methods. The identity function is formed by the composition of a Web service method and its inverse, and can be easily verified by applying the definitions.

The result of the composition  $o_m.F.f \circ o_n.F.g$  is a new Web service object  $o_k = (i, b, l, \Sigma, F, C, M, A)$ . We then define the set of access control processors of  $o_k$  to be the union  $o_k.A \cup o_m.A \cup o_n.A$ .

The recursive nature of Web service composition allows reuse of newly composed Web service objects to participate in other compositions. The order of the composition operation results in a sequence of activities (pipeline), which models an activation chain and can be represented using a directed graph.

A practical example to illustrate on how Web service method composition can be used we introduce two Web service methods:

- `boolean verifyFunds(int amount)`
- `boolean doTransaction(boolean hasFunds)`

The composition then would be

- `boolean doTransaction(verifyFunds(int amount))`

In the model we can rename the result method, that belongs to the newly created composite Web service object as

- `boolean verifyFundsDoTransaction(int amount)`

Looking at the method signature does not reveal its internals. This shows the strength of using composition to build new Web service objects from existing Web service methods. The current Web service implementations and proposals (e.g., WSFL) to model composition are not standardized yet. This is probably one of the reasons why Web service composition is not broadly used.

#### 3.6.2 Specialization of Web services

Just as inheritance is used in object oriented programming languages to foster reuse, specialization applied to Web services allows the reuse of existing Web service objects through specialization to accommodate a particular behavior. For instance, a Web service that returns U.S. stock quotes can be specialized by a new Web service that only returns stock quotes of companies that are traded on NASDAQ. In general, we can either override existing behavior or extend the functionality of an existing Web service.

##### DEFINITION 5. *Web service object specialization*

Let  $o$  be a Web service object. Then  $o_{\prec} = (i, b, l, \Sigma, F, C, M, A)$  is a Web service object that represents the specialization of the Web service object  $o$ .  $o_{\prec}$  has the property that  $o_{\prec}.F \prec o.F$ .

The  $\prec$  notation indicates sub-typing or inheritance. In general, specialization and sub-typing introduces many subtleties that need to be addressed (e.g., how exactly is  $F$  affected, when a type system is involved). There are more interesting related properties of type systems that need further investigation (e.g., subtyping of Web services), but are not directly relevant to the design of a distributed access control processor. In this paper we are focused in how the set of access control processors  $A$  of a Web service component is affected. The set of access control processors for  $o_{\prec}$  is defined as

the union of  $o_{\leftarrow}.A \cup o.A$ . The reason we have to use the union in this case (and not possibly subset), is that  $o$  will get instantiated (executed) when  $o_{\leftarrow}$  gets executed. This is similar to objects in programming languages, where the instantiation of a sub-class will instantiate the super-class in its constructor.

### 3.6.3 Operations on Web services

We introduce a useful operation on Web service objects: The union of two Web service objects, which itself is a Web service object.

Intuitively the union of two Web service objects represents an aggregation that results in a new Web service composite object, which offers functionality of both Web service objects. More formally,

#### DEFINITION 6. Web service union

Let  $o, p$  be Web service objects. Then  $\cup_{op} = (i, b, l, \Sigma, F, C, M, A)$  represents the union of the Web service objects  $o$  and  $p$ , where  $\cup_{op}.F = \cup_{op}.F \cup o.F \cup p.F$ , and  $A = \cup_{op}.A \cup o.A \cup p.A$ .  $\cup_{op}$  is a Web service object.

The union operation can be applied in any context where aggregation is useful. As an example, a company has a Web service object  $a_1$  in department  $A$ , and a Web service object  $b_1$  in department  $B$ . The company decides to make both Web services available externally to the public providing one interface, but wants to set some special authorizations to regulate access, that are not necessary for internal access. The *union* operation allows this conveniently by having the ability to add the appropriate authorizations in the composite Web service object.

There are other plausible operations on Web services (e.g., intersection, subset, and service combinators [8]) that require further research and investigation on how they can be applied in the Web service architecture.

## 4. DESIGNING A DISTRIBUTED ACCESS CONTROL PROCESSOR FOR WEB SERVICES

This part of the paper presents key ideas for the design of a distributed access control for Web services. The design is based on a general Web services model presented in the previous section and accomplishes the desired goals. Trade-off among design decisions and rationale are justified when needed. First, an informal high level overview of the design will be presented. It follows a more formal definition of the different components that comprise the access control processor.

### 4.1 Authorization filter architecture overview

We describe first the authorization architecture based on access control processors informally at a high conceptual level. This provides a description of the relationship between components.

There are two types of components in the proposed architecture that comprise the distributed access control processor:

1. **access control processor** - An access control processor is a Web service object that makes authorization decisions (possibly) together with other access control processors for a Web service component. A Web service component can have multiple access control processors associated with it.
2. **gatekeeper** - A gatekeeper is an access control processor that has to make the final decision of whether a particular request for a Web service component can be granted or denied. It

also authenticates principals of incoming requests, and generates a ticket that can be shown as evidence for authenticity. A Web service component can have at most one gatekeeper associated with it.

The gatekeeper and the access control processor are both Web service objects. By defining them as Web service objects they are loosely coupled and leverage from the open Web service architecture. Thus there is no need to invent a new message passing framework, event notification system, or any other required infrastructure, if we define them to be Web service objects.

The gatekeeper is also an access control processor, but one with additional functionality. Gatekeepers intercept SOAP requests for a Web service object and determine whether that request is granted or denied. They work together with the associated set of responsible access control processors (see definition 7) for that Web service object to make authorization decisions for a specific request.

To implement access control for a Web service object, we need to associate it with at least one gatekeeper. A Web service object can have many associated access control processors, but at most one gatekeeper.

The conceptual model does not enforce any particular implementation restrictions. A preferred embodiment could make use of open Internet standards (e.g., XML, SOAP, WSDL).

An example scenario will help to illustrate the distributed access control processor architecture (see figure 4).

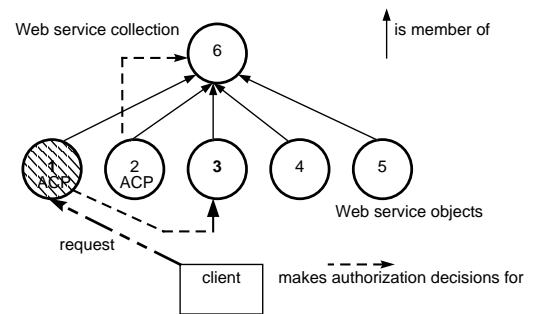


Figure 4: Example scenario of distributed access control processor architecture

In this case a client wants to invoke Web service object 3. The gatekeeper (Web service object 1) intercepts the request. It authenticates the client first, and calculates  $S_{ACL}$ , which are the Web services objects and ACPs  $\{1, 2\}$ . It then routes the request to ACP 2, and collects the authorization decision. The union of the collected authorization decisions is then used to either grant or deny the request of the client. The gatekeeper sends then a response back to the client containing the authorization decision.

Web service object 3 belongs to the Web service collection 6. Both ACP 1 and ACP 2 are also a member of the Web service collection 6. ACP 2 is responsible for the Web service collection 6.

For instance, in a preferred embodiment each Web service object is hosted by an application server to be accessible over the network. This application server typically runs a SOAP router process that accepts incoming SOAP requests and dispatches them to the appropriate Web service object that performs the requested work. A gatekeeper component could be implemented as an extension to that SOAP router, or could be implemented as a separate Web service object. Similarly, the access control processors can be implemented in many different ways to suit the particular needs (e.g., a company uses a proprietary Web services implementation). Al-

though there may be conceptually many access control processors for a Web service object, a particular implementation may want to bundle some of them into one system component for efficiency purposes. In the current major Web service implementations (e.g., MS .NET XML Web services [20], Apache SOAP [40]) there is no support for access control. It can be seen that it is not too difficult to integrate a gatekeeper Web service that acts as a proxy and makes authorization decisions.

## 4.2 Determining the set of responsible authorization decision makers

A Web service object and a Web service collection allow 0 or more associated access control processors. Each Web service object  $o$  has a *set of responsible authorization decision makers*. This is the union of all responsible access control processors of that particular Web service object. In a simple scenario where  $o$  is not a member of any Web service collection, then the set of responsible authorization decision makers is solely  $o.A$ . If  $o$  is a member of the Web service collection  $c$  that has no further parent collections, this set becomes  $o.A \cup c.A$ . Note that at this point we are only interested to determine the set of access control processors and do not consider authorization policies that each of them implement. For instance,  $c.A$  could have authorization policies that do not propagate to its child Web service objects. However, the set of responsible authorization decision makers for  $o$  would not be affected by authorization policies.

If the collection  $c$  is part of a hierarchy the determination of the set of responsible authorization decision makers becomes more complicated to calculate.

For this reason we present an algorithm (see definition 7) that determines for a given Web service object the set of responsible authorization decision makers  $S_{ACL}$ . The algorithm essentially traverses recursively the tree hierarchy and builds the union of all encountered access control processors, where the root of the tree is the Web service object, and the children are Web service collections that belong to the hierarchy.

**DEFINITION 7. Set of responsible authorization decision makers**

*The presented algorithm determines the finite set of responsible authorization decision makers  $S_{ACL}$  for a given Web service object  $o$ :*

**Input:** A Web service object  $o$ .

**Output:** The set of responsible authorization decision makers  $A$  for  $o$ .

**Remarks:** The '+' operation is used to represent the union operation between two operands.

```
A determinesSRADM(WebServiceObject o){
  A sacp = o.A + unionACP(o.C);
  return sacp;
}
```

**Input:** A Web service collection set  $C$ .

**Output:** The set of responsible authorization decision makers  $A$  for  $c$ .

**Remarks:** The '+' operation is used to represent the union operation between two operands.

```
A unionACP (WebServiceCollection c){
  A uacps = nil;
  if (c==null) {
    return null;
  }
```

```
  } else {
    uacps = c.A + unionACP(c.P);
  }
  return uacps;
}
```

## 4.3 Tasks of the gatekeeper

The first task of the gatekeeper when receiving an incoming request is to authenticate the principal if necessary. For instance, the requested Web service object may require subject identification, partially revealed identity, or only anonymous access, in which case authentication would not be necessary at all. The gatekeeper represents an authentication server (e.g., similar to Kerberos [1]), which after authenticating a principal will issue a ticket and associate it with the request, that can be further used to authenticate the request to different access control processors without having to re-authenticate again.

The next task of the gatekeeper is to determine the *set of responsible authorization decision makers*  $S_{ACL}$  for a given Web service object. Once the  $S_{ACL}$  is obtained the gatekeeper will enumerate it in a linear order, where the access control processors on the Web service collection level come first, followed by the access control processors of the Web service object itself. We refer to this list as the *ordered list of responsible authorization decision makers*, or simply *order*( $S_{ACL}$ ). Within a hierarchical level of the Web service model the order is arbitrary.

The gatekeeper analyzes an incoming request (e.g., SOAP message) and a tuple in form of  $t=(principal, resource, action, condition)$  will be composed, which represents the *authorization request* of the incoming request. The authorization request is the basis for an authorization decision based on an authorization policy. Supported actions are *execute*, *replace*, *update*, and *find*, but can be extended for other plausible operations. A condition is something that has to be satisfied to grant the request. This relates to the notion of provisional access control introduced by Kudo and Hada [28]. For instance, a condition could be that a request has to be digitally encrypted, signed, and logged.

As an example, consider Alice wants to access the Web service method `getStockQuote(String symbol)`. Alice sends a well-formed SOAP message to the gatekeeper. The gatekeeper then analyzes the SOAP requests, and creates the authorization request `(Alice, o.getStockQuote("IBM"), execute, log)`. Where *execute* means to execute the Web service method,  $o$  represents the Web service object where the Web service method belongs to, and *log* represents a condition that needs to be fulfilled - in that case logging of the request.

The gatekeeper adds the ordered list based on  $S_{ACL}$ , then generates an authentication ticket, the authorization request, and adds these to the incoming request, digitally signs it (possibly encrypt it before signing) and sends it to the first access control processor on the ordered list  $S_{ACL}$ .

Authorizations can be specified in an authorization specification language (e.g., XACML [34]) and made accessible (e.g., filesystem, database) to each access control processor. Each access control processor that receives such a request will lookup authorization policies that are relevant to the authorization request. It then appends these to the request, digitally signs the modified request, and forwards it to the next access control processor in the ordered list of  $S_{ACL}$ . The last access control processor in that list will forward the SOAP message to the gatekeeper after processing. In an implementation we would need to make sure that the processing of a request message is done efficiently.

The gatekeeper finally decides (after performing the previously described steps) whether a request should be granted or denied. In case it is granted, the request will be passed to the Web service object for processing, otherwise it will be returned to the original requestor along with information about the decision.

For optimization, when the gatekeeper receives a response from an access control processor that denies a particular authorization request, the gatekeeper may decide to stop further processing immediately and sent a response back directly to the originator of the request. This may be useful during parallel processing of an authorization request where a single negative authorization decision means that a request cannot be granted. There is no need in this case then to wait for the outcome of other authorization decisions from different access control processors that are still processing the request.

#### 4.4 Definition of a distributed access control processor

The previous sections presented an informal overview of a distributed access control architecture for Web services. We will now define these components more formally.

##### 4.4.1 Definition of an access control processor

An access control processor receives incoming authorization requests from the gatekeeper of a Web service object and has to make an authorization decision. The authorization decision will then be returned to the gatekeeper.

###### DEFINITION 8. Access control processor (ACP)

Let  $o$  be a Web service object. Further, let  $R$  be the set of all Web service collections and Web service objects, where  $r \in R$ . We define the set  $U = \{r | o \in r.A\}$ . Further, let  $D \subseteq U$ . An access control processor (ACP) is a triple  $a = (o, D, AUTH)$ , where  $D$  is its domain of responsibility,  $AUTH$  is a set of authorization policies (e.g., XACML).  $AUTH$  or  $D$  can be the empty set  $\emptyset$ . An access control processor is a resource.

The paper mentioned before that an ACP is a Web service object itself. Therefore it also has a URN for identification, a set of internal states, a well defined interface and possibly metadata. Note that the empty set is allowed for  $D$  or  $AUTH$ . In this case the ACP would have either no responsibility (e.g.,  $D = \emptyset$ ) for some Web services objects, or there would be no authorization specifications (e.g.,  $AUTH = \emptyset$ ). This means that every incoming request would be allowed. An ACP also has a  $S_{ACL}$ , which allows to define recursively to define access control processors for an ACP. Note that at some point  $S_{ACL}$  has to be empty, since  $S_{ACL}$  has to be finite.

An ACP  $a$  is only responsible for an object  $r \in U$  if and only if  $a \in r.A$ , where  $A$  is the set of ACPs associated to  $r$ . There has to be a two-way agreement: A Web service object has to have  $a$  in its  $S_{ACL}$ . That states that a resource chooses  $a$  to make authorization decisions for it. Conversely,  $a$  has to agree to make authorization decisions for  $r$ . Once this relationship is established an ACP can make authorization decisions for a resource it is responsible for.

Note that an ACP is not authenticating principals. Authentication is done by the gatekeeper, which acts as an authentication server.

Since an access control processor is a Web service object resource, all the operations that apply to the correspondent Web service can be applied to the resource.

##### 4.4.2 Definition of a gatekeeper for a Web service object

Next, we define a special type of ACP, the *gatekeeper*. A gatekeeper has some additional tasks to perform. It needs to authenti-

cate first the principal of an incoming request, and then determine the set of all possible ACPs associated with a particular Web service object ( $S_{ACL}$ ). Second, it needs to coordinate the routing of an incoming authorization request. More formally we define a gatekeeper as follows:

###### DEFINITION 9. Gatekeeper

Let  $a$  be an access control processor,  $o$  be a Web service object that is guarded by  $a$ . A gatekeeper is a specialization  $a_{\prec} = (i, b, l, \Sigma, F, C, M, A)$ . Further,  $a_{\prec}$  is an authentication server that authenticates principals of incoming requests and coordinates the routing of an authorization request with other ACPs. It is a resource.

This definition allows us to reuse all of the previous definitions. A gatekeeper is just a special type of ACP, with some additional functionality (e.g., authentication server). We use specialization (see definition 5) to model the specialized behavior, which lets us conveniently reuse existing definitions.

The gatekeeper will be the one that will determine  $S_{ACL}$ , coordinate the collection of authorization policies for a specific request from all member of  $S_{ACL}$  (routing), and it will (possibly) make an authorization decision based on the collected authorization policies. Note that any  $ACP \in S_{ACL}$  could also make an authorization decision to prevent a negative authorization decision to propagate through the chain.

The proposed model is very flexible to use. For instance, not all hierarchical levels from the Web services model have to be present in a particular scenario. For instance, a small organization hosting only one Web service with one or two Web service objects may decide to need only a gatekeeper. As the organization becomes more complex, more ACPs are added. This allows flexibility to adapt the access control processor architecture to individual needs.

Also, some small organizations or individuals may not have security needs at all. The proposed model allows a Web service object to exist without having to have a gatekeeper. Conversely, A gatekeeper or ACP may exist that is not associated with any Web service object. These are both extreme cases, but show the flexibility of the model.

#### 4.5 Examples

As an example, at the Web service collection level a particular group is not authorized to invoke a Web service object and it is not allowed to overwrite this policy at a lower hierarchical level (hard authorization). In this case if a principal is a member of that group there's no need to propagate the message further along in the chain, and the gatekeeper can stop processing.

In one scenario there is one access control processor per Web service hierarchy level. Each access control processor manages its own set of authorizations. The task of the gatekeeper is to authenticate principals, and collect all relevant authorizations from  $S_{ACL}$ . For an incoming request message the authorization decision is then based on the union of authorizations collected from all levels of access control processors. Conflicts resulting from positive and negative authorizations need to be resolved. The decentralized architecture is scalable and allows the independent management of authorization at various organizational levels.

To illustrate how the architecture could be implemented in a small business scenario consider a simple case where we have two access control processors. One makes authorization decisions for the Web service object  $o$  (gatekeeper). The other one makes authorization decisions for the Web service  $w$ .  $o$  is a member of  $w$ . We use an application server that hosts  $o$ . For efficiency both access control processors are implemented in one Web service object  $a$ .

The gatekeeper itself is implemented as an extension to the application server's SOAP RPC router. A client issues a SOAP request to  $o$  that will be intercepted by the gatekeeper. The gatekeeper would then lookup  $S_{ACL}$ , which is simply  $a$ . It would then forward the SOAP request to  $a$ .  $a$  would lookup the set of relevant authorizations that are stored locally (e.g., file system, database), append these to the SOAP request and return it to the gatekeeper. The gatekeeper then makes the authorization decision and grants access or denies the request.

## 4.6 Benefits of the proposed architecture

We can see that this decentralized approach of authorization filters has several advantages, and achieves many of the desired goals described in section 2.

**Decentralized management of authorizations** Different organization levels can independently specify authorizations. Merging them will be done when the SOAP messages travels along the chain of responsible access control processors.

**Flexibility** Note that a hierarchy level above a Web service is optional. Authorizations can be specified at each level to make it convenient for an administrator to manage them.

**Efficiency** Network traffic will be minimized. SOAP request messages will be intercepted, processed, and forwarded. Note that a particular implementation can decide to combine an access control processor that works at both web service and object/method level.

**Ease of integration** The proposed model of access control processors can be easily integrated into existing Web service infrastructure. Gatekeepers act as a gateway for SOAP messages. The actual Web service implementations at object/method level would not need to be touched at all. The distributed access control processor acts as an authorization service and only forwards SOAP request messages to a Web service object that are authorized.

**Enhanced Security** A gatekeeper can be placed in a firewall zone which enhances security, and the actual Web services behind the firewall. It acts therefore as an additional bastion and keeps the Web services itself protected from direct outside traffic. The Web services behind the firewall can be configured to only accept SOAP request from a gatekeeper (e.g., based on IP address, based on digital signature).

Overall the design represents a foundation on which we can build a framework to achieve the desired goals described earlier. It does not impose restrictions on authentication, and we can chose a suitable authentication system that we can integrate into the gatekeeper component.

## 4.7 Implementation issues

The proposed design does not suggest any particular implementation on how discovery of authorization could be implemented. There are many ways on how to integrate this in the proposed model, for instance on the protocol level (e.g., as a SOAP extension). The gatekeeper could receive such a discovery request, and return a response that indicates what identification type is required (e.g., anonymous, partial revealed identity, full revealed identity). The work related to privacy (e.g., P3P [45]) might be useful in this context, but we leave the work of designing a discovery mechanism open as a future extension to the model.

Emerging standards such as SAML [37] and XACML [34] could be used to specify and exchange authorizations, and manage and exchange trust relationships. Since we prefer XML for the messaging along with XACML fine-grained authorization specification is supported. Clearly, the overall access control processor design is flexible and extensible. Metadata that can be associated to Web services and access control processors help to attach additional semantics and facilitate automatic processing.

There are many technical and practical issues that need to be investigated further. For instance, though a request message conceptually travels in a linear fashion through a list of access control processors, an implementation could do this in parallel. For instance, the gatekeeper could send out a request in parallel and collate responses. There's a trade-off between computation time and network bandwidth. The linear approach consumes less bandwidth (there's only one request message that will be forwarded), but may require more time (since computations of authorizations are done sequentially), depending on  $|S_{ACL}|$ , the amount of available bandwidth, and performance requirements of determining what approach is more efficient.

There are problems that arise in the distributed architecture when an access control processor goes off-line (e.g., server is down for maintenance, network problems). In this case the gatekeeper will wait a predefined time for a response and a time-out occurs. Since the gatekeeper wasn't able then to successfully collect all authorization decision, it cannot make an authoritative decision, and a request should be denied. Similarly, the gatekeeper could be off-line, in which case the client would not be able to access the requested resource.

It might be sufficient in most cases to have at most one access control processor per hierarchical level. In an even more simplistic case there would be only one gatekeeper that makes authorization decisions for a Web service object and no other additional access control processors. Having multiple access control processors per object may result in conflicting authorizations. However, we want to keep the architecture as flexible as possible.

Note that the description of the access control processor architecture presented is done on a conceptual level and offers flexible ways of implementation. On a practical implementation level we may want to limit the number of access control processors in the chain to facilitate efficient processing. Still, in a larger organization we may need to have more complex chains of access control processors.

One technical question is how does a gatekeeper determine the *set of responsible authorization decision makers*  $S_{ACL}$ ? In case of a request we need to merge all relevant authorizations. We know from the definition of a Web service object whether that object is a member of some Web service collections. In this case we can lookup for each Web service collection which ACPs are responsible for making authorization decisions. So for each request for a Web service object we can have some recursive lookup to determine what are the ACPs we need to contact (see algorithm definition 7). In a particular implementation we would be able to use the metadata associated with a Web service object, or Web service collection (e.g., in form of RDF schemas) to store this information. To make things more efficient we could cache some of this information so that for an incoming request a gatekeeper would be able to retrieve  $S_{ACL}$  from a cache.

An ACP can manage multiple Web services objects, or Web service collections. However, another interesting point is that a Web service object could have multiple ACPs with possibly different sets of overlapping or conflicting authorization policies. For instance, we could have an ACP which grants limited public access

and handles incoming requests from outside the firewall. Another ACP is configured to provide team access in a local Intranet environment. This allows access to the same object using different ACPs/gatekeepers. However, this does not allow circumventing of authorizations since all authorization policies are merged and the union of these is used to make an authorization decision. Having different ACPs for the same resource may lead to conflicts, which have to be resolved. A conservative approach would be that negative authorizations have precedence over positive authorizations.

#### 4.8 Authorization on messages

The SOAP messaging framework [22] and XML [44] represent the basis for Web service requests and responses. Every interaction is message based, using SOAP to express the invocation of a particular object. This allows us to use SOAP messages as a basis for authorization decisions. Essentially an incoming request message will be parsed and analyzed by a gatekeeper. Based on the available authorizations, the identity of the subject, and the requested action an authorization decision can be made.

Damiani et al. [14] describe fine-grained authorizations for SOAP messages. They argue that a client's request in form of a SOAP message has an XML structure that is modelled after the interface offered by the remote Web service. Therefore, in their opinion, it makes more sense to consider the requests themselves as objects of an authorization system. In their design a SOAP message is routed through an authorization filter that makes an authorization decision and possibly change the original request in the SOAP message based on that decision.

The approach in this paper is different, since it does not change the the original request. Only information will be added: A gatekeeper will add routing information (e.g., to members of  $S_{ACL}$ ). ACPs are also possibly adding authorization decisions. However, the content of the original request will not be changed. In case a request is denied in part or all, a SOAP response will be sent back by the gatekeeper or another ACP to the client including the result of the service denial. In case a request is granted it will be marked as accepted, digitally signed by the authorization filter and forwarded to the target Web service object.

There are several reasons that motivate our decision to not modify the original message. First, a client is interested in a particular request. In case a request is denied in whole or in parts the client should receive a notification about that and then decide how to proceed. In general, modifying a SOAP request without the consent of the requester may lead to a new request, which may not be in the interest of the requestor. Consider a client that wants to purchase a book and provides a coupon code. If the request is not authorized (e.g., based on the identity of the client) the client would need to be informed about this. Modifying the original SOAP message may lead to a purchase not wanted under the modified terms. Second, modifying information or filtering out information may be appropriate (assuming the consent of the requestor) if only one object would be affected. However, Web service aggregation may cause many objects to be involved. Although some parameter of a request is not authorized it might be needed by some dependent service.

To sum up, modifying the original request by applying an authorization filter on the message content may lead to problems, and may conflict with the client's intention. An access control processor should analyze a SOAP message and then act as a decider. In case access is not granted at all or in only in parts a response should be sent back to the client. In this design the client will need to make the decision of how to proceed.

#### 4.9 Propagation of authorizations

We will use XML as a format of choice to specify authorizations. Authorizations can be specified at various levels:

- Web service collection
- Web service object
- Web service method

This allows someone hosting an object to specify authorizations that are related to that object and its methods. Second, at a higher level authorizations can be specified for a Web service collection. These authorizations will be propagated to its objects.

Propagation helps to exploit the hierarchical structure of the Web services model to facilitate the administrative efforts based on the idea of that the most specific takes precedence. Furthermore, it helps to enhance the overall usability of the system.

We can exploit the following relationships:

- Web service collection to Web service object
- Web service object to Web service method
- Web service method to Web service method property

Damiani et al. [13] allow the specification of *recursive* or *local* authorizations. A recursive specification allows the propagation of authorizations.

However, propagation also introduces additional complexity. How far (number of levels) should an authorization propagate? What about conflicting authorizations? How can we prevent the misuse of overriding authorizations if not desired? Damiani et al. [13] introduce the notion of *hard* and *soft* authorization statements that prevent authorizations at a more general level from being overridden. In this context *soft* means that an authorization should be applied only if nothing else has been stated on a more general level. In contrast, an authorization defined as *hard* should not be overridden by a more specific authorization. This suggests that *hard* authorizations need to be placed at the most general level (Web service collection) to be effective.

This architecture supports and accommodates the hierarchical structure that is used in organizations. For instance, a credit card company may have a Web service collection that comprises all Web services that the company provides. Some general authorizations can be specified at that level. For instance, access is in general allowed, as long as it is not from a specific competitor company. At a departmental level manager can specify authorizations that are associated with particular Web service objects. For instance, the accounting department has some accounting Web service objects, that require stronger authorization needs than what is specified at the higher Web service collection level. In contrast, some other department (e.g., Marketing) provides some Web service objects that are satisfied with a weaker level of authorization. The specification of authorizations on the Web service object level allows organizational units to override or extend authorizations that are specified at a higher level. At the actual implementation level (objects, methods) more specific authorizations can be added at a fine-grain level that even supports a granularity below methods.

#### 4.10 Specification of authorizations

How should authorizations be defined, and how should they be stored? There are different proposals. It might be a reasonable choice to use XML itself to express authorizations. The structural benefits of XML and schema-verification properties would help to

detect errors in authorizations automatically during the parsing process of the access control specification. Damiani et al. [13] introduce XML Access Sheets (XAS), that are associated with the document/DTD. In this case we have a clean separation of representation and storage of authorizations from the document they protect, which conforms to the design principle to separate between data model and access control model.

On the contrary, Kudo et al. [28] are introducing the XML access control language (XACL), which is embedded in the document structure and integrated in the more recent XACML proposal [34]. Authorizations are specified in the document using a special `<POLICY>` tag. Doing this will lead to a change in the data model. A DTD for a given application would need to be changed to accommodate that special tag. If not done, a document with XACLs would no longer be considered valid by an XML parser. Second, unprotected authorizations specified within a document are easy to manipulate by an attacker. Authorizations should be stored in a safe and protected place, so that tampering with them is prevented. For these reasons the separation of data and access control model seems to be more appropriate.

#### 4.11 Performance and scalability considerations

It is clear that the implementation of access control will not come for free. A Web service must pay a computational and network bandwidth penalty for having gatekeepers making authorization decisions. The goal is to minimize that computational effort. Furthermore, we do not want to introduce additional effort and complexity. That means that we do not want to the access control processor to perform work that will be repeated by the gatekeeper. Second, the design of the access control processor has to scale. Adding more Web services that need to be managed or increasing traffic to a Web service should be addressed such that it is easy to simply add more access control processor that share the workload should be an easy undertaken.

Secure communication will be based on encryption of SOAP messages in combination with digital signatures. For instance, the work on XML digital signature [50], XKMS [19] and XML encryption [46] might be used to ensure data integrity, non repudiation, and secrecy of SOAP messages. However, the gatekeeper will need to read the message to be able to make an authorization decision. Thus the gatekeeper will need a secret key that the Web service is using for decryption. At this time we do not worry about key distribution. Decryption and encryption in general is an expensive operation. In case the gatekeeper would decrypt, compute the decision, re-encrypt, digitally sign, and forward the message to the Web service, which in turn would need to perform the same steps, the computational work of the access control processor for decryption and encryption would be lost. We can assume that there's a trust relationship between the gatekeeper and the Web service object, for which it makes authorization decisions (e.g., they have to share a secret key). There should be a secure channel between these two so that there's no need for the gatekeeper to encrypt again the message. Signing of the message with the private key of the access control processor does not only verify the integrity of the authorization, but also the integrity of the message when the access control processor received it.

A gatekeeper may become the bottleneck for a Web service with many requests in parallel. Since the gatekeeper is a Web service itself all the tricks that are done with load balancing for Web servers (e.g., server farms, SMP architecture) can be applied to the gatekeeper and all other ACP components.

Adding more objects that need authorization decisions to an ac-

cess control processor adds more burden to its workload. In this case we would simply need to add more access control processors (maybe on different server machines) that work in parallel. In the extreme case each object within a Web service would have its own access control processor. For a very high load and an expensive computational method this scheme could even be further extended to have an access control processor for each method, or even for parameter value ranges of methods. In this case there would have to be some dispatcher mechanism which itself may be carefully designed to avoid a bottleneck. Overall the access control processor architecture is based on a distributed shared-nothing design. This achieves high scalability. Both approaches of distributing one access control processor over many CPUs or different server machines, with the second approach of assigning one access control processor per object (or even further down to method level) should achieve a high scalability.

Another performance issue is caused by the hierarchical architecture of the access control processor along with its distribution. At most we can have a two layer hierarchy comprising access control processors for a Web service collection, and access control processors for a Web service object.

In the smallest scenario there is only an access control processor for a Web service and one on object/method level. As mentioned earlier a particular implementation may decide to combine those to avoid authorization collection requests.

## 5. RELATED WORK

Oppliger [35] reviews current security mechanisms on the Internet. Sandhu and Samarti [38] survey access control principles and models, and provide a solid background on the topic of access control. In addition, Sandhu et al. [39] present an overview of the role-based access control model. The paper uses these access control principles as a basis for the design of an access control processor (e.g., combination of discretionary and role-based access control model).

Oppliger [35] reviews security mechanisms on the Internet and confirms the lack of security on the Web. In addition, Oppliger provides a comprehensive overview of methods for securing applications on top of the Hypertext Transfer Protocol (HTTP). The original HTTP/1.0 [3] specification provided a simple password-based basic authentication. Web servers are using this to provide basic access control (e.g., to protect files or directories). Authentication information is sent using Base64 encoding. Nothing is being done to prevent passive eavesdropping to collect username and passwords. Therefore this authentication scheme is considered weak similar to other TCP/IP applications, such as TELNET or FTP). The consequence is that our design of a distributed access control processor does not support the simple password-based basic authentication in HTTP/1.0.

The HTTP/1.1 [17] specification introduced an improved authentication scheme called *Digest Authentication*, which uses a more elaborate security protocol that no longer transmits passwords in clear. However, this authentication scheme was not widely adopted by commercial browsers and is still considered weak compared to other technologies (e.g., SSL). The paper adopts digest authentication as a possible useful authentication scheme in the design of a distribute access control processor.

To secure communication on the Internet in general, SSL (Secure Sockets Layer) was invented. Its primary goal was to provide a secure communication channel and to authenticate a Web server (optionally the client). SSL operates between any two applications that do not necessarily need to be on the same (secure) network. After 1994, SSL was widely adopted in commercial browsers. There

were many problems with earlier versions of SSL [48]. SSL itself is quite complicated, and comprises many technical details (e.g., versioning, firewall traversal, error handling). It also requires public key certificates. This introduced new problems on how to manage these certificates securely, which helped certification authorities (CA) and public key infrastructures (PKI) to gain momentum. Furthermore, *IPSec* was introduced [15] to create a secure network of computers over insecure channels by providing security for low-level network packets. The main difference between SSL and *IPSec* is that SSL secures two applications, whereas *IPSec* secures an entire network. Both technologies work well in the HTTP environment and could be used as a basis for secure communication for Web services.

There are two major directions for efforts leading to a security model for Web services. The first one is trying to leverage existing security solutions in the area of Internet technologies and information systems [25] (e.g., HTTP digest authentication, usage of SSL). It proposes usage of authentication features found in the HTTP protocol, Web servers, and in operating systems. However, the problem with this approach is that it might only work together with system level security in a corporate homogeneous Intranet scenario. This is in contradiction with the notion of Web services, that are supposed to foster collaboration across network boundaries in heterogeneous and distributed environments.

The second direction towards a security model for Web service is to build a new infrastructure and framework that works in distributed and heterogeneous environments based on new emerging standards (e.g., XML, RDF) and open Internet technologies (e.g., HTTP, SOAP). For instance, one of the motivating factors behind SAML [37] is to enable interoperability between different systems that provide security services. Traditionally, security has been implemented within a single organization. As a goal it would be desirable to have transactions initiated at one site to be completed at a different site. This requires security information to be shared among the various Web sites involved in a transaction. The overall direction where SAML is heading may be promising, and it needs to be investigated further on how this can be integrated in a security model for Web services, and in the design of the proposed distributed access control processor.

Building new technologies based on open standards towards a security framework for Web services seems to have many benefits compared to trying to patch existing infrastructure. A combination of existing, emerging, and new technologies might be a reasonable choice: For instance, leveraging from existing HTTP protocol mechanisms and its extensibility, the usage of emerging open standard (e.g., XML, RDF Schemas), and the design and development of new protocols and specifications that address shortcomings of the previous two.

The WebDAV [49] Distributed Authoring Protocol allows distributed authoring scenarios where resources may be accessible by multiple principals. To control how these principals can access and alter a resource a system of access controls is needed. The WebDAV Access Control Protocol [30] [10] defines access control extensions to the core WebDAV protocol. These extensions allow WebDAV servers to provide an interoperable mechanism to provide access control for content and metadata, and define what actions a particular principal is allowed to exercise on a particular resource.

The access control work in WebDAV is based on the discretionary model. It supports different notions of principals: user, client software, servers, groups. HTTP scheme URLs are used to identify principals. There is a single access control list (ACL) associated with each resource, that determines the operations a principal is allowed to perform on that resource. WebDAV access control

uses HTTP digest authentication. No fine grained access control is supported (only on resource level, not on properties). Furthermore, no role based security model is supported (a role is dynamically defined collection of principals).

Overall WebDAV Access Control Protocol introduces a set of methods, headers, and message bodies that define access control extensions to the core WebDAV protocol. This protocol centric approach is different to the design of a distributed access control processor presented in this paper. Instead the paper designs a system with standardized infrastructure (e.g., that could be implemented as a plug-in or module for a Web server), that could make use of existing protocols and authorization specification languages (e.g., XACML) to provide a general access control solution for the service oriented architecture. The motivation for this approach is that there are already a variety of protocols and languages available for the Web services architecture. Why add another? For instance, there is XACML to specify authorizations in XML and SAML to share security information among the various Web sites involved in a transaction. Therefore a processor that leverages from XACML (which the proposed distributed access control does) and SAML seems to be more reasonable approach for a general design of a distributed access control processor.

The Lightweight Directory Access Protocol (LDAP) [51] does not provide access control in the core protocol. However, there is a requirement to securely access (replicate and distribute) directory information throughout the network within an organization and the Internet. Similar to the Web service architecture, access control is critical for a successful deployment and acceptance of LDAP in the market place. RFC 2820 [41] enumerates the requirements for an access control model for LDAP. The idea to enhance and extend the core protocol to support access control lists. Since this protocol centric approach is similar to WebDAV we have the same argument here how the presented work in this paper relates to LDAP.

However, we could access LDAP using the Web service architecture by introducing a SOAP/LDAP gateway. Hence the presented model for Web services could be applied to LDAP too. Therefore the proposed design in this paper could be extended to provide an access control solution that is different than the current LDAP protocol centric approach.

Furthermore, there are existing object security models for CORBA. For instance, the CORBA Security Service [21] provides a specification for identification and authentication of principals, authorization and infrastructure based access control, security auditing to make users accountable for their security related actions, security of communication between objects, and non-repudiation. The CORBA security model is security technology neutral, which allows to implement CORBA security on a wide variety of existing systems. However, it would need to be further investigated on how reusing the CORBA security mechanisms will help to further improve the proposed design and implementation of an distributed access control processor for Web services.

## 6. FUTURE WORK

The proposed authorization model is flexible and can be used in different contexts. It will be interesting to develop a prototype and integrate it into a major Web service implementation (e.g., Apache SOAP [40]) to gain more insights on how the architecture behaves that will gradually help to refine the design.

WebDAV can be seen as a specialized Web service for distributed collaborative authoring. Its supported methods are GET, PUT, COPY, MOVE, LOCK, PROPFIND, and PROPPATCH. Possible method parameters are filenames or property names. Because of the similarities to the Web service concepts it would be

interesting to explore how the proposed distributed access control processor could be used to provide an alternative access control solution for WebDAV. For instance, we can perform a mapping from WebDAV into the proposed Web services model, and then apply the proposed authorization model. The advantages here would be that access control is represented as a service, and there's no need to extend the WebDAV core protocol itself to deal with access control. However, it needs to be evaluated how this can be accomplished in more detail. A prototype system will help to gain more insights.

Similarly, the presented work in this paper can be very generally applied to different service based Internet information systems to provide a comprehensive and standardized access control solution. No extensions to the core protocol would be necessary. Instead, a Web service wrapper layer would be needed to provide access to the service in a standardized way (e.g., SOAP over HTTP). Today's Internet information systems and protocols share a common set of access control requirements, but have each their own peculiarities. It needs to be further investigated on how the proposed model along with security specifications (e.g., XACML, SAML) is generic enough to cover all these requirements to be truly universal.

Service combinators [8] represent high level primitives that provide a more convenient and robust way to interact with the Web. It is based on the idea of a service that is unreliable, i.e. there's no guarantee that it returns a result for a query, or terminates at all. Therefore the main underlying assumptions are notions of failure and the rate of communication.

The service combinator model defines services as basic services, gateways, sequential or concurrent execution, time or rate limit, repetition, non-termination, and failure. For instance, the *timeout(t,S)* combinator represents a service *S* that better produce a result within *t* seconds after invocation, or else fails. Another interesting combinator is *repeat(S)*. It repeatedly invokes a service until it succeeds, and has otherwise no associated condition for termination. To terminate such a loop we would need to use other means, e.g., *timeout(t, repeat(S))*.

It would be interesting to explore how service combinators can be used to further enhance the proposed abstract model for Web service, i.e. how service combinators can be extended to work in the context of Web services.

Furthermore, the proposed model already includes metadata as an integral part. Currently the automatic discovery of Web services is limited to its interface, i.e. how to interact with a Web service. However, there are no semantics associated with a Web service, that can be extracted automatically. As an example, RDF Schemas [6] may be useful to specify the metadata along with semantics and relation for Web services. Semantic processing would allow software agents to interact with Web services in a more intelligent way. For instance, technologies related to the semantic Web [42] are addressing these issues, but focus primarily on Web pages and not on Web services.

The paper defined a discovery mechanism for authorization as a goal, but did not pursue this topic in more depth. There are subtleties on to not reveal too much information that could be used by an attacker. The question is what information can be safely revealed that clients can extract to foster automatic processing. How would such a discovery protocol be designed (e.g., XML specification)?

## 7. CONCLUSION

The paper argued that security presents the foundation for the wide spread deployment of Web services. We enumerated design goals for a general Web services access control model, and justified why these goals are important.

Then the paper introduced a formal abstract model for a hierar-

chical Web services architecture. The proposed distributed access control processor architecture based on a general model for Web service components incorporates basic Web service objects, as well as aggregation, composition, operations and specialization on Web services, and works as a SOAP filter gateway that acts as an authorization service for Web services. Design decisions and rationale are justified.

The distributed access control processor comprises two components: An access control processor Web service object and a gatekeeper Web service object. The presented architecture is fully integrated into the existing Web service infra-structure, i.e. gatekeeper and access control processors are Web services itself.

The paper concludes that the proposed solution represents a viable starting point that can be used as an infrastructure for further research in the area of access control and security for Web services.

One of the difficulties that remains is to promote the presented model and design so that it may become broadly accepted and eventually an Internet standard. One approach is to integrate some of the work and ideas into existing standardization efforts (e.g., OASIS). Another way of achieving this is to implement a prototype that proves to be useful in the Internet and Web services community, so that developers start adopting the technology so that it can spread widely. For instance, the Apache SOAP project might be a desired target platform for a first prototype implementation. In the meantime other proposals will be developed independently. The Internet has proven to act as a good filter for emerging technology.

## 8. ACKNOWLEDGEMENTS

I am grateful to Jim Whitehead for his valuable comments and suggestions.

## 9. REFERENCES

- [1] Kerberos: The network authentication protocol. <http://web.mit.edu/kerberos/www/>. last accessed: 3/7/2002.
- [2] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Computer Publishing, 2001.
- [3] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol - HTTP/1.0. *Network Writing Group, Request for Comments*, May 1996.
- [4] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. <http://www.faqs.org/rfcs/rfc2396.html>, August 1998.
- [5] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. <http://www.w3.org/TR/SOAP/>, May 2000.
- [6] D. Brickley and R. Guha. Resource description framework (RDF) schema specification 1.0. <http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>. last accessed: 3/7/2002.
- [7] A. Brown, B. Fox, S. Hada, B. LaMacchia, and H. Maruyama. SOAP Security Extensions: Digital Signature. <http://www.w3.org/TR/SOAP-dsig>. last accessed: 3/5/2002.
- [8] L. Cardelli and R. Davies. Service combinators for web computing. *IEEE Transactions on Software Engineering*, 25(3):309–316, 1999.
- [9] A. Ceponkus, P. Furniss, and A. Green. Business Transaction Protocol. [http://www.oasis-open.org/committees/business-transactions/draft\\_0.9.pdf](http://www.oasis-open.org/committees/business-transactions/draft_0.9.pdf), October 2001.

- [10] G. Clemm, A. Hopkins, E. Sedlar, and J. Whitehead. WebDAV Access Control Protocol. <http://www.webdav.org/acl/>.
- [11] R. Cover. AuthXML Standard for Web Security. <http://xml.coverpages.org/authxml.html>. last accessed: 3/7/2002.
- [12] R. Cover. Security Services Markup Language (S2ML). <http://xml.coverpages.org/s2ml.html>. last accessed: 3/7/2002.
- [13] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Design and implementation of an access control processor for XML documents. *Computer Networks*, 33(1-6):59-75, June 2000.
- [14] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Fine grained access control for soap e-services. *WWW10*, May 2001.
- [15] N. Doraswamy and D. Harkins. IPSEC: The new security standard for the Internet, intranets, and virtual private networks. Prentice Hall, 1999.
- [16] B. A. et al. Web Services Security Language (WS-Security). <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security.asp>, January 2002.
- [17] R. Fielding, J. Gettys, J. Mogul, and H. Frystyk. Hypertext transfer protocol - HTTP/1.1. *Network Writing Group, Request for Comments*, (2068), January 1997.
- [18] J. Fontana. Top Web services worry: Security. *Network World*, <http://www.nwfusion.com/news/2002/0121webservices.html?docid=7747>, January 2002.
- [19] W. Ford, P. Hallam-Baker, B. Fox, B. Dillaway, B. LaMacchia, J. Epstein, and J. Lapp. XML Key Management Specification (XKMS). <http://www.w3.org/TR/xkms/>, March 2001.
- [20] M. N. framework. .NET framework homepage. <http://msdn.microsoft.com/netframework/>. last accessed: 3/7/2002.
- [21] O. M. Group. The CORBA security service specification. <ftp://ftp.omg.org/pub/docs/ptc>.
- [22] M. Gudgin, M. Hadley, J.-J. Moreau, and H. F. Nielsen. SOAP Version 1.2 Part 1: Messaging Framework. <http://www.w3.org/TR/soap12-part1/>, December 2001.
- [23] D. F. S. G. J. Barkley, A. Cincotta and D. Kuhn. Role-based access control for the World Wide Web. *20th National Computer Security Conference*, 1997.
- [24] R. Khare and A. Rifkin. Weaving a Web of Trust. *World Wide Web Journal*, 2(3):77-112, summer 1997.
- [25] M. Kirtland. Authentication and Authorization. [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dn\\_voices\\_webservice/html/service02282001.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dn_voices_webservice/html/service02282001.asp).
- [26] R. Kraft. A model for network services on the web. *The 3rd International Conference on Internet Computing (IC 2002)*, 3:536-541, June 2002.
- [27] R. Kraft. Research and design issues of access control for network services on the web. *The 3rd International Conference on Internet Computing (IC 2002)*, 3:542-548, June 2002.
- [28] M. Kudo and S. Hada. XML document security based on provisional authorization. *CCS'00*, 2000. IBM Tokyo Research Laboratory.
- [29] F. Leymann. Web Services Flow Language (WSFL 1.0). <http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>, May 2001.
- [30] L. Lippert. WebDAV Access Control Goals (Internet Draft). <http://www.webdav.org/acl/goals/draft-ietf-webdav-acl-reqts-00.txt>.
- [31] Microsoft Developer Network (MSDN). An Introduction to GXA: Global XML Web Services Architecture. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dngxa/html/gloxmlws500.asp>, February 2002.
- [32] Microsoft Passport. Microsoft Passport homepage. <http://passport.microsoft.com>. last accessed: April 2002.
- [33] T. Modi. WSIL: Do we need another Web services specification? <http://www.webservicesarchitect.com/content/articles/modi01.asp>, January 2002.
- [34] OASIS. eXtensible Access Control Markup Language (XACML). <http://www.oasis-open.org/committees/xacml/index.shtml>.
- [35] R. Oppliger. Methods of securing applications for the world wide web (WWW). *Computer Security Journal*, 15(1):1-9, Winter 1999.
- [36] Organization for the Advancement of Structured Information Standards (OASIS). OASIS homepage. <http://www.oasis-open.org/>.
- [37] SAML. Security Assertion Markup Language (SAML). <http://www.oasis-open.org/committees/security/docs/draft-sstc-saml-01.pdf>. last accessed: April 2002.
- [38] R. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications*, pages 40-48, September 1994.
- [39] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 20(2):38-47, 1996.
- [40] A. SOAP. Apache SOAP homepage. <http://xml.apache.org/soap/>. last accessed: 3/7/2002.
- [41] E. Stokes, B. Blakley, R. Byrne, R. Huber, and D. Rinkevich. Access control model for LDAPv3. <http://www.ietf.org/internet-drafts/draft-ietf-ldapext-acl-model-08.txt>.
- [42] O. L. Tim Berners-Lee, James Hendler. W3C Semantic Web - Web Site. <http://www.w3.org/2001/sw/>. last accessed: 6/3/2002.
- [43] UDDI. UDDI homepage. <http://www.uddi.org/>. last accessed: 3/5/2002.
- [44] W3C. Extensible markup language (XML). <http://www.w3.org/XML/>.
- [45] W3C. Platform for privacy preferences (P3P) project. <http://www.w3.org/P3P/>.
- [46] W3C. XML encryption WG. <http://www.w3c.org/Encryption/2001/>.
- [47] W3C. XML schema. <http://www.w3.org/XML/Schema>.
- [48] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proceeding of 2nd USENIX Workshop on Electronic Commerce*, November 1996.
- [49] J. Whitehead and Y. Y. Goland. WebDAV: A network protocol for remote collaborative authoring on the Web. 1999.
- [50] XML Signature WG. XML Digital Signatures.

<http://www.w3.org/Signature/>. last accessed: 4/15/2002.

[51] W. Yeong, T. Howes, and S. Kille. Lightweight directory access protocol. <http://www.ietf.org/rfc/rfc1777.txt>, 1995.

## APPENDIX

### A. TERMS AND DEFINITIONS

#### A.1 Web service model

The paper uses and extends a hierarchical containment structure for Web services that was first introduced by Kraft [26]. It comprises the following three Web service components:

1. **Web service method** - represents an operation on a Web service object. A Web service method represents a computational function with a well-defined interface, similar to a method in object-oriented programming (see definition 1)
2. **Web service object** - aggregates a set of Web service methods, and has an internal state, similar to an object in object-oriented programming (see definition 2)
3. **Web service collection** - represents a Container for Web service objects and other Web service collections. Provides a convenient way to group (possibly related) Web services objects (see definition 3).

#### A.2 Security and access control

A common problem comes from the lack of clarity about the participants in a security system and the terminology used. Therefore the paper adopts the following terms and definitions that are common in the literature of security and cryptology [2]:

**Subject** A physical person

**Person** A physical person or a legal person (e.g., company, government)

**Principal** Entity that participates in a security system. A principal can be a subject, a person, a role, piece of equipment, a computational agent, a communication channel, a smartcard, or card-reader terminal. A principal can also be a compound of principals (e.g., group, conjunction, compound role, delegation)

**Group** A Set of principals

**Role** Function assumed by different persons in succession. The same person can assume multiple roles

**Identity** Correspondence between the names of two principals signifying that they refer to the same principal

**Authentication** The process of verifying the identity of a principal

**Privacy** The ability to protect a subject's personal secrets. It can be extended to the ability to prevent invasions of a subject's personal space

**Security object** An entity in a passive role to which a security policy applies

**Access control** The prevention of use of a resource by unidentified and/or unauthorized principals in an unauthorized manner

**Access control policy** A set of rules that are part of a security policy

**Authorization** The granting of access to a principal on a security object

**Authorization policy** An authorization policy states what principals are allowed to perform particular actions on specific objects under certain conditions. We will re-use the work that has been done relating to these definitions in the area of XML. For instance, XACML [34] could be used to express and implement authorization policies. An access control processor will read these policies, and compare them with an authorization request.

**Authorization request** An authorization request can be in the classical form (see Sandhu et al. [38]). In this case it is a triple (principal, object, action). We could also use an enhanced version developed by Kudo et al. [28] that uses a form of provisional access control. In this case we would have a quadruple (principal, object, action, condition), which states that a condition needs to be satisfied before or after the action is granted. We do not introduce a new definition. Instead we are re-using the one of Kudo et al.. In this context objects are resources (e.g., Web service methods, Web services objects, and Web service collections). We further allow fine-grained specification of objects using metadata properties to go below the Web service method level.

**Authorization decision** An authorization decision is the result of comparing an authorization request against the set of all authorization policies. It is either granted or denied.

#### A.3 Terms in Internet information systems

The following definitions refer to some terms commonly encountered in Internet information systems, which we need to define first according to RFC 2396 [4].

**Resource** Any concept that has identity can be a resource. For instance, a document, a video file, a Web service, and a collection of other resources. There is no requirement that a resource has to be network accessible. As an example, books, animals, and vinyl records are also considered resources. A resource represents a conceptual mapping to an entity or set of entities, not the entity itself at a particular moment in time. This allows that the resource itself remains constant over time, even if the actual content of the resource changes.

**URI** A Uniform Resource Identifier (URI) is a compact sequence of characters with restricted syntax for identifying an abstract or physical resource. It can be further classified as a locator, a name, or both, hence a URI represents a superset of URL and URN (see below).

**URL** A Uniform Resource Locator (URL) refers to the subset of URI that identify resources by their primary network access mechanism (e.g., HTTP protocol).

**URN** A Uniform Resource Name (URN) refers to the subset of URI that is not location dependent and remains persistent even if the resource ceases to exist or becomes temporarily unavailable.

The terms aggregation and composition are not always used consistently in the literature, and the difference between these two is not strict. Both share in common the notion of containment, a **has a** relationship. As an example, a computer has a CPU, a cat has a tail, etc.

**Aggregation** Objects being part of one aggregation can potentially be shared by other aggregates.

**Composition** If a composite object in some sense "owns" the component exclusively, then the relationship is described more precisely as one of composition.