Specification and Enforcement of Access Control in Heterogeneous Distributed Applications

Torsten Fink, Manuel Koch, and Cristian Oancea

Institut für Informatik
Freie Universität Berlin, 14195 Berlin, Germany
{tnfink,mkoch,oancea}@inf.fu-berlin.de

Abstract. Security is a crucial aspect in any modern software system. We consider in this article the specification and the management of access control in in-house business applications which are coupled over the Internet using Web services. In-house business applications are usually built on a middleware in which security is an established aspect and security management tools are available. The integration of security in SOAP, however, is still an ongoing activity.
Therefore, we propose an access control model for Web services which originates from CORBA-based applications. This integration has the advantage of a unique access control policy for both the CORBA-based in-house application and the Web services. We implemented an infrastructure to enforce the access control policy and a comprehensive set of powerful XML based management tools.

Keywords: SOAP-based web services, access control, RBAC, CORBA, security management, interoperability.

1 Introduction

Security is an integral part of most modern software systems that are not used in completely trusted environments. We consider in this article in-house business applications that are integrated over the Internet by SOAP-based Web services. There is an established support for security in the middleware, e.g. CORBA [15] or Java RMI, used in in-house business applications, but there is still an ongoing effort in establishing standards for Web service security.

Nowadays, due to the lack of a Web security standard and corresponding implementations, Web security is mostly delegated to the security mechanisms of the Web server which acts as application server. For example, the Tomcat Web server [2] provides a security manager for the administration of users, groups, roles, and permissions to access Java-Web applications. The authorization in Tomcat, however, is coarse-grained, in the sense, that it is not possible to restrict the access on single operations of Web services. Moreover, delegation of access decision is as well a problem as the reuse of access control policies in different frameworks, e.g. .Net [11].

There are emerging XML-based security standards from W3C and OASIS for Web security. The Web Services Security (WSS) specification [13] from OASIS

M. Jeckle and L.-J. Zhang (Eds.): ICWS-Europe 2003, LNCS 2853, pp. 88–100, 2003.
© Springer-Verlag Berlin Heidelberg 2003

describes a mechanism for securely exchanging SOAP messages. It provides the following three main security features: the ability to send security tokens (e.g., user name, certificates) as part of a SOAP message to authenticate users, to provide message integrity, and message confidentiality.

The Security Assertion Markup Language (SAML) [12] from OASIS is an XML framework for exchanging authentication and authorization information, whereas the eXtensible Access Control Markup Language (XACML) [14] presented by OASIS lets you express authorization and access control policies in XML. XACML defines a vocabulary to specify subjects, rights, objects, and conditions. Implementations of the XACML standard already exist, for example Sun's XACML implementation [19] written in Java or the access control engine *jiffyXACML* by Jiffy Software [10]. To our knowledge, however, there is no implementation which enforces a given XACML policy in an application.

Unlike all approaches mentioned above which concentrate on developing new approaches for security in Web services, we propose to integrate a well-established security model from a different domain into Web service applications. This security model is called *View-based Access Control* (VBAC) [6] which originates from CORBA-based applications. VBAC is a variant of role-based access control [16] and provides a specification language, *View Policy Language* (VPL), in which access control policies can be specified. Authorizations can be specified fine-grained on the level of single Web service operations and authorization rights can change dynamically, triggered by operation calls. The VBAC model is implemented in the CORBA-based infrastructure Raccoon [5] accompanied with a comprehensive set of powerful XML based management tools. Raccoon enforces a VPL policy for CORBA-based systems.

We identified the features needed to describe access control for Web services, extended the VPL accordingly and applied the VPL to the specification of access control requirements in Web services. We use the Raccoon infrastructure to enforce a Web service security policy. Therefore we developed a proxy which intercepts SOAP messages. This proxy extracts the authentication and service information and delegates access decision to the Raccoon infrastructure. The Raccoon infrastructure decides if the request is rejected or if the Web service is executed.

The advantages of an integration of a well-established security model into Web services are:

- A unique access control model and access control language for both in-house applications and Web services. This reduces the expert security knowledge required by system designers and administrators.
- A powerful access control language in which dynamic right changes can be expressed and authorization can be explicitly granted or denied on the level of Web service operations. For example, XACML does not provide a possibility for dynamic right changes, Tomcat specifies access control on the level of entire Web applications only.
- The VPL was especially designed for readability and an easy use by developers and system administrators. XACML is mainly designed to describe access

control policies in a format that allows for an efficient machine evaluation of access requests against access requirements. So, the VPL is more suitable for system designers and administrators and reduces design and administration errors.
- VPL supports automatic type and consistency checks. For example, contradicting authorization assignments can be statically detected.
- VBAC and VPL are accompanied by an infrastructure which enforces access control policies written in VPL. There are XML based graphical management tools for policy deployment and policy administration. These tools can be used without any new implementation.

The VPL access control specification and the XACML specification, however, do not mutually exclude, since the VPL specification can be transformed in a XACML specification by an XSL stylesheet. Therefore, the proposed specification language VPL can be also used as a manageable front-end for a XACML specification. Moreover, the VPL is independent of Raccoon and can be used also with other access control enforcement engines.

The remainder of this paper is organized as follows: Section 2 introduces a video rental example which is used throughout the paper. In section 3 we identify the access control requirements of the video rental example and introduce the VPL and the VBAC model. Section 4 presents a brief overview of the Raccoon infrastructure and shows how it can be used to enforce a VPL access control policy for Web services. Section 5 concerns related work and section 6 concludes the paper and points to future work.

2 Example – Video Central

Video Central is a Web service sample application provided by IBM [9]. The Video Central contains a central data repository that can be accessed by registered Web-based applications (see fig. 1).

There is one business-related Web service, the *business registration*, that allows a business (a video store) to register with the Video Central. Once a video store registered with Video Central, it is able to use the following customer-related Web services:

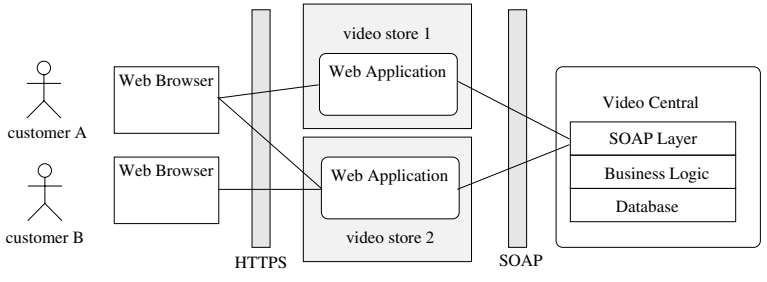


Fig. 1. Video Central architecture.

- *Customer Registration*: Enables the registered video store to register their own customers with Video Central.
- *Customer Infraction*: Enables the registered video store to add and query infraction information to and from the central repository.
- *Customer Rented List*: Enables the registered video store to add and query rented videos for each customer.
- *Customer Wish List*: Enables the registered video store to add and query the titles of videos the customer has expressed an interest in watching.
- *Movie Search*: Enables the registered video store to search for movie titles. This Web Service can be used by the staff or customers of the store.

The UML design of Video Central is given in [9]. Figure 2 shows the part of the class diagram needed for the paper.

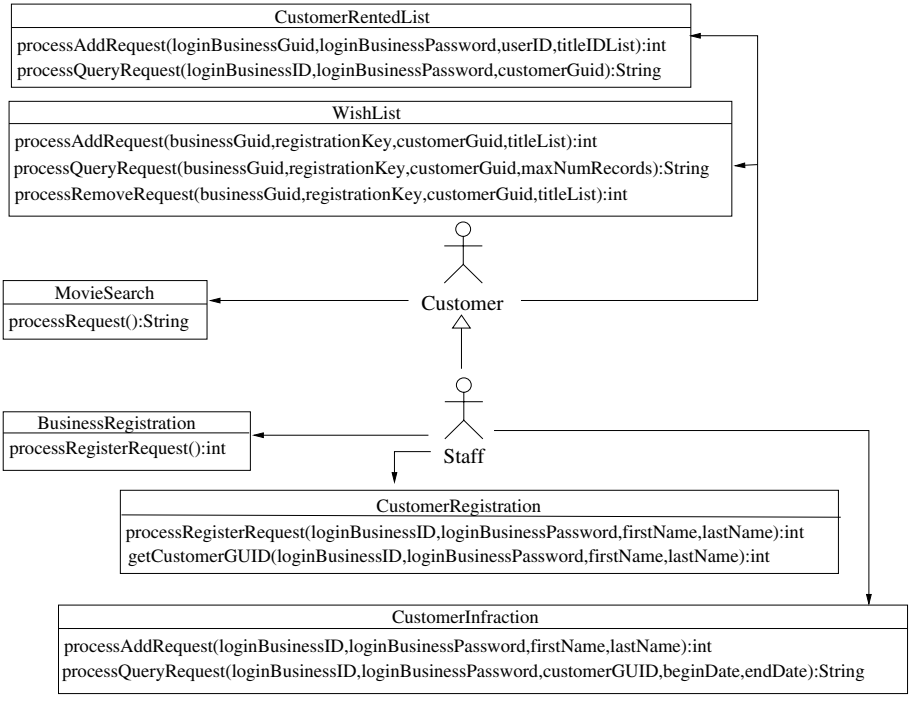


Fig. 2. Video Central class diagram.

It shows the classes for the six Web services and their operations. The operation parameters for the operations of the classes BusinessRegistration and MovieSearch are not shown, since we will not need them later in the paper. The operation parameters of the remaining classes are relevant and are depicted in the class diagram. The arrows between actors and Web services show in which Web services the actors participate.

3 Access Control

We identify in this section the access control requirements in the Video Central application and introduce afterwards the View Policy Language and the View-based Access Control model in which we express the Video Central access control requirements.

3.1 Access Control in Video Central

In the Video Central application, the accesses on the services are performed by video store staff members and video store customers. We get the following access control requirements for the above Web services:

- *Business Registration*: Can be used by a video store staff member (this will be in general a video store manager). The business registration is a prerequisite for all remaining Web services.
- *Customer Registration*, *Customer Infraction*: Staff members can use these Web services, but only if their video store is registered with Video Central. Staff members can use the Web services only with respect to their customers. For example, a staff member of video store A is denied to register a customer for video store B.
- *Customer Rented List*, *Wish List*: Staff members can use these Web services (again only for their customers). Customers can check the list of videos rented by themselves, but they cannot add requests to the list. Customers can add, remove, and query their wish list.
- *Movie Search*: This Web service can be used by staff members and customers.

3.2 View-Based Access Control

View-based access control (VBAC) is an access control model specifically designed to support the design and management of access control policies in distributed systems. VBAC is an extension of *role-based access control* (RBAC) [16, 17] by the concept of views and schemas. A *view* groups permissions or denials to access objects (e.g., a file, a method, printer) and views are assigned to roles. A subject (e.g., a user, a process) can access an object if the subject has a role to which a view with the required permissions is assigned. If such a view does not exist for the role, the subject cannot access the object. A *schema* specifies dynamic assignments or removals of views to and from roles. These assignments are performed automatically whenever a given operation is executed.

The View Policy Language (VPL) is a declarative language to specify VBAC policies [6,4]. The VPL is used to specify roles, views, and schemas. Roles are defined in a role declaration (after the keyword **roles**), the views in several view definitions (introduced by the keyword **view**) and the schemas in several schema definitions (introduced by the keyword **schema**).

The role clause defines the available roles in the policy together with their initial views. Figure 3 depicts the role definition for the Video Central application

in which we have two roles, namely `customer` and `staff`. The role `staff` has initially the view `BusinessRegistration` (keyword **holds**), the role `customer` does not have an initial view.

A VPL view defines the authorization for a subset of operations of a single Web service, which is referenced in the **controls** clause. For example, the view `BusinessRegistration` gives the permission to call the operation *processRegisterRequest* of Web service BusinessRegistration. A VPL view can be statically restricted to certain roles listed after a **restricted to** clause, so that it can only be assigned to these roles. For example, the view `BusinessRegistration` can be assigned only to the role `staff`, but not to `customer`.

The view `CustomerRegistration` gives the permission for the operations *getCustomerGUID* and *processRegisterRequest*. Since a video store can only register its own customers (see requirements in Sec 3.1), the operation can be called only with a subset of the possible operation parameters. In our example, *getCustomerGUID* can be called only with the *loginBusinessID* of the calling video store (a video store gets its *loginBusinessID* after registering with Video Central by the Web service BusinessRegistration). Analog, the operation *processRegisterRequest* can only be called with the *loginBusinessID* of the calling video store.

This restriction of parameters is not necessary in the VPL originating from CORBA-based applications. An operation can be granted so that it can be called with all possible operation parameters or it is denied. This is due to the fact that in CORBA-based applications a common design pattern is used that creates an object on the server side for each client. The client communicates with this object and thus no identification is necessary. For example, in a Video rental CORBA application each client would get its own *CustomerRegistration* object on the

```
role customer
     staff holds BusinessRegistration

view BusinessRegistration controls BusinessRegistration restricted to staff {
        processRegisterRequest }

view CustomerRegistration controls CustomerRegistration restricted to staff
{
        getCustomerGUID(loginBusinessID,_,_,_) if caller = loginBusinessID

        processRegisterRequest(loginBusinessID,_,_,_) if caller = loginBusinessID
}

view CustomerInfraction controls CustomerInfraction restricted to staff {
        processAddRequest( loginBusinessID,_,_) if caller = loginBusinessID

        processQueryRequest( loginBusinessID,_,_,_,_) if caller = loginBusinessID
}
```

Fig. 3. The VPL role and view declaration.

server side. SOAP, however, does not support these object-oriented concepts [21]. Instead a general function is used where the *loginBusinessID* is explicitly given as a parameter.

We extended the VPL with the possibility to specify conditions of the form **if caller** = *param*, where the keyword **caller** represents the ID of the caller of a Web service and *param* is an operation parameter. The caller of a Web service is identified by an integer, i.e., only parameters of type integer can be used in the condition **if caller** = *param*. This is a first attempt sufficient for the Video Central example. If a parameter x_i of an operation $op(x_1, .., x_i, ..., x_n)$ is not restricted (i.e., all values are allowed in an operation call), we denote this by $op(x_1, ..., _, ..., x_n)$. If there is no condition for the parameters of $op(x_1, .., x_n)$ in a view, the operation can be called with all possible parameter values. In this case we simply write the operation name *op* without parameter list (e.g., *processRegisterProcess* in view `BusinessRegistration`).

The view `CustomerInfraction` gives the permission to call the Web service operations *processAddRequest* and *processQueryRequest*, again only for the customers of the video store.

```
view CustomerRentedListFull controls CustomerRentedList
                                restricted to staff {
        processAddRequest( loginBusinessID,_,_,_) if caller = loginBusinessID

        processQueryRequest( loginBusinessID,_,_) if caller = loginBusinessID
}

view CustomerRentedListRestricted controls CustomerRentedList {
        processQueryRequest( _,_, customerGUID) if caller = customerID
}
```

Fig. 4. The VPL view declaration.

There are different access requirements for the roles `staff` and `customer` with respect to the Web service CustomerRentedList. A staff member can call both operations *processAddRequest* and *processQueryRequest*, but a customer can call *processQueryRequest* only. The view `CustomerRentedListFull` is restricted to the role `staff` and gives the permission to call both operations. The view `CustomerRentedListRestricted` permits only to call *processQueryRequest* and is not restricted to a certain role, but to certain parameters. The operation can be called only if the **caller** is the customer for which the rented list shall be checked.

For the Web service WishList, we have again different views for customers and staff members. Staff members can add, remove, and query the wish list of all their customers. A customer can add, remove and query only his/her own wish list. This is reflected in the view `WishListFull` for the role `staff` and view `WishListRestricted` for the role `customer`. The Web service MovieSearch can be used by staff members and customers without any restriction (see fig. 5).

```
view WishListFull controls WishList restricted to staff {
        processAddRequest(businessGuid,_,_,_) if caller = businessGuid

        processQueryRequest( businessGuid,_,_,_) if caller = businessGuid

        processRemoveRequest( businessGuid,_,_,_) if caller = businessGuid
}

view WishListRestricted controls WishList restricted to customer{
        processAddRequest(_,_,customerGuid,_) if caller = customerGuid

        processQueryRequest(_, _, customerGuid,_ ) if caller = customerGuid

        processRemoveRequest(_,_,customerGuid,_) if caller = customerGuid
}

view MovieSearch controls MovieSearch {
        processRequest }
```

Fig. 5. The VPL view declaration.

Initially only the view `BusinessRegistration` is assigned to a role, namely to `staff`. Therefore, users in role `staff` can access the Web service BusinessRegistration but they cannot access any other Web service. Customers can not access any Web service. Customer-specific Web services become available not before a video store has registered with Video Central. After the video store registered with Video Central, the views to access the Web services are assigned to the roles. The dynamic assignment and removal of views (and so of authorizations) to/from roles is specified in *VPL schemas* introduced by the keyword **schema** followed by the schema name and an **observes** clause that specifies the Web service for which the schema is defined. The assignment (removal) of views is triggered by the execution of an operation of a Web service. After the operation name *op*, we specify by **assign** $V_1, ..., V_N$ **to** R that the views V_1,...,V_N are assigned to role R after calling the operation *op*. We specify by **remove** $V_1, ..., V_N$ **from** R that the views V_1,...,V_N are removed from R after calling *op*. Fig. 6 depicts the schema of the Video Central example, which specifies the view assignment after a video store has registered with Video Central.

4 Implementation – Enforcing the Policy

We show in this section the infrastructure to enforce an access control policy specified in VPL. This infrastructure is integrated into the Raccoon infrastructure [5] which enforces VPL policies in CORBA environments. We start with an introduction of Raccoon and show afterwards how we have integrated SOAP messages.

```
schema BusinsessRegistration controls BusinessRegistration {
    processRegisterRequest
        assigns CustomerRegistration, CustomerInfraction,
                CustomerRentedListFull, WishListFull, MovieSearch to staff

        assigns CustomerRentedListRestricted,
                WishListRestricted, MovieSearch to customer
}
```

Fig. 6. The VPL schema declaration.

4.1 Raccoon Infrastructure

The Raccoon infrastructure consists of a deployment tool which processes VPL policies and stores view and role definitions in repositories (role and policy server) that can be managed using graphical management tools. Access decisions are made on the basis of policy information that is supplied by these repositories.

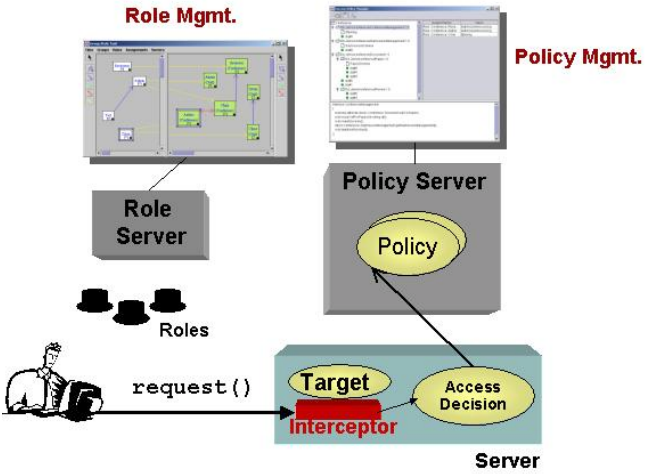


Fig. 7. Security infrastructure.

Figure 7 shows the main parts of the Raccoon infrastructure. A role server contains all the role certificates of a user. When a client authenticates at the system, he/she gets all their role certificates. The role certificates are transmitted when a client calls a server operation. This call is intercepted by a CORBA interceptor which forwards the client information to an access decision object. This access decision object decides on the basis of the given roles and the policy if the client is permitted to call the operation. The current policy can be requested on the policy server. If the policy permits to call the operation for the client, the interceptor forwards the request to the servant. If the client is not permitted to call the operation, the request is denied.

Graphical management tools support the management of access control. There is a tool for the definition of roles and the user role assignment, a tool for the definition of views and a tool for the definition of policies, i.e., the relation of views to roles. Modifications made in these tools become immediately valid and are enforced in the distributed system.

4.2 Access Control Enforcement in Web Services

We use Raccoon for the management and the enforcement of VPL policies in Web service environments. The idea is to intercept SOAP messages and to delegate the access control decision to Raccoon. Depending on the Raccoon decision, the SOAP message is forwarded or rejected. Figure 8 shows the integrated architecture. We use HTTPS as communication protocol, since messages are encrypted and client certificates are transported on the server side.

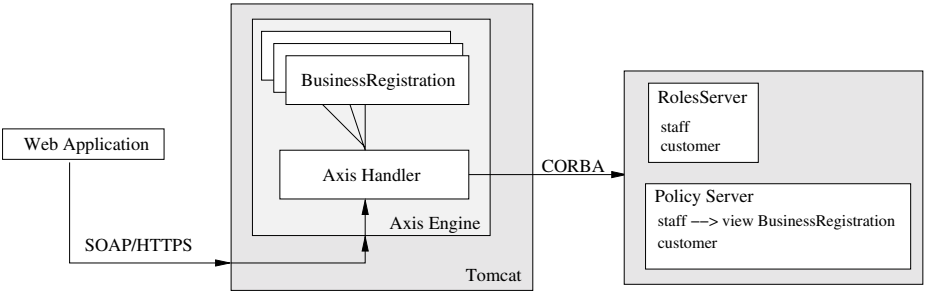


Fig. 8. Security infrastructure.

Deployment of Web Services
The Web services are deployed on the Web server. We use the Apache Tomcat Web Server [2] together with the Axis SOAP implementation [1] in order to take advantage of the concept of Axis *handlers*. A handler is a component that acts on a specific part of a SOAP message; for example, a handler can be in charge of performing authentication on the message's sender before allowing it to be processed by the service provider.

Policy Deployment
The VPL policy for the Web services is deployed in the Raccoon infrastructure. Raccoon is a CORBA-based application and the deployment of the VPL policy needs the CORBA interface repository. The interfaces defined in the Interface Definition Language (IDL) are derived from the WSDL document of the Web services. This transformation in IDL can be done by an XSL stylesheet. For example, WSDL *portTypes* are IDL interfaces, WSDL operations are IDL operations, WSDL <*input*> elements for operations are IDL **in** parameters and WSDL <*output*> elements are IDL **out** parameters.

Axis Handler for Access Decision
We implemented a handler on the server side which is located before the Web service, i.e., the SOAP message must pass this handler before it reaches the service. The handler serves as CORBA client to communicate with the CORBA-based Raccoon infrastructure, i.e., the Raccoon role and policy server. When a SOAP message passes the handler, the handler extracts the user information (e.g., certificates) and the requested method name and the parameters from the SOAP message. This certificates are used to get the role certificates of the user from the Raccoon role server. The role server stores the relation between user certificates and roles. The policy server contains the views and the view-role association, so that the handler can determine, on the basis of the client roles, if the access to the requested Web service operation is permitted or denied. If access is denied, an exception is thrown. Otherwise, the SOAP message is transmitted to the Web service.

5 Related Work

In [3] role-based access control (RBAC) is applied to Web applications. RBAC is implemented as a CGI script for the use with a Web server. The creation of users, roles and authorizations are created by an Admin Tool. Thus, in some aspects this work constitutes a predecessor of our work.

Damiani et. al present in [8,7] an approach for the definition of authorizations at the level of individual XML elements and attributes. On the basis of such an authorization policy, a tool produces a view on an XML document for each requester, showing the XML elements and attributes for which the requester has a permission. In contrast to our approach, where a client can see all Web services but get an exception if a denied service is called, Damiani et. al hide the denied services. Moreover, their approach misses a complete management environment, the underlying access control model does not provide dynamic right changes nor the concept of roles and group definition.

Sirer and Wang present in [18] an approach for formal specification and enforcement of security policies on Web services. The specification language is based on temporal logic. A policy is processed by an enforcement engine which produces platform-specific code that is integrated into the Web server. This approach, however, is difficult to handle in dynamic environments (e.g., changing access control requirements). An authorization change requires to write a new policy, to generate new code, and to integrate it in the Web server. In our approach dynamic right changes can be done by graphical management tools and policy modifications are immediately enforced without changing the web application. Moreover, the policy language in [18] focus more on the specification of sequencing actions by inserting predicates and implications. Actions are URL invocations initiated by clients and server-side script executions.

The IONA Security Framework (iSF) [20] provides security service adapters for Web Services, J2EE and CORBA applications allowing interoperable security enforcement. In contrast to our appraoch, however, the iSF does not support the

designer or administrator with a declarative description language as the VPL. Dynamic right changes specified in VPL schemas and a permission management by views are not possible, either.

6 Conclusion and Future Work

We used the View Policy Language (VPL) to specify access control requirements in Web services. The VPL and the corresponding access control model originate from CORBA-based applications. Extensions of the VPL were necessary to cover all requirements occurring in Web services. We implemented an infrastructure to enforce an access control policy in Web-based applications. We also provide deployment and management tools for Web service security management.

The VPL can be used as a front-end for XACML specifications. We have to investigate the mapping from VPL to XACML in more detail to see if all VPL features can be mapped and if so, how this mapping should look like. Future work will concern also the mapping of the Raccoon infrastructure to the XACML model. This would provide interoperability with other systems, such as jiffyXACML or Sun's XACML implementation.

References

1. Apache. Axis. `http://ws.apache.org/axis/`.
2. Apache. Tomcat. `http://jakarta.apache.org/tomcat/index.html`.
3. J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrila, and D. Kuhn. Role-based Access Control for the World Wide Web. In *Proc. of the 20th National Information System Security Conference*, 1997.
4. G. Brose. *Access Control Management in Distributed Object Systems*. PhD thesis, Freie Universität Berlin, 2001.
5. G. Brose. Raccoon – An infrastructure for managing access control in CORBA. In *Proc. Int. Conference on Distributed Applications and Interoperable Systems (DAIS)*. Kluwer, 2001.
6. G. Brose. Manageable Access Control for CORBA. *Journal of Computer Security*, 4: 301–337, 2002.
7. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Fine Grained Access Control for SOAP E-Services. In *Proc. of 10th WWW Conference*, 2001.
8. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A Fine-Grained Access Control System for XML Documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(2), 2002.
9. IBM. *IBM Video Central for e-business tutorial, Version 7.2*, 2002. `http://www-106.ibm.com/developerworks/webservices/demos/videocentral/`.
10. jiffySoftware. jiffyXACML. `http://www.jiffysoftware.com/xacml/index.html`, April 2003.
11. Microsoft. .Net Framework. `http://www.microsoft.com/net/`.
12. OASIS. *Security Assertion Markup Language (SAML) v1.0 Specification*, November 2002.

13. OASIS. *Web Services Security: SOAP Message Security, Working Draft 11*, March 2003.
14. OASIS. *XACML 1.0 Specification*, February 2003.
15. OMG. *Common Object Request Broker Architecture: Core Specification V.3.0.2*, December 2002.
16. R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST Model for Role-Based Access Control: Towards A Unified Standard. In *Proc. of the 5th ACM Workshop on Role-Based Access Control*. ACM, July 2000.
17. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role–based access control models. *IEEE Computer*, 29(2): 38–47, February 1996.
18. E. G. Sirer and K. Wang. An Access Control Language for Web Services. In *Proc. of 7th ACM Symposium on Access Control Models and Technologies*, pages 23–30. ACM Press, 2002.
19. Sun. *Sun's XACML Implementation*, 2003. `http://sunxacml.sourceforge.net/`.
20. IONA Technologies. Enterprise Security in Web Services – White Paper, 2002.
21. S. Vinoski. Web Services Interaction Models. *IEEE Internet Computing*, pages 89–91, May/June 2002.