Proceedings of the 9th IASTED International Conference
Software Engineering and Applications (SEA 2008)
November 16 - 18, 2008 Orlando, Florida, USA

MODEL-DRIVEN DEVELOPMENT OF ACCESS CONTROL POLICIES FOR WEB SERVICES

Christian Emig[1], Sebastian Kreuzer[1], Sebastian Abeck[1], Jürgen Biermann[2], Heiko Klarl[2]

[1]Research Group Cooperation & Management, Universität Karlsruhe, Germany
[2]iC Consult GmbH, Keltenring 14, 82041 Oberhaching, Germany
{ emig | kreuzer | abeck } @ cm-tm.uka.de, { biermann | klarl } @ ic-consult.de

ABSTRACT
Web service-oriented architecture (WSOA) is a promising paradigm for future software development. Necessary identity management (IdM) architectures for WSOA are just being prepared to enable fine-grained access control. With the loose coupling of Web services with cross-cutting identity services the question arises how to develop access control policies for Web services. In this paper we present a model-driven approach defining access control policies which are independent from the IdM architecture to which they are later applied. Therefore we develop a platform-independent access control model for WSOA and derive a platform-specific model from a given IdM product. We show how to map both models to a concrete language. Access control policies are then defined using our platform-independent language and transformed to platform-specific policies using explicitly defined transformation rules. We present a case study that applies our approach.

KEY WORDS
Access Control, Service-Oriented Architecture, Model-Driven Development

1. Introduction

Currently, most enterprises try to align their business processes with the supporting IT by migrating towards Web service-oriented architecture (WSOA) [1]. Besides the development of WSOA's core concerns (cf. to separation of concerns [2]) there are cross-cutting concerns that have to be addressed before being able to go productive with WSOA, a central one is enabling security, especially access control [3]. Access control consists of authentication and authorization verification. Authorization verifications are made by the IdM architecture, typically at a component called policy decision point (PDP) and are enforced by a component called policy enforcement point (PEP). We presented an architectural blueprint of a WSOA-aware IdM architecture in [4].

In this paper we build upon our conceptual access control metamodel for WSOA presented in [5] and develop a domain-specific language called Web Services Access Control Markup Language (WSACML). While our conceptual access control metamodel defines the sets and relations necessary for the decision on authorization verification requests, WSACML defines an appurtenant language for the expression of such policies. WSACML defines the platform-independent model (PIM) in the context of OMG's model-driven architecture (MDA) [6] considering the IdM product as the exchangeable platform. The goal is that during Web service development its security aspects can be defined in parallel. Policies can then be modeled independently from the IdM architecture that will be used at service runtime. When deploying the Web service, its policy is transformed conforming to the policy metamodel of the given IdM product and deployed there. This allows Web service developers to reduce dependencies towards an IdM product and to incorporate IdM as a loosely-coupled infrastructural service.

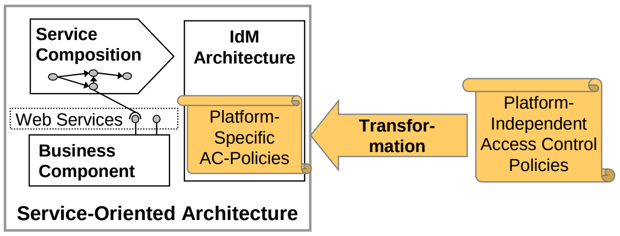
Service Composition
IdM Architecture
Web Services
Platform-Specific AC-Policies
Business Component
Transfor-mation
Platform-Independent Access Control Policies
Service-Oriented Architecture

Figure 1. Transformation of Access Control Policies

The contributions of this paper are:

1. We define a **policy model for Web service-oriented architecture** that is platform-independent from a given IdM product and we present a concrete language which applies this model called **Web Services Access Control Markup Language (WSACML)**.

2. We show how to derive a **platform-specific policy model** of an existing IdM product and how to link these two models using **explicitly defined transformation rules**. This allows for a model-driven development starting with platform-independent policies.

The paper is organized as follows: section 2 gives the background on access control and model-driven development and discusses related work. In section 3 we present WSACML as a platform-independent access control policy language for Web services. In section 4 we show how to derive a platform-specific policy metamodel of a state-of-the-art IdM product and how to couple it with the platform-independent one. In section 5 our approach is applied practically in a case study. A conclusion and an outlook on our future work in this area close the body of the paper.

2. Background and Related Work

2.1 Access Control Models

The purpose of access control models is to define sets and relations for the definition of authorization statements which are used to make access control decisions. The core of access control models is about the definition of the so-called subject/object-relation [7] which formally models an active subject for getting access to a protected object. Role-based access control (RBAC) [8] introduces the role element as an indirection between subjects and objects. Subjects are assigned to (business) roles and authorization is not granted to individuals but to roles which eases administration.

An enhancement of RBAC for Web services is introduced in [9] featuring attribute-based access control (ABAC). It allows for complete decoupling of subjects and objects. Both are characterized by attributes, e.g. their metadata. Policies very abstractly define general authorization statements. While this approach offers much flexibility, the problems not solved yet are about a concrete policy language which needs to be very precise to handle this independence in the subject/object-relation. The decoupling of policies from the objects to be protected adds complexity as policy matching algorithms are needed which are not introduced in [9]. A policy decision point that is able to handle this flexibility has not been presented in this work either.

The OASIS standard eXtensible Access Control Markup Language (XACML) [10] allows for modeling of platform-independent access control policies. XACML does not focus on the Web service domain but is a general policy language. It features the decoupling of access control policies from concrete objects using policy matching algorithms and a generalized way of characterizing subjects and objects using attributes. Focusing on WSOA, the object to be protected is the Web service operation defined by its signature in WSDL. Due to the granularity of a Web service operation it is not enough to only take care of its static aspects but the operation's parameters that are sent during invocation must also be considered. XACML-compliant policy artifacts are voluminous due to XACML's notable syntactical overhead, which hinders its usage for business process designers. IdM products setting up on generalized XACML-based access control policies are rather seldom at present. These characteristics do not favor XACML to be a suitable policy language handling platform-independent analysis and design during Web service development. Considering model-driven development (cf. following section) it is to be positioned at PSM/PSC level.

2.2 Model-Driven Policy Development

Model-driven development is an approach to software development that focuses on models and their transformations as primary engineering artifacts. OMG did a specification of this general approach called model-driven architecture (MDA) [6]. The core elements are platform-independent models (PIM) and platform-specific models (PSM). The link between such two models is drawn by transformation rules, specified between elements of the respective metamodels.

The early treatment of security and especially access control has evolved to be a critical factor in software development projects. The integration of specifying access control in Web service development processes still lacks a standardized approach. Thinking of model-driven development of access control policies for the Web services domain, a domain-specific but platform-independent policy language is a necessary prerequisite. Well-established security infrastructures are already in use for protecting traditional Web applications, so it is reasonable to integrate them into the Web services world. Compliance requirements and therefore the explicit definition of security aspects are drivers for refactoring existing platform-specific policies to platform-independent ones making them understandable in a better way for business analysts.

A contribution to model-driven security is given in [11]. To integrate access control aspects into the Web services development process the authors defined an OCL-like grammar [12] for modeling authorization knowledge and developed a transformation tool which accepts this grammar for the generation of platform-independent XACML-compliant policies. They suggest attaching the authorization knowledge during software design phase to interface models. Proceeding to implementation phases they transform this knowledge into XACML-compliant policies. They suggest employing XACML as target platform but neglect the integration of existing security infrastructures. To be effectively applicable, a further transformation step into a given IdM product is missing.

A further approach to model-driven security is given by [13]. The authors suggest a development process that combines platform-independent models based on OMG MOF [14], one for system design with another one concerning security aspects. Additionally, a tool is presented to transform the platform-independent authorization models into a platform-specific representation for JEE and .NET components. They discuss a generic methodology for integrating security aspects into software development processes, but an employable instantiation of this approach for use in the Web service domain is not presented neither for the system design metamodel nor the security design language. Moreover, MOF-based metamodeling is a heavyweight extension of UML in the sense that it requires superior competences from security architects and lacks tool support. Furthermore, it is complicated to enhance the policy model towards Web service domain, especially to consider Web service invocation parameters. Additionally, the transformation rules presented are very specific to the integrated security models of JEE and .NET platform. A transformation to a loosely-coupled IdM product is somewhat more complicated as policy models are very different - a fact that is not addressed.

3. Web Services Access Control Markup Language (WSACML)

In this section we present Web Services Access Control Markup Language (WSACML), an XML-based language for the definition of access control policies for Web services. WSACML policies are independent from the IdM product which is used later, so they relate to the platform-independent model (PIM) of MDA. They build the first step for model-driven policy development. Using platform-independent policies allows for an abstraction of the highly technical representation inside the conrecte IdM products. WSACML extends our domain-specific conceptual access control metamodel for Web service-oriented architecture as defined in [5] towards a concrete language. First we give an overview of the relevant sets and relations for access control policies in Web service-oriented architecture.

The **objects** to be protected in WSOA are Web service operations provided by atomic Web services or compositions. As their interfaces are technically the same, their access control policies can be handled similarly. Following the WSOA paradigm, a mapping from business processes towards Web services covering the IT-supported parts takes place. Taking access control for Web services into consideration, we focus on so-called usage contexts of a Web service. From a business perspective a usage context directly relates to the invocation of a Web service during a concrete business process. A usage context describes which subjects (users or user agents) should have access to specific objects to accomplish their tasks or to meet other business requirements.

A **subject**, e.g. a human user or a self-acting service, is defined by a collection of various subject attributes forming its digital identity. Considering just the subject's business role does not scale with a constantly growing number of fine-grained Web services as argued in [9]. Furthermore, role-based access control is currently one of the most relevant concepts in enterprise level access control systems and it is strongly related to business process modeling. As practical experience shows, the mapping of business roles to system roles is always cumbersome. Our approach allows a business role to be represented either by a specific subject attribute carrying the role or it can be mapped to a role-specific set of subject attributes.

The signature of a Web service operation aggregates several parameters, input parameters as well as output parameters. Constraining **input parameters** in dependency to a usage context enables fine-grained access control. This allows for instance the comparison between input parameters and attributes of the calling subject. An example for the checking of input parameters is given in the case study. Besides subject attributes and input parameters, constraints on **environment attributes** like time or location are also relevant to describe usage contexts and to make access control decisions.

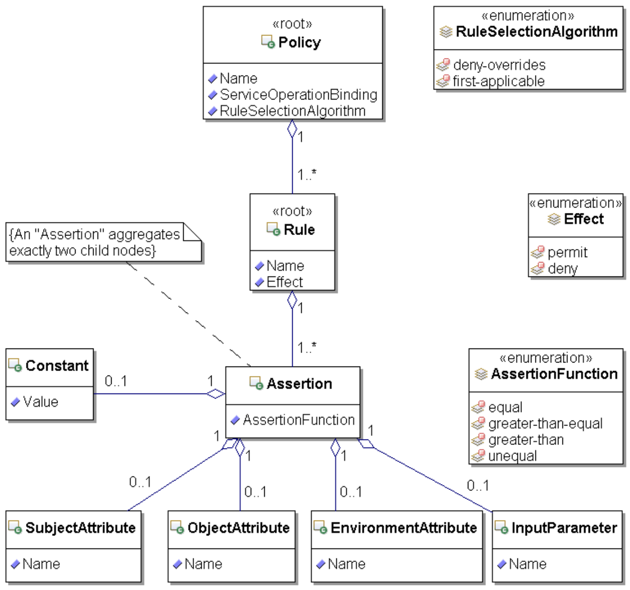

Figure 2. Abstract Syntax of WSACML

In figure 2 the abstract syntax of WSACML is depicted as an UML class diagram. The top-level element *Policy* is identified by its attribute *Name* and bound to an object by its attribute *ServiceOperationBinding* referencing a Web service operation as listed in the service registry. Furthermore, it defines a *RuleSelectionAlgorithm* describing what is done if more than one *Rule* is applicable. *Deny-overrides* implies that explicit prohibitions (deny-rules) have precedence over permissions (permit-rules). *First-applicable* is another algorithm which determines the first matching *Rule* of a *Policy* to define the result of an access control decision.

A *Policy* is an ordered list of *Rules* (cf. to *RuleSeclectionAlgorithm*) which themselves have an identifying *Name* and an *Effect*. The *Effect* describes what happens if a *Rule* is applicable. Most common *Effects* are

Permit and *Deny*. A *Rule* aggregates several *Assertions* and is applicable if all of its *Assertions* evaluate to true. An *Assertion* is a predicate that requires two arguments. It combines two variables or a variable and a constant value using an *AssertionFunction* like *equal*, *unequal*, *greater-than-equal* etc. To reflect the different entities that participate in an access context, variables are divided into four categories *SubjectAttributes*, *ObjectAttributes*, *InputParameters* and *EnvironmentAttributes*. WSACML allows that two variables are treated as arguments of *Assertions*. This enables for instance comparing an *InputParameter* of a service invocation with *SubjectAttributes* of the calling subject. A *Rule*'s *Effect* is only enforced if the *Rule* is applicable, which means that all of its *Assertions* are evaluated to true. If a *Rule* is not applicable, it implies that the Web service's usage context does not match and that the subject is not authorized.

Since a WSACML *Policy* is simply a pointer to concrete *Rules*, the administrative advantage of re-using *Rules* within *Policies* of other Web services is featured. If another Web service is deployed which is used in similar usage contexts, its *Policy* can be built from existing *Rules*. Additionally, a *Policy* of an existing Web service that is required to be accessed in another usage context is just extended by an additional *Rule* which covers the new *Assertions*. If the access requirements of an existing usage context change, only the appropriate *Rule* needs to be updated to adapt the *Policies* of all Web service operations associated with that usage context. The *Policy* of a Web service composition needs to include all *Policies* of the Web services operations it does invoke by all means. This allows for a pre-checking as the first step in a Web service composition and may reduce the need for roll-back operations.

We use XML as the concrete syntax for WSACML, so the abstract syntax depicted in figure 2 has to be transformed to XML Schema (XSD). [15] describes a general way of mapping. The transformation maps UML classes to XSD complex types, UML attributes to XSD attributes and UML aggregations to XSD sequences. *Policy* and *Rule* classes were stereotyped with *<<root>>* to generate XSD top-level elements. Namespaces were retightened manually, as well as constraints like the restricted occurrence of variables and constants as child nodes of an assertion.

It is important to recognize that WSACML only gives the structure of the language but does not give the vocabulary in the sense of the concrete names of the attributes of subjects, objects, parameters and environment that are available to set up access control policies. The vocabulary depends on the given business and IT environment and needs to be created in advance. It contains the information about all relevant business objects and can be enhanced at anytime. In our case study in section 5 we present a snapshot of an exemplary vocabulary.

4. Deriving Platform-Specific Policy Meta models

After having defined a platform-independent policy model, we show in this section how a policy metamodel for an existing security infrastructure can be refactored exemplarily treating CA eTrust SiteMinder [16] being a state-of-the-art IdM suite. Subsequently we show how a model-to-model transformation is utilized to generate platform-specific policies from WSACML ones.

4.1 CA eTrust SiteMinder as a specific IdM Platform

SiteMinder is an enterprise-level policy-based Web access management platform. Web agents are hooked into distributed Web servers and application servers and act as policy enforcement points. They intercept access requests, forward them to the centralized policy decision point and enforce access decisions. Policies can be authored via a Web-based administration applet or using a given Java-based policy API. Policies are maintained in an LDAP-based policy store, which was the starting point for obtaining the subset of SiteMinder's policy model relevant for authorization. Figure 3 depicts the platform-specific abstract SiteMinder policy syntax. As there is neither an explicit definition nor a specification of SiteMinder's backend policy model, we derived the semantics of objects and attributes by reverse engineering using the policy API and the policy design guide.

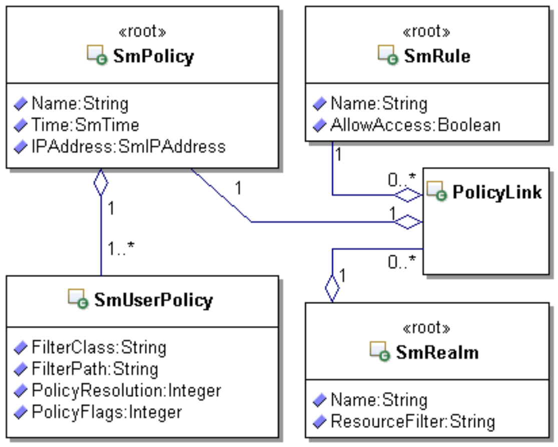

Figure 3. Abstract Syntax of SiteMinder Policy Model

Figure 3 shows the relevant SiteMinder's object classes that are used to formulate authorization statements. In the SiteMinder model authorization is bound to resources by the attribute *ResourceFilter* which is part of the object *SmRealm*. *SmRealm* corresponds to a subset of an URL. The *PolicyLink* links a realm to a policy, called *SmPolicy*. *SmPolicy* allows for time- and location-based restrictions and directly links towards user filters, called *SmUserPolicy*. Authorized users are determined via *SmUserPolicy* objects which allow the definition of LDAP-based user filters. *SmRule* is the third element of the *PolicyLink* and determines whether an authorization statement permits or denies access. Other attributes owned by the related object classes that are not shown here mainly contain configuration settings, e.g. for each *SmUserPolicy* a specific connection between a policy server and different user directories can be defined.

To be usable for policy transformation, the SiteMinder UML diagram has to be transformed to an XSD schema as well by applying the same transformation as we did for the PIM described in the previous section. We ensured that classes and attributes are directly related to their policy API counterparts. When proceeding to deployment a policy import tool only needs to parse the XML-based SiteMinder policy and settings files and generate the appropriate SiteMinder objects forming the platform-specific code (PSC).

4.2 Defining PIM to PSM Transformation

After defining the platform-independent and the platform-specific model, these need to be linked via explicit transformation rules. Development of transformation rules between both policy models primarily consists of three steps:

1. Provide abstract syntax trees for both input models and output models

Since XSLT is selected as the transformation language, source and target trees are provided as XML schema (XSD). The abstract syntax for the source tree is provided by WSACML XSD (generated from figure 2) and the abstract syntax for the target tree is provided by SiteMinder XSD (generated from figure 3).

2. Formulate conceptual transformation rules

In this step concepts of PIM are to be mapped towards PSM. As both models can substantially differ the actual linking should be assisted by an expert of the PSM environment as this model is usually substantially more complicated to understand. First the transformation rules which are to be formalized in the next step are to be formulated on a conceptual basis. Mapping can be initiated by identifying semantically equivalent sub trees in both models which can be started at the root node.

Table 1. Conceptual Transformation Rules

Source: WSACML	Transformation	Target: SiteMinder
Policy		SmRealm
Rule/ @Effect	Permit → true Deny → false	SmRule/ @AllowAccess
Assertion /Subject-Attribute /@Name= role	→ ‘cn =’+ @Value + ‘, ou = roles, dc = com’ →organizationalUnit →5 →0	smUserPolicy/ @FilterPath @FilterClass @PolicyRes. @PolicyFlags

It is useful to work with rule patterns e.g. direct mapping, mapping with type cast, or restructuring rules. Examples of the three kinds of rules are shown in table 1. The first entry is a direct mapping of WSACML *Policy* elements to SiteMinder’s *SmRealm*. If semantics and structure are equivalent but types and names differ, transformation involves type casting and renaming. An example for type casting is the second rule in table 1. It does a mapping of String-typed attribute *Effect* of a WSACML *Rule* to the attribute *AllowAccess* of Boolean type owned by SiteMinder *SmRule*.

The third example in table 1 is about a complex restructuring rule. SiteMinder does not explicitly support WSACML’s concept of *Assertions*. Required attribute statements are distributed over several LDAP objects combined to a SiteMinder policy. The example shows how to map an assertion concerning a concrete subject attribute from WSACML’s model to SiteMinder’s model. It takes the value of the subject attribute “role” and converts it to a LDAP search expression. This affects the four attributes of SiteMinder’s *smUserPolicy: FilterPath* accepts an LDAP search expression, which is concatenated from “cn=” in case of *SubjectAttribute/@Name=role* and a literal derived from *Constant/@Value* and the remainder “ou=roles, dc=com”. *FilterClass* defines the object class of the directory object specified by *FilterPath*, here an LDAP object of type “organizationUnit”. *PolicyResolution* describes the relation between the directory object and the subject. In the actual case “*5*” indicates that the subject’s distinguished name is a member of the directory object (organizationalUnit). *PolicyFlags’s* value of “*0”* indicates that neither exclusion nor recursion due to nested groups is required by the policy.

3. Formalize rules by mapping them to concrete transformation technology

SiteMinder’s view on policy and rule elements is different from the view presented by WSACML, i.e. SiteMinder policies are user centric whereas WSACML policies are resource-centric. So a WSACML *Policy* is mapped to *SmRealm* and a *Rule* is mapped to both *SmRule* and *SmPolicy* which are linked together by *PolicyLink*.

In figure 4 we show an exemplary excerpt of the formalization of the conceptual transformation rules using XSLT. It shows how to map the WSACML *Rules* to SiteMinder’s policy model. The XSLT-based transformation is applied when the XPath expression assigned to the attribute “match” is matched by a node or node set of the source tree. The transformation involves type casting from String-valued WSACML Rule/@Effect to Boolean-valued SiteMinder SmRule/@AllowAccess. The instruction <xsl:apply-templates> causes the XSLT processor to search for applicable transformation rules at this point.

```
<xsl:template match=„Rule">
  <PolicyLink>
    <SmRule>
     <xsl:attribute name=„Name">
       <xsl:value-of select='@Name'/>
     </xsl:attribute>
     <xsl:attribute name=„AllowAccess">
       <xsl:value-of select="@Effect = 'permit'"/>
     </xsl:attribute>
    </SmRule>
    <SmPolicy>
     <xsl:attribute name=„Name">
       <xsl:value-of select='@Name'/>
     </xsl:attribute>
     <xsl:apply-templates/>
    </SmPolicy>
  </PolicyLink>
 </xsl:template>
```

Figure 4. Exemplary XSLT Transformation Rule

5. Case Study

An integration project at our university applies a WSOA-based approach to encapsulate functionality of existing heterogeneous software systems at reusable Web services. These services are then to be composed following specified business processes. These processes are designed with UML use case and activity diagrams and are mapped to component and service diagrams [17].

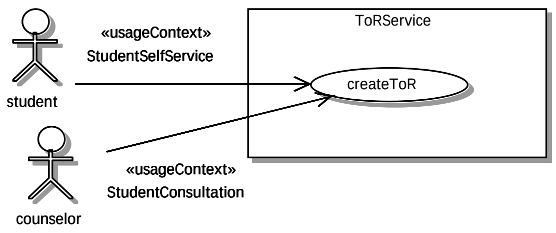

Figure 5. Associating Objects with Usage Contexts

UML use case diagrams which abstractly describe the functional requirements build the starting point of a Web service development project. Access control aspects as non-functional requirements are integrated into such analysis models by enhancing the model semantics using stereotypes. In figure 5, we show our enhancement adding so-called “usage contexts”. The given example addresses a service providing a transcript of records (ToR). It is offered by a Web service called ToRService at the operation “createToR”. This functionality can be accessed by different roles in different usage contexts. A student may get access for self-service and is then allowed to get his ToR only. In contrast a counselor is allowed to access the ToR of all students.

Before being able to specify these usage contexts as WSACML *Rules* that build the WSACML *Policy* for the “createToR” Web service operation, the concrete vocabulary has to be defined. According to our metamodel depicted in section 3, four areas have to be covered: attributes of subjects are to be derived from the user repository, object attributes are an aggregation of existing metadata, input parameters are to be derived from the existing Web service interfaces and environment attributes are to be specified according to the project’s needs. This vocabulary is not fixed but can be flexibly extended by new attributes as this does not affect existing WSACML *Rules*. The attributes provide the vocabulary for the access control language WSACML. Figure 6 exemplarily depicts a reduced view of our WSACML vocabulary.

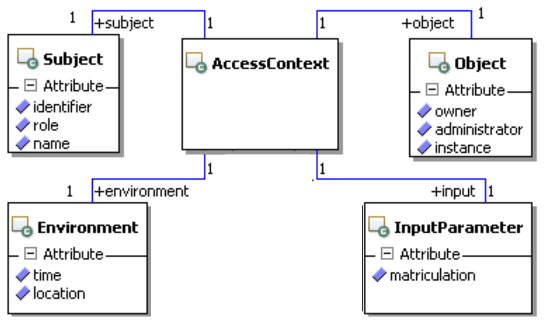

Figure 6. Exemplary WSACML Vocabulary

From the enhanced use case diagram, a WSACML *Policy* is generated for each use case corresponding to a Web service operation. While the aggregation of all usage contexts defines the overall WSACML *Policy* of a Web service operation, each usage context maps to an individual WSACML *Rule*. This *Rule* puts together all necessary *Assertions* that must hold for the *Rule* to be applicable. It sets up on the vocabulary of attributes as defined before.

```
<Policy Name=„createToR_policy"
   ServiceOperationBinding=„ToRService/createToR“
   RuleSelectionAlgorithm=„first-applicable “>
  <RuleRef>StudentSelfService</RuleRef>
  <RuleRef>StudentConsultation</RuleRef>
</Policy>
```

```
<Rule Name=„StudentSelfService“ Effect=„permit“>
   <Assertion AssertionFunction="equal">
     <SubjectAttribute Name="role" />
     <Constant Value="student" />
   </Assertion>
   <Assertion AssertionFunction="equal">
     <SubjectAttribute Name=„identifier“ />
     <InputParameter Name="matriculation" />
   </Assertion>
</Rule>
```

```
<Rule Name=„StudentConsultation“ Effect=„permit“>
   <Assertion AssertionFunction="equal">
     <SubjectAttribute Name="role" />
     <Constant Value="counselor" />
   </Assertion>
</Rule>
```

Figure 7. Exemplary WSACML Policy / Rules

In figure 7 an exemplary WSACML *Policy* and corresponding *Rules* are depicted. The *Policy* has the name “createToR_policy” and is bound to the Web service operation “ToRService/createToR” as specified during service design. From the two usage contexts *StudentSelfService* and *StudentConsultation* as depicted in the enhanced UML use case diagram, two WSACML *Rules* with the corresponding names are derived. Using “first-applicable” as *RuleSelectionAlgorithm* implies the first matching rule defines the *Effect* of the *Policy*. *StudentSelfService* contains two *Assertions* that need to be evaluated to true so that the *Rule* can be applied. First, the attribute “role” of the calling subject needs to be “student”, second the subject’s attribute “identifier” has to be identical to the input parameter with the name “matriculation” of the service invocation. To match the usage context of a student consultation process, the calling subject needs to have the attribute “role” with a value of “counselor”. In case of an access control request, the rule *StudentSelfService* is checked first. If the subject attribute role does not match, this *Rule’s* assertions do not evaluate to true so the *Rule* is not applicable. The next *Rule* is evaluated until one is applicable. Then this *Rule’s Effect* defines the overall access decision. Keep in mind that this WSACML *Policy* consisting of two *Rules* is completely independent from a concrete IdM product which we will consider in the next section.

```
<SmRealm Name=‘createToR_policy’
  ResourceFilter=‘ToRService/createToR’>
 <PolicyLink>
   <SmRule Name=‘StudentConsultation’
    AllowAccess=‘true’ />
   <SmPolicy Name=‘StudentConsultation’>
     <SmUserPolicy
      FilterPath=‘cn=counselor, ou=roles, dc=com’
      FilterClass=‘organizationalUnit’
      PolicyResoultion=‘5’
      PolicyFlags=‘0’ />
   </SmPolicy>
 </PolicyLink>
</SmRealm>
```

Figure 8. Exemplary SiteMinder Policy (PSM)

At present we use SiteMinder as the policy decision point in our IdM architecture. By using WSACML for policy specification and transformations to SiteMinder, we are not limited to this product. When deploying a Web service, its WSACML *Rules* according to its *Policy* are collected and the transformation is applied. We use the XSLT processor of Oracle JDeveloper. Figure 8 shows a SiteMinder policy that has been transformed from the WSACML policy in figure 7 using the transformation depicted in figure 4. With the PSM policies then conforming in a structural and syntactical way to SiteMinder’s policy model we developed a Java-based policy import tool which uses SiteMinder’s policy API for the final import to its policy store.

6. Conclusion and Further Work

In this paper we presented Web Services Access Control Markup Language (WSACML) which allows modeling of access control policies for Web services at design time. It is platform-independent related to the IdM product that is applied at runtime. We designed both an access control model and a concrete language and showed how to refactor an appropriate model of an existing IdM product exemplarily considering CA eTrust SiteMinder. We illustrated how to apply our approach in a case study in a WSOA-based integration project.

Our next steps are further work on how access control policies for Web service compositions relate to the policies of the atomic ones. Additionally, we will focus on policy lifecycle management.

References

[1] Christian Emig, Jochen Weisser, Sebastian Abeck: Development of SOA-Based Software Systems – an Evolutionary Programming Approach, IEEE Conference on Internet and Web Applications and Services ICIW’06, Guadeloupe, February 2006.

[2] Edsger Dijkstra. On the role of scientific thought. EWD 447, 30th August 1974, Neuen, The Netherlands. Appears in: Edsger W. Dijkstra, Selected Writings on Computing: A Personal Perspective, Springer-Verlag, 1982. ISBN 0–387–90652–5, pp. 60–66.

[3] Nick Nikols: Directory Products evolve towards Identity Services, Burton Group Identity and Privacy Strategies, November 2004.

[4] Christian Emig, Frank Brandt, Sebastian Kreuzer, Sebastian Abeck: Identity as a Service - Towards a Service-Oriented Identity Management Architecture, 13th EUNICE Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services (EUNICE 2007), Twente/Netherlands, July 2007.

[5] Christian Emig, Frank Brandt, Sebastian Abeck, Jürgen Biermann, Heiko Klarl: An Access Control Metamodel for Web Service-Oriented Architecture, IEEE Conference on Software Engineering Advances ICSEA’07, Cap Esterel/France, August 2007.

[6] Joaquin Miller, Jishnu Mukerji (Editors): MDA Guide 1.0.1 http://www.omg.org/docs/omg/03-06-01.pdf

[7] B.W. Lampson: Dynamic protection structures, AFIPS conference proceedings, FJCC 1969, pp. 27-38.

[8] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman Role-Based: Access Control Models, IEEE Computer 29(2): pp. 38-47, IEEE Press, 1996.

[9] Eric Yuan, Jin Tong: Attribute Based Access Control (ABAC) for Web Services, IEEE International Conference on Web Services (ICWS 2005), Orlando Florida, July 2005.

[10] OASIS eXtensible Access Control Markup Language (XACML) Version 2.0, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[11] Muhammad Alam, Ruth Breu, Michael Hafner: Modeling permissions in a (U/X)ML world, IEEE First International Conference on Availability, Reliability and Security (ARES’06), 2006.

[12] Object Management Group: Object Constraint Language, Version 2.0, Mai 2006.

[13] David Basin, Juergen Doser, Thorsten Lodderstedt: Model-Driven Security: From UML Models to Access Control Infrastructures, ACM Transactions on Software Engineering and Methodology, Vol. 15, No. 1, pp. 39–91, January 2006.

[14] Object Management Group: Meta Object Facility (MOF) Specification, Juli 2005.

[15] Design XML schemas using UML, 01 Feb 2003. http://www.ibm.com/developerworks/library/x-umlschem/

[16] CA eTrust SiteMinder, Product Homepage. http://www.ca.com/us/internet-access-control.aspx

[17] Christian Emig, Karsten Krutz, Stefan Link, Christof Momm, Sebastian Abeck: Model-Driven Development of SOA Services, C&M Research Report, April 2007.