An Integrated Approach for Identity and Access Management in a SOA Context

Waldemar Hummer[1], Patrick Gaubatz[2], Mark Strembeck[3], Uwe Zdun[2], and Schahram Dustdar[1]

[1]Distributed Systems Group
Information Systems Institute
Vienna University of Technology
{lastname}@infosys.tuwien.ac.at

[2]Software Architecture Group
Faculty of Computer Science
University of Vienna
{firstname.lastname}@univie.ac.at

[3]Information Systems Institute
Vienna University of
Economics and Business
mark.strembeck@wu.ac.at

ABSTRACT

In this paper, we present an approach for identity and access management (IAM) in the context of (cross-organizational) service-oriented architectures (SOA). In particular, we defined a domain-specific language (DSL) for role-based access control (RBAC) that allows for the definition of IAM policies for SOAs. For the application in a SOA context, our DSL environment automatically produces WS-BPEL (Business Process Execution Language for Web services) specifications from the RBAC models defined in our DSL. We use the WS-BPEL extension mechanism to annotate parts of the process definition with directives concerning the IAM policies. At deployment time, the WS-BPEL process is instrumented with special activities which are executed at runtime to ensure its compliance to the IAM policies. The algorithm that produces extended WS-BPEL specifications from DSL models is described in detail. Thereby, policies defined via our DSL are automatically mapped to the implementation level of a SOA-based business process. This way, the DSL decouples domain experts' concerns from the technical details of IAM policy specification and enforcement. Our approach thus enables (non-technical) domain experts, such as physicians or hospital clerks, to participate in defining and maintaining IAM policies in a SOA context. Based on a prototype implementation we also discuss several performance aspects of our approach.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access Control*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server, Distributed applications*; D.2.11 [**Software**]: Software Architectures—*Domain-specific architectures, Languages, Service-oriented architecture*

General Terms

Design, Languages, Management, Security

Keywords

Identity and Access Management, SAML, SOAP, WS-BPEL, WS-Security

SACMAT'11, June 15–17, 2011, Innsbruck, Austria.

1. INTRODUCTION

In recent years, Service-Oriented Architectures (SOA) [24] have emerged as a suitable means to develop loosely coupled distributed systems. Today, Web services are a commonly used technology that build the foundation of SOAs and both intra- and cross-organizational business processes. Electronic business collaborations require enforcement of high-level security constraints such as ensuring the identity and competencies of end users, restricted access to resources, or protection of private data. In our previous work, we identified the need for modeling support of identity and access control models from the experiences gained in the area of role engineering (see, e.g., [34–37]). However, to enforce the corresponding access control policies in a software system, the resulting models must also be mapped to the implementation level.

Different aspects of identity and access management (IAM) in distributed environments and SOAs have been studied previously. In fact, our work builds on a number of existing approaches and standards. An important point with regards to electronic business processes spanning multiple services and cross-organizational units is the concept of Single Sign-On (SSO, e.g., [17, 25]), which simplifies user authentication for the individual services by establishing trust relationships across security domains. SSO allows the business process to obtain a signed authentication token for a security domain d, which is also accepted by other security domains that trust domain d. The Security Assertion Markup Language (SAML) [20] provides a standard way of expressing signed assertions about the identity and attributes of a system participant. The Web Services Security (WS-Security) [21] SAML Token Profile defines how SAML assertions can be transported securely in Web service invocations, i.e., by including a security token element in the header of the SOAP (Simple Object Access Protocol) invocation message.

Cross-organizational IAM involves stakeholders with different background and expertise. The technical IAM model which expresses well-defined semantics and supports detailed security audits may be suited for software architects and developers, but for non-technical domain experts an abstracted view is desirable. In the context of model-driven development (MDD) [28,29,33], a systematic approach for DSL (*domain-specific language*) development has emerged in recent years (see, e.g., [15, 32, 38, 42]). A DSL is a tailor-made (computer) language for a specific problem domain. In general, DSLs provide relevant domain abstractions as first class language elements and can be designed and used on different abstraction layers, ranging from DSLs for technical tasks to DSLs for tasks on the business-level. Thus, DSLs can also be defined for non-technical stakeholders, such as business analysts or biologists, for example. In general, a DSL makes domain-knowledge explicit.

That is, the DSL is built so that domain experts can understand and modify DSL code to phrase domain-specific statements that are understood by an information system. To ensure compliance between models and software platforms, the models defined in a DSL are mapped to source code artifacts of the software platform via automated model-transformations (see, e.g., [14, 30, 41]).

This paper presents an approach to define and enforce IAM policies in cross-organizational SOA business processes. The approach is based on the Web Services Business Process Execution Language (WS-BPEL) [22], which has in the previous years emerged as the de-facto standard for defining Web service compositions and business processes. WS-BPEL is an XML-based special-purpose language whose features range from invocation of external Web services, message correlation and asynchronous invocations to control flow structures (e.g., loops, branches, parallel flows), XML data transformation and modification of SOAP message headers. Our implementation builds on well-established standards including SAML and WS-Security, and supports the concept of single-sign-on (SSO) to authorize and secure the individual steps in the business process. The use of a domain-specific language (DSL) for Role Based Access Control (RBAC) [4, 5, 27] allows us to abstract from technological details and to involve domain experts in the security modeling process. SOA experts and software developers utilize the identity and access models to define security constraints while designing electronic business processes in WS-BPEL. At deployment time, the WS-BPEL process is instrumented with special activities to ensure its compliance to the IAM policies at runtime.

The remainder of this paper is structured as follows. In Section 2, we introduce an illustrative scenario for IAM in a distributed SOA context. We then present in Section 3 our approach for integrated modeling and enforcement of identity and access control in SOA business processes, and discuss the mapping from the modeling to the implementation level. Details on the implementation are given in Section 4, and in Section 5 we evaluate different aspects of our solution. Section 6 contains a discussion of related work, and Section 7 concludes the paper with an outlook for future work.

2. SCENARIO: IAM IN A SOA BUSINESS PROCESS CONTEXT

We illustrate the concepts of this paper based on a motivating scenario taken from the e-health domain. Our example scenario models the workflow of an orthopedic hospital which treats fractures and other serious injuries. The hospital is supported by an IT infrastructure organized in a SOA, implemented using Web services. The SOA provides services for patient data, connects the departments of the hospital and facilitates the routine processes. The hospital exchanges data with other partner hospitals. As patient data constitute sensitive information, security must be ensured and a tailored domain-specific RBAC model needs to be enforced.

A core procedure in the hospital is the patient examination. The corresponding technical business process is depicted in Business Process Modeling Notation (BPMN) in Figure 1. We assume that the process is implemented using WS-BPEL and that each BPMN service task (depicted as gray rounded rectangles) denotes the invocation of a Web service. The arrows between the tasks indicate the control flow of the process. The BPMN groups in the figure are annotated with *Role* and *Context* labels, the purpose of which will be detailed later in this section. Note that all tasks are backed by Web services, however, part of the tasks are not purely technical but involve some sort of human labor or interaction. For instance, the activation of the task *Obtain X-Ray Image* triggers an invocation

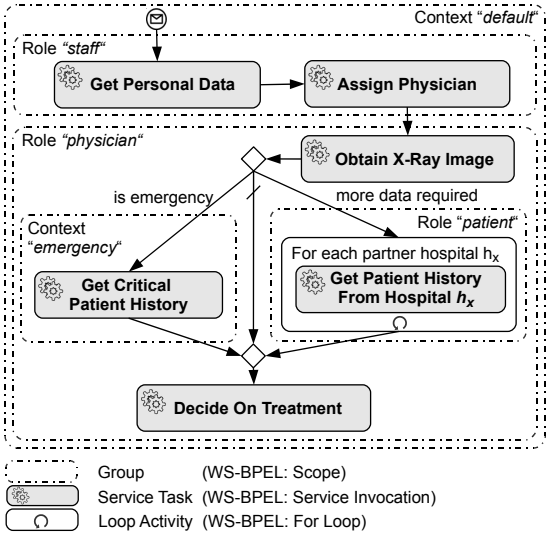

Figure 1: Hospital Patient Examination Scenario in BPMN

to the Web service `http://h1.com/xray`, but the task itself is performed by the hospital staff (and the patient).

The first step in the examination process is to retrieve the personal data of the patient. To demonstrate the cross-organizational character of this scenario, suppose that the patient has never been treated in our example hospital (H1) before, but has already received medical treatment in a partner hospital (H2). Consequently, H1 obtains the patient's personal data from H2 via a Web service residing under the URL `http://h2.com/patients`. Secondly, the patient is assigned to an available physician, which is performed using an `examination` service. These first two tasks need to be performed by a general staff member (role "staff"). In the process definition in Figure 1, this requirement is expressed as a BPMN *group* (rounded rectangle with dashed border) with a corresponding label. In the implementation of the process, this group is mapped to a BPEL *scope* with an extensibility attribute `role`. Similar to a scope in a regular programming language, a WS-BPEL scope embraces a set of instructions and defines boundaries for the lifetime of variables and event handlers defined in this scope. Analogously, the role attribute is valid within the boundaries of its owner scope.

After the patient has been assigned, the responsible physician requests an x-ray image using the Web service of the x-ray department (`http://h1.com/xray`). This activity runs under a new group (or scope), which requires the role "physician". The physician then analyzes the received x-ray image and decides whether additional data are required. For instance, the patient may have had a similar fraction or injury in the past, in which case special treatment is required. Hence, the business process requests historical data from partner hospitals, which also participate in the SOA. Due to privacy issues, the historical data are only disclosed to the patient herself, and the *Get Patient History* service task executes under the role "patient". Note that this role change and the identity management is enforced by the platform, which will be discussed in Section 3. Another situation that requires additional data is the case of an emergency. If the emergency demands for immediate surgery, it is important to determine historical data about any critical conditions or diseases that might interfere with the surgery. This critical information is stored in a secured repository which can be accessed via the Web service `http://h1.com/emergency`. Access to the critical historical data requires the context "emergency", which

is also indicated via an enclosing scope in Figure 1. Finally, after acquiring the necessary data, the process switches back to the context "default" and the role "physician". The invocation of the operation `decideOnTreatment` constitutes the end of the examination and triggers the subsequent treatment activities.

The following list summarizes the stakeholders and their key requirements concerning the SOA-based IT system of the hospital.

- The IT system facilitates the hospital *staff* in their daily work and employs a clear role concept for separation of concerns.
- Besides receiving an efficient treatment, the main interest of the *patient* is that all personal data remain confidential and protected from abuse.
- The *security experts* of the hospital need not necessarily be technical experts and hence require an intuitive interface to model identities, roles and security restrictions in the system.
- The IT *architects and developers* who implement Web services and business processes desire an integrated solution, in which identity and access control can be easily plugged in based on the models defined by the hospital's management.

In the course of this paper, we focus on two aspects concerned with mapping security constraints from a higher-level model to the implementation level: 1) enabling domain experts to map the identity and access model from its abstract representation to a DSL, and mapping of DSL expressions to the implementation level, 2) enabling architects and developers to easily author SOA business processes in WS-BPEL which enforce the security constraints.

3. INTEGRATED APPROACH FOR IAM IN A SOA CONTEXT

This section presents our integrated approach for identity and access management and enforcement in a SOA context. The core assets in a SOA are the services, and the participants that perform operations on these services are either humans or other services. It has been shown that SOA models can be mapped to (extended) RBAC models (e.g., [1]). We build on these findings and provide a declarative DSL for RBAC, integrated with an end-to-end solution for simplified development of secured SOA business processes. The tight integration of the DSL allows to trace identity and access control specifications from the modeling level down to the implementation code, enabling the detailed audit of security compliance.

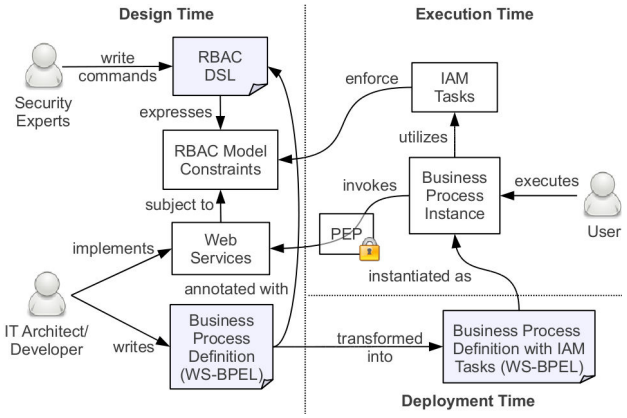

Figure 2: Approach Overview

Figure 2 depicts a high-level overview of our approach, including the involved stakeholders and system artifacts and the relationships between them. At design time, the security experts write RBAC DSL commands to define the RBAC model constraints. The IT specialists implement Web services and define WS-BPEL processes on top of the services. The WS-BPEL definition is annotated with elements from the RBAC DSL, in order to define which parts in the process require which access privileges. At deployment time, the WS-BPEL file is automatically enriched with IAM tasks that conform to the security annotations. The business process is instantiated and executed by human individuals (for example patients and staff members), and the IAM tasks have the process conform to the constraints defined in the RBAC model. A PEP component intercepts all service invocations and blocks unauthorized access.

In the following, we firstly discuss the core language model of the RBAC DSL and show its mapping to the textual representation and further down to the implementation level. Secondly, we present our approach for automatic enforcement of the access control constraints using the extensibility mechanism in WS-BPEL processes.

3.1 DSL-Based RBAC Modeling for SOA

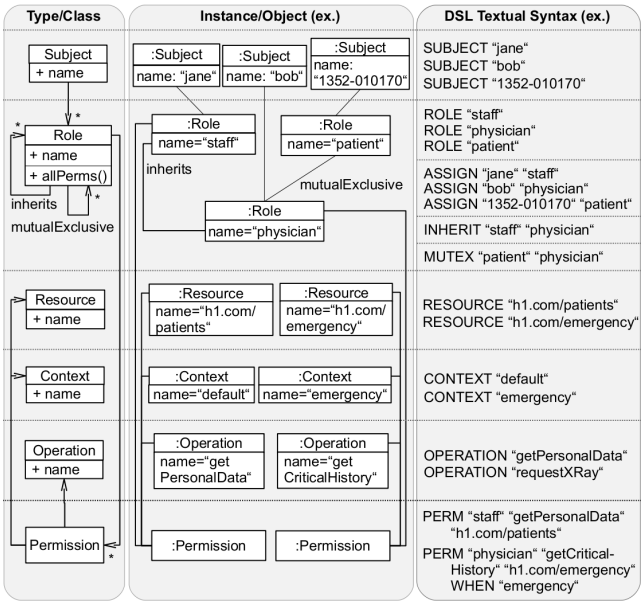

Figure 3: RBAC Model and DSL Language Elements

Figure 3 depicts an example that shows the different abstraction layers of our RBAC DSL. In particular it depicts a (simplified) class diagram of the DSL language elements, an excerpt of the object diagram for the hospital scenario from Figure 1, and a textual representation of the example specified with our RBAC DSL. `Subjects` are identified by a `name` attribute: hospital staff receive a unique name, and for the patients' name we use their social security number, which serves as a unique identifier. Subjects are associated with an arbitrary number of `Roles`, which are themselves associated with `Permissions` to execute certain `Operations`. Roles may inherit from other role instances (association `inherits`), and two roles can be defined as being mutually exclusive (association `mutualExclusive`). We use a context-specific extension of the traditional RBAC model, which has been proposed previously in a similar form (e.g., [6, 8, 26]). The `Context` element allows for a more fine-grained definition of permissions and maps directly to the context requirements in the scenario process definition (see Figure 1). In our approach, we directly associate Web service instances with `Resources`, service invocations with RBAC `Operations`, and `Contexts` with scopes in a Web services business process. A scope in WS-BPEL builds a group of related tasks and limits the lifetime and validness of its enclosed variables, partner links, correlation sets and event handlers. The RBAC permis-

sions are expressed with regard to a certain context in which they are applicable. When a WS-BPEL scope is associated with a certain context (e.g., *emergency*), then all activities (i.e., operations) contained in that scope must execute under this context, and, consequently, the subject executing the process must be allowed to invoke the service operations under this context (see Section 3.2). For instance, when the physician named *bob* is about to retrieve the critical patient history in our scenario, then *bob* needs to have the role *physician*, which allows him to execute the Web service operation `getCriticalHistory` in the context *emergency* (see Figure 3). The *default* context always exists and is automatically assumed if no context is explicitly provided.

DSL Command	Effect (OCL)
`SUBJECT "jane"`	Subject.allInstances()->select(s | s.name='jane')->size() = 1
`ASSIGN "jane" "staff"`	Subject.allInstances()->select(s | s.name='jane').role->select(r | r.name='staff')->size() = 1
`INHERIT "staff" "physician"`	Role.allInstances()->select(r1 | r1.name='staff').allPerms()->forAll(p1 | Role.allInstances()->select(r2 | r2.name='physician').allPerms() ->exists(p2 | p1=p2))
`MUTEX "patient" "physician"`	Subject.allInstances()->forAll(s | not (s.role->exists(r | r.name='physician') and s.role->exists(r | r.name='patient')))

Table 1: Excerpt of RBAC DSL Semantics in OCL

An excerpt of the RBAC DSL constructs and their effect expressed as an OCL (Object Constraint Language) expression is printed in Table 1. The first exemplary command, `SUBJECT "jane"` has the effect that, upon execution, exactly one instance of the class *Subject* with `name` attribute *"jane"* exists. The effect of the second instruction is that the *Subject* named *jane* has an associated *Role* object with name *"staff"*. The `INHERIT` command takes two parameters, a junior-role and a senior-role name, and causes the senior-role to inherit all permissions of the junior-role. The operation *Role.allPerms()* returns all associated permissions of a Role instance and its ancestor roles. Finally, the statically mutual exclusive roles *"patient"* and *"physician"* are defined via the DSL command `MUTEX`, which specifies that no *Subject* instance must ever be assigned both of these roles simultaneously . We currently do not use the alternative form of dynamic mutually exclusive constraints which disallow combinations of certain roles to be activated by one user in the same session or process instantiation, but this is planned for future work. The four OCL constraints illustrate the mapping from the abstract RBAC domain model to the level of an intermediate language (DSL), which is easy to use and comprehend for domain experts, and abstracts from the underlying complexity. The remaining OCL constraints for our example have been left out for brevity.

3.1.1 Collaborative Identity and Access Modeling for Single-Sign On

The goal of the patient examination scenario is that hospitals are able to *collaboratively* model the identity and access control information. To avoid a single point of failure and because each hospital reserves the right to define their own (internal) access control policies, the RBAC information is not stored centrally, but each hospital maintains their own model. However, the ability to retrieve the model data from partner hospitals is vital in order to support SSO and cross-organizational access to resources. For instance, the loop in the business process in Figure 1 retrieves the patient history from partner hospitals using a secured Web service operation `getPatientHistory`, which is provided by all hospitals. The idea is to store data in a decentralized manner, i.e., when a patient is registered or examined in hospital X, then X creates a patient record that is stored locally, but can be accessed by the partner hospitals. The invocation of the `getPatientHistory` operation is secured with a SAML header asserting the identity of the patient. Consider the patient is identified under a subject name "1352-010170" (cf. Figure 3). This requires that the RBAC models of the partner hospitals also contain a subject with this identifier, and that this subject is associated with the role "patient".

To achieve an integrated view on a distributed RBAC model, different strategies have been proposed. The special-purpose language PCL (Policy Combining Language) defined in [10] allows combining of access control policies expressed in XACML. In another work, integration of policies from different organizations is performed based on the similarity of XACML rules [12]. Since the RBAC DSL essentially provides a subset of the functionality of XACML, we are able to utilize these existing solutions for policy integration and collaborative modeling of access control constraints across the different hospitals in the scenario.

3.2 Security Enforcement in WS-BPEL Processes using Annotations

Section 3.1 discussed how the RBAC model is constructed by means of the RBAC DSL, and how the access constraints relate to services, operations, and scopes in SOA business processes. To enforce these constraints at runtime, the business process needs to follow a special procedure. For instance, invoking the `getPersonalData` operation of Hospital 1 requires the process to execute under the role "staff". That is, this service operation requires the presence of a corresponding SAML WS-Security token in the SOAP header of the request. The token contains a SAML assertion that confirms the identity of the subject executing the process operation, as well as the attribute claims for that subject. Integrity of this token and the contained attributes is ensured by applying an XML signature [40] using the X.509 certificate issued for Hospital 1. The attribute claims contain the information under which *role* ("staff") and in which *context* ("default") the subject executes the operation. To obtain the signed SAML assertion, the process needs to invoke the operation `requestSAMLAssertion` of the SAML Identity Provider (IdP) service of Hospital 1. The patient data service relies on the IdP to identify and authenticate the subject (process user), hence the user credentials (e.g. subject name and password) are required for invocation of `requestSAMLAssertion`.

Since one execution of the patient examination process involves different subjects (a staff member, a physician, a patient), the user credentials cannot be hard-coded into the process definition, but are requested from a separate, decoupled *Credentials Provider* (CrP) service. This service offers a `getUserAuthentication` operation, which provides the actual user credentials to be used for a specific process scope. Upon invocation, this operation will cause a username/password input prompt to be displayed to the staff member sitting at the reception desk. After the user has been authenticated, the user credentials can also be stored in a local session configuration file on the reception desk computer. To avoid plaintext passwords from being transmitted over the network, the returned user credentials are encrypted using WS-Security [21]. During execution, the *Credentials Provider* service is always invoked when the process enters a scope that requires a change of subject.

The detailed procedure is illustrated in Figure 4, which shows the sub-part of the hospital scenario process that executes under

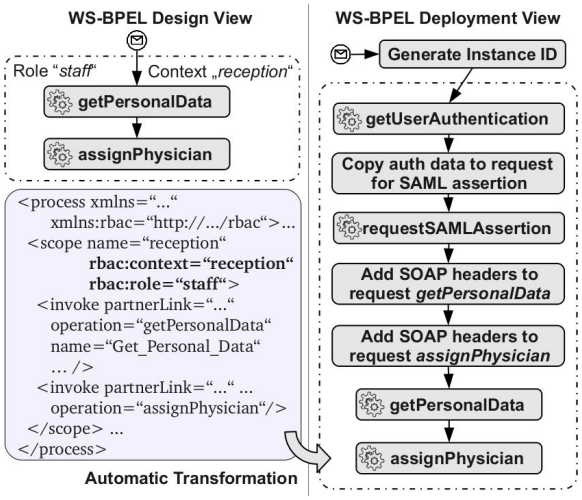

Figure 4: Transformation of WS-BPEL Process Definition

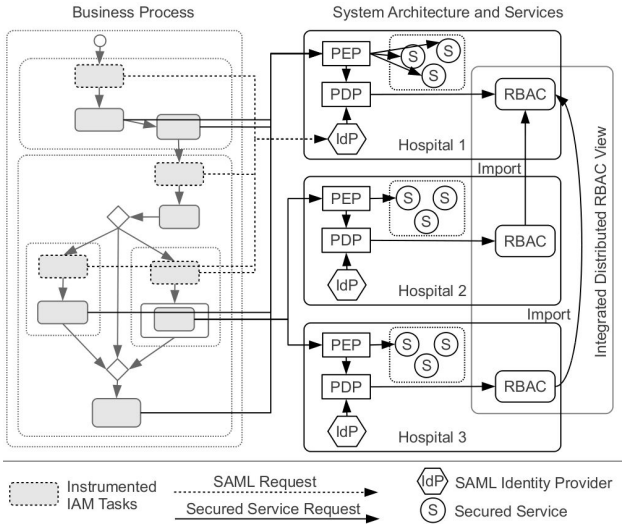

Figure 5: Example Process in System Architecture

the role "staff" and the context "reception". The left part of the figure shows the process definition at design time. Note the annotation attributes `rbac:context` and `rbac:role` which define the required context and role for the scope. At deployment time, the necessary additional process tasks are inserted into the WS-BPEL definition by means of an automatic transformation. At the start of the transformed process, an activity is inserted which generates a unique process instance identifier (ID). The instance ID is sent along as a SOAP header in all subsequent invocations of the WS-BPEL process. This ID helps the CrP service to correlate previous invocations of the process instance, and to keep track of the process state in order to provide the credentials from the correct subject. For instance, when the CrP's operation `getUserAuthentication` is first called with the generated ID, the user credentials are requested from the reception desk employee. The second invocation with the same instance ID will cause the CrP to request the user credentials from the assigned physician, and so on (cf. Figure 1). Note that the CrP service is application-specific and constitutes a tailor-made decoupled component that orchestrates the retrieval of user credentials of changing subjects. The injected process tasks that follow the CrP invocation retrieve the required SAML assertion from the IdP and copy a corresponding SAML header to all service requests of the scope. Details on the implementation of the automatic WS-BPEL transformation are provided in Section 4.

4. IMPLEMENTATION

In the following, we describe the prototype implementation of our approach for integrated SOA identity and access control. This section is divided into four parts: firstly, we outline the architecture of the system and the relationship between the individual services and components; secondly, the SAML-based SSO mechanism is described; the third part briefly discusses the implementation of the RBAC DSL; finally we present the algorithm for automatic transformation of WS-BPEL definitions containing security annotations.

4.1 System Architecture

Figure 5 sketches the high-level architecture and relationship between the example process and the system components. The patient examination example scenario is implemented using WS-BPEL [22] and deployed in a Glassfish[1] server with WS-BPEL module. The example scenario involves three hospitals, which host the protected services for patient management and examination. All service invocations are routed through a Policy Enforcement Point (PEP), which acts as a central security gateway, intercepts every incoming service request and either allows or disallows its invocation. Using the Java API for XML Web services (JAX-WS), the PEP has been implemented as a SOAP message handler (interface `SOAPHandler`). This handler can be plugged into the Web service's runtime engine in a straightforward manner. Once activated, the interceptor is able to inspect and modify inbound and outbound SOAP messages as well as to abort the service invocation.

Each hospital runs an instance of the SAML IdP service, which is used to issue the SAML assertions that are required in the WS-BPEL process. The responsibilities of the IdP are twofold: firstly, it checks whether the subject (i.e., the user currently executing the process) has provided valid credentials; secondly, the IdP assures the identity of a subject and its associated attributes (roles, contexts) by issuing an SAML assertion which is used as a SOAP header in subsequent service invocations by this subject (i.e., the process scope for which it is valid).

The actual decision whether an invocation should be prevented or not is typically delegated to another entity, the Policy Decision Point (PDP). When deciding over the access to a service resource the PDP has to make sure that the subject attempting to access the resource has the permission to do so. In our concrete implementation, the PDP uses the RBAC repository to determine whether the requesting subject is permitted to access the target resource (service) under the specified context and role. Thereby, the PDP can rely on the SAML tokens in the SOAP header of the request messages, which assert the identity of the subject as well as the context and role it operates under. The policy information in the RBAC repository is based on the DSL commands authored by domain experts. Each repository defines both local rules and integrates rules from RBAC repositories of trusted partner hospitals (see, e.g., [10, 12]). The combined information of all RBAC repositories creates an integrated view on the distributed RBAC model.

The advantage of our approach is that changing security requirements in the course of the process execution are handled automati-

[1] https://glassfish.dev.java.net/

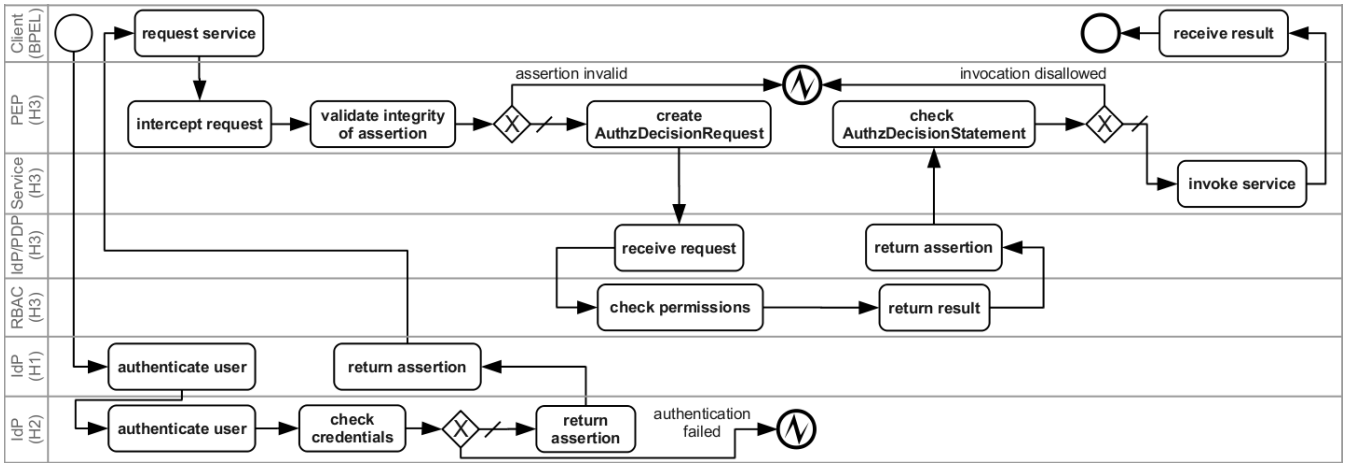

Figure 6: Identity and Access Control Enforcement Procedure

cally. Each time the process changes the scope and requires a new role or context, we utilize the *Instrumented IAM Tasks* which get injected into the WS-BPEL process automatically, as described in Section 3.2. The IAM tasks invoke the IdP and request a new security assertion token for the current subject, role, and context. The security token is then added to the header of all invocations in the same scope. This procedure is repeated for all sub-scopes which require a new role or context. More details concerning the automatic generation of the IAM tasks in WS-BPEL are given in Section 4.4.

4.2 SAML-based Single Sign-On

Figure 6 depicts an example of the Identity and Access Control enforcement procedure modeled via BPMN. To illustrate the SSO aspect of the scenario, we assume that a patient with subject name "1352-010170" (cf. Figure 3), who is registered in hospital 2 (H2), is examined in hospital 1 (H1) and requests its patient history from previous examinations in hospital 3 (H3). The procedure is initiated by the Web service client that demands the execution of a protected Web service. Note that we use the generic term *client*, whereas in our scenario this client is the WS-BPEL engine executing the patient examination process. Prior to issuing the actual service request, the client has to authenticate using the SAML IdP. The latter queries the user database (DB) to validate the credentials provided by the client. In our approach, the credentials (e.g., username-password combinations) are stored in a separate DB and are hence decoupled from the RBAC model. However, the username in the DB equals the subject name in the RBAC model. As the credentials of user "1352-010170" are not stored in the DB of H1, the IdP contacts the IdP of H2, which validates the credentials.

If the user credentials could not be validated, the process is terminated prematurely and a SOAP fault message is returned. In our example scenario, the business process receives the fault message and activates corresponding WS-BPEL fault handlers. Otherwise, if the credentials are valid, the IdP creates a signed assertion similar to the one shown in Listing 1 and passes it back to the client. From now on the business process attaches this assertion to every service request. The request to the protected service is then intercepted by the PEP of H3, which extracts the attached assertion, validates its integrity, and aborts the service invocation if the assertion is invalid (i.e., has been manipulated). Otherwise, it generates an Authorization Decision Request message which is passed to the PDP. The PDP then asks the RBAC repository if the client is allowed to access the requested service. The PDP's decision is expressed as an Authorization Decision Statement. Wrapped into an assertion similar to the one shown in Listing 2, the statement is then passed back to the PEP. Based on the assertion's enclosed information the PEP then either effects the actual service invocation or returns a fault.

The example SAML assertion in Listing 1 illustrates the information that is encapsulated in the header token when the scenario process invokes the `getPatientHistory` operation of the patient Web service of H3. The assertion states that the subject named `1352-010170`, which has been successfully authenticated by the IdP of the hospital denoted by the `Issuer` element (H2), is allowed to use the context `default` and the role `patient`. Note that the subject can be a human being, but it may as well be a service itself that attempts to invoke another service as part of a service composition. The included XML signature element ensures the integrity of the assertion, i.e., that the assertion content indeed originates from the issuing IdP (H2) and has not been modified in any way. When the PEP of H3 intercepts the service invocation with the SAML SOAP header, its first task is to verify the integrity of the assertion. The signature verification requires the public key of the IdP that signed the assertion; this key is directly requested from the corresponding IdP (under `http://h2.com/IdP`) using SAML Metadata [19].

```
1  <Issuer>http://h2.com/IdP</Issuer>
2  <ds:Signature>...</ds:Signature>
3  <Subject><NameID>1352-010170</NameID></Subject>
4  <Conditions NotBefore="2010-12-17T09:48:36.171Z"
5              NotOnOrAfter="2010-12-17T10:00:36.171Z"/>
6  <AttributeStatement>
7    <Attribute Name="context">
8      <AttributeValue>default</AttributeValue>
9    </Attribute>
10   <Attribute Name="role">
11     <AttributeValue>patient</AttributeValue>
12   </Attribute>
13 </AttributeStatement>
14 </Assertion>
```

Listing 1: SAML Assertion Example (1)

After the PEP of H3 has verified the message integrity (and thereby authenticated the subject), it needs to determine whether the subject is authorized to access the requested service operation. This is achieved by the PDP service of H3 that allows the PEP to post an SAML Authorization Decision Query. The PDP answers this query by returning an assertion containing at least one SAML Authorization Decision Statement. Using these Decision State-

ments the PDP is able to express the RBAC service's authorization decision using "plain" SAML. Listing 2 shows an example SAML assertion which informs the PEP that our patient is allowed to invoke the action (operation) `getPersonalData` of the resource (Web service) `http://h1.com/patient`. The `Issuer` name of the PDP is the same as for the IdP (`http://h3.com/IdP`).

```
<Assertion>
  <Issuer>http://h3.com/IdP</Issuer>
  <ds:Signature>...</ds:Signature>
  <Subject>
    <NameID>1352-010170</NameID>
  </Subject>
  <AuthzDecisionStatement Decision="Permit"
    Resource="http://h3.com/patient">
    <Action>getPersonalData</Action>
  </AuthzDecisionStatement>
</Assertion>
```

Listing 2: SAML Assertion Example (2)

4.3 RBAC DSL Implementation

In Section 3.1 we have described the mapping of the RBAC model elements to the textual DSL representation. The mapping of the RBAC DSL commands to executable code on the implementation level is illustrated in Figure 7. We follow a *hybrid* approach to DSL development, which combines *preprocessing* with *embedding* [42]. Embedding means that the DSL platform makes use of an existing *host language* and uses the interpreter and development tools of that language. Preprocessing denotes the process of converting the DSL commands into the machine-readable syntax of the host language using (light-weight) transformations. We evaluated the performance and syntactical flexibility of different (scripting) languages and have chosen Ruby, a language frequently used for DSL development. Ruby provides an interpreter named *JRuby*[2], which is implemented in pure Java and can be integrated using the Java Bean Scripting Framework[3] (BSF).

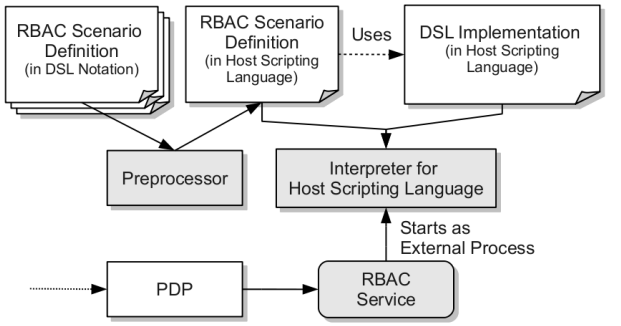

Figure 7: Execution of RBAC Requests

The Preprocessor component transforms the RBAC scenario definition from the DSL notation to the syntax of the host scripting language. An example of a light-weight transformation is to convert `ASSIGN "jane" "staff"` to `ASSIGN "jane","staff"` when using Ruby as the host scripting language. While the first command cannot be parsed by JRuby, the transformed command is well-formed for the interpreter, and is interpreted as a call to the function `ASSIGN` with the two string parameters, which looks up the `Subject` instance and assigns the given role. The remaining DSL constructs are interpreted analogously, and the implementation ensures that all constraints (e.g., mutually exclusive roles, see Table 1) are fulfilled. The preprocessor also serves a second purpose, namely checking whether the DSL code conforms to the allowed syntax or uses any disallowed commands; since the textual DSL is the user interface to the security-critical RBAC model, it is important to identify potentially harmful commands. Another point to consider is that the host language potentially provides features that are undesired for use in the DSL context, such as input/output operations. Hence, the interpreter for the host scripting language executes in a separate Java process, for which we apply restrictive permissions in the Java security policy settings, such as file system access (`java.io.FilePermission`) or network access (`java.net.NetPermission`).

[2]http://jruby.org/

[3]http://jakarta.apache.org/bsf/

4.4 Automatic Transformation of WS-BPEL Process Definition

At deployment time of the business process, the WS-BPEL definition is automatically transformed to ensure correct execution of identity and access control at runtime. Note that the WS-BPEL process is responsible for the choreography of CrP service, SAML IdP, as well as the core business logic services for patient examination.

Figure 8 depicts the relationships between the five scopes (s_1, s_2, s_3, s_4, s_5) of the scenario process. A hierarchical relationship indicates that the child scope (arrow target) is contained in the parent scope (arrow source). Attributes that are not defined in a child scope are inherited from the parent scope. For instance, scope s_3 inherits the context from its parent s_1. The existence of a sequential relationship between two scopes s_x (arrow source) and s_y (arrow target) means that the control flow is passed from s_x to s_y. More specifically, the last task of scope s_x has a control flow link to the first task of s_y in the process definition. This is the case for the scopes s_2 and s_3, where task *Assign Physician* has a control flow link to *Obtain X-Ray Image* (cf. Figure 1). The scope relationships graph is the basis for determining at which points in the process definition the IAM tasks need to be injected.

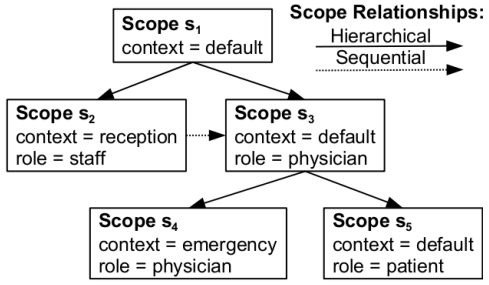

Figure 8: Scope Relationships in Scenario Process

The automatic WS-BPEL transformation is described in Algorithm 1. Variable names are printed in *italics*, and XML markup and XPath expressions are in `typewriter` font. The input is a WS-BPEL document *bpel* with security annotations. Firstly, four required documents need to be imported into the WS-BPEL process using `import` statements: the XML Schema Definitions (XSD) of SAML and WS-Security, and the WSDL (Web Service Description Language) files describing the CrP service and the IdP service.

Then the `partnerLink` declarations for these two services are added to *bpel*, and six `variable` declarations are created (input/output variables for operations `getUserAuthentication` and `requestSAMLAssertion`, a variable to store the assertion, and a variable for additional information such as the instance ID). Next, the algorithm loops over all `scope` elements s with a `role` or `context` attribute, and stores hierarchical and sequential relationships to the array variable *rel*. Although the implementa-

Algorithm 1 WS-BPEL Transformation Algorithm

Input: WS-BPEL document *bpel*
Output: transformed BPEL document
add `<import ../>` statements to *bpel*
add `<partnerLink ../>` definitions to *bpel*
add `<variable ../>` declarations to *bpel*
$rel \leftarrow$ new array // use variable rel for scope relationships
for all *bpel*`//scope` as s **do**
  $rel[s] \leftarrow \emptyset$
  if s`/@role` or s`/@context` **then**
    $rel[s] \leftarrow rel[s] \bigcup s$`/ancestor::scope[1]`
    $rel[s] \leftarrow rel[s] \bigcup s$`/preceding-sibling::scope[1]`
  end if
end for
for all indexes s in rel, r in $rel[s]$ **do**
  if scopes r and s have different security requirements **then**
    // add IAM tasks to s: `<invoke>` for IdP and CrP services, `<assign>` (SAML SOAP header) for each `<invoke>` in s
  end if
end for

tion also considers more complex cases, we assume that a sequential relationship exists if s has a preceding XML sibling element (i.e., an element on the same level as s in the element tree, sharing the parent element with s) named `scope`. The parent scope in a hierarchical relationship can be addressed using the XPath `ancestor::scope[1]`. After all scope relationships have been determined, we loop over all related scopes r and s (conforming to the notation in Figure 8, an arrow points from r to s) and check whether the security requirements are different (in terms of different security annotations). If so, the IAM tasks are injected to the beginning of scope s. The IAM tasks consist of two `invokes` for the invocations to CrP and IdP, as well as several `assign` tasks which add the security SOAP headers to the requests of the remaining service invocations in scope s. Note that the first scope in the process always receives the IAM tasks, although it has neither a parent nor a preceding sibling element.

5. DISCUSSION AND EVALUATION

We evaluated various aspects of the presented solution, and the main evaluation results are discussed in the following. The key aspects are the runtime performance of identity and access control enforcement, and the discussed WS-BPEL transformation algorithm.

To evaluate the scalability of the approach we have defined, deployed, and executed ten test processes with the Glassfish WS-BPEL engine. The processes contain an increasing size (1,2,...,10) of scopes that are annotated with `rbac:role` and `rbac:context` attributes. Each scope contains one `<invoke>` task, which invokes one of the Web service operations of the hospital scenario. The average response time of each service is roughly 200 milliseconds. The processes have been deployed in Glassfish, once with enforced security (i.e., annotated with security attributes, automatically transformed at deployment time), and once in an unsecured version. The deployed processes were executed 10 times and we have calculated the average value to minimize the influence of external effects. Figure 9 illustrates different measurements of the process execution time in milliseconds for both the secured and the unsecured version. The secured version incurs a large overhead, which is hardly surprising considering the fact that for each business logic service the process needs to invoke the CrP, IdP and RBAC services, and applies and checks the XML signatures. However, the measured results indicate that the current implementation leaves room for additional optimization.

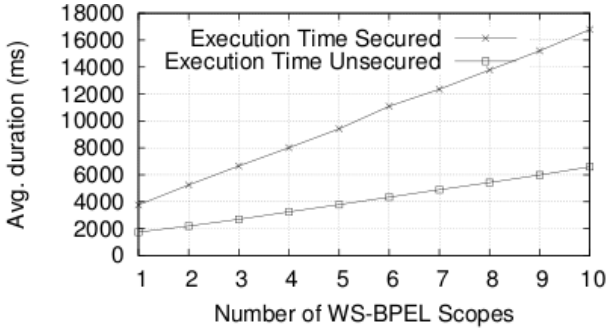

Figure 9: Process Execution Times – Secured vs Unsecured

In Section 4.1 we presented our concrete implementation of an SAML IdP and its duty to issue SAML Assertion tokens. These Assertion tokens are embedded in the header of every subsequent request to secured Web services. Each Assertion contains at least one Attribute Statement that includes the service's required role and context attribute. As the Assertion contains exactly one single context as well as one single role attribute, this means that the Assertion is only valid for one single subject, role, and context. Furthermore this also means, that whenever one of these three change, a new SAML Assertion has to be issued by the IdP. In terms of performance, this approach may not be the most effective, but it has the advantage, that it can be implemented using "plain" SAML. If performance is a critical issue, we propose the following solution: Instead of creating lots of specialized Assertions, the IdP should issue just one generic Assertion per subject. Contrary to the specialized Assertion, the generic one contains a list of all context and role attributes that the subject is allowed to use (instead of just one in each case). This means, that the generic Assertion can be re-used for multiple role/context changes in the WS-BPEL process. Consequently, the IdP's workload can be effectively reduced (provided that there is at least one role/context change present in the WS-BPEL process). The drawback of this solution is that a new custom SOAP header needs to be introduced, in which the client specifies which context and which role (chosen from the Assertion's list of allowed ones) it wants to use. Since the WS-BPEL engine acts as the client, it is the engine's duty to select and attach the correct header to every Web service request. Hence this functionality has to be embedded in the WS-BPEL process definition which, again, increases its size and complexity substantially.

Concerning the evaluation of the WS-BPEL transformation algorithm, we again consider the ten test processes described earlier in this Section. Figure 10 shows the number of WS-BPEL elements of the process definition before and after the automatic transformation. The results indicate that the size of the WS-BPEL definition rises sharply with increasing number of scopes. While our test process with a single scope contains 33/115 WS-BPEL elements before/after transformation, the process definition for 10 scopes grows to 60/484 WS-BPEL elements before/after transformation, respectively. These numbers are determined by counting all XML (sub-)elements in the WS-BPEL file using the XPath expression `count(//*)`. At the beginning of the transformation, 41 elements are added (`import`, `partnerLink` and `variable` declarations), and for each new scope 41 elements are added for the IAM task definitions (note that both values are 41 coincidentally). We observe that the ability to define security annotations in WS-BPEL greatly reduces the required effort at design time.

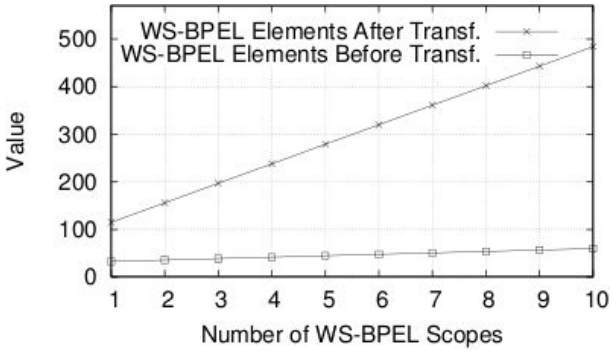

Figure 10: Process Size before and afer Transformation

The textual DSL is used as an interface to the policy rules stored in the RBAC repository. In case a UML binding is required, it is straightforward to integrate our DSL with domain-specific UML extensions for process-related RBAC models (see, e.g., [36]).

6. RELATED WORK

This section discusses related approaches in the area of model-driven IAM and their application to SOA.

Skoksrud et al. present Trust-Serv [31], a solution for model-driven trust negotiation in Web service environments. The platform supports modeling of trust negotiation policies as state machines, and the policy enforcement is transparent to the involved Web services. Different strategies for policy lifecycle management and migration are proposed. Our approach is less concerned with iterative creation of trust relationships, but builds on an IAM model and uses an integrated enforcement in Web service based business processes.

An integrated approach for Model Driven Security, that promotes the use of Model Driven Architectures in the context of access control, is presented by Basin et al. [2]. The foundation is a generic schema that allows creation of DSLs for modeling of access control requirements. The domain expert then defines models of security requirements using these languages. With the help of generators these models are then transformed to access control infrastructures.

The approach by Wolter et al. [39] is concerned with modeling and enforcing security goals in the context of SOA business processes. Similar to our approach, their work suggests that business process experts should collaboratively work on the security policies. A computational independent model (CIM) defines high-level goals, and the CIM gets mapped to a platform independent model (PIM) and further to a platform specific model (PSM). At the PIM level, XACML and *AXIS 2*[4] security configurations are generated. Whereas their approach is more generic and attempts to cover diverse security goals including integrity, availability and audit, we focus on IAM in WS-BPEL business processes.

Kulkarni et al. [9] describe an application of context-aware RBAC to pervasive computing systems. As the paper rightly states, model-level support for revocation of roles and permissions is required to deal with changing context information. Whereas their approach has a strong focus on dynamically changing context (e.g., conditions measured by sensors) and the associated permission (de-)activation, context in our case is a design-time attribute that is part of the RBAC model definitions.

A related access control framework for WS-BPEL is presented by Paci et al. in [23]. It introduces the *RBAC-WS-BPEL* model and the authorization constraint language *BPCL*. Similar to our approach, the BPEL activities are associated with required permissions (in particular, we associate permissions for `invoke` activities that try to call certain service operations). However, one main difference is related to the boundaries of the validity of user permissions: RBAC-WS-BPEL considers pairs of adjacent activities (a_1 and a_2, where a_1 has a control flow link to a_2) and defines rules among them, including separation of duty (a_1 and a_2 must execute under different roles) and binding of duty (a_1 and a_2 require the same role or user); our approach, on the other hand, is to annotate scopes in BPEL processes, which allows to apply separation and binding of duties in a sequential, but also in a hierarchical manner.

A dynamic approach for enforcement of Web services Security is presented in [16] by Mourad et al. The novelty of the approach is mainly grounded by the use of Aspect-Oriented Programming (AOP) in this context, whereby security enforcement activities are specified as *aspects* that are dynamically weaved into the WS-BPEL process at certain *join points*. Essentially, our approach can also be regarded as a variant of AOP: the weaved aspects are the IAM tasks, and join points are defined by security annotations in the process. A major advantage of our approach is the built-in support for SSO and cross-organizational IAM. An interesting extension could be to decouple security annotations from the WS-BPEL definition and to dynamically adapt to changes at runtime.

Various other papers have been published that are related to our work or have influenced it, some of which are mentioned in the following. The platform-independent framework for Security Services named SECTISSIMO has been proposed by Memon at al. [13]. A multilayer mandatory access control (MAC) architecture tailored to Web applications is presented by Hicks et al. [7]. Lin et al. [11] propose policy decomposition to support collaborative access control definition. In [3] an approach to speeding up credential-based access control operations – in particular in the web context – is proposed by Carminati et al.

XACML [18] is an XML-based standard to describe RBAC policies in a flexible and extensible way. Our DSL could be classified as a high-level abstraction that implements a subset of XACML's feature set. Using a transformation of DSL code to XACML markup, it becomes possible to integrate our approach with the well-established XACML environment and tools for policy integration (e.g., [12]).

7. CONCLUSION

We presented an integrated approach for Identity and Access Management in a SOA context. The solution is centered around model-driven development of RBAC constraints, and runtime enforcement of these constraints in Web services based business processes. Our approach fosters cross-organizational authentication and authorization in service-based systems, and greatly simplifies development of SSO-enabled WS-BPEL processes. Although tailor-made SSO solutions (coded explicitly in the business process) may yield a performance gain over the generic approach, from a practical viewpoint our approach has the advantage of being highly reusable and simple to apply. As part of our ongoing work, we are developing alternative ways to define and assign RBAC permissions at runtime, also taking into account dynamic mutual exclusion. We further investigate the use of additional security annotations and an extended view of context information.

8. REFERENCES

[1] M. Alam, M. Hafner, and R. Breu. A constraint based role based access control in the SECTET a model-driven approach. In *Int. Conf. on Privacy, Security and Trust*, 2006.

[4] http://axis.apache.org/axis2/java/core/

[2] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering Methodology*, 15:39–91, 2006.

[3] B. Carminati and E. Ferrari. AC-XML documents: improving the performance of a web access control module. In *10th ACM SACMAT*, pages 67–76, 2005.

[4] D. F. Ferraiolo and D. R. Kuhn. Role-Based Access Controls. In *15th National Computer Security Conference*, 1992.

[5] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Artech House, second edition, 2007.

[6] O. Garcia-Morchon and K. Wehrle. Efficient and context-aware access control for pervasive medical sensor networks. In *IEEE Int. Conf. on Pervasive Computing and Communications Workshops*, pages 322 –327, April 2010.

[7] B. Hicks, S. Rueda, D. King, T. Moyer, J. Schiffman, Y. Sreenivasan, P. McDaniel, and T. Jaeger. An architecture for enforcing end-to-end access control over web applications. In *15th ACM SACMAT*, pages 163–172, 2010.

[8] V. Koufi, F. Malamateniou, and G. Vassilacopoulos. A Mediation Framework for the Implementation of Context-Aware Access Control in Pervasive Grid-Based Healthcare Systems. In *4th Int. Conf. on Advances in Grid and Pervasive Computing*, pages 281–292, 2009.

[9] D. Kulkarni and A. Tripathi. Context-aware role-based access control in pervasive computing systems. In *13th ACM SACMAT*, pages 113–122, 2008.

[10] N. Li, Q. Wang, W. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin. Access control policy combining: theory meets practice. In *14th ACM SACMAT*, pages 135–144, 2009.

[11] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. Policy decomposition for collaborative access control. In *13th ACM SACMAT*, pages 103–112, 2008.

[12] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino. XACML Policy Integration Algorithms. *ACM Transactions on Information System Security*, 11:4:1–4:29, February 2008.

[13] M. Memon, M. Hafner, and R. Breu. SECTISSIMO: A Platform-independent Framework for Security Services. In *Modeling Security Workshop at MODELS '08*, 2008.

[14] T. Mens and P. V. Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.

[15] M. Mernik, J. Heering, and A. Sloane. When and How to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37(4):316–344, December 2005.

[16] A. Mourad, S. Ayoubi, H. Yahyaoui, and H. Otrok. New approach for the dynamic enforcement of Web services security. In *8th Int. Conf. on Privacy Security and Trust*, pages 189 –196, 2010.

[17] B. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33–38, Sept. 1994.

[18] OASIS. eXtensible Access Control Markup Language. http://docs.oasis-open.org/xacml/2.0, 2005.

[19] OASIS. Metadata for the OASIS Security Assertion Markup Language (SAML). http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf, 2005.

[20] OASIS. Security Assertion Markup Language. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf, March 2005.

[21] OASIS. Web Services Security: SOAP Message Security 1.1. http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 2006.

[22] OASIS. Web Services Business Process Execution Language. http://docs.oasis-open.org/wsbpel/2.0/OS, 2007.

[23] F. Paci, E. Bertino, and J. Crampton. An Access-Control Framework for WS-BPEL. *Int. J. f. Web Services Research*, 5(3):20–43, 2008.

[24] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, 2007.

[25] A. Pashalidis and C. J. Mitchell. A taxonomy of single sign-on systems. In *8th Australasian Conference on Information Security and Privacy*, pages 249–264, 2003.

[26] W. rong Jih, S. you Cheng, J. Y. jen Hsu, and T. ming Tsai. Context-aware access control in pervasive healthcare. In *EEE Workshop: Mobility, Agents, and Mobile Services*, 2005.

[27] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *Computer*, 29(2):38 –47, 1996.

[28] D. C. Schmidt. Model-Driven Engineering – Guest Editor's Introduction. *Computer*, 39(2), February 2006.

[29] B. Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5), 2003.

[30] S. Sendall and W. Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5), 2003.

[31] H. Skogsrud, B. Benatallah, and F. Casati. Model-Driven Trust Negotiation for Web Services. *IEEE Internet Computing*, 7:45–52, November 2003.

[32] D. Spinellis. Notable design patterns for domain-specific languages. *J. of Systems and Software*, 56(1):91–99, 2001.

[33] T. Stahl and M. Völter. *Model-Driven Software Development*. John Wiley & Sons, 2006.

[34] M. Strembeck. A Role Engineering Tool for Role-Based Access Control,. In *3rd Symposium on Requirements Engineering for Information Security*, 2005.

[35] M. Strembeck. Scenario-driven Role Engineering. *IEEE Security & Privacy*, 8(1), January/February 2010.

[36] M. Strembeck and J. Mendling. Modeling Process-related RBAC Models with Extended UML Activity Models. *Information and Software Technology*, 53(5), May 2011.

[37] M. Strembeck and G. Neumann. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments. *ACM Trans. on Inf. and System Security*, 7(3), 2004.

[38] M. Strembeck and U. Zdun. An Approach for the Systematic Development of Domain-Specific Languages. *Software: Practice and Experience (SP&E)*, 39(15), October 2009.

[39] C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel. Model-driven business process security requirement specification. *J. Syst. Archit.*, 55:211–223, 2009.

[40] World Wide Web Consortium (W3C). XML Signature Syntax and Processing. http://www.w3.org/TR/xmldsig-core/, 2008.

[41] U. Zdun and M. Strembeck. Modeling Composition in Dynamic Programming Environments with Model Transformations. In *5th Int. Sym. on Software Composition*, 2006.

[42] U. Zdun and M. Strembeck. Reusable Architectural Decisions for DSL Design: Foundational Decisions in DSL Projects. In *14th European Conference on Pattern Languages of Programs (EuroPLoP)*, July 2009.