An Approach to Extract RBAC Models from BPEL4WS Processes

Jan Mendling, Mark Strembeck, Gerald Stermsek, and Gustaf Neumann
Department of Information Systems, New Media Lab
Vienna University of Economics and BA, Austria
e-mail: firstname.lastname@wu-wien.ac.at

Abstract

The Business Process Execution Language for Web Services (BPEL) has become the defacto standard for Web Service composition. Yet, it does not address security aspects. This paper is concerned with access control for BPEL based processes. We present an approach to integrate Role-Based Access Control (RBAC) and BPEL on the meta-model level. Moreover, we show that such an integration can be used to automate steps of the role engineering process. In particular, we extract RBAC models from BPEL processes and present an XSLT converter that transforms BPEL code to the XML import format of the xoRBAC software component.

1. Introduction

Web Services define interfaces for software components that can be accessed via standard Internet protocols. In a Web Service scenario, software components publish their interfaces as a document conforming to the Web Service Description Language (WSDL, see [6]). WSDL interfaces define so-called *port types* as a named set of abstract operations and the abstract messages. An operation in this context is described by the atomic message exchange supported by a service. Typically, Web Services are invoked through messages using the Simple Object Access Protocol (SOAP, see [11]) which can be used in conjunction with Internet protocols like HTTP or SMTP. Since Web Services are stateless in nature, complementary specifications have been defined to provide business process related state control. The Business Process Execution Language for Web Services (BPEL4WS or BPEL) [1] is such a specification that is applicable for both external choreography and internal composition of business processes. A process engine provides the runtime environment for Web Service composition.

Several security requirements of web-based business processes can be addressed via access control measures. Simplified, access control deals with the definition and enforcement of access control policies. Such policies define which subject is allowed to perform which operations on which objects. Thus, a simple access control policy rule breaks down to a $\langle subject, operation, object \rangle$ triple. An access request is granted iff the corresponding subject owns a permission for the requested operation. Role-based Access Control (RBAC) introduces roles as an additional abstraction layer to decouple subjects and permissions [9]. In essence, RBAC roles are a collection of permissions. In RBAC and in the area of workflow modeling roles are also used as an abstract concept for delegation (e.g. [19, 23]) or the assignment of obligations (e.g. [22]). Roles are often closely related to work profiles or job descriptions of an organization (see [14]), and RBAC directly supports the assignment of permissions based on the need-to-know principle. Although BPEL considers security issues to be important (cf. [1]), it leaves all security aspects to the implementation of a BPEL compliant process engine. Accordingly, no security aspects are standardized through BPEL.

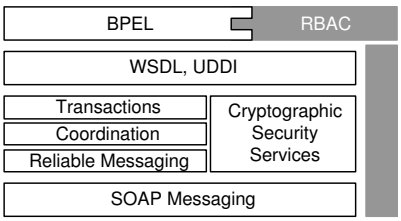


Figure 1. RBAC and the Web Services stack

This paper presents an approach towards the integration of BPEL and RBAC on the meta-model level. Figure 1 indicates that access control measures (e.g. via RBAC) can be sensibly applied on different abstraction levels (actually down to the operating system level). However, in this paper we focus on the relation of RBAC and BPEL only.

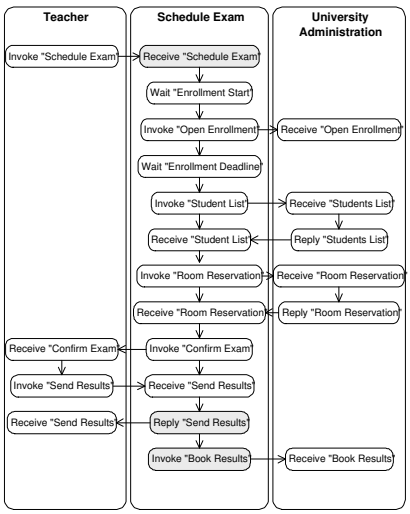
The remainder of the paper is structured as follows. In Section 2 we introduce BPEL and give an example to motivate its integration with RBAC. Section 3 gives an overview of RBAC and Section 4 introduces a mapping between

BPEL and RBAC on the meta-model level. Furthermore, we show how the mapping helps to automate some steps of the role engineering process introduced in [14]. In Section 5 an XSLT transformation script is used to extract RBAC models from BPEL processes. Section 6 gives an overview of related work, and Section 7 concludes the paper.

2. An Introduction to BPEL

BPEL is an XML-based language for the composition of executable business processes based on Web Services. In this section, we use an example to introduce its basic language concepts. A complete specification of BPEL can be found in [1]. Figure 2 shows a "schedule exam" process offered to university lecturers. After a teacher has initiated an exam, the enrollment is opened by the university administration. According to the number of enrolled students, an adequate room is scheduled. Subsequently, the list of students, the room number, and the time of the exam are sent to the teacher. Finally, the teacher conducts the exam and returns a list of the grades to the university administration.

The right column of Figure 2 shows an excerpt of BPEL code to specify this process. In BPEL conversational relationships between two parties are called `partnerLinks`. A `partnerLink` serves as an abstraction for interrelated BPEL roles representing internal (`myRole`) and external parties (`partnerRole`) in a message exchange. So-called `partnerLinkTypes` are BPEL extensions used in WSDL documents that define the port types of each BPEL role involved in a conversation. In Figure 2, lines 1–6 show that a port type is not explicitly defined in the BPEL process definition. It has to be retrieved from a `partnerLinkType` called "Exam" and the port type attached to its "Exam Service"



BPEL

```
1  <process name="Schedule Exam" ...>
2    <partnerLinks>
3      <partnerLink name="Exam"
4        partnerLinkType="uni:ExamLT"
5        myRole="Exam Service"
6        partnerRole="Teacher"/>
       ...
7    </partnerLinks>
8    <partners>
9      <partner name="Academic Staff">
10       <partnerLink name="Teacher"/>
11       <partnerLink name="Author"/>
12     </partner>
13   </partners>
14   <variables>
15     <variable name="Schedule"
16       messageType="uni:ScheduleM"/>
       ...
17   </variables>
18   <sequence>
19     <receive partnerLink="Exam"
20       portType="uni:ExamProcess"
21       operation="scheduleExam"
22       variable="Schedule"
23       createInstance="yes"/>
       ...
24     <reply partnerLink="Exam"
25       portType="tch:Exam"
26       operation="sendResults"
27       variable="ResultList"/>
       ...
28     <invoke partnerLink="Admin"
29       portType="adm:results"
30       operation="bookResults"
31       inputVariable="ResultList"/>
       ...
32   </sequence>
33 </process>
```

Figure 2. A BPEL example process.

role. A `Partner` element (lines 9–11) can be used to group multiple `partnerLinks`. For example, consider another process for gathering information about publications of a faculty which involves a role called "Author". A `partner` element called "Academic Staff" could be used to group the `parnterLinkTypes` that include the roles "Teacher" and "Author". Using `partner` definitions provides for an indirect mechanism to group `partnerRole` elements. The message types used in a BPEL process are described via `variables`. A variable is identified by a unique name and is associated with a message type. Variables store received messages and hold messages to be sent to other parties. In lines 15–16 of Figure 2 a "Schedule" variable is declared. In order to route a message to the correct process instance, a BPEL process includes the definition of correlation sets (not shown in Figure 2). A `correlation set` describes parts of messages which are unique for a process instance. Aliases to these message parts are called `properties`.

A BPEL process describes the execution order of Web Service operations via basic and structured activities. A *basic activity* is either a message exchange between Web Services or a local operation of a BPEL engine. The example illustrates a `receive` activity in lines 19–23. In general, `receive` blocks the process until a matching message arrives. A `reply` activity represents a synchronous response to a preceding `receive` activity (see lines 24–27). Invocations of remote Web Service operations are modeled as `invoke` activities. Lines 28–31 illustrate an asynchronous one-way invocation, i.e. only an input variable, but no output variable is declared. Synchronous request/response interaction can be expressed by including an additional output variable to store the synchronously sent response message.

The control flow logic of a BPEL process is defined through *structured activities*: `flow` for parallel execution; `sequence` for sequential execution; `switch` for branching related to a calculated value; `while` for loops; `compensate` for compensation actions; and `pick`. The `pick` activity is a structured activity similar to `receive`. It defines branching semantics in response to a timer event or to a message receipt. Synchronous `invoke` with an output variable, `receive`, and `pick` define access channels for external messages that are sent to a BPEL process engine. Therefore, they play an important role for BPEL related access control measures. Figure 3 summarizes the BPEL concepts via a simplified meta-model.

There are some aspects of business processes explicitly mentioned to be outside the scope of BPEL, e.g.:

- *Coordinated Transactions:* Coordination of multiple parties in a business transaction is also regarded as being orthogonal to BPEL. BPEL recommends WS-Transaction [5] to be used in order to coordinate multiple participants in a distributed transaction.

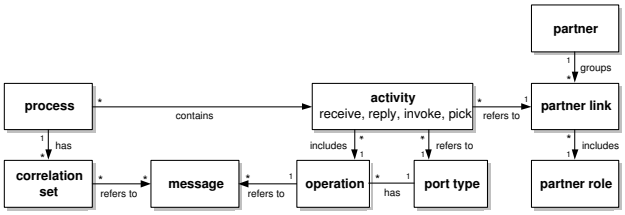


Figure 3. Exerpt from the BPEL meta-model

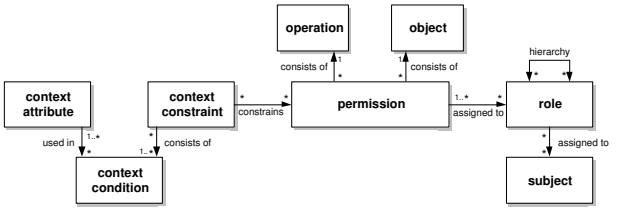


Figure 4. A simple RBAC meta-model

- *Cryptographic Security Services:* The BPEL specification recommends to implement WS-Security [2] in order to grant reliable messaging. WS-Security provides a framework to ensure that messages have not been modified or forged, and to detect duplicate or expired messages.
- *Access Control:* Conceptually, a BPEL process is open to everybody who is able to support the relevant partner link types. In many web-based business scenarios, however, it is necessary to implement a more restrictive access control policy.

If we consider cryptographic security services as digital signatures, security tokens, and encryption to be available; there is still a need to define an access control policy allowing only certain subjects to act according to a specific role. In our example (see Figure 2), this aspect is important in any situation where an external message is routed to a process instance: First, when a Web Service acting as a teacher initiates the process; and second, when the grade list is send to the exam process. Thus, Web Service compositions have security requirements beyond cryptographic security services. Moreover, a permission assigned to a certain subject/role may depend on specific context information (see also [15]), e.g. specific instance data of a process. To put it in terms of the above example, the permission for a teacher T to send a grade list back to the process may depend upon whether T instantiated that exam process and not whether he is known to be a teacher.

3. Role-Based Access Control

Role Based Access Control (RBAC) [9] provides a flexible approach to model access control policies. Permissions are assigned to roles, and roles are assigned to subjects (cf. Figure 4). Roles can be modeled to reflect the work profiles of subjects in an organization (see [14]), and in connection with business processes, roles are equipped with all permissions that a corresponding role owner needs to complete her respective tasks (see [14]). Furthermore, roles tend to change significantly slower than the assignment of subjects to these roles. Thus, the administration of access rights is one of the central strengths of RBAC. A *role hierarchy* is a directed acyclic graph that defines inheritance relations between RBAC roles. A *senior-role* is a role that inherits permissions from one or more *junior roles*. For example, a senior manager role could be modeled as a senior-role to a junior manager role.

Together with various extensions RBAC evolved into a defacto standard for access control in software-based systems. Context constraints (see [15]) are an RBAC extension to model and enforce context-dependent access control policies. In this connection, *context attributes* represent properties of the environment whose values may change dynamically (like time or process state), or which vary for different instances of a certain entity (like location or ownership). A *context condition* is a predicate that compares the current value of a context attribute with at least one other value, e.g. a constant or another context attribute. A *context constraint* consists of one or more context conditions. It evaluates to true iff all its context conditions hold.

4. Mapping of BPEL to RBAC Elements

In Figure 5, dotted lines indicate mappings from BPEL meta-model elements to respective RBAC elements. *RBAC Roles* can be derived from roles and partners in BPEL. A BPEL `partnerRole` relates to a set of Web Service operations that are carried out during a business process. A BPEL `partner` is a container of multiple BPEL `partnerRole` elements which is used to define the capabilities required from a business partner. They represent a meaningful grouping of roles like the "Academic Staff" partner in Section 2 that groups the "Teacher" and the "Author" role. In RBAC, a BPEL `partner` then maps to a senior-role while the respective `partnerRole` represents a junior-role from which permissions are inherited. In the example, an "Academic Staff" senior-role would be created that inherits permissions from its junior-roles "Teacher" and "Author".

Permissions are generated for activities that define a channel for an external party to send messages to a BPEL process instance. Therefore, especially `receive`, `pick`, and synchronous `invoke` need to be considered. Outgoing messages sent by the BPEL engine to external parties via asynchronous `invoke` and `reply` can be neglected (from the perspective of the sending BPEL engine) - assuming the

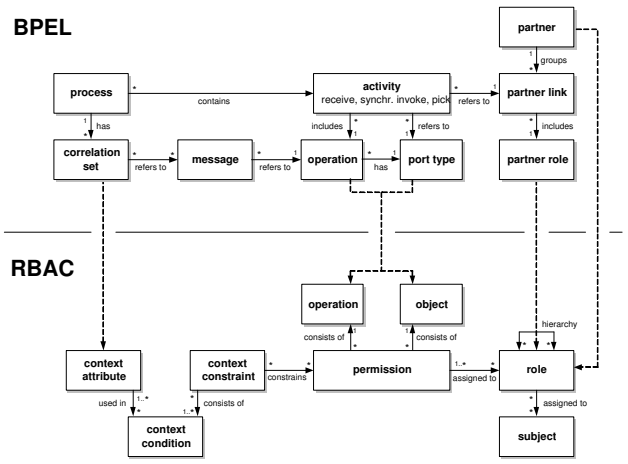


Figure 5. Mappings between BPEL and RBAC

BPEL engine works correctly and BPEL process descriptions have been verified before deployment. *Port types* can be regarded as objects in RBAC terms. Actually, port types are rather interfaces than objects. However, this mapping abstracts from the implementation behind the interface, and therefore maps port types to objects in RBAC permissions. A discussion on Web Services and distributed objects can be found in [21]. Operations of BPEL activities can be mapped to RBAC operations. Accordingly, an RBAC permission - an $\langle operation, object \rangle$ pair - is generated from a BPEL $\langle operation, porttype \rangle$ pair. Via its `partner link` attribute a BPEL operation is associated with a partner role. This relationship can be mapped to a permission-to-role assignment in RBAC. Subjects are not modeled in BPEL, and thus cannot be extracted from BPEL processes. BPEL *properties* are conceptually related to context attributes (see Section 3). A `property` represents a reference to an XPath [8] expression where attributes identifying a process instance can be found. For example, a social security number or an invoice number are natural candidates for such properties.

[14] presents the *scenario-driven role engineering process*. The scenario concept is of central significance for this approach. In essence, a scenario is an action and event sequence, and BPEL processes can be seen as formalized scenario descriptions; accordingly, they are well-suited to serve as input to the role engineering process. In order to perform a given scenario, a subject must possess access rights for all operations associated with that scenario. A role can be seen as a container for this collection of permissions. An adaption of the role-engineering approach for BPEL process descriptions includes the following top-level activities:

1. *Extract Roles and Permissions:* Roles and permissions can be extracted from a BPEL process definition. Section 5 presents the implementation of an extraction program based on XSLT [7]. XSLT is well suited for the transformation of XML-encoded BPEL process models into an XML format that can be read/imported by an RBAC implementation.
2. *Define Preliminary Role-Hierachy:* Then, the preliminary role hierarchy is defined. On this stage other permissions derived from scenarios which are not available as BPEL processes can be added. Furthermore, for each work profile respective roles are created. This may lead to redundancies which are marked for later review. For further details, see [14].
3. *Define RBAC Model:* Finally, the RBAC model is defined. In this step, redundant roles are removed from the preliminary role hierarchy. If necessary, constraints like e.g. Process Instance Ownership are identified and defined. After the constraint catalog has been completed, respective roles can be created and added to the role hierarchy. For details, see [14].

5. Extraction of RBAC Elements from BPEL

We developed an XSLT transformation script that automates the first step of the role engineering process described in Section 4. This script extracts information from a BPEL process and stores it in an XML format that can be read by the xoRBAC component. xoRBAC provides an RBAC-service that enables the definition and enforcement of RBAC policies including context constraints [13, 15]. Basically the script implements three types of mappings that derive RBAC elements from BPEL processes (cf. Figure 6):

- *PortType-Operation Pairs to RBAC Permissions:* In Section 2, we mentioned that `receive`, `pick`, and synchronous `invoke` with an output variable are the BPEL activities that allow for the receipt of external messages. Since these activities are the only activities to provide channels for incoming messages, the corresponding $\langle portType, operation \rangle$ pairs map to permissions (see also Section 4).
- *BPEL Partner Roles to RBAC Roles:* A `partnerRole` identifies an external party in BPEL. For each `partnerRole` an RBAC role is generated. If a BPEL role *R* is associated with a `partner`, a senior-role relationship with this `partner` has to be created for *R*. Furthermore, for each activity that implies an incoming message from *R* a permission is added.
- *BPEL Partner to Senior-Roles:* A `partner` in BPEL groups partner links. For example, Section 2 mentioned an "Academic Staff" `partner` that was used to group "Teacher" and "Author" roles. Partner links include the respective `partnerRole` attribute to identify the external party in a conversation. This defines

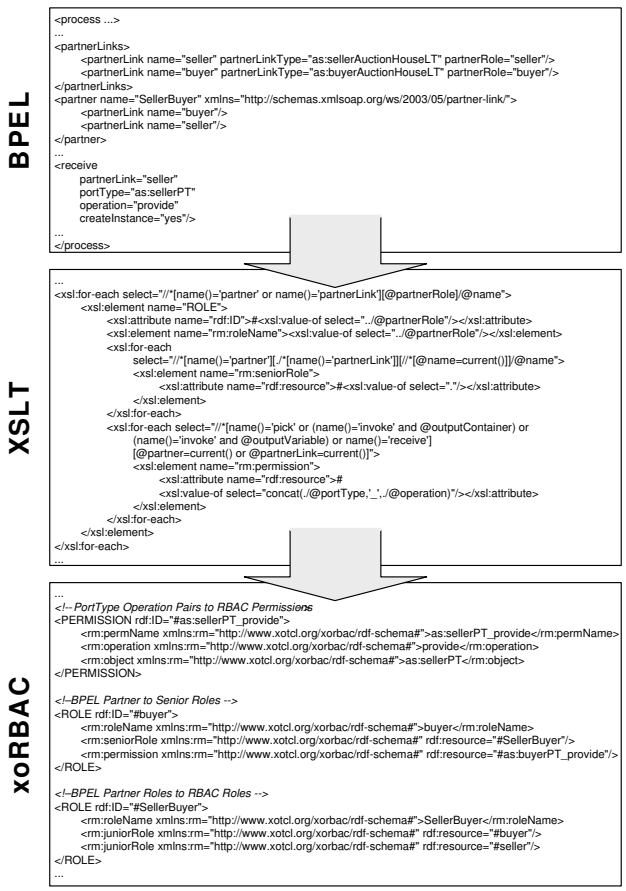


Figure 6. BPEL-to-RBAC XSLT script.

an indirect hierarchical relationship between partners and partner roles. A `partner` is mapped to an RBAC role with junior-role relationships to each indirectly (via a `partnerLinkType`) associated `partnerRole`. The RBAC roles derived from BPEL `partner` elements do not have directly assigned permissions, because a partner element in BPEL is only used for grouping.

While the XSLT script above produces an XML document that can be imported by the xoRBAC component, the approach is general in nature and can be applied to any other RBAC component accordingly.

6. Related Work

Our approach for the integration of BPEL and RBAC on the meta-model level is related to organizational modeling in workflow systems, e.g. as presented by Rosemann and Zur Mühlen [16], and in [22] Yu and Schmid present a framework for workflow modeling which includes permissions that can be assigned to roles. In the context of workflow modeling, roles are frequently discussed from a resource perspective: i.e. work items are routed to available resources that are modeled as roles. Moreover, delegation of tasks between subjects and/or roles is an important issue. Van der Aalst et al. [19] present an organizational model for a workflow system based on the eXchangeable Routing Language (XRL) [20]. Their model is capable to express inheritance and delegation hierarchies of roles. Our paper complements such workflow-routing oriented work.

If messages are exchanged via the Internet, security issues need a stronger emphasis compared to Workflow scenarios deployed on a trusted local network. Skogsrud et al. [17] present Trust-Serv which extends the Self-Serv platform [3] with a trust negotiation framework. In particular, they use credentials to perform access control measures, and identify RBAC as a topic for their future research.

RBAC and its use in a business process context is another area of work related to our paper. Various contributions concerning access control in collaborative environments exist, especially for groupware and workflow systems. For example, Thomas and Sandhu introduced TBAC [18], a family of models that support the specification of active security models. In TBAC permissions are dynamically (de)activated according to the current task/process-state. In [12], an approach for access control in inter-organizational workflows is suggested, and in [4], Bertino et al. describe a well-elaborated language and algorithms to express and enforce constraints which ensure that all tasks within a workflow are performed by predefined users/roles. Neumann and Strembeck presented a scenario-driven role engineering process [14]. It uses scenarios to extract permissions and to define tasks and work profiles. Subsequently, work profiles are used to derive RBAC roles. In our paper, we suggest an approach to automate certain steps of the role engineering process for scenarios that are defined as BPEL processes.

Security aspects of web services have been addressed by specifications complementary to BPEL, e.g. Web Service Security [2]. Other proposals like the Extensible Access Control Markup Language (XACML) [10] may also be applied in a Web Service context. The eXtensible Access Control Markup Language (XACML) [10] is a standard adopted by the Organization for the Advancement of Structured Information Standards (OASIS) and provides an XML-based language for the definition of access control policies. Nevertheless, XACML especially focuses on the definition of the policy language and the corresponding XML schemata. It does not address engineering aspects like elicitation or maintenance of access control policies.

7. Conclusion and Future Work

In this paper we presented an approach for the integration of BPEL and RBAC on the meta-model level. Our work is motivated by two main facets. First, BPEL does not ad-

dress access control measures although access control is an important and integral aspect of business processes. Second, role engineering is a time-consuming task and can be made more efficient through an integration with business process modeling. We use the mappings between RBAC and BPEL to automate steps of the scenario-driven role engineering process presented in [14]. In particular, we described a respective XSLT script which transforms BPEL process descriptions to RBAC models in an XML format that can be imported by the xoRBAC software component [13, 15]

With our approach we aim to enhance the security features of Business Process Management Systems that operate via Web Services. Moreover, our approach provides for more efficient role engineering as it automates steps of the scenario-driven role engineering process. Finally, as RBAC and process models are highly interrelated, automation in role engineering also facilitates consistency between the deployed business processes and corresponding RBAC policies. In our future work we implement an RBAC-aware BPEL engine that reflects the findings of this paper. In particular, the implementation will build on an integrated metamodel of BPEL and RBAC. Another interesting aspect for future work is the continued integration of role engineering activities with BPEL-based processes.

References

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Specification, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems, 2003.

[2] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdellie, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, and D. Simon. Web Services Security. Specification, IBM Corp., Mircosoft Corp., VeriSign, Inc., 2002.

[3] B. Benatallah, Q. Sheng, and M. Dumas. The self-serv environment for web services composition. *IEEE Internet Computing*, 7(1):40–48, January/February 2003.

[4] E. Bertino, E. Ferrari, and V. Atluri. The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. *ACM Transactions on Information and System Security (TISSEC)*, 2(1), February 1999.

[5] F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey, and S. Thatte. Web Service Transaction. Specification, BEA Systems, IBM Corp., Microsoft Corp., 2002.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Service Description Language (WSDL) 1.1. Note, W3C, March 2001.

[7] J. Clark. XSL Transformations (XSLT) Version 1.0. Recommendation, W3C, November 1999.

[8] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. Recommendation, W3C, November 1999.

[9] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and Systems Security*, 4(3), 2001.

[10] S. Godik and T. Moses, eds. eXtensible Access Control Markup Language (XACML). Specification, OASIS, 2003.

[11] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen. SOAP Version 1.2 Part 1 and Part 2. Recommendation, W3C, June 2003.

[12] M. Kang, J. Park, and J. Froscher. Access Control Mechanisms for Inter-Organizational Workflow. In *Proc. of the ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2001.

[13] G. Neumann and M. Strembeck. Design and Implementation of a Flexible RBAC-Service in an Object-Oriented Scripting Language. In *Proc. of the 8th ACM Conference on Computer and Communications Security (CCS)*, 2001.

[14] G. Neumann and M. Strembeck. A Scenario-driveen Role Engineering Process for Functional RBAC Roles. In *Proc. of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2002.

[15] G. Neumann and M. Strembeck. An Approach to Engineer and Enforce Context Constraints in an RBAC Environment. In *Proc. of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2003.

[16] M. Rosemann and M. zur Mühlen. Evaluation of workflow management systems - a meta model approach. In K. Siau, Y. Wand, and J. Parsons, editors, *Proc. of the 2nd EMMSAD Workshop, Barcelona, Spain*, 1997.

[17] H. Skogsrud, B. Benatallah, and F. Casati. Model-driven trust negotiation for web services. *IEEE Internet Computing*, 7(6):45–52, November/December 2003.

[18] R. Thomas and R. Sandhu. Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented authorization management. In *Proc. of the IFIP WG11.3 Conference on Database Security*, August 1997.

[19] W. M. P. v.d. Aalst, A. Kumar, and H. M. W. Verbeek. Organizational modeling in UML and XML in the context of workflow systems. In *Proc. of the ACM Symposium on Applied Computing (SAC)*, pages 603–608, 2003.

[20] H. Verbeek, A. Hirnschall, and W. van der Aalst. XRL/Flower: Supporting Interorganizational Workflows using XRL/Petri-net Technology. In *Web Services, E-Business, and the Semantic Web, CAiSE 2002 International Workshop (WES 2002)*, LNCS 2512, pages 93–108, 2002.

[21] W. Vogels. Web Services are not Distributed Object. *IEEE Internet Computing*, pages 59–66, Nov/Dec 2003.

[22] L. Yu and B. Schmid. A conceptual framework for agent oriented and role based workflow modeling. In *Proceedings of the CaiSE Workshop on Agent Oriented Information Systems (AOIS99)*, 1999.

[23] L. Zhang, G. Ahn, and B. Chu. A Rule-Based Framework for Role-Based Delegation and Revocation. *ACM Transactions on Information and System Security (TISSEC)*, 6(3), August 2003.