An Access-Control Framework for WS-BPEL

Federica Paci, Purdue University, USA
Elisa Bertino, Purdue University, USA
Jason Crampton, University of London, UK

ABSTRACT

Business processes, the next-generation workflows, have attracted considerable research interest in the last 15 years. More recently, several XML-based languages have been proposed for specifying and orchestrating business processes, resulting in the WS-BPEL language. Even if WS-BPEL has been developed to specify automated business processes that orchestrate activities of multiple Web services, there are many applications and situations requiring that people be considered as additional participants who can influence the execution of a process. Significant omissions from WS-BPEL are the specification of activities that require interactions with humans to be completed, called human activities, and the specification of authorization information associating users with human activities in a WS-BPEL business process and authorization constraints, such as separation of duty, on the execution of human activities. In this article, we address these deficiencies by introducing a new type of WS-BPEL activity to model human activities and by developing RBAC-WS-BPEL, a role-based access-control model for WS-BPEL, and BPCL, a language to specify authorization constraints.

Keywords: activity; authorization constraints; business process; permission; role

INTRODUCTION

Business process management (BPM) systems have gained a lot of attention due to the pressing need for integrating business processes of different organizations. Research efforts have been devoted to improve current workflow technologies in order to support collaborative business processes. BPM systems can be considered as an extension of classical workflow management (WFM) systems. Older proprietary workflow systems managed document-based processes where people executed the workflow steps. Today’s BPM systems manage processes that include person-to-person work steps, system-to-system communications or combinations of both. In addition, BPM systems include integrated features such as enhanced (and portable) process modeling, simulation, code generation, process execution, and process monitoring. All these functions and features have resulted in an increased interest in BPM suites because they enhance business process flexibility while at the same time reducing risks and costs. Therefore, BPM suites are a way to build, execute,
and monitor automated processes that may go across organizational boundaries: a kind of next-generation workflows.

Recently, Web services have provided the basis for the development and execution of business processes that are distributed over the network and available via standard interfaces and protocols. Business processes or workflows can be built by combining Web services through the use of a process specification language. Such languages basically allow one to specify which tasks have to be executed and the order in which they should be executed. Because of their importance, process specification languages have been widely investigated, and a number of languages have been developed. One such language is WS-BPEL 2.0 (Web services business process execution language), an XML-based workflow process language, which provides a syntax for specifying business processes in terms of Web services (Jordan & Evdemon, 2007). WS-BPEL resulted from the combination of two different workflow languages, WSFL (Leymann, 2001) and XLANG (Thatte, 2001), and adopts the best features of these languages. WS-BPEL is layered on top of several XML standards, including WSDL 1.1 (Web services definition language; Christensen, Curbera, Meredith, & Weerawarana, 2001), XML Schema 1.0 (Biron & Malhotra, 2004), and XPath 1.0 (Clark & DeRose, 1999), but of these, WSDL has had the most influence on WS-BPEL.

However, despite the significant progress toward the development of an expressive language for business processes, significant challenges still need to be addressed before we see the widespread use of business process management systems in distributed computer systems and Web services. Even if WS-BPEL has been developed to support the specification of automated business processes that orchestrate activities of multiple Web services, there are cases in which people must be considered as additional participants who can influence the execution of a process. Recently, a WS-BPEL extension to handle person-to-person processes has been proposed called BPEL4People (Agrawal et al., 2007b). In BPEL4People, users that have to perform the activities of a WS-BPEL business process are directly specified in the process by user identifiers or by groups of people’s names. No assumption is made on how the assignment is done or on how it is possible to enforce constraints like separation of duties.

WS-BPEL does not provide any support for the specification of either authorization policies or authorization constraints on the execution of activities composing a business process. We believe, therefore, that it is important to extend WS-BPEL to include the specification of human activities and an access-control model able to support the specification and enforcement of authorizations to users for the execution of human tasks within a business process while enforcing constraints, such as separation of duty, on the execution of those tasks (Bertino, Ferrari, & Atluri, 1999; Casati, Castano, & Fugini, 2001; Crampton, 2005a; Wainer, Barthelmess, & Kumar, 2005).

In this article, we propose an approach to extend WS-BPEL with an authorization model that also supports the specification of a large number of different types of constraints.

Role-based access control (RBAC) is a natural paradigm for the specification and enforcement of authorization in workflow systems because of the correspondence between tasks and permissions.

In recent years, several extensions to RBAC have been proposed with the goal of supporting access control for workflow systems (Ahn, Sandhu, Kang, & Park, 2000; Bertino et al., 1999; Wainer et al., 2005). We make use of this work in defining RBAC-WS-BPEL, a role-based access-control model based on the specification of authorization policies for business processes defined in WS-BPEL. However, a role-based model alone is not sufficient to meet all the authorization requirements of business process management systems such as separation-of-duty constraints and binding-of-duty constraints. Separation-of-duty requirements exist to prevent conflicts of interest and to make fraudulent acts more difficult to com-

Copyright © 2008, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
mit. A simple example of a separation-of-duty constraint is to require two different signatures on a check. Binding-of-duty constraints require that if a certain user executes a particular activity, then this user must also execute a second activity in the business process.

In this article, we introduce BPCL (business process constraint language), which can be used to specify authorization constraints for WS-BPEL business processes. This language is influenced by the seminal work of Bertino et al. (1999) on authorization constraints in workflow systems and more recent work on constraint specification and enforcement (Crampton, 2005a).

In the next section, we present an overview of WS-BPEL and introduce an example that we will use throughout the article for illustrative purposes. Then, we define the components of RBAC-WS-BPEL, including authorization policies and authorization constraints. In the subsequent section, we provide an example of an RBAC policy for a purchase-order WS-BPEL business process, specified in XACML (Moses, 2005). After this, we describe BPCL and how it implements the authorization constraints described previously. Next, we illustrate the specification of human activities, and authorization information and constraints in the purchase-order WS-BPEL business process. This is followed by the presentation of an algorithm to evaluate if a request by a user to execute an activity in a WS-BPEL process can be granted, and a discussion on a possible implementation of our model. Finally, we conclude with related work and future research directions.

**INTRODUCTION TO WS-BPEL**

WS-BPEL is an XML-based language to specify business processes that orchestrate the operations of several Web services. The top-level element in the specification is `<process>`. It has a number of attributes that specify the process name, the name spaces being referred to, and whether the process is an abstract process or an executable process. An executable process describes the internal implementation of the process, while an abstract process specifies the external behavior of a process. The `<partner-Links>` element is used to identify the external Web services invoked from within the process. The `<variables>` element defines the data that flow within the process. The `<correlationSets>` element is used to bind a set of operations to a service instance. The `<faultHandlers>` element is used to handle exceptions. The `<compensationHandlers>` element is used to implement actions to be taken in the case of transaction rollback. The `<eventHandlers>` are used to specify actions in response to external events.

The actual business logic is represented as a group of activities, which are executed in a structured way. Activities are executed by invoking Web services’ operations. The business logic includes basic control structures: The `<sequence>` activity contains one or more activities that are performed sequentially, the `<if>` activity is used to specify conditional branching execution, the `<while>` activity supports iterative execution of an activity, the `<pick>` activity is used to trigger an activity following a specified event, the `<repeatUntil>` activity provides for repeated execution of a contained activity, the `<forEach>` activity iterates the execution of an enclosed `<scope>` activity for a fixed number of times, and the `<flow>` activity is used to specify one or more activities to be performed concurrently. The `<links>` elements can be used within a `<flow>` activity to define explicit control dependencies between nested child activities; each specifies that the activity that contains its `<source>` element must be executed before the one that includes the link’s `<target>` element. These activities, in turn, may contain basic activities: the `<invoke>` activity that allows the business process to invoke a one-way or request-response operation on a communications channel offered by a partner, the `<receive>` activity that allows the business process to wait in a blocking mode for a matching message to arrive, and the `<reply>` activity that allows the business process to send a message in reply to a message that was received via a `<receive>` activity. The `<scope>` activity defines a subprocess with its own variables, partner links, message exchanges, correlation...
sets, event handlers, fault handlers, compensation handler, and termination handler.

The creation of a business process instance in WS-BPEL is always implicit; activities that receive messages (that is, <receive> activities and <pick> activities) can be annotated to indicate that the occurrence of that activity results in a new instance of the business process to be created. When a message is received by such an activity, an instance of the business process is created if it does not already exist. A business process instance is terminated when one of the following conditions hold: The last activity in the process terminates, a fault occurs and it is not handled appropriately, or a process is terminated explicitly by a terminate activity.

To provide concrete examples of the proposed extensions to WS-BPEL, we introduce, as a running example, a purchase ordering process being part of a purchase ordering and financial system.

**Example 1.** There are six activities involved in ordering and paying for goods:

- the creation of a purchase order requesting goods from a supplier (crtPO),
- the approval of the purchase order prior to dispatch to the supplier (apprPO),
- the acknowledgement of delivery of the goods by signing a goods-received note (signGRN),
- the creation of a payment file on receipt of the supplier’s invoice for the goods (crtPay), and
- the approval of the payment to the supplier (subject to receipt of goods: apprPay).

An informal specification of the process is shown in Figure 1: An arc from one activity to another means that the former activity must be executed before the latter. Hence, the execution of the crtPO activity must precede that of the apprPO activity, while the signGRN and crtPay activities can be executed in parallel because no execution order is specified. Figure 2 shows how the purchase-order process is expressed in WS-BPEL. The process orchestrates the operations of four Web services: a Web service that provides the operation crtPay to create the payment file, a Web service providing the operation apprPO to approve the order and to send it to the supplier, a Web service that provides the operation signGRN that allows the signature of the good-request note, and a Web service that offers the operation apprPay to approve the payment for the order. The <process> element is the root element and represents the whole business process specification. The structure of the main processing section is defined by the outer <sequence> element, which states that the contained activities are to be sequentially executed.

The <sequence> element contains, in the following order, a crtPO <receive> activity representing the receipt of an order from a customer and the creation of the purchase-order

---

**Figure 1. A purchase-order process specification**

![Figure 1. A purchase-order process specification](chart.png)
Figure 2. The purchase-order process expressed in WS-BPEL

```
<process name="purchaseOrderProcess"
    targetNamespace="http://acme.com/ws-bp/purchase"
    xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns="http://manufacturing.org/wsdl/purchase/>

<partnerLinks>
<partnerLink name="customer"
    partnerLinkType="PurchaseOrderPartnerLT"
    myRole="PurchaseOrderService" />
<partnerLink name="approverPO"
    partnerLinkType="ApproverPOLT"
    partnerRole="approverPO" />
<partnerLink name="approverPOPayment"
    partnerLinkType="ApproverPOPaymentLT"
    partnerRole="approverPOPayment" />
<partnerLink name="creator"
    partnerLinkType="creatorLT"
    partnerRole="creator" />
<partnerLink name="signerGRN"
    partnerLinkType="signerGNRLT"
    partnerRole="signerGNR" />
</partnerLinks>

<variables>
<variable name="PO" messageType="POMessage"/>
<variable name="GRN" messageType="GRNMessage"/>
<variable name="PF" messageType="PFMessage"/>
</variables>

<sequence>
<receive partnerLink="customer" portType="crtPOPT" operation="crtPO"
    variable="PO" name="Create Purchase Order" createInstance="yes">
</receive>
<invoke partnerLink="approverPO" portType="approvePOPT"
    operation="apprPO" inputVariable="PO" name="Approve Purchase Order">
</invoke>
<flow>
<link name="signGRN-to-ctrsignGRN"/>
<invoke partnerLink="signerGRN" portType="signGRNPT" operation="signGRN"
    inputVariable="GRN" name="Sign GRN">
<source linkName="signGRN-to-ctrsignGRN"/>
</invoke>
<invoke partnerLink="signerGRN" portType="ctrsignGRNPT"
    operation="ctrsignGRN" inputVariable="GRN" name="CounterSign GRN">
<target linkName="signGRN-to-ctrsignGRN"/>
</invoke>
<invoke partnerLink="creator" portType="ctrPayPT" operation="ctrPay"
    outputVariable="PF" name="Create Payment File">
</invoke>
</flow>
<invoke partnerLink="ApproverPOPayment" portType="apprPayPT"
    operation="apprPay" inputVariable="PF" name="Approve Payment">
</invoke>
</sequence>
</process>
```
signGRN and ctrsignGRN since the execution of signGRN must precede the one of ctrsignGRN. The execution dependency between the signGRN and ctrsignGRN <invoke> activities is expressed using a <link> element: The signGRN <invoke> activity is the source activity of the link signGRN-to-ctrsignGRN, while the ctrsignGRN <invoke> is the target activity.

Note that all <invoke> activities in the purchase-order process require interactions with humans. Later we show how the execution of these activities is handled.

**RBAC-WS-BPEL: AN AUTHORIZATION MODEL FOR WS-BPEL**

A WS-BPEL process is a representation of a business process and is typically specified as a set of activities and a set of dependencies between the activities. The dependencies fall into two broad categories: those determined by the application logic of the process such as the order of execution of the activities (Rusinkiewicz & Sheth, 1995), and those determined by security requirements. WS-BPEL supports only the first of those dependency categories.

In this article, we deal with the second category and focus on developing a role-based authorization model for WS-BPEL. The model, referred to as RBAC-WS-BPEL, inherits all the components of traditional RBAC models: users, roles, permissions, role hierarchies, user-role assignment, and role-permission assignment relations. Users are assigned to roles and roles are assigned to permissions. An RBAC-WS-BPEL permission represents the ability to execute an activity of a WS-BPEL business process. A user acquires the permission to execute a business process’ activity only if he or she is assigned to a role that has the permission to perform that activity. RBAC-WS-BPEL roles are structured in a hierarchy that reflects the different responsibilities associated with a business process and defines a permissions inheritance relation among the roles. Moreover, RBAC-WS-BPEL
allows one to specify authorization constraints like separation-of-duty requirements that place restrictions on the potential inheritance of permissions from opposing roles.

The main difference with respect to traditional RBAC models is that WS-BPEL business processes coordinate the operations of Web services provided by different organizations. Therefore, roles do not represent job functions within a single organization like in traditional RBAC models, and potential users of the business process are not the employees of an organization performing job functions identified by roles. With traditional RBAC, business process users may not be known a priori and there is a need for a mechanism to identify users and to assign them to roles. In RBAC-WS-BPEL, users are thus identified by means of digital credentials. A credential contains a set of attributes characterizing the owner specified via name-value pairs. An RBAC-WS-BPEL role identifies a set of conditions on users’ attributes. A user is assigned to a role if the user’s credentials match the user’s attribute conditions associated with the role.

Figure 3 illustrates the relations among the various RBAC-WS-BPEL components, which are defined in what follows.

**Definition 1: RBAC-WS-BPEL Permission**

Let BP be a WS-BPEL business process. An RBAC-WS-BPEL permission is a tuple \((A_i, \text{Action})\) where \(A_i\) is the identifier of an activity in BP and \(\text{Action}\) identifies the type of action that can be performed on activity \(A_i\).

To render our specification open to future extensions, we do not specify the types of action that can be performed on an activity. In the following examples, we will consider the type of action \(\text{execute}\), allowing a user to carry out an activity of the business process.

**Definition 2: RBAC-WS-BPEL Role**

An RBAC-WS-BPEL role \(r\) is a set of attribute conditions \(r = \{ac_i | ac_i \equiv \text{AttrName}_{op}, \text{AttrValue}_{i}\}\), where \(\text{AttrName}_i\) identifies a user attribute name, \(op\) is a comparison or a set operator, and \(\text{AttrValue}_{i}\) is a value, a set, or a range of attribute values.

Two roles \(r\) and \(r'\) can be identified by the same set of attribute names. However, it is required that at least one of the values that the attributes of \(r\) and \(r'\) assume must be different.

Since we assume that a set of attribute conditions univocally identifies a role, a user can be assigned only to one role while two users identified by the same attributes with the same values are assigned to the same role.

---

**Figure 4. RBAC-WS-BPEL role hierarchy for the purchase-order process**

<table>
<thead>
<tr>
<th>Manager</th>
<th>{Employment = Manager, Company = Electronics&amp;CO, Age = 42}</th>
</tr>
</thead>
<tbody>
<tr>
<td>FinAdmin</td>
<td>{Bank = Chase, Employment = Branch Director}</td>
</tr>
<tr>
<td>FinClerk</td>
<td>{Bank = Chase, Employment = Employee}</td>
</tr>
<tr>
<td>POAdmin</td>
<td>{Employment = Administrator, Company = Electronics&amp;CO, Age = 42}</td>
</tr>
<tr>
<td>POClerk</td>
<td>{Employment = Employee, Company = Electronics&amp;CO}</td>
</tr>
</tbody>
</table>
Definition 3: RBAC-WS-BPEL Role Hierarchy

Let R be a partially ordered set of roles. A role hierarchy defined over R is the graph of the partial-order relation between the roles in R. If \( r_0 \in R \) and \( r < r_0 \), then we say \( r_0 \) dominates \( r \).

Example 2. Figure 4 illustrates the RBAC-WS-BPEL role hierarchy for the purchase-order business process. It consists of five different roles. The most senior role is Manager; which dominates the roles FinAdmin and POAdmin; FinAdmin and POAdmin, in turn, dominate, respectively, roles FinClerk and POClerk. For example, a user that wants to be assigned to the Manager role must provide digital credentials containing an Employment attribute equal to Manager; a Company attribute equal to Electronics&CO, and an Age attribute equal to 45.

Definition 4: RBAC-WS-BPEL User-Role Assignment Relation

Let U be the set of all potential users and R be a partially ordered set of roles. The RBAC-WS-BPEL user-role assignment relation is the set of tuples \( UA = \{ (u, r) \in U \times R \mid \forall \text{attr}_{i} \equiv \text{AttrName}_{i} \text{opAttrValue}_{i} \in r, \exists \text{attr}_{j} \in \text{CredSet}(u) | \text{attr}_{j} = \text{AttrName}_{j} \land \text{attr}_{j} \text{is evaluated true according to the value of attr}_{j} \} \).

Definition 5: RBAC-WS-BPEL Authorization Schema

Let BP be a WS-BPEL business process. A RBAC-WS-BPEL authorization schema for BP is a tuple \( (R, P, RA, UA) \) where \( R \) is a partially ordered set of roles associated with BP, \( P \) is the set of permissions defined for the activities in BP, \( RA \subseteq R \times P \) is a role-permission assignment relation, and \( UA \) is the user-role assignment relation.

One advantage of the role-based paradigm is that more senior roles inherit permissions assigned to more junior roles. This significantly reduces the number of permission-role assignments.

Example 3. Figure 5a illustrates the set of permissions associated with the purchase-order process comprising the ability to execute each of the activities in the purchase order. Figure 5b illustrates a typical permission-role assignment relation. Note that no permissions are explicitly assigned to the Manager role, although the role does implicitly have the rights to execute all activities in the process. Similarly, the FinAdmin role has the permission to execute the apprPay <invoke> activity.

The above authorization model is complemented by a language supporting the specification of constraints. In particular, RBAC-WS-BPEL allows the specification of two different types of authorization constraints: role authorization constraints and user authorization constraints. We now formally introduce these two types of constraint.

Definition 6: RBAC-WS-BPEL Authorization Constraints

Let U be a set of users and R be a partially ordered set of roles. A role authorization constraint is a tuple \( (D, (A_1, A_2), \rho) \), where \( D \subseteq R \) is the domain of the constraint and \( \rho \subseteq R \times R \). A user authorization constraint is a tuple \( (D, (A_1, A_2), \rho) \), where \( D \subseteq U \) is the domain of the constraint and \( \rho \subseteq U \times U \). A constraint \( (D, (A_1, A_2), \rho) \) is satisfied if, whenever \( x \in D \) performs \( A_1 \) and \( y \) performs \( A_2 \), \( (x, y) \in \rho \). Given a constraint \( C \equiv (D, (A_1, A_2), \rho) \), we say that \( C \) applies to \( A_2 \).

An authorization constraint places some restrictions on the users and roles who can perform \( A_2 \) (the consequent activity) given that the user \( u \in D \) or the role \( r \in D \) has executed \( A_1 \) (the antecedent activity). We will use the notation \( (D, (A_1, A_2), \neq) \) to denote a separation of duty constraint and \( (D, (A_1, A_2), =) \) to denote a binding of duty constraint. Moreover, we can specify constraints that restrict the execution of two activities by users or roles, where that restriction can be expressed as a binary relation on the set of users or roles. Such relations
Definition 7: RBAC-WS-BPEL Authorization Specification

An RBAC-WS-BPEL authorization specification is a tuple (BP, AS, AC) where BP is a WS-BPEL business process, AS is the authorization schema defined for BP, and AC is the set of authorization constraints that apply to the activities in BP.

RBAC-XACML AUTHORIZATION SCHEMA

The first extension we propose to WS-BPEL is the specification of the RBAC-WS-BPEL authorization schema associated with a WS-BPEL business process. In our approach, this component of the language is specified using the RBAC XACML policy language (Crampton, 2005b) proposed as an alternative to the RBAC profile for XACML (Anderson, 2005).

Figure 7 shows how the RBAC-WS-BPEL authorization schema reported in Figures 4 and 5 can be encoded in XACML. The autho-
<table>
<thead>
<tr>
<th>Role Set</th>
<th>Permissions</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager</td>
<td>permissions:Manager</td>
<td>Approve, Sign</td>
</tr>
<tr>
<td>FinAdmin</td>
<td>permissions:FinAdmin</td>
<td>Approve, Sign</td>
</tr>
<tr>
<td>POAdmin</td>
<td>permissions:POAdmin</td>
<td>Create, Sign</td>
</tr>
<tr>
<td>FinClerk</td>
<td>permissions:FinClerk</td>
<td>Create, Sign</td>
</tr>
<tr>
<td>POClerk</td>
<td>permissions:POClerk</td>
<td>Create, Sign</td>
</tr>
</tbody>
</table>

Figure 7. RBAC-WS-BPEL authorization schema expressed in pseudo-XACML
permission assignment <PolicySet> element. It includes a <PolicySet> subelement for each role to which the relation RA assigns a permission. Each <PolicySet> subelement contains a <PolicySetIdReference> child node for each permission assigned to the role.

<PolicySetIdReference> refers to the permission <Policy> element that represents the permission. Finally, a role <PolicySet> element represents a role in the hierarchy. For example, the Manager role is represented by the role <PolicySet> element having the <PolicySetId> attribute equal to role:Manager. The <Target> subelement limits the applicability of the role <PolicySet> to users satisfying the specified attribute conditions. The <Target> subelement of the role <PolicySet> representing the Manager role has two <Subject> subelements that specify the attributes’ conditions that a user has to satisfy to be assigned to the Manager role: Only the directors working for the Electronics&CO company who are older than 45 years can be assigned to the Manager role. <PolicySetIdReference> subelements are used to refer to the permission assignment <PolicySet> element containing the set of permissions associated with the Manager role. In addition, they are used to represent the role hierarchy by referencing immediate junior roles. The Manager role, for example, references the role <PolicySet> elements for roles FinAdmin and POAdmin.

BUSINESS PROCESS CONSTRAINT LANGUAGE

We now introduce an XML-based language for the specification of authorization constraints such as separation of duty and binding of duty. We call this language the business process constraint language. BPCL provides an XML schema template for specifying authorization constraints, reported in Appendix A.

According to the proposed XML schema, an <AuthorizationConstraints> element contains all the authorization constraints that apply to the activities in a WS-BPEL business process. Each constraint C ≡ (D, (A₁, A₂), ρ) is represented by a <Constraint> element having an ID attribute by which it is referenced. The <Constraint> element has three subelements: <Domain>, <Activities>, and <Predicate>. The <Domain> element represents the domain D of the constraint C. It has two child elements, <Type> and <Subject>. The data content of the <Type> element specifies the type of the constraint C: It contains the value of role if C is a role authorization constraint and the value of user if C is a user authorization constraint. The content of the <Subject> element is either a set of roles or a set of users, and depends on the contents of the <Type> element.

The <Activities> element specifies the two activities A₁ and A₂ to which the constraint C applies. In particular, <Activities> has two child nodes, <AntecedentActivityReference> and <ConsequentActivityReference>, containing, respectively, an XLink reference to the XML element representing activities A₁ and A₂ in the WS-BPEL specification. Finally, the <Predicate> element data content identifies the relation ρ in C: For example, the string equal identifies the relation =, while the string not equal identifies the relation ≠.

Figure 8 illustrates the use of BPCL in defining the authorization constraints associated with the purchase-order process. For example, the constraint (U, (Approve Payment, Sign GRN), ≠) is represented by the <Constraint> element with ID attribute C₁. Notice that the content of the <Subject> element can be empty as in both C₁ and C₄, in which case all elements in the appropriate domain are considered. In C₄, the <AntecedentActivityReference> and <ConsequentActivityReference> elements respectively refer to the crtPay and apprPay activities (in the purchase-order WS-BPEL specification), to which the constraint is applied. The relation < is represented by the string seniority in the <Predicate> element data content. Furthermore, since (R, (Create Payment File, Approve Payment), <) is a role authorization constraint, the <Type> data content element is equal to role.
In this section, we illustrate how our extensions are incorporated into the purchase-order-process WS-BPEL specification introduced in previously. WS-BPEL has been designed to be extensible. Extensions to WS-BPEL could include anything ranging from new attributes to new elements, to extended assign operations and activities to enable restrictions or extensions of run-time behavior and so on. The <process> element contains an <extensions> element having an <extension> child element that is used to declare name spaces of WS-BPEL extension attributes and elements, and indicate whether they carry semantics that must be understood by a WS-BPEL processor. WS-BPEL allows, also, the definition of new types of activities by placing them inside the <extensionActivity> element. The contents of an <extensionActivity> element must be a single element qualified with a name space different from the WS-BPEL name space. We have applied the WS-BPEL extension rules to specify which activities require the interaction with users and to specify the authorization information and the authorization constraints applied to these activities.

First, we declared into the process specification an <extension> element specifying the name space http://www.example.org/rbac-ws-bpel that identifies our extensions. Then, we introduced a new type of WS-BPEL activity called <HumanActivity> to specify the activities that must be performed by humans: <HumanActivity> contains the activity that to be performed requires interaction with a user. Finally, we added two new child elements to the <process> element, <authorization_schema> and <authorization_constraints>, to include the references to the authorization information necessary to state which roles or users are allowed to execute the business process activities, and the authorization constraints that apply to the activities in

```
<AuthorizationConstraints>
  <Constraint Id="C1">
    <Domain>
      <Type>User</Type>
      <Subject/>
    </Domain>
    <Activities>
      <AntecedentActivityReference xlink:type="simple"
        xlink:href="purchase_order_wsbpelspec.xml#xpointer(//*[@name="Approve Payment "])"/>
      <ConsequentActivityReference xlink:type="simple"
        xlink:href="purchase_order_wsbpelspec.xml#xpointer(//*[@name="Sign GRN"])">
    </Activities>
    <Predicate>not equal</Predicate>
    <Constraint>
      <Domain>
        <Type>Role</Type>
        <Subject/>
      </Domain>
      <Activities>
        <AntecedentActivityReference xlink:type="simple"
          xlink:href="purchase_order_wsbpelspec.xml#xpointer(//*[@name="Create Payment File"])"/>
        <ConsequentActivityReference xlink:type="simple"
          xlink:href="purchase_order_wsbpelspec.xml#xpointer(//*[@name="Approval Payment"])"/>
      </Activities>
      <Predicate>seniority</Predicate>
      <Constraint>
      </Constraint>
    </Constraint>
  </Constraint>
  ... ...
  <Constraint Id="C4">
    ... ...
    <Constraint>
    </Constraint>
  </Constraint>
</AuthorizationConstraints>
```
the process. The `<authorization_schema>` and `<authorization_constraints>` elements have a `ref` attribute of type URI respectively pointing to the XML document defining the RBAC policy and to the BPCL representation of the authorization constraints.

As we can see from the RBAC-WS-BPEL specification of the purchase order, our approach for associating authorization information and authorization constraints with business process human activities is characterized by some interesting features. First, the specification of authorization information and authorization constraints in WS-BPEL does not require a significant modification to the syntax of the language. We simply require the inclusion of two new XML elements in the WS-BPEL syntax. Hence, the specification of a WS-BPEL business process that includes authorization information and authorization constraints is modular. Furthermore, with this approach it is easy to modify the authorization information and authorization constraints associated with the business process since only the references to them need to be modified. Second, the language we have proposed for the specification of authorization constraints is very expressive. It supports the specification of binding-of-duty constraints, separation-of-duty constraints, and constraints that restrict the execution of two activities by users or roles, whereby such restriction can be expressed as a binary relation on the set of users and roles.

**RBAC-WS-BPEL ENFORCEMENT**

In the previous sections, we have presented the main components of RBAC-WS-BPEL, the access-control model we proposed for WS-BPEL. In this section, we describe the algorithm we use to determine if a user request to execute an activity $A_i$ in a WS-BPEL process can be enforced.

**Figure 9. Extended WS-BPEL purchase-order business process**

```xml
<process name="purchaseOrderProcess" targetNamespace="http://acme.com/ws-bp/purchase">
    <bpel:extension namespace="http://www.example.org/rbac-ws-bpel" mustUnderstand="yes"/>
    <rbac-ws-bpel:authorization_schema ref="http://www.example.org/rolehierarchy.xml"/>
    <rbac-ws-bpel:authorization_constraints ref="http://www.example.org/authorization_constraints.xml"/>
    <partnerLinks>
        ...
    </partnerLinks>
    <variables>
        ...
    </variables>
    <sequence>
        <receive partnerLink="customer" portType="crtPOPT" operation="crtPO" variable="PO" name="Create Purchase Order" createInstance="yes">
            ...
        </receive>
        <extensionActivity>
            <rbac-ws-bpel:HumanActivity>
                <invoke partnerLink="approverPO" portType="approvePOPT" operation="apprPO" inputVariable="PO" name="Approve Purchase Order">
                    ...
                </invoke>
            </rbac-ws-bpel:HumanActivity>
        </extensionActivity>
        ...
    </sequence>
</process>
```
satisfied. When a user $u$ issues a request to perform an activity $A_i$ of a WS-BPEL process, the enforcement system evaluates the identity of the requester, assigns him or her to a role, and checks the permissions he or she has. Furthermore, it has to verify that the execution of the activity $A_i$ by $u$ does not violate any authorization constraints and does not prevent some other subsequent activities from completing because certain constraints are violated. Hence, for a given instance of the WS-BPEL process, when the enforcement system receives a request to perform an activity $A_j$ by a user $u$, it has to verify that:

- $u$ is authorized to perform $A_j$,
- all the constraints in which $A_j$ is the consequent activity are satisfied, and
- the WS-BPEL process instance can complete if $u$ performs $A_j$.

In what follows, we present an algorithm to evaluate whether a request to execute an activity by a user can be granted. Our algorithm verifies whether the WS-BPEL process instance will complete and that no authorization constraints will be violated. The algorithm is performed before executing any request $(u, r, i, A_j)$ to check if the execution of the request does not prevent the WS-BPEL process instance from completing. To guarantee completeness, before granting a request, the authorization schema associated with the WS-BPEL process is updated with the fact that the user $u$ under role $r$ has executed activity $A_j$. Then, the algorithm computes for each activity in the WS-BPEL process the set of roles and users that are entitled to perform them. If one of these sets is empty, the request cannot be granted. After each request is granted, the authorization schema AS is updated with the fact that the user $u$ under role $r$ has executed activity $A_j$, to ensure that the fact that a particular user and role has executed a particular activity is considered in enforcing constraints that apply to subsequent activities.

In what follows, we denote with $((A_j, A_i), \rho)$ both role and user authorization constraints.

The algorithm receives as input an RBAC-WS-BPEL authorization specification $(BP, AS, AC)$, an instance $i$ of the WS-BPEL process $BP$, and a request $(u, r, i, A_j)$ by a user $u$ to execute activity $A_j$ in $BP$ under the role $r$. When the request $(u, r, i, A_j)$ is received, the algorithm first adds to the authorization schema the fact that the role $r$ of user $u$ is executing the activity $A_j$. To represent this, we have added a function $I_k$ that associates with each activity $A_i$ in BP the role that has executed $A_i$. This step is important to guarantee the completeness of the instance $i$ (Line 1).

Then, for each pair of activities $A_j$ and $A_i$, the algorithm builds $V_{v_k}(A_j, A_i)$, where $V_{v_k}(A_j, A_i)$ is the set of roles that can execute $A_j$ and $A_i$ (in that order) given the authorization schema AS and the role authorization constraints in AC. The basic strategy to compute $V_{v_k}(A_j, A_i)$ is to initialize each $V_{v_k}(A_j)$ to the set of roles that are authorized to perform $A_j$ (Line 3) and to apply all possible role constraints defined for each pair of activities, including those derived from authorization information (Lines 5-6). If for some $A_j$ and $A_i$ $V_{v_k}(A_j)$ is empty, then the algorithm terminates (Line 7) since no pair of authorized roles exists that complies with the role authorization constraints, and therefore no valid execution assignment exists for the instance $i$. Otherwise, for each task activity $A_j$ we (re)compute the set of roles that can perform $A_j$ (Lines 10-11). The same steps (Lines 12-22) are repeated to compute for each activity $A_i$ the set $V_{v_i}(A_i)$ of users authorized to perform $A_i$. If the role $r$ played by $u$ belongs to $V_{v_i}(A_i)$, the set of roles authorized to execute the activity $A_i$, and $u$ belongs to $V_{v_i}(A_i)$, the set of users that are authorized to perform $A_i$, the request $(u, r, i, A_j)$ can be granted.

**RBAC-WS-BPEL SYSTEM ARCHITECTURE**

In this section, we describe a possible implementation for the proposed RBAC-WS-BPEL access-control model on top of a WS-BPEL engine. Figure 10 represents the architecture that implements the access-control enforce-
The main component described previously. The main components are a WS-BPEL engine, a Web service called RBAC-WS-BPEL Enforcement Service that is the core of the architecture, and three repositories—XACML policy store, BPCL constraints store, and the history store. The WS-BPEL engine is responsible for scheduling and synchronizing the various activities within the business process according to the specified activity dependencies, and for invoking Web service operations associated with activities. The RBAC-WS-BPEL Enforcement Service has two tasks. First, it manages the execution of the business process’s <HumanActivity> activity. It is important to notice that our enforcement service is able to manage such execution without requiring any extensions to legacy WS-BPEL engines. Second, it acts as a reference monitor: When a user claims a <HumanActivity> activity, it verifies that the user is authorized to perform it according to the authorization schema and authorization constraints.

The RBAC-WS-BPEL Enforcement Service offers two WSDL interfaces. The first interface provides the operations to start and complete the execution of a human activity.
This interface provides two operations: initiateActivity and onActivityResult. initiateActivity is a one-way operation that is invoked within a WS-BPEL process to start the execution of a <HumanActivity> activity. The invocation message of initiateActivity contains a set of information about the activity and the business process, and references to authorization schema and authorization constraints. Figure 11 illustrates the content of the initiateActivity invocation message for <HumanActivity> representing the execution of the apprPO <invoke> activity. The onActivityResult callback operation is performed to notify the WS-BPEL engine that the execution of a <HumanActivity> activity is complete. The message received by the WS-BPEL process contains the business process variables modified by the <HumanActivity>.

The second interface of the RBAC-WS-BPEL Enforcement Service allows users to display the activities they can claim, and to claim and execute them. This interface offers two operations: listActivity and claimActivity. The listActivity operation returns the list of the activities a user can claim. To claim an activity in the list, a user invokes the claimActivity operation. When this activity is executed, the RBAC-WS-BPEL Enforcement Services selects all possible BPCL constraints and the information in XACML authorization schema, and runs Algorithm 1 to determine if the user request can be granted or not. If the user is authorized, the WSDL operation providing the interface of the <HumanActivity> activity that is specified in the invocation message of the initiateActivity operation is invoked.

The XACML policy store contains the RBAC-WS-BPEL authorization schema associated with the business process, while the BPCL constraints store contains the authorization constraints. The history store is used to record the past executions of each human activity, including the user who has performed it and if the execution of the activity was successful or not. History information is used to enforce authorization constraints.
Figure 11. Example of initiateActivity invocation message

```xml
<initiateActivityInputMsg>
  !--WS-BPEL Business process variables the task has to manipulate--!
  <variables>
    <variable name = "PO" > value<variable>
  </variables>
  <HumanActivityInterface portType="approvePOPT" operation="apprPO" inputVariable="PO" name="Approve Purchase Order ">
  </HumanActivityInterface>
  <rbac-ws-bpel: authorization_schema
    ref="http://www.example.org/rolehierachy.xml"/>
  <rbac-ws-bpel: authorization_constraints
    ref="http://www.example.org/authorization_constraints.xml"/>
</initiateActivityInputMsg>
```

HANDLING <HUMANACTIVITY> ACTIVITY EXECUTION AND RBAC-WS-BPEL ENFORCEMENT

We now describe how the execution of <HumanActivity> activities is realized and how the RBAC-WS-BPEL authorizations and authorization constraints are enforced on them.

To perform a <HumanActivity> activity, it is first necessary to add this activity to the <partnerLinks> list in the WS-BPEL process specification the RBAC-WS-BPEL Enforcement Service. Then, each <HumanActivity> is replaced with the WS-BPEL code to fill in the fields of RBAC-WS-BPEL Enforcement Service’s initiateActivity operation invocation message, followed by the <invoke> activity to perform initiateActivity, the <receive> activity to receive the message of RBAC-WS-BPEL Enforcement Service’s onResultActivity operation, and the code to update the business process variables with the value contained in the message (see Figure 12).

The execution of a <HumanActivity> starts when the WS-BPEL process invokes the initiateActivity operation. The process then waits to be called back by the RBAC-WS-BPEL Enforcement Service. The RBAC-WS-BPEL Enforcement Service adds the name of the activity contained in the invocation message to the list of <HumanActivity> that can be claimed. When a user requests the <HumanActivity> by invoking the claimActivity operation, the RBAC-WS-BPEL Enforcement Service queries the XACML policy store to determine on the basis of the digital credentials provided by the user.

Figure 12. <HumanActivity> code translation in standard WS-BPEL code

```xml
<extensionActivity>
  <rbac-ws-bpel: HumanActivity>
    <invoke partnerLink="approverPO " portType="approvePOPT"
      operation="apprPO" inputVariable="PO" name="Approve Purchase Order ">
    </invoke>
  </HumanActivity>
<extensionActivity>
```
users to which role the user can be assigned. If the user’s credentials match the attribute conditions of a role, then the RBAC-WS-BPEL Enforcement Service executes Algorithm 1 to verify that the user is authorized to perform the activity without preventing the end of the business process execution. If the user can perform the activity, the RBAC-WS-BPEL Enforcement Service removes the activity from the list of the <HumanActivity> that can be claimed and then invokes the operation that provides the user interface of the activity. If the execution of the operation completes successfully, the RBAC-WS-BPEL Enforcement Service calls back the WS-BPEL process performing the operation on ActivityResult. The message content associated with this operation is used by the WS-BPEL process to update the status of its variables. Finally, the RBAC-WS-BPEL Enforcement Service updates the history store, recording the name of the user that performs the activity and if the execution was successful or not.

**RELATED WORKS**

The problem of associating an authorization model with a workflow has been widely investigated. Atluri and Huang (1996) have proposed a workflow authorization model (WAM) that supports the specification of authorizations in such a way that users gain access to required objects only during the execution of a task, thus synchronizing the authorization flow with the workflow. To achieve such synchronization, their approach associates an authorization template (AT) with each task in the workflow, which allows appropriate authorizations to be granted only when the task starts and be revoked when the task finishes. They have proposed an implementation of WAM using Petri nets in order to be able to perform safety analysis because the safety problem in WAM is equivalent to the reachability problem in Petri nets.

Compared to the work of Atluri and Huang (1996), RBAC-WS-BPEL does not allow the specification of temporal authorizations that have validity only within the expected duration of a certain task or activity; however, both models support the specification of role-based authorizations and separation-of-duties constraints. In both models, the assignment of authorized users to activities is done during the execution of the business process.

Arguably, the most sophisticated approach to the problem of authorizing users to execute tasks within a workflow, while enforcing constraints, is the one by Bertino et al. (1999). According to such approach, a workflow is a list of task-role specifications. A task-role specification identifies a task, specifies the roles authorized to perform the task, and specifies the maximum number of activations of the task that are permitted in an instance of the workflow. The model, however, supports only a sequential task execution. As part of such approach, a language for defining constraints on role assignment and user assignment to tasks in a workflow has been developed. Such a constraint language supports, among other functions, both static and dynamic separation-of-duty constraints. The authors have also shown how such constraints can be formally expressed as clauses in a logic program; such reduction makes it possible to exploit results from logic programming and deductive databases. A further contribution of the approach is the development of algorithms for planning role and user assignments for the various tasks. The goal of algorithms is to pre-compute all the possible role-task assignments and user-task assignments so that all constraints stated as part of the authorization specification are satisfied. A drawback of this approach is that the algorithms for role and user assignment to tasks run in a time exponential in the number of tasks in the workflow.

Both the model of Bertino et al. (1999) and RBAC-WS-BPEL are role-based authorization models and allow the definition of authorization constraints restricting the set of users and roles that can execute an activity. Unlike the model of Bertino et al., in RBAC-WS-BPEL, a role represents a set of user attributes and conditions a user has to satisfy in order to be assigned to the role. Moreover, RBAC-WS-BPEL supports the sequential, concurrent, conditional, and iterative execution of activities, while the
model of Bertino et al. supports only sequential execution. Finally, the model of Bertino et al. determines all the possible assignments of users and roles to activities before the execution of a workflow is started, while in RBAC-WS-BPEL, the assignment is done at run time when a user claims the execution of an activity.

Another interesting work is by Crampton (2005a). He has proposed an expressive method for specifying authorization constraints in workflow systems. In particular, his model allows the specification of separation-of-duty constraints, weak separation-of-duty constraints, binding-of-duty constraints, constraints on the relative seniority of users who perform different tasks, and constraints determined by contextual user-based information. All the constraints are expressed as binary relations on the set of users. The model has the advantage of being independent from any computational model or access-control mechanism. As part of such an approach, an algorithm has been proposed for the assignment of authorized users to tasks in a workflow that guarantees the workflow instance completes. The algorithm runs in a time polynomial in the number of users and tasks, unlike the equivalent procedure in the model proposed by Bertino et al. (1999).

Unlike the work of Crampton (2005a), RBAC-WS-BPEL assumes role-based access-control enforcement. Both models specify authorization constraints as a binary relation on the set of users. RBAC-WS-BPEL, however, also allows the definition of authorization constraints on the set of roles. Moreover, in the models, the assignment of authorized users to activities in a business process is performed at run time in a way that guarantees that the business process instance completes.

Casati et al. (2001) have proposed an authorization framework for the assignment of tasks to roles, organizational levels, and agents. Roles and organizational levels are structured into hierarchies to facilitate the assignment of tasks to agents. Authorizations for agents to play roles and levels, and for roles and levels to execute tasks can be specified for all instances of a given workflow process independent of time and workflow execution history. Then, the framework enables the definition of instance-dependent, time-dependent, and history-dependent authorizations in the form of constraints: Authorizations can be modified depending on the state or history of a workflow instance, the time, or the content of process data. Authorization constraints are enforced as event-condition-action (ECA) rules. The event part denotes when an authorization may need to be modified. The condition part verifies that the occurred event actually requires modifications of authorizations, and determines the involved agents, roles, tasks, and processes. The action part enforces authorizations and prohibitions. Active rules are also exploited for managing authorization inheritance along the role and level hierarchies of the framework. Active database technology has been adopted for the implementation of the framework; it has been used in particular to support the definition and execution of ECA rules. Finally, Casati et al. have presented the implementation of the authorization framework within the WIDE workflow management system.

RBAC-WS-BPEL and the model of Casati et al. (2001) are role-based access-control models. In the model of Casati et al., authorization constraints also model time- and history-dependent authorizations; RBAC-WS-BPEL supports only the specification of history-dependent authorization constraints. In both approaches, authorization and authorization constraints are evaluated at run time during the execution of a business process.

With the widespread adoption of Web services to implement complex business process and of WS-BPEL as the standard language to specify business processes based on Web services, the problem of how to associate authorized users with the activities of a WS-BPEL process is gaining attention. The RBAC-WS-BPEL authorization model that has been introduced in this article is one of the few approaches that address this problem. Another similar approach is the one of Koshutanski and Massacci (2003). They propose an authorization model for business processes based on Web services. In this
approach, the authorization logic is decoupled from the application logic of the business process. Access-control enforcement is based on two types of policies: access-control policies and release policies. Both types of policy are expressed as logic rules specifying conditions on the credentials a user must submit to invoke business process activities. Access-control policies are used to decide if a user request can be granted or not. A request is granted if it is a logical consequence of an access-control policy and of the credentials submitted by the user with the request. Release policies are used when a user request is denied to determine the additional credentials that a user has to provide for the request to be granted. The enforcement process involves different components. A policy evaluator is associated with each Web service, the activities of which are orchestrated in a business process: It takes local authorization decisions. A policy orchestrator defines an authorization business process that orchestrates the authorization processes performed by the policy evaluators of the Web services invoked to fulfill a user’s request. The authorization business process is executed by a third component called the authorization server that returns the result of the execution to the user. If the user request is denied, the user receives a business process that defines the further actions that he or she has to execute in order to see the request accepted.

Both the model of Koshutanski and Massacci (2003) and RBAC-WS-BPEL assume a role-based access-control model. Both models support the specification of authorizations to roles and to users by assigning a user to a role and the specification of authorizations constraints on the set of users and roles.

In both models, the assignment of users to roles is attribute based.

Authorizations in the model of Koshutanski and Massacci (2003) grant the execution of a Web service to a user or role, while in RBAC-WS-BPEL, authorizations grant the execution of a Web service’s operation.

In the model of Koshutanski and Massacci (2003), authorizations and authorization constraints are both expressed using logic rules, while in RBAC-WS-BPEL, RBAC-XACML is used to model authorizations and BPCL to specify authorization constraints. Moreover, the model of Koshutanski and Massacci supports the specification of release policies to determine the additional credentials that a user has to provide to be granted a previously denied request. RBAC-WS-BPEL does not allow the specification of release policies: If a user cannot be assigned to a role on the basis of the credentials he or she submits, the request to perform an activity is denied.

Another interesting proposal is BPEL4People, recently proposed by IBM and SAP. BPEL4People includes extensions to WS-BPEL that are required in order to support user interactions. BPEL4People is comprised of the two following specifications:

- The WS-BPEL extension for people (Agrawal et al., 2007b) layers features on top of WS-BPEL to describe human tasks as activities that may be incorporated as first-class components in WS-BPEL process definitions.
- Web services human task (WS-HumanTask; Agrawal et al., 2007a) introduces the definition of stand-alone human tasks, including the properties, behavior, and operations used to manipulate them. Capabilities provided by WS-HumanTask may be utilized by Web-services-based applications beyond WS-BPEL processes.

The WS-BPEL extension for people introduces a new basic WS-BPEL activity called <people activity> that uses human tasks as an implementation, and allows specifying tasks local to a process or user tasks defined outside of the process definition. The definition of stand-alone human tasks is given in WS-HumanTask specification. A local task can be (a) an in-line task declared within the <people activity>, (b) an in-line task declared within either the scope containing the <people activity> or the process scope, or (c) a stand-alone task identified using a QName. The element <task> is used to define an
in-line task within a <people activity>. The elements <localtask> and <remotetask> are used to specify, respectively, stand-alone tasks that do not offer a callable Web service interface and those that offer a callable Web service interface. The users entitled to perform a <people activity> are specified by a <peopleAssignment> element that associates with the activity a query on an organizational directory.

Both RBAC-WS-BPEL and BPEL4People introduce a new type of WS-BPEL activity to handle human interactions in WS-BPEL and specify how the execution of this activity is performed.

In RBAC-WS-BPEL, human activities always have a callable Web service interface, while BPEL4People supports human activities without a callable Web service interface. Moreover, BPEL4People does not allow the specification and the enforcement of authorizations and authorization constraints on human activities while RBAC-WS-BPEL does.

CONCLUSION

In this article, we have proposed two extensions to WS-BPEL. The first extension is the introduction of a new type of WS-BPEL activity to specify those activities that are not automatically executed but require interaction with a user to be completed. The second extension is RBAC-WS-BPEL, a role-based access-control model for WS-BPEL that supports the specification of authorization information necessary to state if a role or a user is allowed to execute human activities composing the processes. The authorization information comprises a role hierarchy reflecting the organizational structure, a permission-role assignment relation, and a set of permissions that represent the ability to execute activities. The authorization information is encoded using XACML. We have also defined a schema for BPCL, a new XML-based language for describing authorization constraints. Such constraints place restrictions on the roles and users that can perform the activities in the business process. Furthermore, we have illustrated how these components can be included in the business process specification. We have proposed an algorithm to evaluate if a request by a user to perform an activity in a WS-BPEL process can be granted or not. The algorithm verifies whether the execution of a WS-BPEL process will complete without violation to the authorization constraints. Finally, we have proposed an implementation of access-control enforcement on the WS-BPEL engine that does not require any modification to existing engines.

One of the advantages of our approach is that the resulting specification, including a WS-BPEL business process specification, authorization information, and authorization constraints, is modular. It is thus possible for the same business process specification to have different authorization information: Different organizations may define different roles and different assignments of activities to roles. Moreover, different organizations may have different access-control policies that require the specification of different authorization constraints. A further advantage is the expressiveness of BPCL, which enables us to support specification constraints that go beyond basic separation- and binding-of-duty constraints.

We are currently extending this work in several directions. The first extension deals with the introduction of a new type of authorization constraint. Currently, in the constraint $(D, (A_1, A_2), \rho)$, the execution of activity $A_2$ is constrained by the execution of a single antecedent activity $A_1$. We plan to extend our language so that the execution of $A_2$ may be dependent from the execution of several antecedent activities. We also want to extend BPCL to support both positive and negative authorizations. Another direction on which we will focus is the development of more sophisticated algorithms for the assignment of users and roles to activities that satisfies authorization constraints. A final possibility for future work is to consider the use of BPCL as a general authorization constraint language and investigate how it could interoperate with XML-based authorization languages such as XACML.
REFERENCES


**ENDNOTE**

1. CredSet(u) is the set of digital credentials associated with user u.
Here we report the XML schema for the authorization constraints introduced in the article.

```xml
<xsd:schema xmlns:xsd= "http://www.w3.org/2001/XMLSchema">
  <xsd:element name="AuthorizationConstraints" type="Auth_constrType"/>
  <xsd:complexType name="Auth_ConstrType">
    <xsd:element name="Constraint" type="ConstrType" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:complexType>
  <xsd:complexType name="ConstrType">
    <xsd:sequence>
      <xsd:element name="Domain" type="DomainType"/>
      <xsd:element name="Activities" type="ActivitiesType"/>
      <xsd:element name="Predicate" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="Id" type="xsd:ID"/>
  </xsd:complexType>
  <xsd:complexType name="DomainType">
    <xsd:sequence>
      <xsd:element name="Type" type="xsd:string"/>
      <xsd:element name="Subject" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ActivitiesType">
    <xsd:element name="AntecedentActivityReference">
      <xsd:complexType>
        <xsd:attribute name="xlink:type" fixed="simple" type="xsd:string"/>
        <xsd:attribute name="xlink:href" type="xsd:anyURI"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="ConsequentActivityReference">
      <xsd:complexType>
        <xsd:attribute name="xlink:type" fixed="simple" type="xsd:string"/>
        <xsd:attribute name="xlink:href" type="xsd:anyURI"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:complexType>
</xsd:schema>
```