Access Control and Authorization Constraints for WS-BPEL

Elisa Bertino
Cerias and Computer Science Department
Purdue University
West Lafayette, IN
bertino@cerias.purdue.edu

Jason Crampton
Information Security Group
Royal Holloway, University of London
Egham, United Kingdom
jason.crampton@rhul.ac.uk

Federica Paci
Dipartimento di Informatica e Comunicazione
Universita' degli Studi di Milano
Milano, Italy
paci@dico.unimi.it

Abstract

Computerized workflow systems have attracted considerable research interest in the last fifteen years. More recently, there have been several XML-based languages proposed for specifying and orchestrating business processes, culminating in WS-BPEL. A significant omission from WS-BPEL is the ability to specify authorization information associating users with activities in the business process and authorization constraints on the execution of activities such as separation of duty. In this paper, we address these deficiencies by developing the RBAC-WS-BPEL and BPCL languages. The first of these provides for the specification of authorization information associated with a business process specified in WS-BPEL, while BPCL provides for the articulation of authorization constraints.

1. Introduction

Recent advances in web services, associated technology and standards, such as XML, and recent research on web service semantics are enabling new, high-level approaches to programming the web. The main idea of such approaches is that 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 those tasks 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 (Web Services Business Process Execution Language), an XML-based workflow process language, which provides a syntax for specifying business processes based on web services [12]. WS-BPEL is a synthesis of two rival workflow languages, WSFL [9] and XLANG [14], and adopts the best features from these two. The language is layered on top of several XML specifications, including WSDL 1.1 [6], XML Schema 1.0 [16] and XPath 1.0 [15], but of these, the WSDL has had the most influence on WS-BPEL.

However, there remain significant challenges to be resolved before we see the widespread use of workflow technology in distributed computer systems and web services. Of particular interest to the security community is the problem of authorizing users to execute tasks within a workflow while enforcing constraints such as separation of duty on the execution of those tasks [2, 3, 4, 5, 17].

The WS-BPEL language 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 couple WS-BPEL with a model for expressing such authorization policies and constraints, and a mechanism to enforce them. It is important that such an authorization model be high-level and expressed in terms of entities that are relevant from the organizational perspective. In this paper, we propose an approach to extend WS-BPEL syntax 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 to apply to authorization in workflow systems because of the correspondence between tasks and permissions. In recent years, a considerable amount of work has been done on the use of RBAC to support access control in workflow

systems [1, 3, 17]. We make use of this work in defining RBAC-WS-BPEL, a language for 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 workflow 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 commit. A simple example of a separation of duty constraint would be to require two different signatures on a cheque. Binding of duty constraints require that if a certain user executed a particular task then that user must also execute a second task in the workflow. In this paper, we introduce BPCL (business process constraint language), which can be used to articulate authorization constraints for business processes. This language is influenced by the seminal work of Bertino *et al* on authorization constraints in workflow systems [3] and more recent work on constraint specification and enforcement [7].

In the next section we provide an overview of WS-BPEL and introduce an example that we will use throughout the paper for illustrative purposes. In Section 3 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 workflow, specified in XACML [11]. In Section 5 we describe the BPCL language and how it implements the authorization constraints described in Section 3. We then complete the specification of our example purchase order workflow. Finally, we conclude with suggestions for future work.

2 Introduction to WS-BPEL

WS-BPEL is an XML-based language to specify business processes. The top level element in the specification is `<process>`. It has a number of attributes, which specify the process name, the namespaces being referred to, and whether the process is an abstract process or an executable process. The `<partnerLinks>` element is used to identify the external web services invoked from within the process. The `<variables>` element defines the data that flows within the process. The `<correlationSets>` element is used to bind a set of operations to a service instance.

Most importantly from our point of view, 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. The business logic includes basic control structures: the `<sequence>` element contains one or more activities that are performed sequentially; the `<switch>` element is used to specify conditional branching execution; the `<while>` element supports iterative execution of an activity; the `<pick>` element is used to trigger an activity following a specified event; and the `<flow>` element is used to specify concurrent execution of a set of activities. These activities, in turn, may contain basic activities, which are specified using one of the following elements: the `<invoke>` element, that allows the business process to invoke a one-way or request-response operation on a communications channel offered by a partner; the `<receive>` element that allows the business process to wait in a blocking mode for a matching message to arrive; and the `<reply>` element that allows the business process to send a message in reply to a message that was received via a `<receive>` activity.

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 causes a new instance of the business process to be created. 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 the WS-BPEL language, we introduce a purchase ordering process as a running example. Consider a simple process forming 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 despatch to the supplier (*`apprPO`*);*
- *the acknowledgement of delivery of the goods by signing a goods received note (*`signGRN`*);*
- *the acknowledgement of delivery by countersigning the goods received note (*`ctrsignGRN`*);*
- *the creation of a payment file on receipt of the supplier's invoice for the goods (*`crtPay`*);*
- *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 first activity must be executed before the other. Hence, the execution of the `crtPO` activity must precede that of `apprPO`, while `signGRN` and `crtPay` can be executed in parallel because no order of execution is specified.

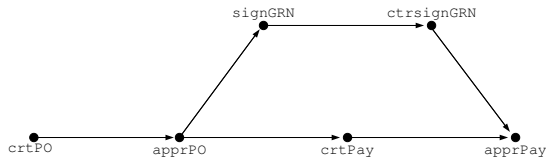


Figure 1. A purchase order process specification

Figure 2 shows the purchase order process rendered in WS-BPEL. The <process> element is the root element of the document 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 sequentially executed.

The <sequence> element contains, in the following order: two <invoke> elements, representing, respectively, the activities crtPO and apprPO; an <assign> element; a <flow> element; and another <invoke> element representing the activity apprPay.

The <flow> element contains three activities: signGRN, ctrsignGRN and crtPay. The activities in the <flow> element are concurrently executed, except for signGRN and ctrsignGRN since the execution of signGRN must precede the one of ctrsignGRN. The execution dependency between the activities signGRN and ctrsignGRN is expressed using the <links> element: signGRN is the source activity of the links, while ctrsignGRN is the target activity.

The <while> element is used to specify that the signGRN activity should be performed twice. The value of CT attribute is set to 0 by the <assign> activity before the <flow> element. It is incremented by 1 following each execution of the signGRN activity, as specified in the second <assign> element.

```
<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"/>

  <variables>
    <variable name="PO" messageType="POMessage"/>
    <variable name="GRN" messageType="GRNMessage"/>
    <variable name="PF" messageType="PFMessage"/>
    <variable name="CT" type="xsd:integer"/>
  </variables>

  <sequence>
    <invoke partnerLink="" portType="crtPOPT"
            operation="crtPO" outputVariable="PO"/>
     <invoke partnerLink="" portType="approvePOPT"
             operation="apprPO" inputVariable="PO"/>

    <assign>
      <copy><from expression="0"/><to variable="CT"/></copy>
    </assign>
    <flow>
      <links>
        <link name="multipleexecsignGRN-to-ctrsignGRN"/>
      </links>
      <while condition="CT < 2">
        <sequence>
          <invoke partnerLink="" portType="signGRNPT"
                  operation="signGRN" inputVariable="GRN"/>
          <assign>
            <copy>
              <from expression="bpws:getVariableData('CT') + 1"/>
              <to variable="CT"/>
            </copy>
          </assign>
        </sequence>
        <source linkName="multipleexecsignGRN-to-ctrsignGRN"/>
      </while>
      <invoke partnerLink="" portType="ctrsignGRNPT"
              operation="ctrsignGRN" inputVariable="GRN">
        <target linkName="multipleexecsignGRN-to-ctrsignGRN"/>
      </invoke>
      <invoke partnerLink="" portType="ctrPayPT"
              operation="ctrPay" outputVariable="PF"/>
    </flow>
    <invoke partnerLink="" portType="apprPayPT"
            operation="apprPay" inputVariable="PF"/>
  </sequence>
</process>
```

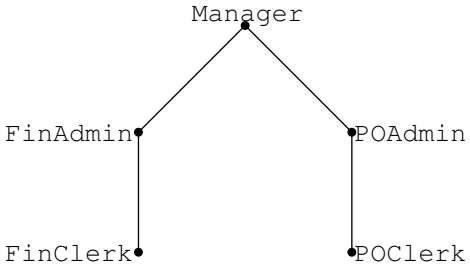
Figure 2. The purchase order business process expressed in WS-BPEL

3. RBAC-WS-BPEL – A Language for Business Processes with Constrained Authorization

A WS-BPEL process is a representation of an organizational or 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 [13], and those determined by security requirements. WS-BPEL addresses the first of these categories.

In this paper we deal with the second category and we focus on developing authorization extensions to WS-BPEL. The proposed extensions include the specification of authorization information and authorization constraints. Authorization information associates activities with authorized users and enables a reference monitor to reach a decision about the legitimacy of a user request to execute an activity. Authorization constraints include separation of duty requirements, where two different users must execute two different activities, and binding of duty constraints, in which the same user is required to perform two different activities. In what follows, we formally define the main components of the RBAC-WS-BPEL language.

These components support the specification of a business process and the specification and enforcement of an RBAC authorization policy for the activities composing



(a) The role hierarchy

P	
P_1	(crtPO, $execute$)
P_2	(apprPO, $execute$)
P_3	(signGRN, $execute$)
P_4	(ctrsignGRN, $execute$)
P_5	(crtPay, $execute$)
P_6	(apprPay, $execute$)

(b) Permissions

RA	
FinClerk	P_5
FinAdmin	P_3
FinAdmin	P_6
POClerk	P_1
POClerk	P_3
POAdmin	P_2
POAdmin	P_4

(c) Role-permission assignment

C_1	$(U, \texttt{crtPO}, \texttt{signGRN}, =)$
C_2	$(U, \texttt{signGRN}, \texttt{ctrsignGRN}, \neq)$
C_3	$(U, \texttt{crtPO}, \texttt{crtPay}, \neq)$
C_4	$(R, \texttt{crtPO}, \texttt{apprPO}, <)$
C_5	$(R, \texttt{crtPay}, \texttt{apprPay}, <)$

(d) Authorization constraints

Figure 3. RBAC-WS-BPEL authorization schema and authorization constraints for the purchase order process

the business process. First, we provide the formal definition of *RBAC-WS-BPEL permission*, defining an action that can be performed on an activity in a business process and the definition of *RBAC-WS-BPEL role hierarchy* specifying the partial order relation between the roles associated with a business process; then, we introduce the definition of *RBAC-WS-BPEL authorization schema* collecting all the authorization information related to a business process. After that, we introduce the definition of *RBAC-WS-BPEL authorization constraints*, that place restrictions on the role and user assignment to activities in a business process. Finally, we define an *RBAC-WS-BPEL authorization specification*, which incorporates a WS-BPEL business process, an *RBAC-WS-BPEL authorization schema* and a set of *RBAC-WS-BPEL authorization constraints*.

Definition 1 (**RBAC-WS-BPEL permission**) Let BP be a WS-BPEL business process. An *RBAC-WS-BPEL permission* is a tuple $(A_i, Action)$ where A_i is the identifier of an activity in BP and $Action$ identifies the type of action can be performed on activity A_i.

To render our specification open to future extensions, we do not specify which are the types of actions that can be performed on an activity. In the following examples, we will consider the type of action `execute`, that allows a subject to carry out an activity of the business process.

Definition 2 (**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 $r, r' \in R$ and $r < r'$, then we say r' *dominates* r.

Definition 3 (**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) where R is a partially ordered set of roles associated with BP, P is the set of permissions defined for the activities in BP and $RA \subseteq R \times P$ is a role-permission 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.

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 4 (**RBAC-WS-BPEL authorization constraints**) Let U be a set of users and let be R 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/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 $antecedent\ activity$). Using this formalization for representing authorization constraints, $(D, (A_1, A_2), \neq)$ defines a separation of duty constraint and $(D, (A_1, A_2), =)$ defines 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 could include "`belongs-to-same-department-as`" or "`is-line-manager-of`".

Example 2 *Figure 3 illustrates the various components of an RBAC-WS-BPEL authorization schema and the authorization constraints for the purchase order example. Figure 3(a) illustrates the role hierarchy and defines* 5 *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`*. The set of permissions comprises the ability to execute each of the activities in the purchase order process and is shown in Figure 3(b). Figure 3(c) 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* `crtPay` *activity. Finally, Figure 3(d) shows how the authorization constraints associated to the purchase order process are represented according to Definition 4.* C_1 *is a binding of duty constraint, requiring that the user that creates a purchase order must sign for the goods.* C_2 *and* C_3 *are separation of duty constraints.* C_2 *imposes that the user that countersigns the GRN must be different from the user that signed the GRN, while* C_3 *assesses that the user that creates the purchase order cannot create the payment for the goods. Finally,* C_4 *and* C_5 *are seniority constraints.* C_4 *states that the role that approves a purchase order must be more senior than the role that creates it, while* C_5 *requires that the role that approves the payment must be more senior than the role that creates it.*

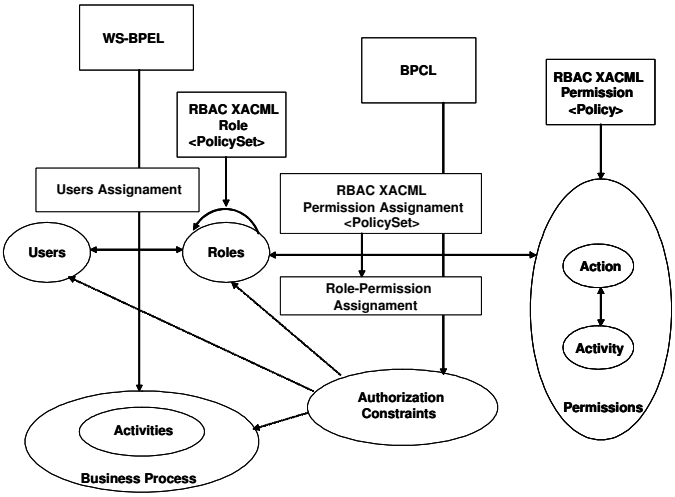


Figure 4. RBAC-WS-BPEL component representation

We can now formally introduce the notion of RBAC-WS-BPEL authorization specification that combines all the previous notions.

Definition 5 (**RBAC-WS-BPEL authorization specification**) A *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.

In the following sections we illustrate how the RBAC-WS-BPEL authorization schema and RBAC-WS-BPEL authorization constraints are incorporated into a WS-BPEL process definition. In Section 4 we describe the representation of RBAC-WS-BPEL authorization schema using XACML. In Section 5, we describe a new XML schema we have developed for specifying authorization constraints in business processes, which we call BPCL (business process constraint language). Figure 4 illustrates the components of the RBAC-WS-BPEL language and the relations existing between them.

4. RBAC XACML

The first extension we propose to the WS-BPEL language 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 [8] recently proposed as an alternative to the RBAC profile for XACML [10]. Figure 5 shows how the RBAC-WS-BPEL

authorization schema reported in Figure 3 can be encoded in XACML.

```
<!-- Role set -->
<PolicySet ... PolicySetId="set:roles" ... >
 <PolicySet ... PolicySetId="role:Manager" ... >
  <Target>Any user with role attribute = "Manager"
  </Target>
  <PolicySetIdReference>permissions:Manager
  </PolicySetIdReference>
  <PolicySetIdReference>role:FinAdmin
  </PolicySetIdReference>
  <PolicySetIdReference>role:POAdmin
  </PolicySetIdReference>
 </PolicySet>
 <PolicySet ... PolicySetId="role:FinAdmin" ... >
    <Target>Any user with role attribute = "FinAdmin"
    </Target>
  <PolicySetIdReference>permissions:FinAdmin
  </PolicySetIdReference>
  <PolicySetIdReference>role:FinClerk
  </PolicySetIdReference>
 </PolicySet>
  <PolicySet ... PolicySetId="role:FinClerk" ... >
    <Target>Any user with role attribute = "FinClerk"
    </Target>
   <PolicySetIdReference>permissions:FinClerk
   </PolicySetIdReference>
  </PolicySet>
    .
    .
    .
</PolicySet>

<!-- Role-permission assignment relation -->
<PolicySet ... PolicySetId="relation:ra" ... >
  <PolicySet ... PolicySetId="permissions:FinAdmin" ... >
    <PolicySetIdReference>permission:pay:approve
    </PolicySetIdReference>
    <PolicySetIdReference>permission:sign:grn
    </PolicySetIdReference>
 </PolicySet>
    .
    .
    .
</PolicySet>

<!-- Permission set -->
<PolicySet ... PolicySetId="set:permissions" ... >
  <Target>Any subject, any resource, any action
   </Target>
  <Policy ... PolicyId="permission:po:create" ... >
    <Rule ... >
      <Target> execute create purchase order activity
      </Target>
    </Rule>
  </Policy>
    .
    .
    .
</PolicySet>
```

Figure 5. RBAC-WS-BPEL authorization schema expressed in pseudo-XACML

The authorization policy uses three different kinds of XACML policies, each one represented by a `<PolicySet>` element. The set P of permissions associated with a WS-BPEL business process is represented by a `Permission <PolicySet>` containing a `Permission <Policy>` element for each permission in P. The RA role-permission assignment relation is represented by a `PermissionAssignment <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 `<PolicySetId>` attribute equal to `role:Manager`. The `<Target>` subelement limits the applicability of the `Role <PolicySet>` to users holding the associated role attribute and value. `<PolicySetIdReference>` subelements are used to refer to the `PermissionAssignment <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`.

5. BCPL – Business Process Constraint Language

Now we introduce the second extension to WS-BPEL, that is, an XML-based language for the specification of authorization constraints such as separation of duty and binding of duty. We call this language BPCL (business process constraint language).

In BPCL, an `<AuthorizationConstraints>` element contains all the authorization constraints that apply to the activities in a WS-BPEL business process. Each constraint $C \equiv (D, (A_1, A_2), \rho)$ 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 subelements, `<Type>` and `<Subject>`. The `<Type>` data content specifies the type of the constraint C: it contains the value "role" if C is a role authorization constraint, and the value "user" if C is a user authorization constraint. The contents of `<Subject>` element will either be a set of roles or a set of users and will depend on the contents of the `<Type>` element.

The `<Activities>` element specifies the two activities A_1 and A_2 to which the constraint C is applied. In particular, `<Activities>` has two child nodes, `<AntecedentActivityReference>`

and <ConsequentActivityReference>, containing, respectively, an XLink reference to the XML element representing activities A_1 and A_2 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 $\neq$.

```
<AuthorizationConstraints>
  <Constraint Id="C1">
    <Domain>
      <Type>User</Type>
      <Subject/>
    </Domain>
    <Activities>
     <AntecedentActivityReference xlink:type="simple"
        xlink:href="..............................
              #xpointer(//*[@operation="crtPO"])"/>
     <ConsequentActivityReference xlink:type="simple"
        xlink:href="..............................
              #xpointer(//*[@operation="signGRN"])"/>
    </Activities>
    <Predicate>equal</Predicate>
  </Constraint>
  ⋮
  <Constraint Id="C5">
    <Domain>
      <Type>Role</Type>
      <Subject/>
    </Domain>
    <Activities>
     <AntecedentActivityReference xlink:type="simple"
        xlink:href="..........................
              #xpointer(//*[@operation="crtPay"])"/>
     <ConsequentActivityReference xlink:type="simple"
        xlink:href="..........................
              #xpointer(//*[@operation="apprPay"])"/>
    </Activities>
    <Predicate>seniority</Predicate>
  </Constraint>
</AuthorizationConstraints>
```

Figure 6. BPCL representation of the constraints in Table 6

Figure 6 illustrates the use of BPCL in defining the authorization constraints associated with the purchase order process. For example, the constraint $(R, \texttt{crtPay}, \texttt{apprPay}, <)$ is represented by the <Constraint> element with Id attribute "C5". Notice that the <Subject> can be empty, as in both C1 and C5, in which case all elements in the appropriate domain are considered. In C5, the <AntecedentAtivityReference> 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. Further, since $(R, \texttt{crtPay}, \texttt{apprPay}, <)$ is a role authorization constraint, the <Type> data content element is equal to "role".

6. Illustrating RBAC-WS-BPEL authorization specification

In this section we illustrate how our extensions are incorporated into the purchase order process specification introduced in Section 2. Figure 2, represents the business logic of the process. In Figure 7, in accordance with Definition 5, we include (references to) the authorization information necessary to state which roles or users are allowed to execute the activities, and the authorization constraints that apply to the activities in the process.

Notice that the specification of the business logic remains unchanged. We believe that it is important that our authorization extensions should not require significant changes to WS-BPEL syntax. Accordingly, we only add two new child nodes to the <process> element: <authorization_information_reference> and <authorization_constraints_reference>. The <authorization_information_reference> element contains an XLink reference to the XML document defining the RBAC policy associated with the purchase order process. Again, note the use of XACML, an existing standard, to articulate the RBAC policy, rather than the definition of another XML-based policy language. The <authorization_constraints_reference> element contains a XLink reference to the BPCL representation of the authorization constraints applied to the activities in the business process order.

```
<process name="purchaseOrderProcess" ... >

  <authorization_information_reference xlink:type="simple"
               xlink:href="role_hierarchy.xml"/>
  <authorization_constraints_reference xlink:type="simple"
              xlink:href="authorization_constraints.xml"/>

  <variables>
    <variable name="PO" messageType="POMessage"/>
    <variable name="GRN" messageType="GRNMessage"/>
    <variable name="PF" messageType="PFMessage"/>
    <variable name="CT" type="xsd:integer"/>
  </variables>

  <sequence>
    <invoke partnerLink="" portType="crtPOPT"
            operation="crtPO" outputVariable="PO">
    </invoke>
     ⋮
  </sequence>
</process>
```

Figure 7. Incorporating authorization information into WS-BPEL

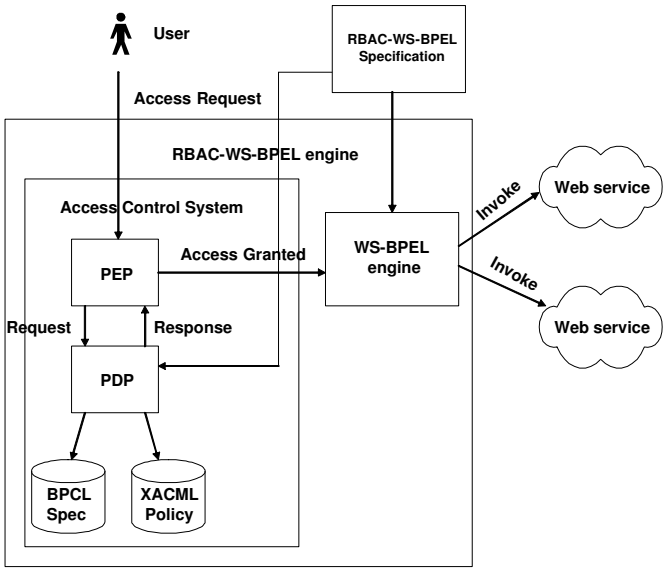


Figure 8. RBAC-WS-BPEL architecture

As we can see from the RBAC-WS-BPEL specification of the purchase order, our approach for associating authorization information and authorization constraints with a business process is characterized by some interesting features. First, the specification of authorization information and authorization constraints in the WS-BPEL specification 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, which provide references to the authorization information and the authorization constraints. 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 it only needs to modify the references to them. 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, where that restriction can be expressed as a binary relation on the set of users or roles.

7. RBAC-WS-BPEL system architecture

In this section, we briefly discuss the RBAC-WS-BPEL engine, which manages the execution of business processes subject to the authorization policy and constraints. The engine extends a conventional WS-BPEL engine with an *access control* system. The architecture of the engine is shown in Figure 8.

The WS-BPEL engine is responsible for scheduling and synchronizing the various activities within the business process, in accordance with specified activity dependencies, and for invoking web service operations associated with activities. The engine receives the RBAC-WS-BPEL process specification as input and creates a representation of WS-BPEL process.

The access control system, according to the XACML paradigm, is composed of a PEP (Policy Enforcement Point) and a PDP (Policy Decision Point). When a subject S_j submits a request to perform an activity A_i, the PEP traps the request and creates an XACML request based on the subject's attributes, the requested activity, the type of action, and other information pertaining to the request. Then, it forwards it to the PDP. The PDP, having received the request, retrieves the XACML authorization schema and BCPL constraints referenced in the RBAC-WS-BPEL process specification, and decides if the subject is authorized or not.

In order to make a decision, the PDP must ensure that granting the request does not violate the authorization constraints and does not prevent the business process from completing. The strategy is to initialize the set of users that are authorized to perform activity A_i with S_j and to compute for each pair of activities (A_k, A_m) in the RBAC-WS-BPEL specification, the set of subjects that can execute A_k and A_m applying all possible BCPL constraints defined for A_k and A_m, including those derived from XACML authorization schema. If one of these sets is empty, S_j cannot execute activity A_i, since it means that for one pair of activities, there

does not exist a pair of authorized users that comply with the constraints. Hence, there does not exist a valid user assignment ensuring the completion of the business process.

Having made a decision, the PDP transmits it to the PEP. If the access is granted, the PEP communicates the decision to the WS-BPEL engine, which invokes the web service operation associated with the activity A_i.

One of the advantages of the proposed architecture is that is modular and allows an easy integration of the access control system with existing WS-BPEL engines. A second advantage is that the decision-making algorithm implemented by the PDP does not require any binding of roles or users to activities before process execution, unlike previous approaches [3]. Moreover, the algorithm runs in time polynomial in the number of activities and users [7](rather than exponential [3]). Further, this strategy allows the PDP to take a decision in a time polynomial in the number of activities and users.

8. Conclusions

In this paper we have proposed RBAC-WS-BPEL, an extended version of the WS-BPEL language, that supports the specification of business processes and the specification of authorization information necessary to state if a role or a user is allowed to execute the 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 which 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 who can perform the activities in the business process. Further, we have illustrated how these components can be included in the business process specification. Finally, we have proposed the RBAC-WS-BPEL engine, which manages business process execution respecting authorization constraints. 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. In particular, it is possible for the same business process specification to have different authorization information: different organizations may define different roles and different assignment of activities to roles. Moreover, different organizations may have different security policies and controls, which require the articulation of different authorization constraints. A further advantage is the rich expressiveness of the BPCL language, which enables us to articulate 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 constraints. Currently, 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 on the execution of several antecedent activities. Another direction on which we will focus is the development of more sophisticated algorithms for the assignment of users and roles to the activities that satisfies authorization constraints. A final possibility for future work would be to consider the use of BPCL as a general authorization constraint language and how it could inter operate with XML-based authorization languages such as XACML.

9. Acknowledgments

The work of Elisa Bertino is partially supported by the US National Science Foundation under Grant No. 0430274 and by the sponsors of CERIAS. The work of Federica Paci is partly funded by the European Commission under the Contract 001945, Integrated Project TRUSTCOM.

References

[1] G.-J. Ahn, R. Sandhu, M. Kang, and J. Park. Injecting RBAC to secure a web-based workflow system. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, pages 1–10, 2000.

[2] V. Atluri and W. Huang. An authorization model for workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security*, pages 44–64, 1996.

[3] 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*, 2(1):65–104, 1999.

[4] R. Botha and J. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.

[5] F. Casati, S. Castano, and M. Fugini. Managing workflow authorization constraints through active database technology. *Information Systems Frontiers*, 3(3):319–338, 2001. Also available as Technical Report HPL-2000-156, Hewlett Packard Laboratories.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language 1.1*. W3C, 2001. Available at `http://www.w3.org/TR/wsdl`.

[7] J. Crampton. A reference monitor for workflow systems with constrained task execution. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, 2005.

[8] J. Crampton. XACML and role-based access control, 2005. Presentation at *DIMACS Workshop on Security of Web Services and e-Commerce*.

[9] F. Leymann. *Web services flow language (WSFL 1.0)*. IBM Software Group, 2001. Available at

http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[10] OASIS. *Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML, Version 2.0*, 2004. OASIS Committee Draft (A. Anderson, editor). Available at http://www.oasis-open.org/committees/documents.php?wg_abbrev=xacml.

[11] OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0*, 2005. OASIS Committee Specification (T. Moses, editor). Available from http://www.oasis-open.org/committees/documents.php?wg_abbrev=xacml.

[12] OASIS. *Web Services Business Process Execution Language Version 2.0*, 2005. OASIS Committee Working Draft. Available at http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel.

[13] M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In *Modern Database Systems: The Object Model, Interoperability, and Beyond*, pages 592–620. Addison-Wesley, 1995.

[14] S. Thatte. *XLANG – Web Services for Business Process Design*. Microsoft Corporation, 2001. Available at http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

[15] W3C. *XML Path Language (XPath) Version 1.0*, 1999. W3C Recommendation (J. Clark and S. DeRose, editors). Available at http://www.w3.org/TR/xpath.

[16] W3C. *XML Schema*, 2004. Available at http://www.w3.org/XML/Schema.

[17] J. Wainer, P. Barthelmess, and A. Kumar. W-RBAC – A workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems*, 12(4):455–486, 2003.