User Tasks and Access Control over Web Services

Jacques Thomas*, Federica Paci**, Elisa Bertino*, and Patrick Eugster*

*Purdue University, CERIAS
**Universita degli Studi di Milano

Abstract

Web services are a successful technology for enterprise information management, where they are used to expose legacy applications on the corporate intranet or in business-to-business scenarios. The technologies used to expose applications as web services have matured, stabilized, and are defined as W3C standards. Now, the technology used to build applications based on web services, a process known as orchestration*, is also maturing around the Web Services Business Process Execution Language (WS-BPEL). WS-BPEL falls short on one feature though: as it is focused on orchestration of fully automatic web-services, WS-BPEL does not provide means for specifying human interactions, even less their access-control requirements. Human interactions are nonetheless needed for flexible business processes. This lacking feature of WS-BPEL has been highlighted in a white paper issued jointly by IBM and SAP, which "describes scenarios where users are involved in business processes, and defines appropriate extensions to WS-BPEL to address these." These extensions, called BPEL4People, are well explained, but their implementation isn't. In this paper, we propose a language for specifying these extensions, as well as an architecture to support them. The salient advantage of our architecture is that it allows for the reuse of existing BPEL engines. In addition, our language allows for specifying these extensions within the main BPEL script, hence preserving a global view of the process. We illustrate our extensions by revisiting the classic loan approval BPEL example.*

1. Introduction

Web Services Business Process Execution Language (WS-BPEL, BPEL for short) [12] is a language to specify the *orchestration* of web services. A BPEL specification defines how a *business process* can be created by composing web services. A BPEL process specification thus contains the following elements: (1) declaration of the web services that will be orchestrated, (2) declaration of the control flow between the invocation of the different web services, (3) declaration of the variables used to maintain the state of the business process, and (4) declaration of the data flow.

BPEL is an orchestration language well suited for automated business processes. Many business processes, however, require human interactions. Human interaction can be required either to provide more flexibility, or because the very nature of the activity to be performed requires human interaction; this is where BPEL falls short. Let us consider the loan approval example process from the BPEL specification (fig. 1) to illustrate this claim. The example deals with a process that could be used in a bank to process loan applications. When the requested amount is superior to $10 000 or if the risk associated with the loan (as returned, after processing, by the loan assessment web service) is high, then the process calls a loan approval web service. Clearly, this approval can require human interaction, but there is no provision in the BPEL standard to specify this interaction.

BPEL4People [14] characterizes scenarios that can not be specified with BPEL and makes a compelling case for the need to extend BPEL with support for specifying human activities. But BPEL4People is still only a functional specification. The white paper mostly describes *which* interactions should be supported, not *how* they should be supported or their access be controlled.

Currently, there are three main approaches for supporting human interactions with web services: server-side, middleware, and client-side. The server-side approach consists in extending a BPEL engine, to have it support BPEL4People; this is the path taken by Intalio with the release of Intalio|Tempo [13], which is a BPEL engine

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

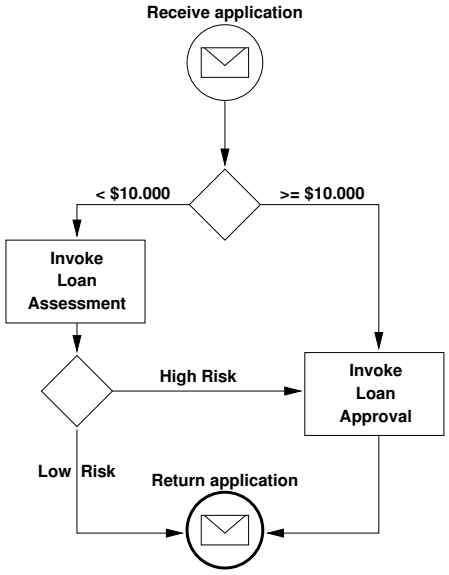

Figure 1. The BPEL 1.1 loan approval example

extended to support BPEL4People (although there is still no specification of BPEL4People besides the white paper). The middleware approach was taken by Oracle with the TaskManager web service [22]. The TaskManager can be viewed as a request queue, where the BPEL process can insert tasks that have to be performed by humans. In the client-side approach, the user interface becomes a full-fledged software running on the computer of the user and it exposes a web service that can be used to push tasks directly to the user. MagooClient, from MagooSoftware, is an example of this approach [18].

The server-side approach has the following advantages. The definition of the user tasks is tightly integrated in the definition of the business process, the extended BPEL engine can enforce access control on the tasks, and no additional software is required. At the same time, replacing the BPEL engine would void the time previously invested in integrating it in the IT infrastructure, together with introducing new bugs that always come with new software.

Both the middleware approach and the client-side approach have the advantage of overcoming the limitations of an existing BPEL server, without requiring its replacement. Such an approach however has the disadvantage that the logic of human tasks is now handled by another entity, hence loosing the global view on the process logic. In addition, the client-side approach runs into the problem of having to schedule users to performs tasks without violating separation of duty constraints, which has proven to be a hard problem [6, 17].

In this paper, we propose a solution that combines the benefits of all the three approaches mentioned above:

- We propose an architecture to extend the functionalities of a standard BPEL engine without modifying it
- We describe a language to specify these functionalities: user tasks and their access control

The language that we propose has a deliberate strong focus on the access control aspect of a business process, as new regulations like the Sarbanes Oxley act of 2002 [27] moved the need for high assurance from the military and governmental space into the corporate world. We designed this language to offer a rich support for modeling exceptions to access control rules, as exceptions are needed to properly reflect business practices, e.g. that a hierarchical superior can override a decision from one of his subordinates. At the same time, we decided against integrating any of the corresponding rules *a priori* in the system, to preserve its flexibility.

In order to extend the capabilities of the BPEL engine rather than replacing it, the handling of user tasks is delegated to a task manager web service. To keep the entire specification of the process contained in the BPEL script, the language extensions that describe a user task are inserted as annotations of a regular `invoke` BPEL activity, which we "compile" down to pure BPEL syntax using an XSLT transformation [32]. We present these annotations and illustrate their usage by revisiting the classic loan approval example from the BPEL specification [12], where we replace the automatic loan approval web service by user tasks.

The rest of this paper is organized as follows: architecture of the system and language integration in section 2; language extensions in section 3; structure and runtime interactions of the task manager in section 4; compilation of the extended language into pure BPEL in section 5; related work in section 6. Section 7 concludes our paper and includes a presentation of the related work.

2. Architecture

In this section, we describe the design of our extensions to BPEL. Namely, we first present the architecture used to support BPEL processes extended with user tasks. Then we describe how we add our extensions into the BPEL language.

2.1. BPEL4People as a Service

In our system, people activities are supported by a web service –the people activity manager–, which offers services

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

similar to Oracle's TaskManager [22], except that it *also* enforces access control based on the business process specification, hence hoisting this responsibility from the process-specific code. The policy decision point as well as the enforcement point are thus provided as a service offered by the infrastructure. Factoring out the enforcement mechanism guarantees a *consistent* and *systematic* enforcement of the policy, as opposed to relying on the application developer to insert the proper security controls in the application code.

We therefore propose an architecture based on the following components (fig. 2):

- an unmodified BPEL engine, which runs the BPEL process.
- the people activity manager, which is responsible for providing an interface for users to perform tasks, managing the user's tasklists, and enforcing the security policy. The people activity manager is described further in section 4.
- an identity management infrastructure, which provides user attributes to the people activity manager, so that it can make access-control decisions based on these attributes. For instance, the group membership information could be used for deciding whether to let a user perform a task.

2.2. Language Integration

The last integration design choice concerns the specification language. We want to avoid modifying the BPEL specification language, so that existing BPEL engines can be used to run our system. Fortunately, the BPEL XML Schema definition was designed in an extensible fashion: the root `tExtensibleElement` is extensible, as the name implies. The most interesting parts in the schema definition of `tExtensibleElement` are the following:

```
<sequence>
    <any namespace="##other" minOccurs="0"
     maxOccurs="unbounded" processContents="lax"/>
</sequence>
<anyAttribute namespace="##other"
              processContents="lax"/>
```

The meaning of this declaration is that any number of attributes from any namespace can be placed inside any element that extends `tExtensibleElement`. For our purpose, we want to be able to extend the `invoke` activity. Since the type of an `invoke` activity (`tInvoke`) extends the type of an activity (`tActivity`), which itself extends `tExtensibleElement`, the net result is that we can insert our extensions directly into a BPEL specification, provided they are defined in another namespace than BPEL.

Moreover, the `processContents="lax"` option indicates that the engine does not need to process extensions that it does not know of. This enables us to add our extensions directly inside the `invoke` activity specification and at the same time still comply with the BPEL XML schema definition.

Inserted as such, these activities would be ignored by a standard BPEL engine. We thus describe in section 5.2 how we use XSLT to automatically compile an extended BPEL process definition, where the extensions are ignored, into a BPEL process that delegates all the people activities to the people activity manager.

3. Extensions to the BPEL Language

In this section we describe the extensions to BPEL that we propose in order to specify people activities. These activities are modeled as editing tasks. We believe that this model encompasses all the cases of human interaction with a web service (actually with any computer program, for that matter).

In order to specify people activities, we therefore have to specify which values will be edited by the task (the business logic aspect of the task) and, since we are interested in access control, the access control requirements that the user must meet to be allowed to perform the task (the access control aspect of the task). The extensions for task specification fall in two categories: the business logic extensions, which specify the data edition performed by the task, and the access control extensions, which specify the conditions that a user must meet to perform the task.

Usually, some of the conditions that a user must meet are criterias pertaining to his digital identity. The domain of digital identity management is already well developed, with both models and technologies. Rather than redefining and then reimplementing existing solutions, our system aims at integrating with them.

The first step in extending a BPEL process with a user task is to add an invocation to the task manager. With the loan approval example (fig. 1), this leads to replacing the invocation of the loanApproval web service, by an invocation of the task manager, which entails replacing the original `invoke`:

```
<invoke partnerLink="approver"
        portType="lns:loanApprovalPT"
        operation="approve" ../.. >
    ../..
</invoke>
```

by the following `invoke`:

```
<invoke partnerLink="taskManager"
        portType="tm:TaskRequestPT"
        operation="taskRequest" ../.. >
    ../..
</invoke>
```

Below, we present our extensions and illustrate them by completing this extension of the loan approval example.

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

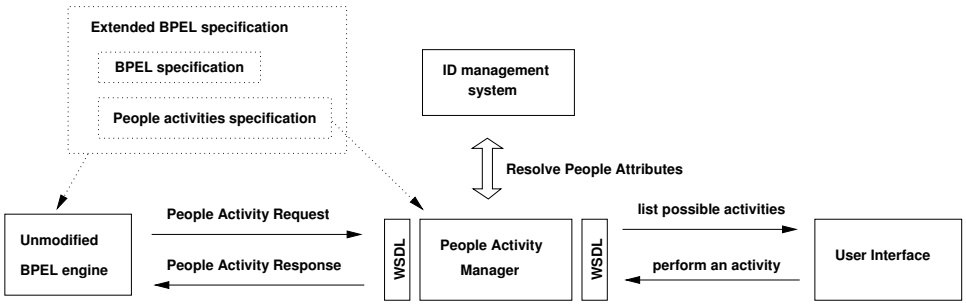

Figure 2. System Architecture

3.1. Business Logic Extensions

These extensions specify which data will be exchanged between the business process and the user task: they describe the variables that will be visible inside the form that will be presented to the user. For each variable, the following information is specified:

Variable name: this should be the identifier of a variable that is visible from the lexical scope where the task is defined in the process.

Interaction mode: input, output, input/output. This is very similar to the directional attribute for parameters of an operation when declaring an interface in CORBA's interface description language. More precisely:

- input: the data is provided to the task as a parameter; modifications to this variable are not propagated back to the process.
- output: this data will be produced by the task.
- input/output: this data will be edited by the task.

From the user perspective, input, output and input/output modes are respectively equivalent to read, write, and read/write modes.

Name mapping: this allows for renaming the variable for the duration of the task. This way, the user can be presented with variable names that, in the context of the task, are more meaningful than the names used inside the business process definition.

Default value: for editing tasks, it is often useful to have fields be pre-filled with a value. Variables from the process can be used as default values, using the facilities already provided in the BPEL specification.

We specify business logic extensions using the following syntax:

```
<variables>
    <var name="ncname" mapsto="ncname"? mode="i|o|io"
        default="XPathQueryString" />
</variables>
```

For our example, the fact that a user task is used to decide the acceptance or rejection of a loan application can be expressed as follows:

```
<invoke ../.. >
    <peopleAspect>
        <variables>
            <var name="request" mode="i" />
            <var name="approval" mode="o" />
        </variables>
    </peopleAspect>
</invoke>
```

3.2. Access Control Extensions

The access-control extensions are composed of two parts: the authorizations describe the set of users allowed to perform a task, while the constraints describe how these permissions should be filtered. The filtering can be used to enforce separation of duty (e.g. "a user can not perform the second approval if he already performed the first one") or content-based access-control (e.g. "this clerk can only manipulate checks below a certain amount").

3.2.1. Authorizations

Authorizations can be based either on the identity of the user requesting the access, referred to as IBAC, the role of the user, or other attributes of the user. Our work so far only considers role based access control (RBAC) and IBAC. It can, however, be easily extended to support attribute-based access control, building on the methods used to provide RBAC support. The syntax for specifying authorizations is given in fig. 3.

The simplest way to specify these authorizations is by statically specifying the user allowed to perform a task; this can be done using the `<constant>` element. A more flexible way to specify the groups of authorized users is to declare them as queries that are run against an organizational

```
<authorizations>
    Authorization+
</authorizations>

Authorization ::= BaseAuthorization
      | <or> Authorization Authorization </or>
      | <and> Authorization Authorization </and>
      | <not> Authorization </not>

BaseAuthorization ::= <id> AuthorizationSpec </id>
      | <role> AuthorizationSpec </role>

AuthorizationSpec ::=
       <constant> <!— identifier —> </constant>
      | <variable> <!— XPath query —> </variable>
      | <query> <!— LDAP filter string —> </query>
```

Figure 3. Syntax for authorizations

```
<invoke  ../.. >
    <peopleAspect>
        <variables> ../.. </variables>
        <authorizations>
            <or>
                <role> <constant> "loanRiskManager"
                       </constant> </role>
                <role> <constant> "branchManager"
                       </constant> </role>
            </or>
        </authorizations>
    </peopleAspect>
</invoke>
```

Figure 4. Only users with the role "loanRiskManager" or "branchManager" can perform this task

directory, like an LDAP directory [16]; this can be achieved by using the <query> element. In BPEL4People, performing a query against an organizational directory is called *People Resolution*; the query itself is called a *People Link*. This runtime evaluation, rather than a static definition in the business process, allows changes to the groups of authorized users without the need to modify the definition of the business process. Finally, one more way to specify authorizations is by referring to a variable of the business process, using the <variable> element around an XPath query that retrieves the variable. Referring to process variables is used to support delegation and escalation, where tasks have to be re-assigned from within the business process.

This syntax allows to specify both constant terms, for cases where the authorizations are static, and queries, so that the authorizations can be more dynamic by being the result of a query against an organizational directory. For the queries, we chose to use LDAP filter strings [16].

Fig. 4 shows an example of how to express that only loan risk managers or local branch managers can approve a loan.

3.2.2. Authorization constraints

Authorization constraints can be based on the execution history, in which case they are either Separation of Duty con-

```
<constraints>
     Constraint+
</constraints>

Constraint ::=  BaseConstraint
     |  <or> Constraint Constraint </or>
     |  <and> Constraint Constraint </and>
     |  <not> Constraint </not>

Baseconstraint ::= HistoryConstraint
     | ContentConstraint
     | IDConstraint
     | True
     | False

HistoryConstraint ::=
    <happened>
        <task>
            <name>
                <!— task name —>
            </name>
            <performed_by>
                (<sameuser/> | <samerole/>)
            </performed_by>
            <outcome>
                (<success/>|<failure/>)
            </outcome>
        </task>
    </happened>

ContentConstraint ::= <content constraint="queryString"/>

IDConstraint ::= Authorization
```

Figure 5. Syntax for authorization constraints

straints (SoD) [24] or Binding of Duty constraints (BoD) [11]. SoD constraints express the fact that a set of tasks have to be executed by different users; BoD constraints express the dual: a set of tasks that must be executed by the same user. Authorization constraints can also be based on other aspects of the activity: the content of the document being edited, the time of the access request, or even the location from where the access is requested. We integrated only the content-based constraints, which fit naturally inside BPEL by formulating them as XPath queries; the time-based or location-based constraints require some more infrastructure that will be the topic of future work. These constraints would, however, be easily integrated in our framework by extending its constraint language. The syntax used to express authorization constraints is given in fig. 5.

For our example, if we want to enforce dual controls (separation of duty) on very large amounts, we will need to chain two human tasks, one for each approval, as illustrated in fig. 6.

3.2.3. Modeling Exceptions

Our language can model exceptions to both the authorizations and authorizations constraint.

Negative Permissions A negative permission [7] specifies that a certain group of individuals should be denied ac-

```
<sequence>
    <invoke  name="approval1" ../.. >
        <peopleAspect>
            <variables> ../.. </variables>
            <authorizations> ../.. </authorizations>
        </peopleAspect>
    </invoke>
    <invoke  name="approval2" ../.. >
        <peopleAspect>
            <variables> ../.. </variables>
            <authorizations> ../.. </authorizations>
            <constraints>
                <not>
                    <happened>
                        <task>
                            <name>"approval1"</name>
                            <performed_by> <sameuser/>
                            </performed_by>
                            <outcome> <success/>
                            </outcome>
                        </task>
                    </happened>
                </not>
            </constraints>
        </peopleAspect>
    </invoke>
</sequence>
```

Figure 6. Example of a dual control for the approval task

cess. As such, a negative permission is an exception to an otherwise permissive policy. If we first specify the target group as an `authorization`, say A_G, then we can negate this `authorization` as in the term $\neg A_G$, hence denying access to members of the group.

Exceptions to the Constraints Our language is also able to model exceptions to the constraints. Suppose we have a constraint C that applies to all users, but we would like to lift it for only some users. Such a case can be modeled as follows. First, an `IDConstraint`, say IDC, is created based on an authorization specification that matches the group of users that should be exempted from the constraint. Then, the rule augmented with the exception can be represented as the disjunction of C and IDC: $C \vee IDC$.

4. The People Activity Manager

In this section, we explain in more detail the central component of our architecture: the people activity manager. We first describe its structure; then we show the steps involved in handling a people activity.

4.1. Structure

The people activity manager serves two purposes in our system. First, it serves a structural purpose. By exposing its interface as a web service, we can use it to extend the functionalities provided on top of a legacy BPEL engine, without extending the engine itself. Second, taking advantage of its placement as a mediator of all interactions between humans and the business process, the people activity manager serves its functional purpose: it is responsible for enforcing the security policy for people activities. In other words, the people activity manager is a reference monitor [2]. For processes that require high availability, the people activity manager would therefore have to be replicated to avoid constituting a single point of failure.

In order to fulfil its functional and structural purposes, the people activity manager comprises the elements illustrated in fig. 7. There are four information repositories, to store the four categories of data that are needed for the people activity manager to function. The *current activities* repository stores the set of activities that are not yet completed; these activities can be freshly submitted ones, activities for which a user claimed ownership (to perform the activity), or revoked activities (when a user decides not to perform a task after claiming ownership on it). The *configuration* repository stores the specification of each type of people activity. As we explained when presenting the specification language in section 3, this specification includes both the business logic and the access control aspects. The *directory server*, which is part of an external identity management system, contains the assignments of users to organizational roles; the people activity manager queries the directory server to verify that the access control specification and the role assignments grant the user the right to perform the task he or she claims. The *history* repository is used to record the history of the actions that were performed on people activities; the people activity manager queries the history repository to enforce the separation of duty constraints.

The people activity manager exposes two interfaces, one that provides services to the BPEL engine (on the left in fig. 7), the other that provides services to the user interface (on the right in fig. 7). Following our discussion of the integration strategies in the introduction, the people activity manager is implemented as a web service, hence the declaration of these interfaces use the web service definition language (WSDL).

4.2. Runtime Interactions

Now we describe the interactions that happen at runtime when a people activity needs to be performed:

- On the BPEL engine
    1. The business process is at a stage where a people activity should be invoked.
    2. The invocation message is prepared by filling it with the values required for a user to perform the task and for the access-control to be performed.

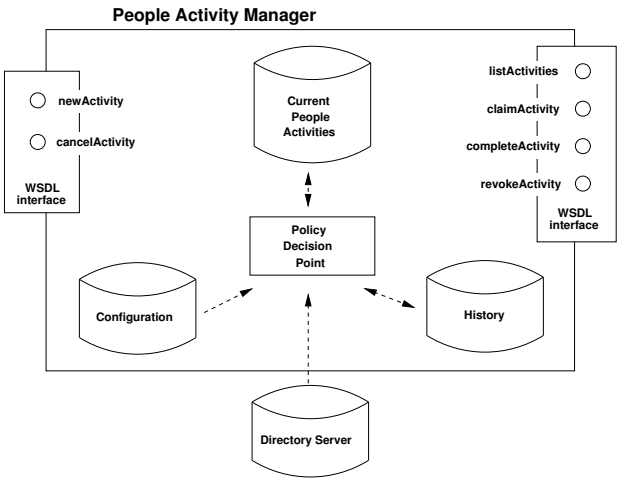

Figure 7. Structure of the People Activity Manager

The format of the message is explained in section 5.1; in section 5.2, we explain how the code that prepares the invocation message is generated from the extended BPEL process.

3. The invocation message is sent to the people activity manager

- The people activity manager does the following, based on the invocation message:
  1. Evaluate the authorizations by querying the organizational directory.
  2. Filter the authorizations based on the authorization constraints. In that case, two types of queries may have to be emitted: queries to the organizational directory and calls to the history database.
  3. For each user that is authorized and that satisfies the constraints, the task is added in the user's task list.
- Through the user interface, a user requests to perform a task:
  1. The people activity manager verifies that the user is still authorized to perform the task. This is needed since the history of the system can change between the time when a task is added to a user's task list and the time when this user requests to perform this task. For instance, this user could have performed in the meantime a task that conflicts (due to an SoD constraint) with the task he is requesting to perform.
  2. The interface implementation retrieves the task parameters from the activity manager, and the user becomes the *owner* of the activity.
  3. The people activity manager removes the task from the task lists of the other users that were authorized to perform it.
  4. Depending on the setting of the variables (`in`, `out`, `inout`), the user can or cannot view and edit the variables he is presented with. Some of these fields may be pre-filled.
  5. Ideally, the user interface enforces the XMLSchema type of the variables (input validation).
  6. The user submits the completed task, which results in the user interface sending back the updated variable values to the task manager.
- The people activity manager performs the following actions, based on the updated values it received from the user interface:
  1. Validate the values with respect to their XMLSchema types.
  2. Verifies the values against the content-based access control constraints (some users may be restricted in the range of values that they can enter for a variable)
  3. If the validation and the verification succeed, the values are passed back to the BPEL engine. Locally, the task is recorded in the history as being successfully completed.
  4. If either the validation or the verification fails, the task is recorded in the history as failed. The people activity manager has to re-evaluate the list of users that can perform the task, and repopulate the task lists. There can be constraints that filter out users that have already failed to successfully perform the task (the number of failed attempts can be a parameter of such a constraint), as suggested by Bertino et al. [6].
- Once the task has either raised a fault or succeeded, the BPEL engine does the following:
  1. If the task raised a fault, the proper fault handler is invoked.
  2. If the task succeeded, the updated values are copied into the state of the business process. The execution of the business process is then resumed.

5. Implementation Over a Legacy BPEL Engine

In this section, we describe how the features from the previous sections can be implemented over a legacy BPEL

```
<message>
   <process_data>
      <!— data needed to perform the task —>
      <var name="ncname">value</var>+
   </process_data>

   <access_control_data>
      <!— data needed to enforce the constraints —>
      <process      name="ncname"/>
      <instanceid   id="ncname"/>
      <taskid       id="ncname"/>
      <var          name="ncname">value</var>+
   </access_control_data>
</message>
```

Figure 8. Message format between the BPEL engine and the people activity manager

engine. Since the support for user tasks is provided as a web service, we decided to overload the `invoke` activity as a way to integrate user tasks in BPEL. This overloading actually requires three parts in the implementation: (1) defining a message format for the invocation of the people activity manager, (2) inserting extra BPEL code in the process definition so that the process interacts nicely with the people activity manager, and (3) implementing the required access control and business logic features inside the people activity manager. The following focuses on the first two parts as they are within our main contributions. Section 6 comes back to the third part.

5.1. Message Format for People Activities Invocation

While running a BPEL process, when the BPEL process engine encounters an `invoke` that is a people activity, it has to send sufficient information to the people activity manager, so that the people activity manager can properly assist in performing the task. The activity manager should receive two sets of variables from the business process: the variables that will be either displayed or edited, and the variables that are needed for the access control (either to perform authorization or enforcement of the authorization constraints). The message format used to transfer this information is illustrated in fig. 8.

We show below how we use XSLT to transform a BPEL specification with our extensions into a pure BPEL specification that runs on a legacy BPEL engine. The weaving process we are about to describe will inject into the BPEL specification the code that is required to generate the invoke messages in such a way that they contain enough information for the people activity manager to properly manage people activities.

```
<invoke inputVariable="peopleMessage"
        outputVariable="peopleMessage">
   <people_aspect>
      <!— task specification —>
   </people_aspect>
</invoke>
```

Figure 9. A people activity, before weaving in the people aspect

```
<sequence>
  <!— message preparation code —>

  <!— original invoke,
   whose extensions will be ignored —>

  <!— message extraction code —>
</sequence>
```

Figure 10. A people activity, after weaving in the people aspect

5.2. Weaving in the People Aspect, using XSLT

As we explained in section 2.2, the extensions we propose for BPEL will be ignored by all BPEL engines that do not support them natively. To solve this problem, we use an XSLT stylesheet to transform the extended process definition into a process definition that relies only on the standard BPEL features. The principle of the transformation is, for each people activity, to gather all the variables of the process that will either be edited or that are needed for access control, include these variables in an invocation message, and then invoke the people activity manager.

People activities are easily identifiable as they contain a `people_aspect` section, which contains the task specification, expressed in the language described in section 3 (cf fig. 9).

The XSLT processing wraps this `invoke` activity into a `sequence` activity. In this `sequence` activity, the `invoke` is surrounded by generated code (fig. 10). The message preparation code populates the invocation message with the process data that the task needs; the message extraction code copies back the values modified by the user.

6. Related Work

Adding user tasks to BPEL processes amounts to retrofitting workflows on top of a Service Oriented Architecture (SOA). The study of workflows [30, 31] is a well established field.

The need to use access control and Separation of Duty (SoD) to guarantee the integrity of information processing was highlighted early on by Saltzer and Schroeder [24];

many other works followed, including the work by Clark and Wilson [10] and Nash and Poland [19], which both discussed the use of SoD to gain assurance in the integrity of the data manipulated by business processes. The work on ADAGE presented in [25] strongly influenced the design of our language. The recent work by Li and Wang [17] contains a thorough overview of SoD research. More recently, there has been a renewed interest in workflows and SoD [3, 6, 11, 5], together with the seminal work from Li and Wang [17] on the complexity of evaluating SoD constraints when they are used for scheduling.

There have been many proposals to add security to web services, starting with the integration of RBAC in a web server [4]: formulating RBAC policies in XML [8] [23]; wrapping access control proxies around a web service [29]; injecting policy-enforcing code in a web service [26]. Finally, the XACML standard [20] offers a generic way to express security policies together with their semantics; the SAML standard [21] offers a generic means of exchanging authentication and authorization information between domains.

As BPEL (and service orchestration in general) is fairly new, there is little related work, both on the access-control issues that arise when a composite process spans several administrative domains [9, 15], and on the support of user tasks [5, 18, 13]. [9] and [15] address issues related to trust-negociation and federated identity, which complement our work. The recent work by Bertino et al. [5] proposes a model for the specification of authorizations and authorization constraints in the context of workflows on top of BPEL, building on [11]. Its implementation would however require replacing the BPEL engine. Besides Oracle's TaskManager [22], two commercial products deal with handling of user tasks for BPEL processes: Magoo Client [18] and Intalio|Tempo [13]. Magoo Client is a user-interface that is reachable as a web service, so that the BPEL process can directly push task assignments to the user. With this approach, all the user scheduling and access control would have to be expressed in BPEL, which does not seem realistic. Intalio|Tempo is supposed to be a full implementation of BPEL4People, with architectural choices similar to ours, but at the time of this writing we were not able to find a version to evaluate.

We used ideas from work on aspect oriented programming for XML [1]. Unlike AspectXML, we chose to include the people aspect directly in the business process definition, in order to maintain a global view of the process within the BPEL script.

7. Conclusions

We presented an architecture that adds support of user tasks to a BPEL engine, together with a language to specify the access-control requirements of these interactions. A distinct advantage of our design choices is that they are compatible with existing BPEL engines.

We recognize that the task manager could be a single point of failure. This perceived limitation, however, can be solved by replicating the task manager and using Web Services Reliable Messaging (WS-RM) [28] to ensure graceful failover from one replica to the other.

We are currently investigating the addition of full end-to-end validation of data types. We are also looking into expressing the authorizations using a standard language. There are two candidate languages: XACML and WS-Policy. It seems that the focus of XACML on access control will fit our purpose better than WS-Policy.

8. Acknowledgements

We would like to thank Laurent Caillette, Marin Markov, Massimo Mecella, and Jan Vitek for valuable feedback and insightful conversations.

References

[1] Aspect XML. `http://www.aspectxml.org`, 2007.

[2] James P. Anderson. Computer security technology planning study. Technical report, Air Force Electronic Systems Division, 1972.

[3] V. Atluri and W. Huang. An authorization model for workflows. In *ESORICS '96*, 1996.

[4] J. F. Barkley, A. V. Cincotta, D. F. Ferraiolo, S. Gavrilla, and D. R. Kuhn. Role based access control for the world wide web. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, 1997.

[5] E. Bertino, J. Crampton, and F. Paci. Access control and authorization constraints for WS-BPEL. In *ICWS'06*, 2006.

[6] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow managem ent systems. *TISSEC*, 2(1), Feb. 1999.

[7] E. Bertino, P. Samarati, and S. Jajodia. An extended authorization model for relational databases. *TKDE*, Volume 9 , Issue 1, 1997.

[8] R. Bhatti, J. B.D. Joshi, E. Bertino, and A. Ghafoor. Access control in dynamic xml-based web-services with x-rbac. In *ICWS'03*, 2003.

[9] P. A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security*, 10(3), 2002.

[10] D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. In *S&P'87*, 1987.

[11] J. Crampton. A reference monitor for workflow systems with constrained task execution. In *SACMAT'05*, 2005.

[12] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. Business process execution language for web services version 1.1, May 2003.

[13] Intalio. Intalio|tempo. http://tempo.intalio.org/, 2007.

[14] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. WS-BPEL extension for people – BPEL4People, 2005.

[15] Hristo Koshutanski and Fabio Masacci. An access control framework for business processes for web services. In *ACM Workshop on XML security*, 2003.

[16] LDAP series of RFCs: RFC 4510 to 4519, June 2006.

[17] N. Li and Q. Wang. Beyond separation of duty: An algebra for specifying high-level security policies. In *CCS'06*, 2006.

[18] Magoo Software Limited. Integrating user tasks into bpel processes. `http://www.magoosoft.com/pdf/BpelUserTasks.pdf`, June 2006.

[19] M. J. Nash and K. R. Poland. Some conundrums concerning separation of duty. In *S&P'90*, 1990.

[20] OASIS. eXtensible Access Control Markup Language, version 2.0, February 2005.

[21] OASIS. Security Assertion Markup Language, version 2.0, March 2005.

[22] Oracle Inc. BPEL tutorial; tutorial 6: Working with the TaskManager service. `http://www.oracle.com/technology/products/ias/bpel/pdf/orabpel-Tutorial6-TaskManagerServiceTutorial.pdf`, 2006 .

[23] A. Ghafoor R. Bhatti, E. Bertino. A trust-based context-aware access control model for web-services. In *ICWS'04*, 2004.

[24] J. H. Saltzer and M. D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 1975.

[25] R. T. Simon and M. E. Zurko. Separation of duty in role-based environments. In *CSFW'97*, 1997.

[26] Emin Gü Sirer and Ke Wang. An access control language for web services. In *SACMAT '02*, 2002.

[27] Sarbanes-Oxley Act of 2002. Pub. L. No. 107-204, 116 Stat. 745, 2002.

[28] Oasis Web Services Reliable Messaging TC. Ws-reliability 1.1, 2004.

[29] Jeroen van Bemmel, Maarten Wegdam, and Ko Lagerberg. 3PAC: Enforcing access policies for web services. In *ICWS'05*, 2005.

[30] W. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and System*. The MIT Press, 2002.

[31] W. van der Aalst and K. van Hee. Workflow patterns. `http://www.workflowpatterns.com`, 2007.

[32] W3C. XSL transformations (XSLT) version 2.0, November 2005.