Authorization and User Failure Resiliency for WS-BPEL business processes

Federica Paci [1], Rodolfo Ferrini [2], Yuqing Sun[3], and Elisa Bertino [1]

[1] Cerias and Computer Science Department, Purdue University,
{paci,bertino}@cs.purdue.edu
[2] Department of Computer Science, University of Bologna,
ferrini@csr.unibo.it
[3] School of Computer Science and Technology (SCST), Shandong University,
sun_yuqing@sdu.edu.cn

Abstract. We investigate the problem of WS-BPEL processes *resiliency* in RBAC-WS-BPEL, an authorization model for WS-BPEL that supports the specification of authorizations for the execution of WS-BPEL process activities by roles and users and authorization constraints, such as separation and binding of duty. The goal of resiliency is to guarantee that even if some users becomes unavailable during the execution of a WS-BPEL process, the remaining users can still complete the execution of the process. We extend RBAC-WS-BPEL with a new type of constraints called *resiliency constraints* and the notion of *user failure resiliency* for WS-BPEL processes and propose an algorithm to determine if a WS-BPEL process is user failure resilient.

1 Introduction

Several XML-based languages have been proposed for specifying and orchestrating business processes, resulting in the WS-BPEL standard language. WS-BPEL has been developed to specify automated business processes that orchestrate activities of multiple Web services. There are, however, cases in which people must be considered as additional participants to the execution of a process. Therefore, it is important to extend WS-BPEL to include the specification of activities that must be fully or partially performed by humans. The inclusion of humans, in turn, requires an access control model to support the specification and enforcement of authorizations to users for the execution of human activities while enforcing constraints, such as separation of duty, on the execution of those activities. One such model is RBAC-WS-BPEL, a role based access control model for WS-BPEL, that supports the specification of authorization information stating which role or user is allowed to execute which human activities in a process [6]. 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. Authorization constraints place restrictions on the roles and users that can perform the activities in the business process. RBAC-WS-BPEL includes also a mechanism to

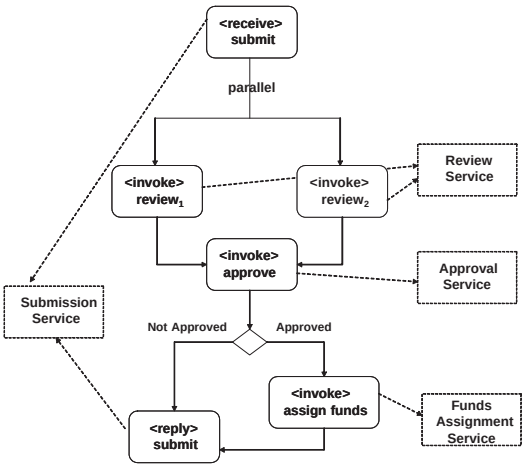


Fig. 1. Project Submission process specification

determine if user requests to perform an activity in a WS-BPEL process can be granted or not; requests are granted if the execution of a WS-BPEL process will complete without violation to the authorization constraints.

In many situations, it is however necessary to make sure that a process can complete even if certain users become unavailable to execute critical activities in the process. The set of available users may change during the execution of a WS-BPEL process for a large variety of reasons. Therefore, resiliency to user unavailability is an important requirement for WS-BPEL processes.

In this paper, we investigate the problem of *resiliency* for WS-BPEL processes. The goal of resiliency is to guarantee that even if some users become unavailable, the remaining users can still complete the activities according to the stated authorization constraints. To address such goal, we extend RBAC-WS-BPEL with a new type of constraints, referred to as *resiliency constraints*, that specify the minimum number of users that must be associated with the execution of the activities in order to give some assurance that even if some users are not available, the WS-BPEL process can terminate. We also define an algorithm that generates assignments (if such assignments exist) of users to activities that satisfy both the authorization constraints and the resiliency constraints.

The remainder of the paper is organized as follows. Section 2 introduces a running example. Section 3 presents the main components of RBAC-WS-BPEL authorization model. Section 4 investigates the problem of resiliency for WS-BPEL processes. Section 4 describes the approach to check if a WS-BPEL process is user failure resilient and presents some complexity results. Sections 6 and 7 discuss the system architecture and report experimental results respectively. Section 8 outlines related work. Section 9 concludes the paper and outlines future research directions.

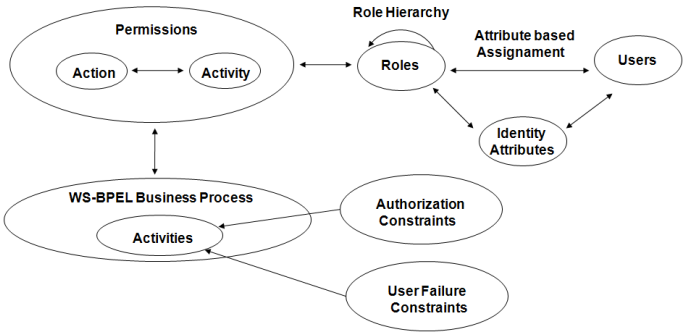


Fig. 2. RBAC-WS-BPEL main components

2 Running Example

In this section we show an example of WS-BPEL process that implements the submission of a research project in an academic institution (see Figure 1). The process orchestrates the following operations:

- the `submit` operation, by the `Submission` service, to submit a project proposal and check if the proposal satisfies various regulations;
- the `review` operation, by the `Review` service, that allows one to review the project proposal;
- the `approve` operation, by the `Approval` service, that allows a faculty member to check the reviews and decide if the project must be supported or not;
- the `assign_funds` operation, by the `Fund Assignment` service, that allows one to revise the funds available and to determine the amount of funds that can be assigned to the project.

The project submission process is organized as follows. First, a faculty member or a phd student submits a project proposal to the academic institution by invoking operation `submit` (the `<receive> submit` activity). Then, the two `review` operations (`<invoke> review` activity) are executed in parallel. After the review process is completed, the `approve` operation is executed (`<invoke> approve` activity): if the project is approved, the operation `assign_funds` (`<invoke> assign_funds` activity) is performed and a notification is sent back to the project investigator (`<reply> submit` activity).

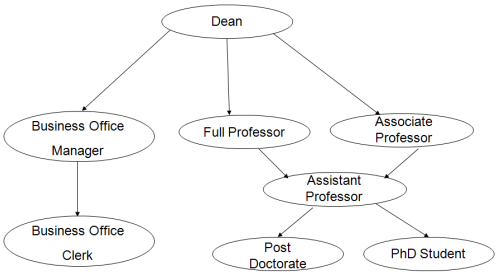
3 RBAC-WS-BPEL Authorization Model

RBAC-WS-BPEL applies to WS-BPEL business processes deployed in a single organization composed of different organizational units. RBAC-WS-BPEL

inherits all the components of traditional RBAC models: users, roles, permissions, role hierarchies, user-role assignment and role-permission assignment relations. Moreover, RBAC-WS-BPEL supports the specification of authorization constraints such as separation of duty and binding of duty that restrict the set of users that can perform a given activity (see Figure 2). In particular, RBAC-WS-BPEL associates with a WS-BPEL business process a set of roles R. Each role is associated with a set of conditions on users' properties that users must satisfy in order to be assigned to that role. Examples of such properties are "social security number", "birth-date" and "employment". Users' properties, that are referred to as *identity attributes*, are conveyed in digital credential issued by trusted third parties called Certification Authorities. Therefore, the assignment of a user to a role is executed by evaluating the user's attributes against the conditions associated with the role. If the user attributes satisfy such conditions, the user is assigned to the role. In RBAC-WS-BPEL, permissions represent the ability to

Roles	**Users**
`Dean`	{John }
`Full professor`	{Mary, Jane}
`Associate professor`	{Chris, Irini }
`Assistant professor`	{Anna, Dan }
`Post Doctorate`	{Ellen, Doug}
`PhD Student`	{Ashish, Melanie, Kara}
`Business Office Manager`	{Tammy }
`Business Office Clerk`	{ Robynne, Leslie}

(a) Roles



(b) The role hierarchy

Fig. 3. RBAC-WS-BPEL role hierarchy and role-permission assignment relation for the project submission process

execute an activity of a WS-BPEL business process and are specified as tuples of the form $(A_i, Action)$ where A_i identifies an activity and *Action* identifies the execution of the activity. Permissions are assigned to roles that are structured in a *role hierarchy* that defines a permission inheritance relation among the roles. RBAC-WS-BPEL supports separation of duty and binding of duty constraints and any authorization constraint that can be expressed as a binary relation on the set of users or roles. An authorization constraint is represented by a tuple $< D, (A_1, A_2), \rho >$, where D is the user or role who has executed activity A_1, called the *antecedent activity*, A_2 is the *consequent activity* to which the con-

Roles	**Permission**
`Post Doctorate`, `PhD Student`	(`<receive> submit`, execute)
`Assistant professor`, `Associate professor`	(`<invoke> review`$_1$, execute)
`Assistant professor`, `Associate professor`	(`<invoke> review`$_2$, execute)
`Full professor`	(`<invoke> approve`, execute)
`Business Office Manager`	(`<invoke> assign_funds`, execute)
`Business Office Clerk`	(`<reply> submit`, execute)

(a) Role-permission assignment relation

	Authorization Constraint
C_1	<U, (`<invoke> assign_funds`, `<reply> submit`), = >
C_2	<U, (`<invoke> review`$_1$, `<invoke> review`$_2$), $\neq$ >
C_3	<U, (`<receive> submit`, `<invoke> review`$_1$), $\neq$ >
C_4	<U, (`<receive> submit`, `<invoke> review`$_2$), $\neq$ >
C_5	<U, (`<receive> submit`,`<invoke> approve`,) $\neq$ >
C_6	<U, (`<invoke> review`$_1$, `<invoke> approve`), $\neq$ >
C_7	<U, (`<invoke> review`$_2$, `<invoke> approve`), $\neq$ >

(b) Authorization constraints

Fig. 4. RBAC-WS-BPEL role-permission assignment relation and authorization constraints for the project submission process

straints is applied and ρ is a relation on U, the set of users, or on R, the set of roles. A constraint $< D, (A_1, A_2), \rho >$ is *satisfied* if, whenever $x \in D$ performs A_1 and y performs A_2, then $(x, y) \in \rho$. Authorization information is encoded in RBAC-XACML [3] while authorization constraints are represented in BPCL, a special purpose XML language for specifying authorization constraints [6]. Finally, RBAC-WS-BPEL supports an algorithm to evaluate at runtime whether a request to execute an activity by a user can be granted or not on the basis of the authorizations the user has and on the basis of authorization constraints defined for the activity.

Example 1. Figures 3 and 4 show the RBAC-WS-BPEL components defined for our running example. Figure 3 (b) shows the role hierarchy that specifies the different positions in an academic institution. Figure 3 (a) lists for each role the users explicitly assigned to each role. [1] Figure 4 (a) illustrates a typical role-permission assignment relation. For example, activity `<invoke> review`$_1$ can be performed both by an `Assistant professor` and by an `Associate professor`. Figure 4 (b) reports the authorization constraints defined for the project submission process. C_1 is a binding of duty constraint, requiring that the same user

[1] A user is *explicitly* assigned to a role if the user's attributes satisfy the conditions associated with the role. A user is *implicitly* assigned to the roles that are dominated in the role hierarchy by the role to which the user is explicitly assigned.

that assigns the funds to the project must notify if the project proposal has been approved or not. C_2, C_3, C_4, C_5 and C_6 are separation of duty constraints. For example, C_2 states that the users who perform `<invoke> review`$_1$ and `<invoke> review`$_2$ must be different.

4 Process User Failure Resiliency

In this section we introduce the key notions of our resiliency model.

Definition 1 (Resiliency constraint). *Let U be the set of users and let BP a WS-BPEL process. A resiliency constraint is a tuple $<A_i, n_i>$, where $A_i \in BP$ and, $n_i \in \mathbb{N}$, n_i denotes the minimum number of users that must have the authorization [2] to perform A_i.*

We now introduce the notion of *user failure resiliency* for a WS-BPEL business process.

Definition 2 (User Failure Resiliency). *Let U be the set of users, BP be a WS-BPEL business process, U_{A_i} be the set of users authorized to perform activity $A_i \in BP$, and $<A_i, n_i>$ be a resiliency constraint for activity A_i. We say that BP is* user failure resilient *if for each $A_i \in BP$ such that a resiliency constraint $<A_i, n_i>$ exists, then $|U_{A_i}| \geq n_i$. Moreover, the* maximum resiliency *of a WS-BPEL business BP, denoted as $MaxRes$, is defined as the maximum over the set $\{n_i \mid \exists <A_i, n_i>$ such that $A_i \in BP\}$.*

If a WS-BPEL process is user failure resilient, there is a sufficient number of authorized users to perform the process so that authorization constraints are satisfied and the process terminates even if some users become unavailable.

A relevant concept to determine whether a WS-BPEL process is user failure resistant is the concept of *configurations*.

Definition 3 (Configuration). *Let U be the set of users, BP be a WS-BPEL business process and U_{A_i} be the set of users authorized to perform activity $A_i \in BP$. Let C be the set $\{<A, u> \mid A \in BP \wedge u \in U_A\}$. We say that c, $c \subseteq C$, is a* configuration *for BP if $\forall$ $A_i \in BP$, $\exists$ one and only one tuple $<A, u> \in c$ such that $A = A_i$.*

The above definition states basically that a configuration must specify a user assignment for each activity in the process.

To determine if a WS-BPEL process is user failure resilient a possible approach is to compute the set S of all possible configurations and then evaluate if the resiliency constraints are satisfied. All such configurations would be then

[2] To determine if a WS-BPEL business process is user failure resilient, we assume that a user has the authorization to execute an activity A_i if he/she is assigned to a role which has the permission to perform A_i. The authorization to execute the activity A_i is effectively granted to the user only at runtime when he/she claims the execution of A_i.

stored to decide which user has to substitute another user if the latter becomes unavailable at run-time. However, rather than computing and storing all the possible configurations, it is sufficient to compute only a subset S_c of S that satisfies the following property:
for each activity $A_i \in BP$ such that a resiliency constraint $<A_i, n_i>$ exists, $|\bigcup_{c \in S_c} \{<A_i, u> \mid <A_i, u> \in c\}| = n_i$.
A resiliency constraint $<A_i, n_i>$ for an activity A_i is satisfied if it is possible to find at least n_i users authorized to perform A_i and therefore n_i configurations. It is trivial to prove that if S_c exists, the cardinality of S_c is equal to $MaxRes$.

In the next section we evaluate the complexity of computing the configurations in S_c.

4.1 Computational complexity of Checking User Failure Resiliency

The complexity of checking whether a WS-BPEL process is user failure resilient is given by the following lemmas.

Lemma 1. *Checking whether a WS-BPEL process is user failure resistant, which is called the* user failure resiliency checking problem *(RCP for short), is* **NP**-*Complete in RBAC-WS-BPEL.*

Lemma 2. *RCP is* **P** *in RBAC-WS-BPEL if only binding of duty constraints are specified on the process activities.*

Lemma 3. *RCP is* **NP**-*Complete in RBAC-WS-BPEL if only separation of duty constraints are specified on the process activities.*

See [7] for the proofs.

5 Constraints evaluation and Planning

As we proved in the previous section, the complexity of computing the configurations to check whether a WS-BPEL process is user failure resilient is **NP**-Complete. In fact, in the worst case, the complexity is $O(|U|^{|A|} * MaxRes)$, where $|U|$ is the number of potential users and $|A|$ is the number of activities in the process, because to compute a configuration, all the possible assignments of users to activities are tried for all the activities in the process and this step is iterated a number of times equal to $MaxRes$.

To reduce the complexity of computing configurations, we thus introduce two heuristics that reduce the number of assignments of users to activities. First, for all the activities that are linked by a binding of duty constraint, the set of users that are authorized to perform these activities is set to the intersection of the sets of users who are authorized to perform each single activity. For example, if the binding of duty constraints $< U,(A_1, A_2),= >$ and $< U,(A_1, A_3),= >$ are specified for activities A_1, A_2 and A_3, the sets of users U_{A_1}, U_{A_2} and U_{A_3} that are authorized to perform A_1, A_2 and A_3 are equal to the intersection $V_{A_1} \cap$

Algorithm 1: Process User Failure resiliency satisfaction

Require: AC set of authorization constraints,
RC set of resiliency constraints,
$Activities$ set of activities ordered according the business process specification
Ensure: BP is user failure resistant

```
1: MaxRes = Max(RC)
2: for A_i ∈ Activities do
3:    V_{A_i} = getAuthorizedUsers(A_i)
4:    if |V_{A_i}| < RC_{A_i} then
5:       exit
6:    else
7:       V. add(V_{A_i})
8:    end if
9: end for
10: LinkedActivities = getSubActivities(Activities)
11: satisfiable = true
12: while (Num < MaxRes AND satisfiable) do
13:    for LinkedActivities_i ∈ LinkedActivities do
14:       Current_Activity = LinkedActivities_i.head()
15:       SubConfig_i = build_config(Current_Activity, V, AC, RC, Conf_Set,
          SubConfig_i, LinkedActivities_i)
16:       SubConfig.add(SubConfig_i)
17:    end for
18:    Current_Config = merge(SubConfig)
19:    if Current_Config.size() == Activities.size() then
20:       satisfiable = true
21:       Conf_Set.add(Current_Config)
22:       Num = Num+1
23:    else
24:       satisfiable = false
25:    end if
26: end while
27: if | Conf_Set | < MaxRes then
28:    for A_i ∈ Activities do
29:       if | Conf_Set | < RC_{A_i} then
30:          Missing_Users = RCA_i - | Conf_Set |
31:          Roles=ua-update(Missing_Users)
32:       end if
33:    end for
34: end if
```

$V_{A_2} \cap V_{A_3}$. V_{A_1}, V_{A_2} and V_{A_3} are, respectively, the set of users that are authorized to execute A_1, A_2 and A_3 because they are assigned to a role that has the permission to execute A_1, A_2 and A_3. The adoption of this heuristic increases the success rate of assignment of users to activities and therefore minimizes the number of user assignments.

The second heuristic groups the activities that are linked by authorization constraints. For example, activities A_4 and A_5 are in the same subset of activities

if there is a separation of duty constraint $< U,(A_4, A_5),\neq >$ between A_4 and A_5. For each subset of activities, a partial configuration is computed and then a complete configuration for the process is generated by merging the partial configurations. This optimization reduces the number of user assignments to activities because, when the assignment of a user to an activity fails, the reassignment of a user is performed only for the antecedent activities that are in the same subset of the activity for which the assignment fails and not for all the other antecedent activities. The computation of the set of users authorized to perform the activities linked by a binding of duty constraint and of the subsets of activities has complexity $|AC|^2$, where $|AC|$ is the number of authorization constraints. The computation of the sub-configurations for each subset of activities has complexity $O(|U_{subset}|^{|A_{subset}|})$, where $|U_{subset}|$ and $|A_{subset}|$ are respectively the number of candidate users and the number of activities in each subset while the complexity of combining the sub-configurations together to obtain a configuration for the whole business process is $|A|$. Therefore, by adopting these heuristics, the complexity of calculating the configurations necessary to assure that a WS-BPEL process is user failure resilient becomes $O(|AC|^2 + MaxRes * \{|U_{subset}|^{|A_{subset}|} + |A|\})$.

Algorithm 1 adopts the heuristics we have described to compute a number of users-to-activities assignment configurations equal to $MaxRes$. First of all the algorithm, computes $MaxRes$ (line 1). Then, for each activity A_i, the procedure `getAuthorizedUsers` returns the set of users V_{A_i} that are authorized to perform activity A_i because they are assigned to a role that has the permission to execute A_i (lines 2-3). If the cardinality of V_{A_i} is lower than the resiliency value specified for A_i, the algorithm terminates, otherwise V_{A_i} is added to V, that is a vector containing for each activity A_i the set V_{A_i} (line 7). Then, the procedure `getSubSetActivities` calculates the subsets of activities $SubSetActivities$ on the basis of the authorization constraints that are applied to them. Each $SubSetActivities$ is saved in the $LinkedActivities$ set (line 10). Then, the algorithm iterates till a number of user configurations equal to $MaxRes$ is not found (line 12) or it is not possible to find such a number of configurations because all the possible combinations of users-to-activities assignment have been tried. The configurations are computed by the recursive procedure `build_config`. `build_config` is executed for each subset $LinkedActivities_i$ and returns a partial configuration $SubConfig_i$. Once a partial configuration $SubConfig_i$ is computed for each subset $LinkedActivities_i$, the procedure `merge` combines all $SubConfig_i$ in one configuration $Current\_Config$ that is added to the configurations set $Config\_Set$. If Algorithm 1 is not able to compute a number of configurations equal to $MaxRes$, it determines the activities for which the resiliency constraint is not satisfied. These activities are the activities whose resiliency value is lower than the number of configurations in $Config\_Set$. For each of these activities, Algorithm 1 calculates the number of additional potential users should be associated with the execution of the activities. Then, `ua-update` checks the logs associated with the activities and returns the roles

for which the user failure assignment has more frequently failed. The additional potential users needed must be added to these roles.

Example 2. Assume that for the activities `<invoke> review`$_1$, `<invoke> review`$_2$ and `<invoke> approve` the following resiliency constraints are specified: (`<invoke> review`$_1$, 3), (`<invoke> review`$_2$, 3) and (`<invoke> approve`, 2). Since $MaxRes$ is equal to three, to prove that the project submission process is user failure resilient, we need to find three different users-to-activities assignment configurations. An example of such configurations is:

1. (`<receive> submit`, Irini), (`<invoke> review`$_1$, Mary),(`<invoke> review`$_2$, Anna), (`<invoke> approve`, Jane), (`<invoke> assign_funds`, John), (`<reply> submit`, John)
2. (`<receive> submit`, Kara), (`<invoke> review`$_1$, Chris), (`<invoke> review`$_2$, Mary), (`<invoke> approve`, John), (`<invoke> assign_funds`, Tammy), (`<reply> submit`, Tammy)
3. (`<receive> submit`, Irini), (`<invoke> review`$_1$, Jane), (`<invoke> review`$_2$, John), (`<invoke> approve`, Mary), (`<invoke> assign_funds`, Tammy), (`<reply> submit`, Tammy).

Therefore, the project submission process is user failure resilient. Consider a different scenario, in which the resiliency constraints (`<invoke> review`$_1$, 4), (`<invoke> review`$_2$, 4) and (`<invoke> approve`, 3) are applied to activities `<invoke> review`$_1$, `<invoke> review`$_2$ and `<invoke> approve`. Now, we have to find four configurations to prove that the process is resilient since $MaxRes$ is equal to four. In this case, the following configurations are generated:

1. (`<receive> submit`, Jane), (`<invoke> review`$_1$, Dan), (`<invoke> review`$_2$, Chris), (`<invoke> approve`, Mary), (`<invoke> assign_funds`, Tammy), (`<reply> submit`, Tammy)
2. (`<receive> submit`, Anna), (`<invoke> review`$_1$, Irini), (`<invoke> review`$_2$, John), (`<invoke> approve`, Jane), (`<invoke> assign_funds`, John), (`<reply> submit`, John)
3. (`<receive> submit`, Kara), (`<invoke> review`$_1$, Jane), (`<invoke> review`$_2$, Mary), (`<invoke> approve`, John), (`<invoke> assign_funds`, null), (`<reply> submit`, null)
4. `<receive> submit`, Ashish), (`<invoke> review`$_1$, Chris), (`<invoke> review`$_2$, Anna), (`<invoke> approve`, null), (`<invoke> assign_funds`, null), (`<reply> submit`, null).

It's easy to see that the process is not user failure resilient because the first two configurations are complete but for the other ones the assignment of a user to activities `<invoke> approve`, `<invoke> assign_funds` and `<reply> submit` fails.

6 System architecture

The main components of the RBAC-WS-BPEL architecture (see Figure 5) are the *WS-BPEL engine*, the *XACML Policy Store*, *BPCL Constraints Store* repos-

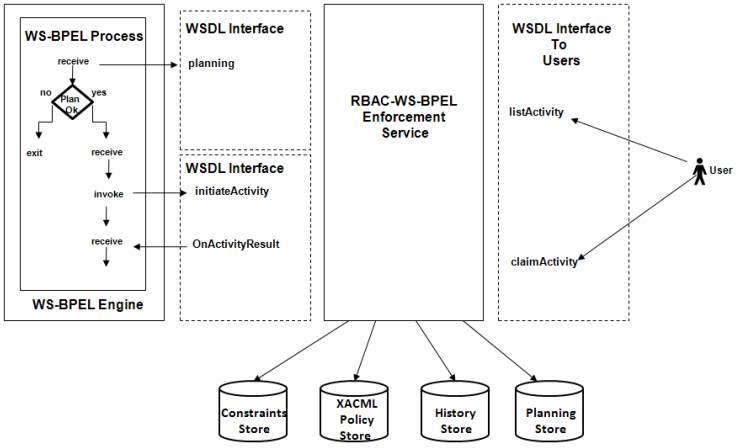


Fig. 5. RBAC-WS-BPEL architecture

itories, the *History Store* and the *RBAC-WS-BPEL Enforcement Service*. 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 services operations associated with activities. The XACML Policy Store records the RBAC-WS-BPEL authorization schema associated with the business process, whereas the BPCL Constraints Store records the authorization constraints. The History Store records the users who have performed an activity and whether the execution of the activity has been successful or not. The RBAC-WS-BPEL Enforcement Service supports the WS-BPEL process administrators both at deployment time and at runtime. When the process is deployed, the RBAC-WS-BPEL Enforcement Service checks if the process is user failure resilient and, hence, if there is a number of users sufficient to start the execution of the process, while during the execution of the process the RBAC-WS-BPEL Enforcement Service checks whether the execution of an activity by a user violates authorization constraints. The RBAC-WS-BPEL Enforcement Service offers three WSDL interfaces. The first interface provides the operations for starting and completing the execution of a WS-BPEL activity that must be performed by a user. The second interface allows users to visualize the activities they can claim, and to claim and execute them [6]. The third interface provides functions for determining if a WS-BPEL process is user failure resilient.

In what follows, we focus on the description of the third interface, because it is the most relevant for the discussion in the paper.

Such interface provides a single operation, called `planning`, that implements Algorithm 1. The `planning` operation notifies the WS-BPEL process administrator if the process is user failure resilient and, if this is not the case, displays the activities for which the resiliency constraints are not satisfied and the roles authorized to perform the activities that should be populated with additional users. The WS-BPEL process administrator can decide to proceed with the ex-

Test Case	Business Process	Num of BoD	Num of SoD	$MaxRes$	Num of Users
1	21 activities	4	4	6	50..140
2	21 activities	4	4	[3,9]	50
3	21 activities	[0,5]	4	6	50
4	21 activities	0	[3,6]	6	50

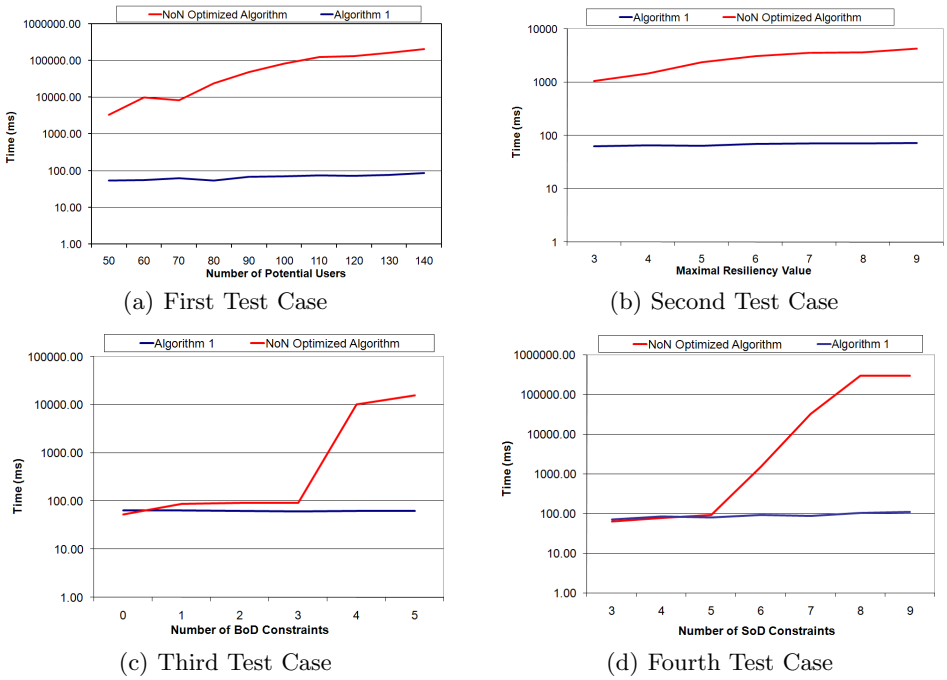
Table 1. Test cases parameters

ecution of the process or to halt the execution and modify the set of potential users associated with the execution of the process. To enable the execution of the `planning` operation, the WS-BPEL process designer has to include in the `<partnerLinks>` list, the RBAC-WS-BPEL Enforcement Service. Moreover, the WS-BPEL specification must be such that the start activity of the process is a `<receive>` activity that executes the `planning` operation. The `<receive>` `planning` activity is followed by an `<if>` activity that performs the subsequent activities that implement the business process only if the execution of `planning` is successful. The `planning` operation retrieves from the XACML Policy Store the hierarchy of roles and the list of users assigned to the roles and selects from the BPCL Constraints Store the set of authorization and resiliency constraints that are necessary for executing Algorithm 1. All configurations computed by the `planning` operation are stored in an additional repository, referred to as *Planning Store*, while the logs of unsuccessful assignments of users to activities are recorded in the History Store.

7 Experimental evaluation

We have carried out several experiments to evaluate the impact of the heuristics for reducing the cost of the configuration computation and to prove the effectiveness of our approach. To execute the tests we have implemented Algorithm 1 and, in addition, an algorithm, referred to as $NoNOptimized$, which computes a number of configurations equal to $MaxRes$ as Algorithms 1 but without adopting the heuristics to reduce the complexity. We have also generated a WS-BPEL process composed by 21 activities, a set of 50 potential users, a role hierarchy of 7 roles, 4 separation of duty and 4 binding of duty constraints. Such process has a $MaxRes$ value equal to 6. We have considered four test cases that are summarized in Table 1. In particular, we have measured in CPU time (milliseconds) the execution time of Algorithm 1 and of the $NoNOptimized$ algorithm in the following cases:

1. we varied the number of potential user from 50 to 140 and we kept the number of separation of duty and binding of duty constraints equal to 4, and the value of $MaxRes$ equal to 6.



(a) First Test Case (b) Second Test Case

(c) Third Test Case (d) Fourth Test Case

Fig. 6. Experimental results

2. we varied the value of $MaxRes$ from 3 to 6 and we kept the number of separation of duty and binding of duty constraints equal to 4 and the number of potential users equal to 50.
3. we varied the number of binding of duty constraints defined for the process from 0 to 5 and we set the number of separation of duty constraints to 4, the value of $MaxRes$ to 6 and the number of potential users to 50.
4. we varied the number of separation of duty constraints defined for the process from 3 to 6 and we set the number of binding of duty constraints to 4, $MaxRes$ value to 6 and the number of of potential users to 50.

The experiments have been run on a PC with operating system WINDOWS XP SP2, a 2Gz T7200 processor and 2GB RAM. Moreover, for each test case we have executed twenty trials, and the average over all the trial execution times has been computed. Figures 6 (a) and (b) report the execution times measured for test cases 1 and 2. The execution times of Algorithm 1 are almost constant for increasing values of the number of potential users and $MaxRes$ value, while the execution time of the $NoNOptimized$ algorithm increases. The reason is that in Algorithm 1 the first heuristic reduces the number of unsuccessful users assignments. Moreover, the second heuristic reduces the number of activities for which we try to reassign a user in case of failure. Instead, for the $NoNOptimized$

algorithm, the time increases because in case of user assignment failure for an activity A_i, a reassignment of a user is tried for all the antecedent activities A_{i-1}, A_{i-2} ... rather than only for the antecedent activities that are linked to A_i by an authorization constraint. The experimental results reported in Figure 6 (c) show the advantage of adopting the first heuristic about the activities that are linked by a binding of duty constraint. When the number of binding of duty constraints is equal to 0, the execution times of Algorithm 1 and of the *NoNOptimized* algorithm are the same; while when the number of binding of duty constraints increases, the execution times of the *NoNOptimized* algorithm become greater than Algorithm 1. When increasing the number of binding of duty constraints, the user assignment success rate for Algorithm 1 increases and, as a consequence, the number of user assignments is minimized. This is the reason why Algorithm 1's execution time is almost constant, while the execution time of the *NoNOptimized* algorithm increases. Finally, Figure 6 (d) shows the impact of generating the subsets of activities. The increase of the number of separation of duty constraints restricts the number of users authorized to perform the activities and, as a consequence, the probability that a user assignment fails is very high. Therefore also the number of reassignments of users to activities is high. The execution time of Algorithm 1 is lower than the time of the *NoNOptimized* algorithm because the number of activities for which the user reassignments is performed is minimized; the user reassignment is tried only for the antecedent activities in the same subset of the activity for which the user assignment fails. Note that the execution time of Algorithm 1, regardless of the test cases we have performed, is under 100 ms. Such results show that our approach to check whether a WS-BPEL process is user failure resilient is applicable to real case scenarios.

8 Related Work

With the widespread adoption of Web services composition to implement complex business processes 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. Koshutanski et al. [5] propose an authorization model for business processes based on Web services. Both the model of Koshutanski et al. and RBAC-WS-BPEL assume an RBAC model and support authorizations constraints on the set of users and roles. They also consider the problem of taking authorization decision on the execution of business process's activities. The main difference with RBAC-WS-BPEL is in the approach to take authorization decision. In the model by Koshutanski et al., an authorization decision is taken by orchestrating the authorization processes of each Web service, the activities of which are orchestrated in the business process, while in RBAC-WS-BPEL an authorization decision is taken independently for each activity in the process.

Xiangpeng et al. [8] propose an RBAC access control model for WS-BPEL business process. Roles correspond to `<partnerRole>` elements in the WS-BPEL

specification and are organized in a hierarchy. Permissions correspond to the execution of the basic activities in the process specification. In addition, separation of duty constraints can be specified. A main difference with respect to our approach is that RBAC-WS-BPEL's BCPL constraints language supports the specification of a broader range of authorizations constraints than the model by Xiangpeng et al.

BPEL4People [1] is a recent proposal to handle person-to-person WS-BPEL business process. With respect to RBAC-WS-BPEL, in BPEL4People users that have to perform the activities of a WS-BPEL business process are directly specified in the process by user identifier(s) 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.

The workflow authorization model proposed by Wang et al. [9] is probably the one that is most closely related to RBAC-WS-BPEL. Wang et al. propose the role-and-relation-based access control (R^2BAC) model for workflow systems. In R^2BAC, in addition to a users role memberships, the users relationships with other users help determine whether the user is allowed to perform a certain step in a workflow. Wang et al. investigate the workflow satisfiability problem, which asks whether a set of users can complete a workflow. They also investigate the resiliency problem in workflow systems, which asks whether a workflow can be completed even if a number of users may be absent. The notion of resiliency supported by RBAC-WB-BPEL is slightly different from the one proposed by Wang et al. They propose a notion of resiliency parametrized in the number of absent users. A workflow is resilient, if it is satisfiable in any configuration where any set of users of cardinality equal to the parameter is not available. Instead, in RBAC-WS-BPEL, a WS-BPEL process is resilient if it is possible to find a number of configurations that satisfy both resiliency constraints and authorization constraints.

9 Conclusions

In this paper, we have investigated the resiliency problem for WS-BPEL business processes. Resiliency in the context of business process means that even if some users become unavailable, the remaining users can still complete the execution of the process according to the stated authorizations and authorization constraints. To address such problem, we have extended RBAC-WS-BPEL, which is an authorization model for WS-BPEL business processes, with the notions of *resiliency constraints* for activities and *user failure resiliency* for a business process. We have proposed an algorithm that allows to statically determine if a WS-BPEL is user failure resilient. The algorithm verifies there is a number of users-to-activities assignment configurations equal to $MaxRes$. These configurations are computed by assuming that users are assigned to the execution of an activity because they cover a role that is granted the execution of the activity. The authorization to execute an activity is effectively granted to users only at runtime when they claim the execution of the activity. Though, the complex-

ity of computing such configurations is NP-complete in the most general case, the proposed algorithm adopts some heuristics that reduce the computational complexity. The experimental results, we have performed, have shown that the algorithm is efficient in most practical cases.

We are planning to extend this work in several directions. We are currently investigating how the RBAC-WS-BPEL Enforcement Service can be implemented on top of ODE BPEL engine[4]. We also want to extend RBAC-WS-BPEL to support authorizations for cross-organizations business processes. Currently, RBAC-WS-BPEL is applied to inter-organization business processes. As we have been told by the main companies providing solutions for WS-BPEL processes, WS-BPEL is mainly used to specify inter-organization business processes rather than cross-organizations business processes. We are also planning to extend RBAC-WS-BPEL with more sophisticated authorization constraints.

References

1. Agrawal, A. et al.: WS-BPEL Extension for People (BPEL4People), Version 1.0, . Online at: http://www.adobe.com/devnet/livecycle/pdfs/bpel4people_spec.pdf,(2007).
2. Alves. A. et al.: Web Services Business Process Execution Language, Version 2.0, OASIS Standard, April 2007. Online at: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.
3. Anderson, A.: Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML, Version 2.0, OASIS Standard, 2005. Online at: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf.
4. Apache ODE BPEL engine. http://ode.apache.org/bpel-extensions.html
5. Kostutanski, H., Massacci, F.: An Access Control Framework for Business Processes for Web Services. In Proceedings of ACM Workshop on XML Security, George W. Johnson Center at George Mason University, Fairfax, Va, USA, October 2003, 15−24.
6. Paci, F., Bertino, E, Crampton, J.: An Access Control Framework for WS-BPEL, International Journal of Web service Research, 5 (3), pp. 20−43 (2008).
7. Paci, F., Ferrini, R., Sun, Y., Bertino, E.: Authorization and User Failure Resiliency for WS-BPEL business processes, Cerias Technical report(2008).
8. Xiangpeng, Z., Cerone, A., Krishnan, P.: Verifying BPEL Workflows Under Authorisation Constraints. In Proceedings of Fourth International Conference on Business Process Management (BPM 2006), Vienna, Austria, September 2006.
9. Wang Q., Li, N.:Satisfiability and Resiliency in Workflow Systems, in Proceedings of ESORICS 2007, pp. 90-105.