Department of
Information Engineering
and Computer Science
DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive 14 - 38123 Povo - Trento (Italy)
http://www.disi.unitn.it

DYNAMIC RESILIENCY TO USER ASSIGNMENT

Fabio Massacci, Federica Paci, and Olga Gadyatskaya

December 2010

Technical Report # DISI-10-074

Dynamic Resiliency to User Assignment

Fabio Massacci, Federica Paci, and Olga Gadyatskaya

DISI, University of Trento, Italy
{massacci, paci, gadyatskaya}@disi.unitn.it

Abstract. Nowadays organizations face fast organizational changes and employee turnover, e.g user are unavailable or they change roles because of a promotion, that might compromise the execution of a business process instance and, thus, the achievement of organizational business goals. In this paper, we investigate the problem of *dynamic resiliency* to changes in the assignment of users to roles. The goal of dynamic resiliency is to guarantee that when the assignment of users to roles changes during the execution of a business process instance, it is possible to find a user to perform the activities whose execution is still pending. We propose an approach to verify that the execution of a business process instance can terminate when there are changes in the in the assignment of users to roles.

1 Introduction

Moderns organizations often follow the ValIT framework [4], the CoBIT [4] model or other management models that require an organization to identify the business goals it has to achieve, the business process activities which achieves those goals, the roles that are responsible for those achievement, and the (security) controls to make sure that no unwanted deviations occur. In many cases the achievement of the organizational goals is even translated into monetary compensation on top of the salary for the users holding the corresponding roles. ERP companies such as SAP are now providing modules (HCM) to manage the link between goals and human resources.

Unfortunately, fast organizational changes and employee turnover, e.g user are unavailable or they change roles because of a promotion, might compromise the execution of a business process instance and, thus, the achievement of the business goals. A common solution that is used by HR staff is to appoint a new user to the role, for example a former co-worker, and delegate to it the execution of the activities that were assigned to the user who is no longer available. However, the selected user might not be able to execute the assigned activities because of the presence of binding-of-duty or separation of duty constraints.

In this paper we investigate the problem of *dynamic resiliency* of a business process instance to changes in the assignment of users to roles: when users' assignment to roles changes during the execution a business process, we want to find the "right" user to perform the activities whose execution is still pending. This means finding a user who is assigned to a role that is entitled to execute the pending activity and is responsible for the achievement of the goal fulfilled by the execution of the activity.

To check whether a business process instance is dynamically resilient, we model an organization as a digraph where the vertexes represent the organizational business

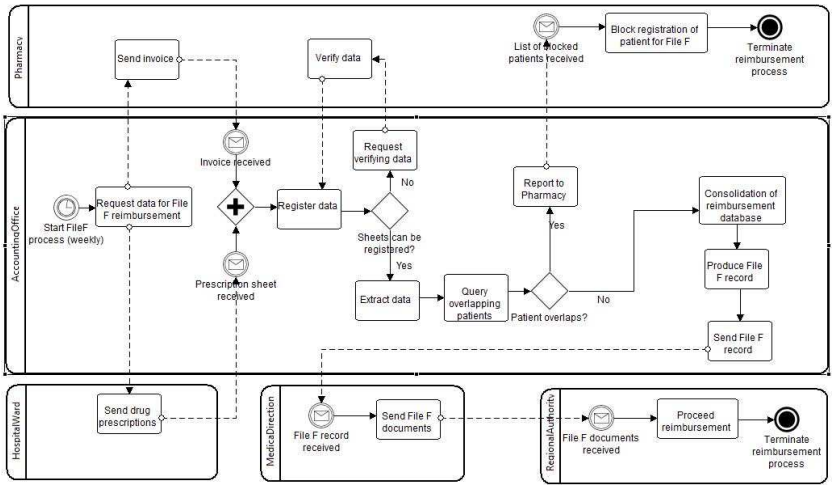

Fig. 1. File F business process

goals, the activities that implement such goals, the organizational roles and the users assigned to these roles. Then, for each activity we traverse the graph to find those paths that have the activity as initial vertex and go back to the activity by traversing the node representing a goal fulfilled by the execution of the activity, and the nodes representing a user, and a role which is both responsible for fulfilling the goal and it is authorized to execute the activity.

The reminder of the paper is organized as follows. Section 2 introduces the running example. Section 3 presents the main components of the organization model. Section 4 presents our approach to verify that a business process instance is dynamically resilient. Sections 5 discusses the system architecture. Section 6 outlines related work. Section 7 concludes the paper and outlines future research directions.

2 The Drug Reimbursement Process

In this paper we use as illustrative example a business process [1] for drug reimbursement called File F process shown in Figure 1. It consists of four macro-phases executed by different actors. The first phase is the drug prescription done by a doctor to a patient to care patient's disease, while the second phase is the drug dispensation to the patients. During the third phase a number of reports are generated and audited. The last phase is about cost reimbursement by local health authorities. Here we focus on the third phase which involves multiple actors, under different trust domains: the Hospital Ward, the Pharmacy, the Accounting Office, the Medical Direction, and the Regional Authority. The Hospital Ward sends drug requests and notifies the pharmacy about the drug availability in the ward stock; it is also responsible to send the dispensation documents to the Accounting Office. The Pharmacy is responsible for the update of all drug data (price,

[1] Adopted from the MASTER project http://www.master-fp7.eu/

codes, etc.). The Accounting Office performs the File F data extraction and generates File F records that are later sent to the Regional Authority. The Medical Direction has to send the File F records to the Regional Health Care Authority.

In this process, people can change role. For example doctors can be promoted to the medical direction or can leave the hospital to work for their own practice or another hospital. Administrative staff can be re-assigned to the planning or accounting. No matter what happens, at the end of the month the hospital must be able to complete the process for drug reimbursement. At the same time the presence of significant privacy, security and medical regulations the hospital as whole cannot just assign the first available user to a task in order to conclude a subprocess. The user must have enough the privileges to do so.

3 The Organization Model

We model an organization starting from the organizational business goals.

Definition 1 (Goal Model). *Let $\mathcal{G}$ be the set of business goals of an organization Org. A goal model* GM *is a directed acyclic hypergraph $\langle \mathcal{G}, \mathsf{Decompose} \rangle$ with one sink G_{top} where* $\mathsf{Decompose} \subseteq \wp(\mathcal{G}) \times \mathcal{G}$ *is the* ***Decomposition*** *relation between goals.*

Intuitively the sink represents the top goal of the organization while vertexes model intermediate business goals. The hyperarcs correspond to the decomposition of the goals into subgoals: $\langle \{G_1, \ldots G_k\}, G_t \rangle$ denotes that the goal G_t is AND decomposed into subgoals $G_1, \ldots, G_k$. Alternative decomposition is represented by different arcs incoming to the same goal.

In an hospital, typically, the head of the hospital might be assigned the achievement of a set of goals (e.g manage the drug dispensation, compile a drug summary request form, and save 5% from last year budget etc.) which he then decomposes into subgoals and assign them to the employees of the hospital such physicians, nurses and so on.

The achievement of goals depends on to the execution of the organizational business process. We consider business processes that are implemented by orchestrating Web services operations and are specified in the WS-BPEL standard [2]. Some of the activities in the process can be automatically executed while other activities require the intervention of humans.

The formal relation that associates goals with the business process activities that implement them is captured by the ***Implementation*** relation $\mathsf{Impl} \subseteq \mathcal{A} \times \mathcal{G}$. We say that an activity A implements a goal G_k if $\langle A, G_k \rangle \in \mathsf{Impl}$.

The next component of our organizational model is the RBAC model. For sake of simplicity, we consider as the only permission type the execution of the activities in the business process[2]. Permissions of executing an activity are assigned to the roles that identify organization's job positions. Users are assigned to roles, and through role assignments they acquire the permissions to perform the business process activities.

[2] In the real scenario we need a more refined classification such as the distinction between logging an activity and executing it, or deploying the activity and invoking it etc.

Definition 2 (Authorization Model). *Let $\mathcal{R}$ be the set of roles and $\mathcal{U}$ be the set of users inside the organization Org. Let* $\mathsf{PA} \subseteq \mathcal{R} \times \mathcal{A}$ *be the* ***Roles to Activities Assignment*** *relation,* $\mathsf{UA} \subseteq \mathcal{R} \times \mathcal{U}$ *be the* ***Roles to Users Assignment*** *relation, and* $\mathsf{UA}^* \subseteq \mathcal{U} \times \mathcal{R}$ *be the symmetric relation of* UA*. The authorization model* AM *is a directed acyclic graph* $\langle \mathcal{R} \cup \mathcal{U} \cup \mathcal{A}, E_{\mathsf{AM}} \rangle$ *where* $E_{\mathsf{AM}} = \mathsf{PA} \cup \mathsf{UA} \cup \mathsf{UA}^*$[3]*.* AM *has one sink* R_{chief}*.*

The authorization model also allows to specify *authorization constraints* on the execution of business process activities.

$SoD_x(A_i, A_j)$ is a **separation of duty constraint** and $BoD_x(A_i, A_j)$ is a **binding of duty constraint**, where pedex x denotes whether the constraint is applied to users (u) or roles (r) and A_i, A_j are the activities to which the constraint is applied. We say that a separation of duty constraint $SoD_x(A_i, A_j)$ is *satisfied* if, whenever U executes activity A_i and U' performs A_j , then $U \neq U'$. A binding of duty constraint $BoD_x(A_i, A_j)$ is *satisfied* if, whenever U executes activity A_i and U' performs A_j, then $U = U'$. We denote with $\mathcal{C}$ the set of constraints which apply to the activities in $\mathcal{A}$.

The execution of an activity A is *granted* to a user U if U is assigned to a role R which has the permission to execute A, and the execution of A by U does not violate any authorization constraint.

We can now formally define the organization model as follows.

Definition 3 (Organization Model). *Let* $\mathsf{Resp} \subseteq \mathcal{G} \times \mathcal{R}$ *be the* ***Goals to Roles Assignment*** *relation, and* $\mathsf{Impl} \subseteq \mathcal{A} \times \mathcal{G}$ *be the* ***Implementation*** *relation. The* ***Organization model*** *is a digraph* $\mathcal{M} = \langle N_{\mathcal{M}}, E_{\mathcal{M}} \rangle$ *where* $N_{\mathcal{M}} = \mathcal{A} \cup \mathcal{G} \cup \mathcal{R} \cup \mathcal{U}$ *and* $E_{\mathcal{M}} = \mathsf{Decompose} \cup \mathsf{PA} \cup \mathsf{UA} \cup \mathsf{UA}^* \cup \mathsf{Resp} \cup \mathsf{Impl}$.

The relation Resp associates with a goal the role which is responsible of its fulfillment.

4 Business Process Dynamic Resiliency

The assignment of users to roles dynamically changes over time in an organization. Users can permanently be assigned to a new role as effect of a promotion or temporary assigned because of an emergency situation. Users can also become unavailable because they get sick, they are overwhelmed by work or they go on holiday. These changes in UA relation can be modeled as follows:

- **User U is Unvailable**: $\mathcal{U} = \mathcal{U} \setminus \{U\}$, $\mathsf{UA} = \mathsf{UA} \setminus \{\langle R, U \rangle\}$. If a user U playing the role R becomes unavailable, the user U is removed from the set of users U and the tuple $\langle R, U \rangle$ is removed from UA relation.
- **User U Changes Role**: $\mathsf{UA} = \mathsf{UA} \setminus \{\langle R, U \rangle\}$ and $\mathsf{UA} = \mathsf{UA} \cup \{\langle R', U \rangle\}$. If a user U playing the role R is assigned to another role R' the tuple $\langle R, U \rangle$ is removed from UA relation and the tuple $\langle R', U \rangle$ is added to UA.

[3] The presence of both UA and its symmetric relation is necessary to evaluate the resiliency of a business process by finding paths that traverse nodes representing roles and users assigned to the roles.

When the relation UA changes during the execution of a business process instance, there might not be assignments of users for the activities not executed yet. As a consequence, the business process instance does not terminate and some of the organizational business goals are not satisfied causing a damage to the organization.

Informally a business process instance is *dynamically resilient* to changes in UA relation, if for each activity A_k that has not been executed yet, there is at least a *potential owner*. A potential owner is a user who is assigned to a role that a) is authorized to execute the pending activity without violating authorization constraints, and b) is responsible for the achievement of the goal fulfilled by the execution of the activity.

Formally, checking if for each activity A_k not yet executed it exists at least a potential owner, means traversing the organizational model $\mathcal{M}$, and find a path $A_k \rightarrow G \rightarrow R' \rightarrow U' \rightarrow R' \rightarrow A_k$ that has A_k as initial and final node and traverses the nodes G, R', U' and R' where G is a goal fulfilled by the execution of A_k, R' is a role who has to achieve goal G and is authorized to execute A_k, and U' is a user assigned to the role R'.

In the formal definition of *dynamic resiliency* we will use the notation $\langle A_i, U, R\rangle@t$ to indicate that a user U assigned to the role R has executed the activity A_i at time t.

Definition 4 (Dynamic Resiliency). *Let Org be an organization, and $\mathcal{M} \equiv \langle N_{\mathcal{M}}, E_{\mathcal{M}}\rangle$ be the organizational model of Org. Let $\mathsf{BP}_{\mathsf{inst}}$ be a running instance of the business process BP, and $\mathcal{A}_{Notexecuted}$ be the set of activities of $\mathsf{BP}_{\mathsf{inst}}$ not executed yet. We say that $\mathsf{BP}_{\mathsf{inst}}$ is* **dynamically resilient** *to changes in UA if $\forall A_k \in \mathcal{A}_{Notexecuted}$ one of the following conditions is satisfied:*

- *a) $BoD_r(A_i, A_k)$, and $\langle A_i, U, R\rangle@t$ implies $A_k \rightarrow G \rightarrow R \rightarrow U' \rightarrow R \rightarrow A_k \in \mathcal{M}$ for some U'*
- *b) $BoD_u(A_i, A_k)$, and $\langle A_i, U, R\rangle@t$ implies $A_k \rightarrow G \rightarrow R' \rightarrow U \rightarrow R' \rightarrow A_k \in \mathcal{M}$,*
- *c) $SoD_r(A_i, A_k)$, and $\langle A_i, U, R\rangle@t$, implies $A_k \rightarrow G \rightarrow R' \rightarrow U' \rightarrow R' \rightarrow A_k \in \mathcal{M}$ for some U' and $R' \neq R$,*
- *d) $SoD_u(A_i, A_k)$, $\langle A_i, U, R\rangle@t$ implies $A_k \rightarrow G \rightarrow R' \rightarrow U' \rightarrow R' \rightarrow A_k \in \mathcal{M}$ for some $U' \neq U$,*
  otherwise
- *e) $A_k \rightarrow G \rightarrow R' \rightarrow U' \rightarrow R' \rightarrow A_k \in \mathcal{M}$ for some U' and R'.*

To evaluate whether a business process instance $\mathsf{BP}_{\mathsf{inst}}$ is dynamically resilient, before $\mathsf{BP}_{\mathsf{inst}}$ is executed, we compute for each activity A_k in $\mathsf{BP}_{\mathsf{inst}}$, all the possible hyperpaths $A_k \rightarrow G \rightarrow R' \rightarrow U' \rightarrow R' \rightarrow A_k$ through which activity A_k can be reached. To compute these paths we have adapted an algorithm for optimal path traversal proposed in [3]. Then, for each activity A_k we store in an array $REACH[A_k]$ a list of tuples $\langle U', R'\rangle$ corresponding to the arcs $U' \rightarrow R'$ which belong to one of the computed hyperpath $A_k \rightarrow G \rightarrow R' \rightarrow U' \rightarrow R' \rightarrow A_k$. During the execution of $\mathsf{BP}_{\mathsf{inst}}$, each array $REACH[A_k]$ is dynamically updated when an activity is executed or when UA relation changes. Algorithm **Update** describes the steps to update arrays $REACH[A_k]$. **Update** takes as input the arrays $REACH[A_k]$ for each activity A_k that is part of the business process instance $\mathsf{BP}_{\mathsf{inst}}$, and $Updates$, the list of UA changes. If a user U playing the role R is unavailable, the tuple $\langle U, R\rangle$ is added to $Updates$: in this

Algorithm 1: Update

Require: $REACH[A_1], \ldots REACH[A_n]$ where $A_i \in \mathsf{BP_i}$, $Updates$.

while $Updates$.`next()` $\neq null$ **do**
  if $Updates$.`next()` $== \langle U, R\rangle$ **then**
    for $A_i \in A_{NotExecuted}$ **do**
      if $REACH[A_i]$.`contains`($\langle U, R\rangle$) **then**
        $REACH[A_i]$.`remove`($\langle U, R\rangle$);
  if $Updates$.`next()` $== \langle U, R, R'\rangle$ **then**
    for $A_i \in A_{NotExecuted}$ **do**
      if $REACH[A_i]$.`contains`($\langle U, R\rangle$) **then**
        $REACH[A_i]$.`remove`($\langle U, R\rangle$);
      if $\langle R', A_i\rangle \in \mathsf{PA}$ **then**
        $REACH[A_i]$.`add`($\langle U, R'\rangle$);

Algorithm 2: $\mathsf{BP_{inst}}$ **Execution**

Require: $\mathsf{BP_{inst}}$, $\mathcal{A}_{Notexecuted} = \emptyset$, $\mathcal{A}_{Executed} = \mathcal{A}$, $REACH[A_1], \ldots REACH[A_n]$ where $A_i \in \mathcal{A}$, $History$.

$A_{current} = A_{start}$
repeat
  Find $(R, U) \in REACH[A_{current}]$; `//Find a potential user`
  Execute($A_{current}$);
  $\mathcal{A}_{Notexecuted} \leftarrow \mathcal{A}_{Notexecuted} \backslash A_{current}$;
  $History$.`add`($\langle A_{current}, U, R\rangle$);
  $REACH[A_{current}]$.`removeall`();
  $REACH[A_{current}]$.`add`($\langle U, R\rangle$);
  for $BoD_r(A_{current}, A_k)$ **do**
    for $\langle U', R'\rangle \in REACH[A_k]$ such that $R' \neq R$ **do**
      $REACH[A_k]$.`remove`($\langle U', R'\rangle$);
  for $BoD_u(A_{current}, A_k)$ **do**
    for $\langle U, R'\rangle \in REACH[A_k]$ such that $U' \neq U$ **do**
      $REACH[A_k]$.`remove`($\langle U', R'\rangle$);
  for $SoD_r(A_{current}, A_k)$ **do**
    for $\langle U', R'\rangle \in REACH[A_k]$ such that $R' = R$ **do**
      $REACH[A_i]$.`remove`($\langle U', R'\rangle$);
  for $\exists SoD_u(A_{current}, A_k)$ **do**
    for $\langle U', R'\rangle\rangle \in REACH[A_k]$ such that $U' = U$ **do**
      $REACH[A_k]$.`remove`($\langle U', R'\rangle$);
  if Change() $\neq 0$ **then**
    IsResilient($\mathcal{A}_{Notexecuted}$);
  $A_{current} = A_{next}$; `//Find next activity to execute`
until $\mathcal{A}_{Notexecuted} == \mathcal{A}_{end}$

case, **Update** cancels the tuple $\langle U, R\rangle$ from each array $REACH[A_i]$ since activity A_i can no longer be reached through the corresponding arc $U \rightarrow R$. If a user U changes his role from R to R', the tuple $\langle U, R, R'\rangle$ is added to $Updates$: **Update** removes the

tuple $\langle U, R\rangle$ from each array $REACH[A_i]$ while the tuple $\langle U, R\rangle$ is added to all the arrays $REACH[A_i]$ of all the activities A_i that the role R' is authorized to perform.

Algorithm 2 illustrates the execution steps of a business process instance $\mathsf{BP}_{\mathsf{inst}}$. When an activity has to be executed, first the algorithm finds a potential owner for the activity. Once the activity is performed, the algorithm modifies $REACH[A_i]$ so that the array contains only the tuple $\langle U, R\rangle$ corresponding to the user U who has executed it. If A_i is in separation of duty or binding of duty with other activities A_k, the algorithm modifies $REACH[A_k]$ so that A_k can be reached only through arcs $U' \rightarrow R'$ that satisfy the constraints. After the update is complete, the algorithm checks if there were changes in UA relation. The function Change() returns 0 if there are no updates to UA, 1 otherwise. If there are no updates, the algorithm 2 proceeds with the execution of the next activity, otherwise the algorithm **IsResilient** first checks whether $\mathsf{BP}_{\mathsf{inst}}$ is dynamically resilient.

Algorithm 3: IsResilient

Require: $REACH[A_1], \ldots REACH[A_n]$ where $A_k \in \mathsf{BP_i}$, $A_{NotExecuted}$.
Ensure: A_k activity identifier.

1: $Resilient$:= `false`;
2: **for** $A_k \in A_{NotExecuted}$ **do**
3: **if** $REACH[A_k] \neq$ empty **then**
4: $Resilient$:= `true`;
5: **for** $BoD_r(A_i, A_k)$ **do**
6: **if** $\langle A_i, U, R\rangle \in History$ and $\neg REACH[A_k]$.`contains`($\langle U', R\rangle$) **then**
7: $Resilient$:= false;
8: `break`;
9: **for** $SoD_r(A_i, A_k)$ **do**
10: **if** $\langle A_i, U, R\rangle \in History$ and $\neg(REACH[A_k]$.`contains`($\langle U', R'\rangle$) and $R' \neq R$) **then**
11: $Resilient$:= false;
12: `break`;
13: **for** $BoD_u(A_i, A_k)$ **do**
14: **if** $\langle A_i, U, R\rangle \in History$ and $\neg(REACH[A_k]$.`contains`($\langle U, R'$) **then**
15: $Resilient$:= false;
16: `break`;
17: **for** $SoD_u(A_i, A_k)$ **do**
18: **if** $\langle A_i, U, R\rangle \in History$ and $\neg(REACH[A_k]$.`contains`($\langle U', R'\rangle$) and $U' \neq U$) **then**
19: $Resilient$:= false;
20: `break`;
21: **if** $Resilient$ == false **then**
22: **return** A_k;

IsResilient verifies that each activity A_k in the set $A_{NotExecuted}$ can be reached through an hyperpath $A_k \rightarrow G \rightarrow R' \rightarrow U' \rightarrow R' \rightarrow A_k$ that does not violate any

authorization constraint. If for some activity A_k does not exist an hyperpath that satisfy the authorization constraints, the business process instance $\mathsf{BP}_{\mathsf{inst}}$ is not resilient to the changes in UA relation, and the algorithm returns the identifier of the activity A_k.

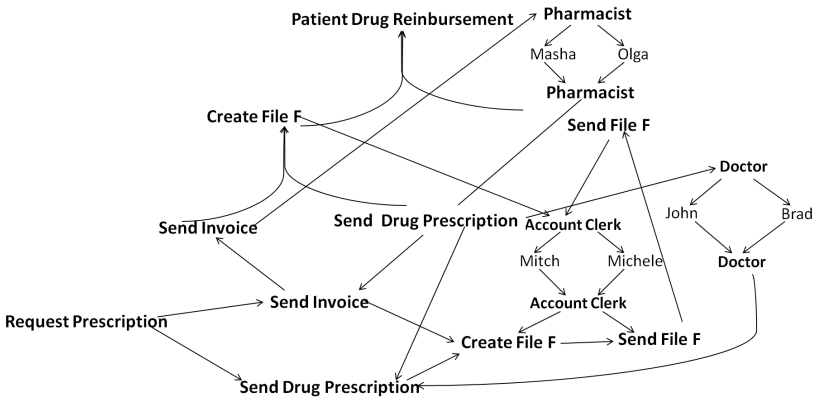

Fig. 2. Hospital Organizational Model

$REACH$[Send File F]	{Michele, Account Clerk },{Mitch, Account Clerk }
$REACH$[Create File F]	{Michele, Account Clerk },{Mitch, Account Clerk }
$REACH$[Send Invoice]	{Masha, Pharmacist }, {Olga, Pharmacist }
$REACH$[Send Drug Reimbursement]	{John, Doctor }, {Brad, Doctor }

Fig. 3. Arrays $REACH[A_k]$ for File F process

Example 1. Figure 2 illustrates the organizational model of an hospital in the North of Italy where the File F business process is deployed, while Figure 3 lists the arrays $REACH[A_k]$ for the activities "Send Invoice", "Send Drug Prescription", "Create File F", and "Send File F" of the File F process. The top goal of the hospital is "Patient Drug Reimbursement" that is decomposed in two subgoals "Send File F" and "Create File F" which on its turn is decomposed in the goals "Send Invoice", and "Send Drug Prescription". The goals "Send File F", "Create File F", "Send Invoice", and "Send Drug Prescription" are satisfied when the corresponding activities are executed. The set of roles inside the hospital consists of the roles "Pharmacist", "Doctor", and "Account Clerk": "Pharmacist" is responsible for the fulfillment of the goal "Send Invoice" and it is authorized to execute the activity "Send Invoice"; "Doctor" has to satisfy the goal "Send Drug Prescription" and it is granted the execution of activity "Send Drug Prescription"; "Account Clerk" has to fulfill the goals "Send File F" and "Create File F" and is authorized to perform the activities "Send File F", and "Create File F". Assume that

there is a separation of duty constraint $SoD_u(CreateFileF, SendFileF)$ between the activities "Create File F" and "Send File F". Moreover, assume that the execution of an instance of the File F business process has started and that the activities "Send Invoice", "Send Drug Prescription", and "Create File F" have been executed by the users Masha, John and Michele respectively. Thus, the arrays $REACH$[Send Drug Reimbursement], $REACH$[Send Invoice], $REACH$[Create File F], and $REACH$[Send File F] are updated as follows:

- $REACH$[Send Drug Reimbursement] := {Masha, Pharmacist }
- $REACH$[Send Invoice]:= {John, Doctor }
- $REACH$[Create File F]:= {Michele, Account Clerk }
- $REACH$[Send File F]:= {Mitch, Account Clerk }

During the execution of the activity "Create File F", the user Mitch becomes unavailable. Thus, since Mitch is the only user who can perform the activity "Send File F" because of the separation of duty constraint between activities "Create File F" and "Send File F", but he is not available the instance of the File F business process is not resilient to the changes in UA relation.

5 System Architecture

We propose here a system architecture that allows to integrate our approach to resiliency into a business process management system for SOA-based business processes where some activities are performed by users. We assume that the SOA-based business process is specified using the WS-BPEL [2] standard and the BPEL4People [1] specification to denote the activities in the process that need to be performed by a user. The system architecture consists of three main components (see Figure 4): the *Business Process Execution Engine*, the *Monitor Service*, and the *Enforcement Service*. The business process engine is responsible for scheduling and synchronizing the various activities within the business process according to the specified control flow. The *Monitor Service* provides to the business process administrator a graphical interface which allows the administrator to import and visualize the organizational model and apply changes to it. When the organizational model is imported into the *Monitor Service*, the service computes the arrays $REACH[A_i]$ for each activity in the business process and stores them in a repository which is shared with the *Enforcement Service*. If the business process administrator changes the relation UA the service automatically updates the arrays $REACH[A_i]$. The *Enforcement Service* has two WSDL interfaces, one for the process and one for the users that are involved in the execution of a business process instance. The interface for the process provides the operations for starting and completing the execution of a business process activity that must be performed by a user. When a human activity A_i has been executed, the interface updates the array $REACH[A_i]$, and if the activity A_i is in binding of duty or separation of duty with other activities A_k, the interface updates also the arrays $REACH[A_k]$ as described by the algorithm 2. The second interface allows users to visualize the activities they can claim, and to claim and execute them. When a user claims the execution of an activity A_i, the interface verifies that the user is entitled to execute the activity, and then checks whether the *Monitor Service* has

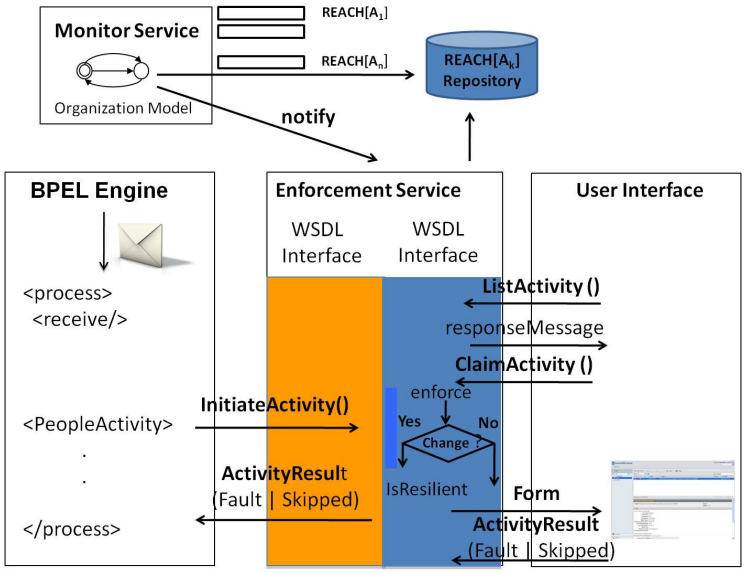

Fig. 4. System architecture

notified a change to the UA relation. If the UA has changed, the interface checks if the business process instance is resilient. If not, the interface notifies the *Monitor Service*.

6 Related Works

The problem of how to include humans as additional participants of a SOA-based business process and how to verify the authorizations users have on the execution of business process activities is gaining attention.

BPEL4People [1] is a recent proposal to handle person-to-person WS-BPEL business process. With respect to our proposal, 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 duty.

The proposals of Wang et al.[6] and Paci et al. [5] are the one more closely related to our work. Wang et al. investigate the resiliency problem in workflow systems, and propose three different notions of resiliency and investigated the computational complexity of checking resiliency. In static resiliency, a number of users are absent before the execution of a workflow instance, while remaining users will not be absent during the execution; in decremental resiliency, users may be absent before or during the execution of a workflow instance, and absent users will not become available again; in dynamic resiliency, users may be absent before or during the execution of a workflow instance and absent users may become available again. Paci et al. have investigated the statical

resiliency problem in the context of 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, authorization constraints, such as separation and binding of duty, and resiliency constraints that specify the minimum number of users that have to be available for the execution of an activity. They propose an algorithm to determine if a WS-BPEL process is statically resilient to user unavailability.

With respect to Wang et al. and Paci et al. proposals, in this paper we have proposed an approach to investigate that a business process instance is dynamically resilient not only to users unavailability but also to users changing their role.

7 Conclusions

In this paper, we have investigated the problem of *dynamic resiliency* to changes in the assignment of users to roles. We have proposed an approach to verify that when there are changes in the assignment of users to roles, the execution of a business process instance can still terminate. The approach is based on finding a potential owner for those activities the execution of which is still pending. The potential owner is a user who is assigned to a role that is entitled to execute the pending activity and is responsible for the achievement of the goal fulfilled by the execution of the activity.

To check whether a business process instance is dynamically resilient, we have modeled an organization as a digraph where the vertexes represent the organizational business goals, the activities that implement such goals, the organizational roles, and the users assigned to these roles. Then, for each activity we have traversed the graph to find those paths that correspond the activity's potential owners.

We are planning to extend this work in several directions. First, we would like to consider the problem of *escalation*. Escalation takes place if an activity does not meet its modeled time constraints. If this occurs, a notification is sent to the users responsible for the fulfillment of the goal the achievement of which depends on the execution of the activity. We also want to represent the changes in the users to roles assignment relation as graph patterns and apply graph transformation techniques to automatically update the organizational model and the arrays $REACH[A_i]$ on the basis of which we evaluate whether a business process instance is dynamically resilient.

References

1. A. Agrawal et al. Ws-bpel extension for people (bpel4people), version 1.0, 2007. http://www.adobe.com/devnet/livecycle/pdfs/bpel4people_spec.pdf.
2. A. Alves et al. Web services business process execution language, version 2.0, oasis standard, April 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.
3. Giorgio Ausiello, Roberto Giaccio, Giuseppe F. Italiano, and Umberto Nanni. Optimal traversal of directed hypergraphs. Technical report, 1992.
4. IT Governance Institute. *IT Governance Implementation Guide: Using COBIT and ValIT*. Isaca, 2007.
5. F. Paci, R. Ferrini, Y. Sun, and E. Bertino. Authorization and user failure resiliency for ws-bpel business processes. In *Sixth International Conference on Service Oriented Computing (ICSOC)*, 2008.

6. Q. Wang and N. Li. Satisfiability and resiliency in workflow systems. In *Proceedings of ESORICS*, pages 90–105, 2007.