Enhancing Workflow Systems Resiliency by Using Delegation and Priority Concepts

Hanan El Bakkali
Information Security Research Team (ISeRT)
Université Mohammed V – Souissi
ENSIAS, Tunisia
elbakkali@ensias.ma

Journal of Digital Information Management

ABSTRACT: *Enforcing dynamic access control constraints in workflow management systems (WFMS) is a very important requirement with regard to security issues. However, respecting those constraints may prohibit the completion of a workflow instance in the case of the lack of authorized users. Such situation is known in the literature as a WSP (Workflow Satisfiability Problem). The ability of a WFMS to use different methods to bypass a WSP situation is often seen as a resiliency property.*

In this work, we propose a new approach that aims to enhance the resiliency of a WFMS while meeting –at run time- the main workflow dynamic access control requirements. In fact, by using both delegation and priority concepts it is possible to find a user which is as suitable as possible to perform the current task instance with lesser security risks.

Categories and Subject Descriptors:
D.4.6 [Security and Protection]: Access Controls, **H.4 [Information Systems Applications]**: Workflow Management

General Terms:
Security and Access Control, Workflow Management

Keywords: Workflow Satisfiability Problem, Dynamic Access Control Constraints, Resiliency, Delegation, Priority

Received: 18 April 2013, Revised 4 June 2013, Accepted 9 June 2013

1. Introduction

Workflow management systems (WFMS) are more and more used in today's organizations. This success is mainly due to the significant gain of productivity that they lead to and to the level of maturity they have reached. However, from the security point of view, many efforts need to be done in order to respond to the security requirements of workflow systems.

Among these requirements, Access Control seems to be the most critical. In many WFMS, it is enforced on the basis of the well known RBAC model. By assigning permissions to roles played by users rather than directly to users, it has greatly facilitated the access control administration.

However, enforcing access control constraints in a Role-based WFMS may prevent a workflow instance from being completed. In fact, in many cases a workflow instance execution might be stopped if the system fail to find an appropriate user to assign to the current task instance with regard to dynamic access control requirements or availability constraints (due to sickness, overwork, holiday,..).

Thus, it is an important requirement of a workflow specification to guarantee the satisfiability of the workflow, which requires that some set of authorized users can complete a workflow as it is highlighted in [1].

In this paper, we will call as '*WSPS*' (Workflow Satisfiability Problem situation) a situation where the WFMS is unable to find (for availability or security reasons) an authorized user to assign to the current task instance in a workflow instance. For example, a situation where a critical medical diagnostic (which is specified as a workflow process) is delayed due to the lack of an authorized doctor to manipulate a specific material is a WSPS where both high failure resiliency and security requirements are important.

Resiliency refers to the ability of a WFMS to use suitable strategies to bypass a WSPS and find out a user who can execute a task instance haven reached an impasse

(as specified in [2-3]).

Let us consider the following example to illustrate the importance of such strategies. We suppose that we have an "*Order process*" workflow $W1$ composed of 4 sequential tasks ($T1$, $T2$, $T3$, $T4$) with $T3$ optional. As shown in Figure 1 below, a workflow instance execution of $W1$ has encountered a WSPS when attempting to assign an authorized user to a task instance of $T4$. In fact, we assume a scenario where the initially authorized users (with regard to their roles) to execute $T4$, which are $\{U1, U2, U3\}$, are no longer able to execute it. For example, we have a dynamic access control constraint that specifies that $T1$ and $T4$ have to be executed by different users. Moreover, $U2$ is sick and $U3$ is very busy so they couldn't execute $T4$.

As a solution to this situation, we can either relax the access control constraint and then assign U1 to T4 (which is a flexible but very risky choice) or assign another user U4 to T4 (on the basis of a specific delegation) that is as suitable as possible to this task (which is both flexible and less risky choice). This choice is based on both access control constraints (managed generally by an '*authorization module*' of the WFMS that has to support role delegation) and performance or quality requirements.

The goal of bypassing a WSPS is to reconcile security constraints as dynamic access control requirements and performance/failure resiliency aspects. Indeed, a lack of flexibility results in low performance while excessive flexibility could allow the wrong and/or unauthorized users to execute tasks within a workflow leading to low quality and potential security risks.

In this paper, we present a new approach that is based on both delegation and priority concepts to bypass such situations.

We will first describe in section 2 useful RBAC and delegation backgrounds. In section 3, we present some related work. Our contribution is presented in section 4 and the architecture proposed for the implementation of our approach is presented in section 5. Finally, we summarize the discussions and conclude in section 6.

2. RBAC and Delegation Background

RBAC is adopted as an ANSI/INCITS standard since 2004 [4]. It is widely considered as the most suitable access control model for today business organizations. By granting permissions to roles played by users rather than to users themselves, it has greatly facilitated the security administration. Likewise, the modification of access controls is not required each a user joins or leaves an organization or a role within the organization.

RBAC consists of 4 main entities: users, roles, permissions and sessions. Authorization decisions are based on the triple (UA, PA, RH) where UA is the user-role assignment relation, PA is the permission-role assignment relation and RH represents hierarchical relations between roles. A session is a concept that is bound to a single user and allows the user to activate the permissions (for example, when willing to execute a task that needs that permissions) of a subset of roles to which he/she belongs.

When RBAC is used as an access control model in WFMS, tasks become the fifth main entity to consider. Thus, a lot of researchers have proposed new extensions and variants for RBAC to support access control in workflows.

Among the important issues related to RBAC and its new proposed extensions is delegation. Delegation which is largely recognized as an important mechanism to provide resiliency and flexibility in the assignment of tasks to authorized users or roles especially in case of lack of resources, urgent delays, etc. Nevertheless, current WFMS still do not integrate delegation concept sufficiently.

Delegation generally implies two users, a delegator (the user who performs a delegation) and a delegatee (the user who receives a delegation). It could concern an elementary permission (or right), a role or even a role hierarchy. It could be permanent or temporary, grant-based (the delegator keeps its delegated permissions) or transfer-based (the delegator loses its delegated permissions during delegation), discretionary or administrative, etc. In [5], one could find a list of principal characteristics of the delegation concept.

In the literature, administrative delegation allows an administrative user that does not necessarily have the role or the permissions to perform the delegated task to assign it to a delegatee, however, discretionary delegation (also called user-delegation) allows the delegator to assign a subset of his available permissions to another user. In this paper, we will use a special case of administrative

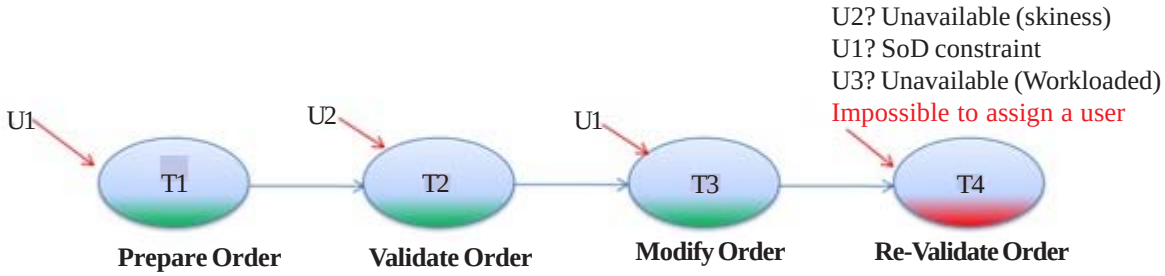

Figure 1. An example of a WSPS scenario

delegation where the delegator is not an administrative user but the WFMS itself. We will call it a WFMS delegation (cf. section 4).

To summarize, from a manager viewpoint, delegation is a suitable mechanism to offer resiliency and flexibility in workflow systems. However, from an access control perspective, a delegation operation may lead to violation of security policies, especially SoD (Separation of Duties) and LP (Least Privileges) constraints.

SoD aims to prevent a single user to hold enough power or privileges to commit a fraudulent act. For example, a user must not be allowed to make an order and then validate the same order. In the literature, there is always a distinction between two kinds of SoD: Static SoD (SSoD) which occurs at administration-time and Dynamic SoD (DSoD) which occurs at run-time. In workflow systems, DSoD constraints are essential to consider tasks dependencies and task execution history in a particular workflow instance.

Enforcing SoD constraints is often based on the definition of conflicting entities (roles, users, permissions). For example, the same user is never allowed to be a member of (in SSoD) or to activate (in DSoD) conflicting roles simultaneously. In some works, it is also question of conflicting tasks (as in [6-8]) and conflicting workflows (as in [6]). For instance, conflicting tasks may not be performed by the same user or conflicting users in the same workflow instance and the same user mustn't participate in two conflicting workflows. A trivial example of two conflicting workflows is a workflow that represents an audit process of tasks belonging to the second workflow. LP aims to ensure that each user has only the needed permissions to perform the task he is preparing to execute and no more. Enforcing LP is particularly important when dealing with role hierarchy and role delegation in RBAC systems. So, LP restrictions must be done to prevent that privileges acquisition through role inheritance or delegation could give a user more privileges than he needs to carry out his current tasks.

3. Related Work

Workflow systems are becoming in the last few years an indispensable tool for automating business processes in large scale organizations. Several researchers -from the security field- have focused their work on security issues in those systems and many of them are interested in reducing the gap between security and flexibility requirements.

Crampton et al. addressed the satisfiability problem in the context of role-based workflow systems while supporting user delegation mechanisms [1]. The authors showed how it is important to ensure that permitting a user delegation request does not render the workflow unsatisfiable with regard to authorization information. However, in our work, we focus on how delegation (more precisely WFMS delegation) could be a solution to a WSPS. Authors of [1] judge that there are six questions a reference monitor must consider while supporting user delegation in Workflow systems. This work focus only on this question: Are all access control and business constraints satisfied if a current task instance t is delegated to a user u by the WFMS?

Barka and Sandhu [5] proposed a role-based delegation model based where users could only delegate their roles but delegating a piece of permission from a role is not supported. In our approach, to enforce LP principle, delegating a role by a WFMS to a user occurs in the context of the execution of a task instance and allows only the activation of the permissions required to execute this task.

Lowalekar et al. proposed in [3] techniques which focus on finding user-task assignments in workflow systems satisfying policy and SoD constraints such that the workflow can be completed even if K users 'fail'. They considered two types of scenarios: Purely static (all user-task assignments are fixed at administration time) and purely dynamic (user-task assignments can change at runtime to get more failure resiliency). However, their work doesn't address delegation issues.

Atluri and Warner [9] proposed a model that extends the notion of delegation in workflow systems to allow conditional delegation that depends on time, workload and task attributes. They addressed the problem of assigning users to tasks preventing the violation of authorization constraints. Their model considers only user delegation as [1].

Wainer et al. [10] focused on the set of users that can perform workflow tasks. They consider that determining - with an order of preference - who among the users are the most appropriate to execute each task is one of the main responsibilities of a workflow system. They also a model called DW-RBAC which allows for user-grant delegations (that could be multiple) to be specified for a particular workflow instance and later revoked when it is no longer required. As explained above, our purpose is to use WFMS -transfer- delegation to bypass WSP situations, so there is no need of multiple-step delegations that increase the complexity and the security risks of such approaches.

In their paper [11], Kumar et al. viewed work distribution in WFMS from both security and performance perspectives. They tried to combine the security advantages of a push approach (A task instance is pushed to a single user) with the performance benefits of a pull mechanism (A user pulls task instances from a view of a common pool of tasks instances) by considering both the deadlines of various tasks instances and the workloads and suitability of users. The distribution mechanisms they proposed are closely linked to delegation. In our approach, a WMFS consider workload of users in addition to information related to roles, historic of tasks execution

and workflows and tasks priorities.

Cao et al. [12] highlighted the importance of a good design of work-resource allocation strategies in WFMS. They proposed a policy-based authorization model for workflow-enabled dynamic business processes that considers three types of policy: requirement policy, scenario policy and substitution policy. Each policy is specified in a Task Authorization Policy Language (TAPL). For instance, a substitution policy states that if roles defined by requirement policies cannot be found, the roles can be substituted by other roles. In our work, we propose that for each role we assign - at the administration-time- a set of *potential role delegates.*

In [13], Wang et al. proposed a novel source-based enforcement mechanism for workflow authorization systems that aims to achieve both security and efficiency. They also showed how malicious users may collude to violate access control policies by using user delegation and proposed a formal definition of secure delegation. Our work uses Workflow delegation to enhance flexibility and resiliency without violating dynamic access control constraints and carefully chooses delegates (at both administrative and run time) so that no "*risky*" role delegation would occur.

In [14], Toahchoodee et al. provided a formal approach for choosing delegatees that first evaluates the trustworthiness of candidates -in the context of the task to be executed- before verifying that the chosen candidate (the most trustworthy) does not violate security constraints. This trustworthiness depends on factors as attributes (or roles), experiences and recommendations. In our work, we do not specify yet how the choice between several delegates that do not violate any dynamic access control constraints will be done (cf. the step 2.2. in the algorithm bellow). In future work, this choice could be based on some criteria as user qualifications, workload, … and depending on the nature of the Workflow, trustworthiness could also be a choice criterion.

The above cited works reflect the increasing importance accorded by researchers -particularly from the security field- to resiliency/flexibility issues in workflow systems and particularly to delegation issues. In this work, delegation is used carefully by the WFMS in order to bypass WSP situations without violating access control constraints.

4. The Proposed Approach for bypassing WSPS

An automated business process or workflow could be seen as a set of ordered tasks that must be carried out by users. Each Task requires a set of permissions to complete. Often, in RBAC-based systems, tasks are also linked to roles that could execute them.

A WFMS, at each workflow execution tries to assign each task instance to a single user depending on his roles and availabilities. This assignation must respect the access control policy of the enterprise and particularly SoD and LP constraints.

In what follows, we present the proposed delegation-based approach that takes place at **run-time** in order to increase the flexibility of the WFMS. In fact, as it has shown before, a WSPS could occur at the level of any task instance. In such a case, the proposed approach could be useful by delegating this task instance to another user that will not create a conflict situation and that respects some useful conditions.

In this approach, we introduce a WFMS delegation that depends on roles. This role delegation allows only the activation of the required permissions to the execution of the current task and only during this execution (temporary delegation).

Let's note that we consider in this approach only tasks that require human intervention.

4.1 Preliminaries

4.1.1 Assumptions

We assume that all work about role engineering is already done at administration time with respect to access control constraints. We also assume that the WFMS is able to identify conflicting entities (users, roles, tasks, workflows, etc) at both administration and run times. It is also supposed to be informed about the work charge and the availability of each user at each instant.

Moreover, we suppose that each workflow within the workflow system has a task granularity that respects both the business and access control constraints (mainly, SoD). In other words, each workflow specification should not contain a composite task that requires absolutely (for business or SoD reasons) more than one user.

As most existing WFMS, we assume that the WFMS logs status changes in the execution history of each workflow instance.

Finally, we suppose that each workflow within the WFMS has task-roles and task-permissions assignments that respect –at administration time- both business and access control constraints.

4.1.2 Preparation step at roles level

Our approach is based on a preparation step that is complementary to "*role engineering*" steps, which takes place at administration-time and it is divided on two sub-steps:

4.1.2.1 Role-type/task association

This sub-step is somewhat supported by most RBAC based WFMS. It consists of assigning to each task a role or a set of roles that can -a priori- perform the task T. This assignation function is denoted here by $R(T)$. All the users belonging to a role from $R(T)$ have the capability to perform

the task *T*. In [11], they introduce the notion of suitability as the inherent qualification of a user to perform a task instance which could be inherited from a role/task suitability table.

In fact, in most organizations there is often a most suitable role for each task. For example, in hospitals, the role "*Doctor*" is typically responsible for "*making a diagnostic*" for patients. Thus, in our approach, $R(T)$ is ordered regarding to role suitability for performing T. R-Type(T) denotes the first element of $R(T)$ which is the most suitable role for the execution of T from both security and business efficiency perspectives.

4.1.2.2 Potential role delegates

This second sub-step allows to assign to each role R an ordered set of potential delegatees roles per task T (for which $R \in R(T)$) and per company environment. This set denoted $Delg(R, T, CE)$ could contain roles that have not enough qualifications or permissions to execute T. Lack of permissions is not a problem, because during run-time the user chosen by the WFMS as a delegatee will '*receive*' temporarily the role R and obtain the permissions necessary to execute T. However, lack of qualifications is the price to pay in order to have flexibility and resiliency especially in emergency contexts.

$Delg(R, T, CE)$ may contain some roles of $R(T)$, but it is not necessarily a subset of it. We recommend also that $Delg(R, T, CE)$ contains some of senior roles of R in the RH. But, it has to be free of any conflicting role with R. In this paper, we propose five possible values for CE parameter (company environment): emergency, lack of users, enough users, confidentiality and quality. This parameter permits a delegation that is more flexible and/or more secure.

Let's note that we can reduce this step by assigning to a role the same set of delegatees roles independently of the task or the CE parameter. This could be better only if such fine-grained delegation is not required (i.e. $Delg(R)$ instead of $Delg(R, T, CE)$).

To summarize, the goal of this step is to allow a faster choice -during the delegation process- by the WFMS of an adequate user (that belongs to a role which is as suitable as possible from both business and security viewpoints) for executing the current task in the case of a WSPS.

4.1.3 Preparation step at workflows level

At administration-time, we assign to each workflow W specified within the WFMS a certain criticality: $cr(W)$ with 4 possible values {0.25 (reduced), 0.5 (medium), 0.75 (important), 1 (critical)}.

This value will help the WFMS during the delegation process of bypassing a WSPS situation to decide which task execution (inside a workflow instance) to suspend if necessary.

4.2 Some Definitions

4.2.1 Workflow definitions

We define first, $WF = \{W1, \ldots, Wn\}$ the set of all workflows modeling correctly business processes in the enterprise.

For the purpose of this paper, we propose a simple workflow definition in which a workflow is represented as a set of tasks. A workflow instance is a specific workflow execution of a workflow.

4.2.2 Task characteristics

We retain seven characteristics for a task:

- **Atomic vs composite:** some tasks are composed of other sub-tasks however others are atomics and then indivisibles.

- **Delay sensitive vs delay-insensitive:** some tasks have delay-constraints while others could be delayed.

- **Delegable vs non-delegable:** some tasks could be non-delegable, for importance or confidentiality reasons.

- **Critical vs optional:** some tasks could be optional as in article review processes, a review by a third reviewer is sometimes optional.

- **Human vs automatic Task:** A human task represents a piece of work that may be performed by a human worker (which we call a user) either assisted by an information system tool or without any information system support. Automatic tasks are beyond the scope of our approach (because there are generally not concerned by SoD constraints nor suitable for role delegation)

- **Interruptible vs non- interruptible:** A task is said to be non-interruptible if once it starts execution it cannot be interrupted and its execution resumed later. Most pure human tasks are in theory interruptible (by definition) but for example, a surgery mustn't be interrupted.

- **Preemptable vs non-preemptable:** According to [15] a task is said to be non-preemptable if once it begins execution using a specific user, it has to be completed without replacing that that user.

4.2.3 Task Representation

A task T is represented by a triplet (W, PT, T-$kind$) where:

• W: the workflow that contain T

• PT: The set of all permissions which are necessary to perform T.

• T-$kind$: A succession of 7 digits that identify the seven characteristics of the task cited above. If the characteristic holds the digit is 1 elsewhere it will be 0.

4.2.4 Task instance states

The basic states for a task instance are Initial, Assigned, Executed, Cancelled, Failed, and Completed [16]. Here we consider the following states:

- **Initiated (or 'activated'):** it is its turn in the execution plan of the workflow instance but is not yet assigned to a user

- **Assigned:** its execution has not started yet but a user was assigned to it.

- **Started:** the assigned user has started its execution.

- **Resumed:** temporarily stopped and the following tasks could not start until its execution.

- **Canceled:** the execution of the task is stopped for good but the following task within the workflow could begin.

- **Failed (or aborted)**: the task execution was stopped for good before its completion and causing by the way the whole workflow failure.

- **Completed:** the task execution was successively completed.

4.2.5 Task instance representation

A task instance refers to a single instance of a particular task definition within a particular workflow instance. We represent a task instance t by the tuple (T, w, st) where:

• T is the task for which t is an instantiation

• w is the workflow instance that contains t.

• st indicates the status of t (see the definition above).

We consider a function *status* that returns the status of a task.

4.2.6 User workload definition

To each user u we associate a workload status denoted as $L(u)$. We only consider three possible load-status values which are {available, loaded, unavailable}. It is intended in future work to extend these three qualitative values to numeric ones (i.e. a percentage) for more precision. The status '*unavailable*' concerns cases in which a user is unable to execute any task.

This user information aims to facilitate dynamic load balancing by the WFMS delegation process among users that can perform the current task. It has to be estimated on the basis of user-task assignations information and other useful information (illness, vacation, efficiency …). However, its calculation is beyond the scope of this paper.

4.2.6 WFMS delegation definition

We define a WFMS delegation as an administrative delegation where the delegator is the WFMS itself and the delegatee is a user that is chosen with respect to some conditions. It is a role delegation which is temporary (during the execution of the current task), transfer-based (only the delegatee user have the delegated permissions during delegation) and partial (no inheritance of junior roles). This delegation is a single-step one that cannot be further delegated.

We recommend that this WFMS delegation be unilateral (without negotiation) in the sense that the WFMS alone can decide on delegating a role to a user in order to execute a task and the chosen delegatee has to accept its choice.

4.2.7 Context definitions

We define first the concept of '*conflict window*' denoted by CW that represents a period of time beyond it no conflict of interest could subsist. Indeed, after a certain period the participation of a user in a workflow instance w (via task execution) will not still have an effect in matter of conflict and security risks on its participation in even a workflow in conflict with w. We define two kinds of contexts:

- A local context to a workflow instance w, denoted by $Ctl(w)$, that depends only on the history and the current state of w. It represents the set of pairs: (u, t) where u is a user who already participated in w by completing a task instance t.

- A global context that concerns the whole enterprise and all its workflows (WF). This context denoted CT is the set of all local contexts of all executed or under-execution workflows of WF during the conflict window.

We define also a set $U(CT)$ that represents all the users who are executing current task instances in under-execution workflows instances of CT. This set will be useful for bypassing some WSPS.

4.2.8 Task priority definitions

A task instance priority is calculated on the basis of the workflow criticality (cf. section 4.1) and the task priority as follows:

$$pr(t) = pr(T) \times cr(W(t)) \qquad (1)$$

pr is a function that returns a value in [0,1].

For each task T, we propose the following algorithm to calculate $pr(T)$ with regard to the value of T-kind:

IF T is optional then $pr(T) := 0$
ELSE (T is critical)
  IF T is delay-sensitive **then**
    IF T is non-resumable **AND** non-preemptable **then**
      $pr(T) := 1$
    Else
      IF T is non-delegable then $pr(T) := 1$
      Else
        IF T is resumable **AND** preemptable **then**
          $pr(T) := 0,5$
        Else $pr(T) := 0,8$
Else (T is non- delay-sensitive)
  IF T is non-resumable **AND** non-preemptable **then**
    $pr(T) := 0,8$
  Else
    IF T is non-delegable **then** $pr(T) := 0,8$
    Else
      IF T is resumable **AND** preemptable then
        $pr(T) := 0,25$
      Else $pr(T) := 0,5$

4.3 Applying the delegation Process

4.3.1 Identifying a WSP situation

We talk about a WSPS with regard to a set of users U, when the WFMS attempts to assign a user from U to an initiated instance t within a workflow instance w and then it encounters that each user of U couldn't execute t because he is unavailable or he corresponds to one of the following conflict situations:

- $S1$: the conflict is due to precedent users activities in $Ctl(w)$ (due to SOD or cardinality or binding constraints related to conflict between users or tasks or roles)

- $S2$: the conflict is due to precedent or current users activities in conflicting instances of workflows with W that are in CT.

Describing in details how these kinds of conflicts are detected by the WFMS authorization/conflict module is beyond the scope of this paper.

For clarity reasons, a WSPS will be denoted in what follows by $WSPS(U, t)$ = true. When $WSPS(U, t)$ is false, the biggest sub-set of U that not causes any conflict with regard to t is denoted non-conflict (U, t) .

4.4 Delegation process algorithm

Before a critical task instance t of T (that belongs to a workflow instance w) is assigned to a user, the WFMS determines the set of users that are assigned to its role-type: $U(R\text{-}type(T))$. If the WFMS identifies a WSPS for $U(R\text{-}type(T))$ with regard to CT and $Ctl(w)$, it starts a delegation process that returns true if it was able to bypass this WSPS and false elsewhere. This algorithm $DP(R\text{-}type(T))$ operates as follows:

Initially:

status (t) = initiated

WSPS $(U(R\text{-}type(T)), t)$ = true (with regard to CT and $Ctl(w)$)

$Delg(R\text{-}type(T), T, CE) = \{R1,\ldots,Rn\}$ (with $n > 0$).

1. i: = 1, DP: = false // *a boolean value that concerns the success or the failure of the delegation process*
2. **WHILE** $i <= n$ **DO**
   2.1. **IF** WSPS $(U(Ri), t)$ = true **THEN** i: = $i + 1$
   2.2. **ELSE**
      a. ***Select*** uij that his load-status=available from non-conflict $(U(Ri), t)$ // *uij exists and its selection is based on some calculation*
      b. $t \leftarrow uij$ // *uij is assigned to t*
      c. $uij \leftarrow R\text{-}type(T)$ //*a role delegation to uij with only the needed permissions to complete t*
      d. status (t): = assigned
      e. DP: = true
      f. **RETURN** (as value) DP (i.e. true)
3. $(i > n)$ **IF** WSPS $(U(CT), t)$ = true **THEN** RETURN (as value) DP (i.e. false)
4. **ELSE**
   4.1. $U0$: = non-conflict $(U(CT)) \cap U(R\text{-}type(T))$
   4.2. i: = 1
   4.2. **WHILE** $i <= n$ **DO**
      a. Ui: = non-conflict $(U(CT)) \cap U(Ri)$;
      b. i: = $i + 1$
   4.3. i: = 0;
   4.4. **WHILE** $i <= n$ **DO**
      a. **IF** $Ui \neq \varnothing$ **THEN**
         $u\text{-}pr$: = priority-$DP(Ui)$
         IF $u\text{-}pr \neq$ '*null*' **THEN**
            DP: = *true*
            $t \leftarrow u\text{-}pr$
            $u\text{-}pr \leftarrow R\text{-}type(T)$ //a role delegation with only the needed permissions to complete t
            status (t): = assigned
            RETURN DP (i.e. true)
         ELSE $(u\text{-}pr =$ '*null*') i: = $i + 1$
      b. **ELSE** $(Ui = \varnothing)$ i: = $i + 1$
   4.5. $(i > n)$ **RETURN** DP (i.e. false)

The call of priority-$DP(Ui)$ corresponds to the following steps (presented separately for the sake of clarity):

1. **FOR** each $u'j \in Ui$ who was previously assigned to the current task instance $t'j$ (which has a status '*assigned*' or '*started*') of a workflow instance $w'j$ that is not yet completed, **DO**
   1.1. **IF** $pr(t) > pr(t'j)$ **THEN**
      a. **IF** $t'j$ is resumable and non-preemptable **THEN** status $(t'j)$: = resumed; **RETURN** $u'j$ (as the value of the function)
      c?. **IF** $t'j$ is optional **THEN** status $(t'j)$: = cancelled ; **RETURN** $u'j$ (as the value of the function)
2. **RETURN** (as value) '*null*'

If the returned value of the delegation process algorithm is false, this means that it has failed to find an authorized and suitable user to perform t and in this case the WFMS could suspend t if it is resumable, or apply another strategy to resolve this situation (as in [17]) or declare the failure of the whole workflow instance completion.

As mentioned before, in step '2.2', the WFMS selects a user among $U(Ri)$ ($Ri$is the i^{th} potential role delegatee of $(R\text{-}type(T)$ by using some calculation on the basis of useful criteria (user qualifications, workload, etc.) that the algorithm could do quickly. The details of this calculation are outside the scope of this paper.

The complexity of this algorithm (with a specific calculation for the step '2.2' that has at most a complexity in $O(m)$) is in the order of $O(n * (2 * m + 1))$ (more briefly $O(2 * n * m)$) where:

n = *the cardinality of* $Delg(R\text{-}type(T), T, CE)$.;

m = *the Max of cardinality* $(U(Ri))$ *from* i=1 *to* n);

In real-world, n and m are normally small.

5. The proposed implementation architecture

The implementation/integration of our algorithm does not

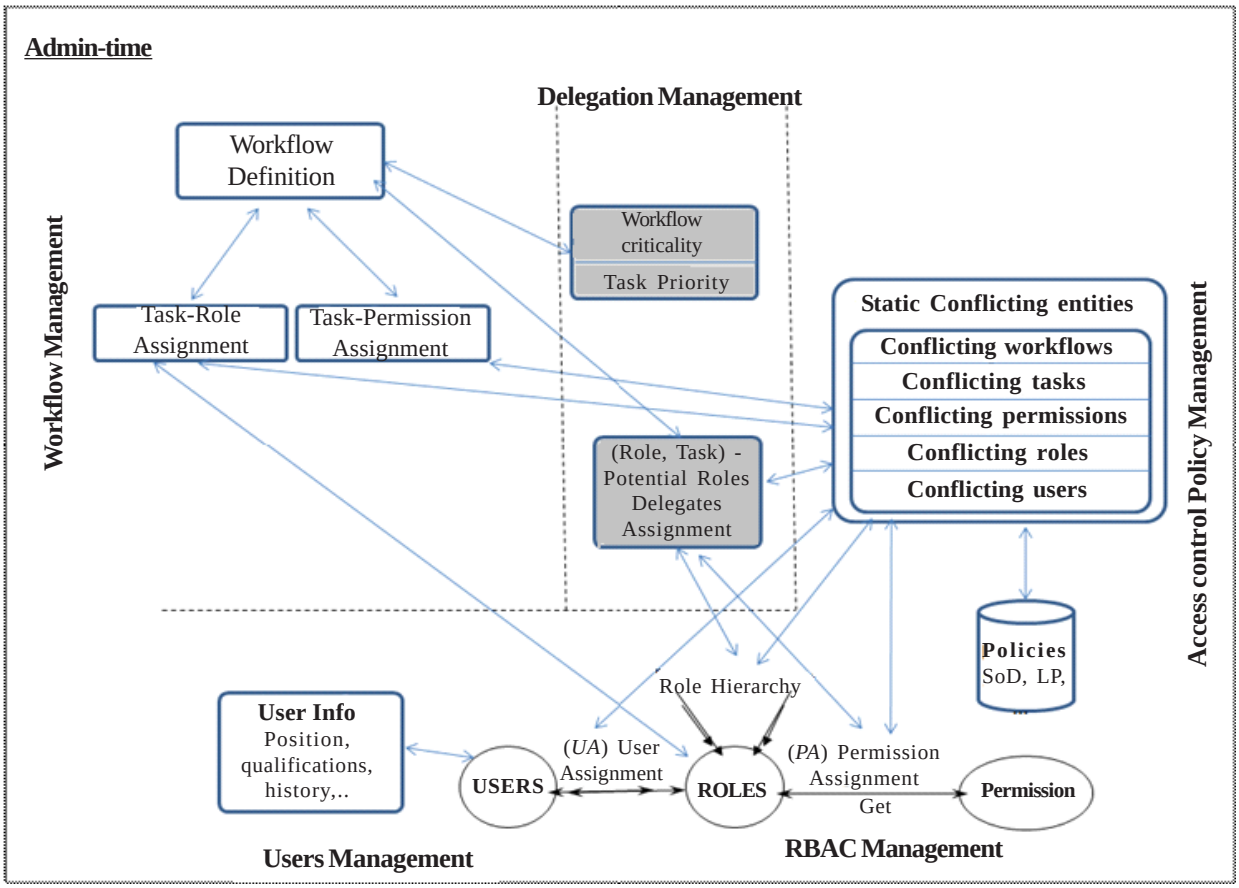

Figure 2. The proposed architecture at administration-time

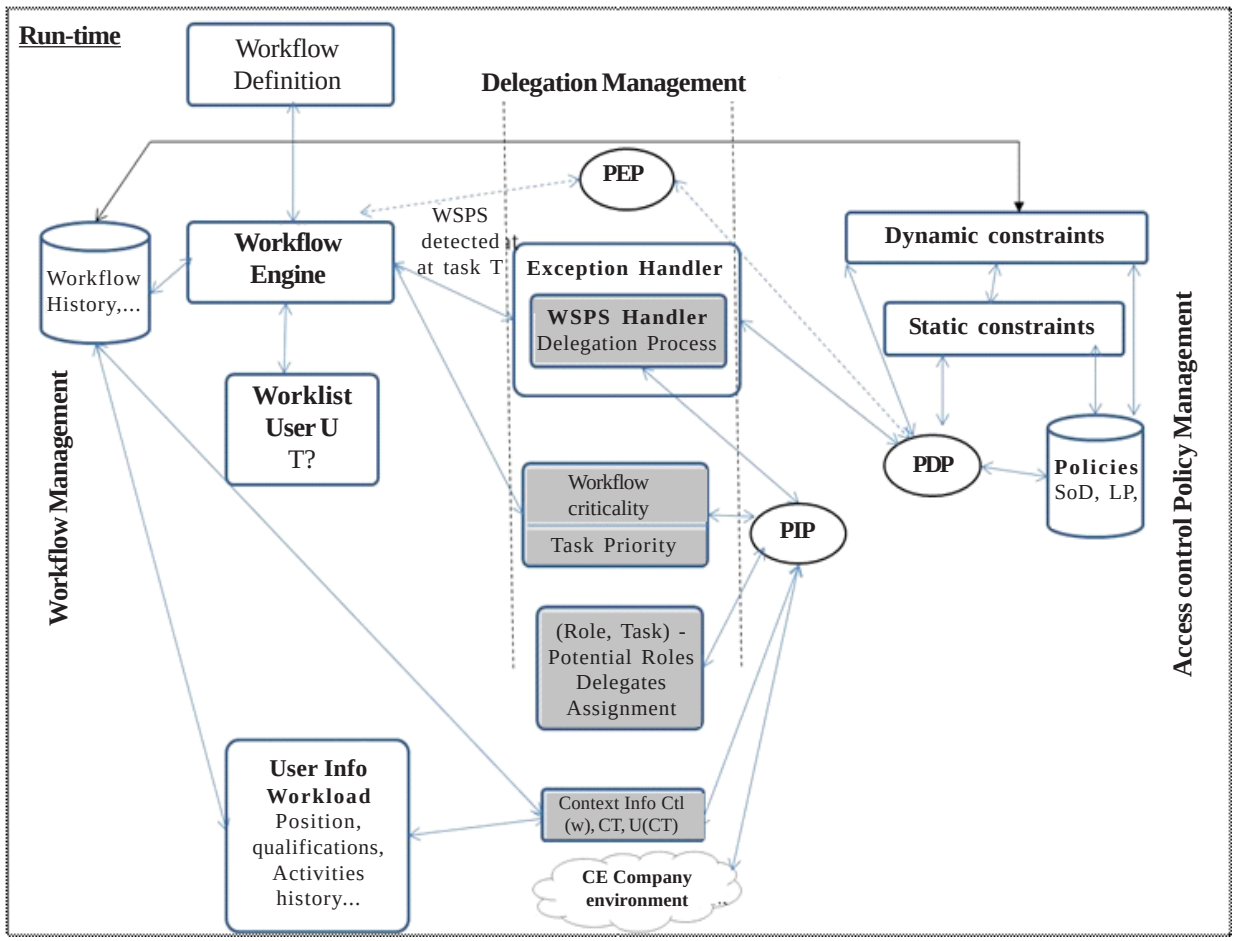

Figure 3. The proposed architecture at run-time

require changing existing implementation of the WFMS modules that will support it. All we need to do is to add a '*module*' to handle WSPS as a kind of exception. Indeed, exceptions often concern the inability to execute some particular task in the workflow.

The proposed architecture for this integration is illustrated in figure 2 (admin-time) and figure 3 (run-time) bellow. PEP (Policy Enforcement Point), PDP (Policy Decision Point) and PIP (Policy Information Point) are classical access control components as in [18].

When a task is instantiated by the workflow engine and a WSPS is detected, the WSPS handler is invoked to apply the Delegation process. Let's note that assigning a user found by the WSPS handler to a task could imply removing other workflow's task (with lower priority) from this user's worklist.

To implement our approach, we are planning to use the open source WFMS YAWL [19] which has a good support for exception handling.

In fact, by enabling the Exception Service of YAWL, it is possible to define exception handling processes for parent workflow instances when certain events occur (in our case a WSPS event). Moreover, it is possible to pause, resume, cancel or restart the task or the workflow instance that triggered the exception**.**

6. Conclusion and Future Works

This work aims to enforce dynamic access control constraints in a workflow system without creating workflow satisfiability problem situations. Indeed, we have presented a new approach which uses a specific delegation process to bypass WSP situations and thus enhancing the workflow system flexibility. This approach requires a specific work to be done at -administration time- that concerns among others the mapping between a role and a set of potential delegatee roles for a specific task. It relies also on the priority concept between tasks in different workflows.

Future work will focus on enhancing our approach by extending it with the study of this important question: Can the workflow instance complete if the WFMS delegates an instance of a task t to a user u selected by the proposed algorithm? In fact, our approach is 'local' to the current task instance that had produced the WSPS, so, we are motivated to extend it in order to take into account constraints on the following tasks in the workflow instance.

The duration of each task execution, on one hand, and the duration of the availability of each user on the other hand, are parameters that we plan to consider in the future to give more flexibility to our approach.

Finally, we project to implement the proposed architecture and achieve the integration of our delegation algorithm into an open source WFMS. This integration would allow testing and simulating our approach benefits in terms of resiliency and security.

References

[1] Crampton, J., Khambhammettu, H. (2008). Delegation and satisfiability in workflow systems. *In:* Proc. Of the 13th ACM SACMAT, p. 31-40.

[2] Wang, Q., Li, N. (2007). Satisfiability and Resiliency in Workflow Systems, *In:* Proc. Of ESORIC'07, p. 90–105.

[3] Lowalekar, M., Tiwari, R., Karlapalem, K. (2009). Security Policy Satisfiability and Failure Resilience in Workflows. IFIP Advances in Information and Communication Technology, (298) 197-210.

[4] American national standard for information technology: Role based access control. (2004). ANSI INCITS 359.

[5] Barka, E., Sandhu, R. (2000). Framework for role-based delegation models. *In:* Proc. of the 16th Annual Computer Security Applications Conference, p. 168-176, *IEEE Computer Society*.

[6] El Bakkali, H., Hatim, H. (2009). RB-WAC: New approach for access control in workflows. *In:* Proc. of the 7th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'09), p. 637- 640.

[7] Perelson, S., Botha, R. A. (2000). Conflict Analysis as a Means of Enforcing Static Separation of Duty Requirements in Workflow Environments. *South African Computer Journal*, (26) 212 – 216.

[8] Wei, X., Jun, W., Yu, L., Jing, L. (2004). SOWAC: a service-oriented workflow access control model. *In:* Proc. of the 28th Annual International Computer Software and Applications Conference, p. 128-134.

[9] Atluri, V., Warner, J. (2005). Supporting conditional delegation in secure workflow management systems. *In:* Proc. of the 10th ACM symposium on Access Control Models and Technologies (SACMAT'05), p. 49-58.

[10] Wainer, J., Kumar, A., Barthelmess, P. (2007). DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Information System*, 32 (3) 365-384

[11] Kumar, A., Van Der Alst, W. M. P., Verbeek, H. M. W (2002). Dynamic Work Distribution in Workflow Management Systems: How to balance quality and performance? *Journal of Management Information Systems*, 18 (3) 157-194.

[12] Cao, J., Chen, J., Zhao, H., Li., M. (2009). A policy-based authorization model for workow-enabled dynamic process management. *Journal of Network and Computer Applications*, 32 (2) 412- 422.

[13] Wang, K., Li, N., Chen, H. (2008). On the Security of Delegation in Access Control Systems. *In:* Proc. of the 13th European Symposium on Research in Computer Security (ESORICS'08), p. 317-332. Springer-Verlag.

[14] Toahchoodee, M., Xie, X., Ray, I. (2009). Towards Trustworthy Delegation in Role-Based Access Control Model. *Information Security, Lecture Notes in Computer Science* 5735, Springer 379-394.

[15] Delias, P., Doulamis, A., Doulamis, N., Matsatsinis, N. (2011). Optimizing Resource Conflicts in Workflow Management Systems, *IEEE Transactions on Knowledge and Data Engineering*, 23 (3) 417-432.

[16] WFMC. (1999). The Workflow Management Coalition. Workflow Management Coalition Terminology and Glossary, Document Number WFMCTC-1011.

[17] Hamid, H., El Bakkali, H,. Berrada, I. (2012). Enforcing Access Control in Workflow Systems with a Task Engineering Approach. *International Journal of Internet Technology and Secured Transactions* (IJITST) 4(1), Inderscience.

[18] OASIS eXtensible Access Control Markup Language (XACML) Version 2.0, 2005. OASIS Committee Specification (T. Moses, editor).

[19] Van der Aalst, W. et al. (2010). Modern Business Process Automation: YAWL and its support environment, Springer.