A logical framework for reasoning about delegation policies in workflow management systems

Khaled Gaaloul* and H.A. Proper

CRP Henri Tudor,
L-1855 Luxembourg-Kirchberg, Luxembourg
E-mail: khaled.gaaloul@tudor.lu
E-mail: erik.proper@tudor.lu
*Corresponding author

Ehtesham Zahoor, François Charoy and Claude Godart

LORIA,
Nancy University, UMR 7503, BP 239,
F-54506 Vandœuvre-lès-Nancy Cedex, France
E-mail: ehtesham.zahoor@loria.fr
E-mail: charoy@loria.fr
E-mai: godart@loria.fr

Abstract: Task delegation presents one of the business process security leitmotifs. It defines a mechanism that bridges the gap between workflow and access control systems. Delegation completion and authorisation enforcement are specified under specific constraints so-called events. In this article, we aim to reason about delegation events to model task delegation and to specify delegation policies using a logical framework. To that end, we propose an event-based task delegation model to control the delegation execution. We then identify relevant events responsible for the dynamic enforcement of delegation policies. Further, we define a task-oriented access control model to specify delegation constraints into authorisation policies. Finally, we propose a technique to automate the delegation policies integration. Using event calculus, we develop a reasoning tool to control the delegation execution and to increase the compliance of all delegation changes in the existing policy of the workflow.

Keywords: workflow; task; delegation; access control; AC; authorisation policy; event calculus; EC.

Reference to this paper should be made as follows: Gaaloul, K., Proper, H.A., Zahoor, E., Charoy, F. and Godart, C. (2011) 'A logical framework for reasoning about delegation policies in workflow management systems', *Int. J. Information and Computer Security*, Vol. 4, No. 4, pp.365–388.

Biographical notes: Khaled Gaaloul is a Researcher at the Public Research Centre Henri Tudor in Luxembourg. Prior to this, he has done his PhD in the French National Institute for Research in Computer Science and Control (LORIA-INRIA) in 2010. During his Doctoral project, he has been working as a Research Associate at SAP Research in Karlsruhe, Germany. His main research interests include enterprise engineering, service innovation, business process management and security.

H.A. (Erik) Proper is a Senior Research Manager at the Public Research Centre Henri Tudor in Luxembourg. He is also a Professor at the Radboud University Nijmegen, The Netherlands, where he heads up the Theories for Enterprise Engineering Group. He has a mixed background, covering a variety of roles in both academia and industry. He has co-authored several journal papers, conference publications and books. His main research interests include enterprise architecture, enterprise engineering, business/IT alignment and conceptual modelling.

Ehtesham Zahoor is a PhD student in the French National Institute for Research in Computer Science and Control (LORIA-INRIA). His main research interests include web services composition and monitoring.

François Charoy is an Associate Professor at the University of Nancy where he leads the Service and Cooperation Team from the French National Institute for Research in Computer Science and Control (LORIA-INRIA). He has co-authored several journal papers, conference publications and books. His research interest includes service oriented computing, business process management and computer supported cooperative work.

Claude Godart is a Full-time Professor at the University Henri Poincar, Nancy 1. His main research areas are business process management, web service computing, non-functional dimensions of e-services: transactional, security, and timing constraints.

1 Introduction

With the broad adoption of workflow management systems to model and automate business processes cross organisations, security becomes a crucial and essential topic (Hung and Karlapalem, 2003). Organisations establish a set of security policies that regulate and secure the flow of information between workflow's tasks and users that constitute the business process (Atluri andWarner, 2005). The execution of these tasks may require an access to specific and potentially sensitive data that have to be controlled when deploying processes within a workflow (Zur Muehlen, 2004).

While the modelling of business processes and workflows is well researched, the organisational dimension has been less investigated in workflow systems (Clavel et al., 2008). Organisational processes are determined by a mix of ad-hoc as well as human-centric processes (Gaaloul, 2010). Human involvement introduces authorisation concerns, requiring restrictions on who is allowed to perform which tasks at what time. This highly dynamic environment must be supported by mechanisms allowing flexibility, security and on-the-fly shift of rights and responsibilities both on a (atomic) task level and on a (global) process level. One specific approach ensuring human-centric interactions is that of task delegation.

We define task delegation as a means for assigning tasks and its access rights from one user to another user. The user who performs a delegation is referred to as

a 'delegator' and the user who receives a delegation is referred to as a 'delegatee'. Delegation can be very useful for real-world situations where a user who has to perform a task is either unavailable or too overloaded with work to successfully complete it. It is frequently the case that delaying these tasks executions would violate time constraints, thereby impairing the entire process execution. Therefore, delegation is a suitable approach to handle such situations and to ensure alternative scenarios by making process execution more flexible and secure within a workflow.

To the best of our knowledge, most of the work done in the area of workflow and access control (AC) systems does not treat delegation in sufficient details and deserves more investigations. On the one hand, existing work in the domain of organisational management in workflows suffers of rigidity and does not support human-centric processes (Atluri and Warner, 2005; Crampton and Khambhammettu, 2008a). On the other hand, current AC mechanisms are relatively static and have tackled delegation in a different way from the business process perspective (Seitz et al., 2005; Wainer et al., 2007). Most of the AC models are role-based and do not consider task delegation constraints (Barka and Sandhu, 2000; Zhang et al., 2003). Moreover, the derived security requirements from delegation are manually integrated and lack of compliancy with the existing policies (Gaaloul, 2010).

In this article, we aim to address the organisational and security requirements for task delegation within workflow systems. In doing so, we have to tackle two major issues, namely allowing task delegation to complete, and enforcing a secure delegation. On the organisational level, we have to identify the list of potential delegatees having the ability to execute the delegated task and to complete it. In doing so, we present an event-based task delegation model (TDM) to monitor the execution of the delegated task. On the security level, it implies the controlled propagation of authority (access right) during task execution and its specifications within the existing policy.

The contribution of this work is to develop a logical framework for reasoning about delegation policies. The novelty of our approach consists of reasoning on authorisation based on the event-based delegation model, and specifying them in terms of delegation policies dynamically. Based on an AC model we defined, we analyse and specify delegation constraints into authorisation policies. Additionally, we propose a technique that automates delegation policies using event calculus (EC) to control the delegation execution and to increase the compliance of all delegation changes in the existing policy of the workflow.

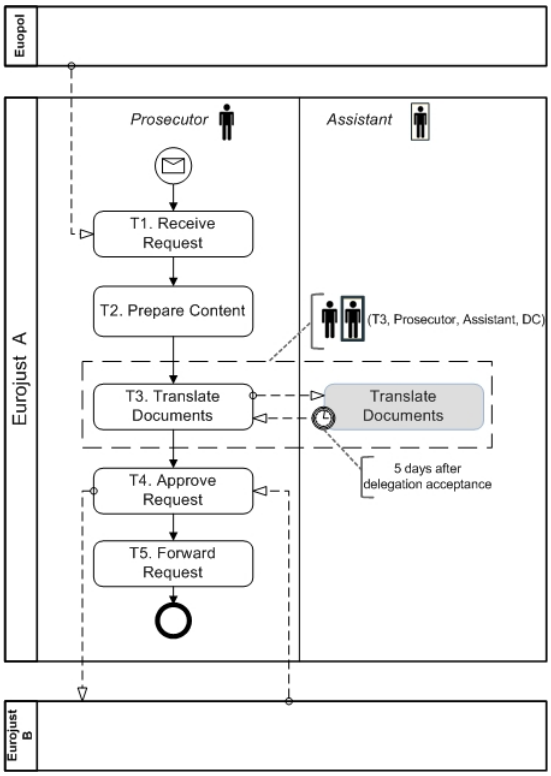
The remainder of this article is organised as follows. Section 2 presents an example requiring the integration of delegation policies. Section 3 focuses on modelling task delegation in workflows. In Section 4, we present a task oriented AC model to reason about delegation constraints based on delegation events. This material is used in Section 5 to specify authorisation policies. In Section 6, we leverage an EC technique to support delegation automation and to increase the compliance of delegation policies. Section 7 presents the implementation of the logical framework. Section 8 discusses related work. Finally, we conclude and outline topics of potential future work.

2 Delegation in an e-government context

To understand the motivation of our research, we present a scenario from an e-government case study supporting delegation (R4eGov, 2006). Mutual legal assistance

(MLA) defines a workflow scenario involving national authorities of two European countries regarding the execution of measures for protection of a witness in a criminal proceeding. Here we describe the MLA process part in the Eurojust organisation A. Users with roles *prosecutor* and *assistant* are assigned to execute the MLA process. Activities which are part of the process are represented as tasks (see Figure 1).

Figure 1 MLA delegation scenario



2.1 Task delegation scenario

The MLA process consists of receiving the request of assistance from Europol member in order to process it and send it to the concerned authority in Eurojust B. The task 'translate documents' T3 is originally only accessible by the user Alice member of role *prosecutor*, a fact defined in the workflow security policy. We define a workflow policy as a mean for defining access rights to a task's resources also called *authorisation policy*.

Let P an authorisation policy for the MLA process. This task is a long-running task and is expected to take five working days to complete. The *Prosecutor* Alice is unavailable to execute this task due to illness, and will delegate it to a subordinate involved in the MLA process. *Assistant* is a subordinate to *prosecutor* in the organisational role hierarchy (RH).

During delegation, the policy P is updated so that user Bob with role *assistant* is now allowed to complete task T3. The policy P will grant an access right to Bob to execute the task T3. As such, users with roles *prosecutor* and *assistant* are here the delegator and the delegatee, respectively.

In the meanwhile, Alice interrupts her sick leave and returns to work. Once again, Alice is able to claim T3. Due to qualification considerations, it is decided that Alice should complete the task, and that her *Assistant* Bob should revoke his actions, and free the task. The policy P needs to be updated to reflect that only Alice has access to the task. As such, the grant access to Bob would now evaluate to a deny decision which defines a policy rule. In addition, the authorisation policy P needs to reflect the new requirements for delegation. In order to derive a delegation policy from the existing policy, we have to specify additional authorisation rules to support delegation, where a rule defines the policy decision effect (e.g., permit, deny). Delegation rules depend on several delegation constraints such as time (i.e., five days) and have to be included in the delegation policy to define specific conditions to validate the policy decision effect.

2.2 Discussion

Delegation policies depend on delegation constraints. Such constraints define the delegation behaviour in the business process. This behaviour interprets human interactions when deploying a workflow. Here, delegation constraints aim to automate delegation policies from existing policy specifications. Accordingly, it is not possible to foresee a deny rule for revocation during the policy definition. Moreover, a manual review of the current AC rights and task executions is costly, labour intensive, and prone to errors. At present, responses arising from AC requests are stateless such that a response changes due to a policy adaptation will not support a dynamic delegation of authority (Pfleeger and Pfleeger, 2006; Barka and Sandhu, 2000; Seitz et al., 2005).

Based on the delegation scenario, we aim to address issues related to task delegation in workflows. At the *organisational level*, we have to identify the list of potential delegatees having the ability to execute the delegated task. The delegatee Bob is a subordinate to the delegator Alice based on the RH definition of the Eurojust organisation. At the *security level*, we have to compute the required authorisation (privileges) to execute the delegated task.

Delegation policies are defined from existing policies and are dynamically specified. Additionally, the integration of such policies has to be computed and compliant with the existing policy based on the delegation constraints. Hence, the essence of our work is to answer the following interrogations:

1 How to model task delegation within a workflow?

2 How to compute delegated privileges with regards to users, tasks and rights?

3 How to monitor delegation when deploying a workflow?

4 How to specify and integrate delegation policies in a compliant manner?

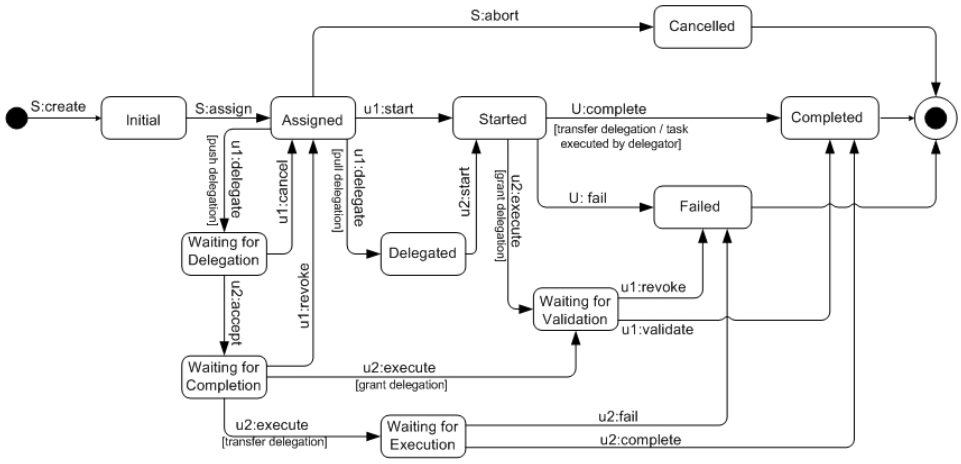
3 Modelling task delegation in workflows

In this section, we aim to model task delegation supporting human-centric interactions in workflows. A workflow is defined as a set of coordinated activities (tasks) composing a business process. We consider a task as a single unit of work (Russell et al., 2005). The basic states for a task life cycle are *initial, assigned, started, cancelled failed*, and *completed* (WFMC, The Workflow Management Coalition, 1999).

3.1 Task delegation model

We present a *TDM* to control the delegation execution. The delegation completion is specified under control constraints defined in the TDM. Control constraints define a fixed set of delegation events to monitor the delegation execution. Control constraints are illustrated in Figure 2. It defines the life cycle of our TDM from the time that a task is created through its final completion, cancellation or failure. It can be seen that there are series of potential states that comprise this model. A task, once created, is generally assigned to a user. The assigned user can choose to start it immediately or to delegate it. Delegation depends on the assignment transition, where the assigned user has the authority to delegate the task to a delegatee in order to act on his behalf.

Figure 2 Task delegation model



Intermediate events define constrained delegation operations within a workflow (*u1:delegate, u1:cancel, u1:revoke*). For instance, the delegator might want to cancel the delegated task. Our TDM would then go back to the previous state (*assigned* state). The delegation control-flow behaviour remains internal according to the task model, where *completed*, *cancelled* and *failed* are the final states. Moreover, additional delegation constraints define how a delegation request is issued based on the TDM. In the following, we detail the main constraints presented in Figure 2.

- Delegation mode: it defines how a delegator selects a delegatee. The *pull* mode assumes that a delegator has at his disposal a pool of delegatees to be selected to work on his behalf. However, the *push* mode assumes that a delegator is waiting for an acceptance from a potential delegatee. For instance, derived events from the Push mode are *u2:accept*, *u1:cancel* and *u1:revoke* in Figure 2.
- Delegation type: it refers to the delegation of privileges. It may be classified into *grant* or *transfer* (Schaad, 2003; Crampton and Khambhammettu, 2006). A grant delegation type allows an instantiated task to be available for both delegator and delegatee. As such, the delegator is still having the control to *u1:validate* or *u1:revoke* the task, and the delegatee to execute it. However, in a Transfer delegation type, the ability to use a delegated access right is transferred to the

delegatee; in particular, the delegated access right is no longer available to the delegator. There is no validation required and the task is terminated (*u2:complete*/*u2:fail*) by the delegatee.

Note that TDM is prefixed with either an *S* or an *U* indicating that the transition is initiated by the workflow system or the human resource, respectively. We define u_1 and u_2 belonging to the set of users *U*, where u_1 is the delegator and u_2 the delegatee.

3.2 Delegation relation

Task delegation defines a user-to-user delegation under specific constraints. Constraints depend on both the delegation behaviour within TDM and the process constraints (i.e., deadline). We define a delegation relation including the delegator, the delegatee, the delegated task and the delegation constraints as follows:

Definition 3.1: We define a delegation relation $DR \subseteq T \times U \times U \times 2^{DC}$ where *T* defines a set of tasks, *U* a set of users and *DC* a set of delegation constraints. A task delegation relation is defined as $dr = (t, u_1, u_2, \{dc\})$, *t* is the delegated task and $t \in T$, u_1 the delegator, u_2 the delegatee $\in U$ and *dc* the delegation constraints.

Delegation constraints can be related to time or evidence specifications. In the MLA example, evidence can be related to the language of translated documents or the number of translated documents within a period of time. In addition, organisational constraints define the delegation relation condition in a user-to-user delegation. For instance, an RH is defined between the *prosecutor* and *the assistant* when delegating task T3. The delegation relation for T3 is defined as follows:
$(T3, u_1{:}\text{Prosecutor}, u_2{:}\text{Assistant}, \{\text{RH}, 5\text{days}\}) \in DR$.

3.3 Revocation

Revocation is an important event/action that must accompany the delegation life cycle. A vast amount of different views on the topic can be found in literature (Zhang et al., 2003; Hagstrom et al., 2001). For simplification, the decision of revocation is issued from the delegator in order to take away the delegated privileges, or the desire to go back to the state before privileges were delegated.

The revocation factors can be identified depending on the delegation context. For example, a delegator can take back his task if his workload is being decreased or if the delegatee is not efficient anymore for performing the delegated task. For instance, Alice cancels Bob's work and will revoke the delegated task T3. The operation of revocation can be defined manually when the delegator decides to revoke the selected task by removing the delegatee's privileges or automatically by bending a time constraint to delegation. This constraint will affect the delegating access rights by deriving additional authorisation requirements in the policy *P* (see Section 6).

4 Access control over task delegation

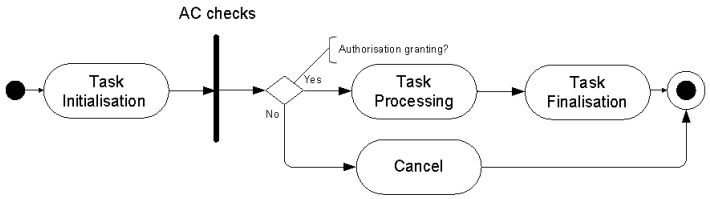
In this section, we aim to reason about task delegation from a task's resource perspective to analyse and specify task delegation constraints while accessing workflow's data. Data access defines permissions on business objects related to workflow's tasks. In current

workflow management systems, the RBAC model is widely adopted, where system administrators assign roles to users. It is more convenient for administrators to manage roles than to manage users directly (Sandhu et al., 1996). In our work, we define a task oriented AC model based on the RBAC model. We aim to support security delegation constraints with regards to potential delegatees and their required privileges.

4.1 Task execution model

We define a task execution model using an activity diagram composed of three main activities: *initialisation*, *processing* and *finalisation* (see Figure 3). During the initialisation of the task, a task instance is created and then assigned to a user. During task processing, the assigned user can start or delegate the task which gathers all operations and rights over the business objects related to task's resources. Finally, the task finalisation would notice the workflow management system that the task is terminated, where termination defines completeness, failure or cancellation.

Figure 3 Task execution model



Seeing the task as a block that needs protection against undesired accesses, the activity diagram includes an AC transition which is in charge with granting access to the task. AC defines the transition from the creation of a task to its assignment to a user. This assignment will lead to the processing or the cancellation of the task. Cancellation can be triggered when an assigned user does not fulfill the required authorisation to execute a task.

The AC transition defines the required authorisation also called permissions when executing a task. Authorisation makes an explicit binding between a user, a task resource (object) and his rights over it (action). For instance, the authorisation to execute the task T3 defines a permission to *translate()* (action) the 'request document' (business object) of task T3. Such authorisation information may be specified using a simple AC list where users may perform tasks for which they are authorised (Crampton and Khambhammettu, 2008a).

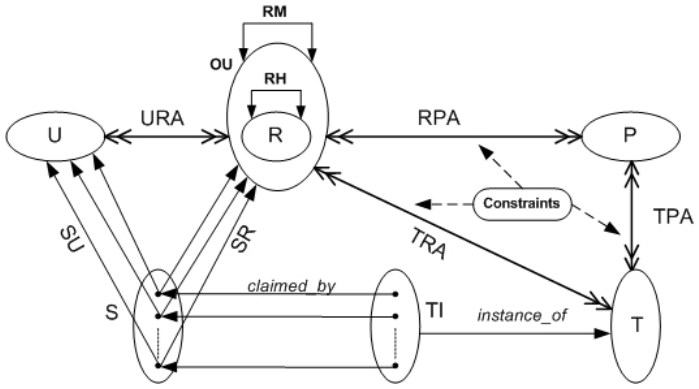
4.2 A task-oriented access control

We propose a task-oriented access control (*TAC*) model to support authorisation requirements in workflow systems (see Figure 4). Authorisation information will be inferred from AC data structures, such as user-role assignment (URA) and task-role assignment (TRA) relations. In addition, we model permission assignment relations for tasks and roles in order to support the task execution context. The remaining relations are generic relations based on the RBAC model (Sandhu et al., 1996).

Formally, we define sets *U, R, OU, T, P, S* and *TI* as a set of users, roles, organisations units, tasks, permissions, subjects and task instances, respectively. We use

a subject to denote the time a user selects roles for a session. During the the task instantiation assignment, we create a user's current active role set.

Figure 4 TAC model



We define *RH*, where *RH* is a partial order on *R*, r_i and $r_j \in R$. *RH* denotes that r_i is a role superior to r_j, as a result, r_i automatically inherits the permissions of r_j.

We define role mapping (*RM*), where $RM \subseteq OU_i \times OU_j$ with OU_i and OU_j two organisations units. *RM* defines external roles accessing distributed resources cross-organisations (Freudenthal et al., 2002). It provides a decentralised AC mechanism where externally known roles are publicly available: $r_k \in OU_i$ and $r_l \in OU_j$, *RM* denotes that r_l is a role mapped to r_k, as a result, r_l shares potentially the permissions of r_k.

4.2.1 Definitions of map relations

- $URA \subseteq U \times R$, the user role assignment relation mapping users to roles they are member of.
- $RPA \subseteq R \times P$, the permission role assignment relation mapping roles to permissions they are authorised to.
- $TPA \subseteq T \times P$, the task permission assignment relation mapping tasks to permissions. This defines the set of permission required to execute a task.
- $TRA \subseteq T \times R$ the task role assignment relation mapping roles to tasks they are assigned to.

4.2.2 Definitions of functions

- $SU : S \rightarrow U$ a function mapping a subject to the corresponding user.
- $SR : S \rightarrow R$, a function mapping each subject to a role, where $SR(s) = r, (SU(s), r) \in URA\}$ with s having a permission $p|(r, p) \in RPA\}$.
- $instance_{of} : TI \rightarrow T$, a function mapping a task instance to its task type.
- $claimed_{by} : TI \rightarrow S$, a function mapping a task instance to a subject to execute it. It defines the user-task assignment condition:

$$s = claimed_{by}(t_{i1})$$

where

$$\{t_i = instance_{of}(t_{i1}), (r,u) \in URA | (SR(s) = r \bigwedge SU(s) = u), (t_i, r) \in TRA\}.$$

4.2.3 Definitions of constraints

Here we discuss separation of duty (SoD) and Binding of duty (BoD) constraints. It defines a security constraint between two tasks that compose a business process (Botha and Eloff, 2001). We define an exclusive relation between tasks for SoD, and a binding relation between tasks for BoD:

$$TT_{SOD} : \{(t_i, t_j) \in T \times T | t_i \textit{is exclusive with} t_j\}$$

$$TT_{BOD} : \{(t_i, t_j) \in T \times T | t_i \textit{is binding with} t_j\}$$

If $(t_i, t_j) \in TT_{SOD}$, then t_i and t_j cannot be assigned to the same user. For instance, (T3, T4) $\in TT_{SOD}$ cannot be assigned to the same user Bob with the role *assistant* in the MLA process (see Figure 1). Regarding the policy, a subordinate (assistant) is not allowed to approve a request that he prepared on behalf of his superior (prosecutor).

If $(t_i, t_j) \in TT_{BOD}$, then t_i and t_j must be assigned to the same user which defines a binding relation between tasks.

4.3 Model contributions

The main contribution of the TAC model is to specify the task assignment condition. Actually, two conditions have to be verified. The first condition is related to task resources requirements. The role's permissions defined in role-permission assignment (RPA) needs to satisfy the permissions defined in task-permission assignment (TPA). If this condition is satisfied, the task is executed if and only if the user/role is assigned to it. Basically, having a permission to execute a task but not being assigned to it will not satisfy the outlined conditions and, therefore, will deny the access to task resources.

Definition 4.1: A task instance t_i is assigned to a user u if and only if:

$$(t,r) \in TRA \Rightarrow \{p \in P | (t,p) \in TPA\} \subseteq \{p | (r,p) \in RPA\} \bigwedge claimed_{by}(t_i) = s,$$

where $(SR(s) = r \bigwedge SU(s) = u)$.

The user-task assignment requires the $claimed_{by}$ function. Returning to the motivating example, T2 'check request' is assigned a set of permissions based on the TPA relation in order to carry out this task. The user Alice with the role *prosecutor* is assigned to T2 since Alice verifies the TRA and $claimed_{by}$ conditions. However, if we consider another user member of the role *prosecutor* from Eurojust B, he is not allowed to execute T2 since he does not fulfill the user-task assignment conditions.

4.4 Access control over task delegation using TAC model

Delegation is a mechanism that permits a user to assign a subset of his assigned permissions (privileges) to other users who currently do not possess it. The TAC model allows computing the list of potential delegatees using the RPA relation that may satisfy the delegated task requirements based on the TPA relation. Here, we define a method for

AC over task delegation using our TAC model. We present the main steps to describe how valid delegatees are checked and whether they need delegated privileges grant as follows:

1 defining roles and permission assignments for each user ($\{(u_1, r_1), (u_2, r_2)\} \subseteq URA$ and $\{(r_1, p_{r1}), (r_2, p_{r2})\} \subseteq RPA$)

2 instantiating the task t_{i1} and assigning it to the delegator s_1 who is the current user u_1 ($s_1 = claimed_{by}(t_{i1})$)

3 checking security constraints before delegation (SoD and BoD)

4 computing the delegatee s_2, who is the current user u_2, based on his permissions assignment ($(t_i, p_{r2}) \in TPA$)

5 granting privileges for s_2 based on the task instance permissions assignment ($p'_{r2} \leftarrow p_{r2} \cup p_{ti}$) which is defined in the $claimed_{by}$ function

6 defining the delegation relation instance: dr_1 = (t_{i1},s_1,s_2,{DC}).

The main contribution of this method is to specify the delegated task assignment conditions based on Definition 4.1. If the conditions are satisfied, then the task t_i is delegated to the delegatee u_2. However, if u_2 does not have the permission required and there is no conflicts (BoD or SoD) to execute t_i. Then the delegated privileges are granted for u_2 based on the $claimed_{by}$ function.

The computation of the privileges is based on the TRA and $claimed_{by}$ specifications defined in our TAC model. Basically, we provide an optimised method to compute the least privileges to delegate based on the current requirements of the task instances t_{i1} and not the full requirement of the task type t_i. The task instance is generated from the delegated task. We aim to optimise the delegated privileges based on what the delegated task instance defines (see $claimed_{by}$ condition for permissions).

For instance, the *Prosecutor* Alice delegates T3 to his *Assistant* Bob. We assume that Bob does not have the permissions to access T3 resources. Alice will grant just the permission *translate()* on the business object type 'request document' while keeping additional privileges related to the MLA request treatment that have to be protected due to privacy reasons. By computing the task instance requirements for delegation, we ensure the grant of the least privileges for the delegatee: (Assistant,translate()) $\in$ RPA.

In summary, we have analysed task authorisation constraints to support security requirements for delegation. We have presented a TAC model to support AC over workflows. The novelty of this model is the ability to reason about task delegation from human (users) and material resources (tasks). Our TAC model allows us to compute delegatees and their delegated privileges. This model will help us to specify the delegation polices within workflows in the next section.

5 Specifying delegation policies

In this section, we discuss authorisation policy specifications and particularly delegation policies. We introduce authorisation policies based on our AC (TAC) model. We then identify the delegation constraints that have to be specified in the delegation policies and updated in the existing authorisation policy.

5.1 Authorisation policy specification

Authorisation policies may be specified using a simple AC list or more complex role-based structures. An AC has to be defined to check the authorisation of the initiating user so-called subject. An authorisation makes an explicit binding between a role (subject), a task resource (object) and his rights (action) over it. This binding is defined based on the main relations: URA, TPA and TRA in our AC model (TAC). Subsequently, an authorisation expresses a user's permissions on a task's resources, where a permission is the right to execute an action on a resource.

Definition 5.1: We define a policy $P \subseteq target \times rule \times 2^C$, where *target* defines where a policy is applicable, *rule* is a set of rules that defines the policy decision result, and C the policy constraints set that validates the policy rule.

A target defines the entities of an access request. It is composed of a role associated to the subject and an action on a business object of a task type. A rule effect defines an authorisation decision. It can return as a result a permit, a deny or an indeterminate request (Moses, 2005). Constraints are related to the workflow authorisation specifications. For instance, the SoD is a constraint for a user-task assignment. For instance, the policy decision returns the result 'permit' where the user Alice member of role *Prosecutor* can access to the resource MLA1 (Request Document) of task T1 and read it (see Figure 1).

5.2 Policy decision changes

We define delegation transitions as events ruling delegation behaviour. The internal behaviour based on events may be a source to a policy change, thereby introducing advanced security requirements in AC systems. From our TDM, we analyse security requirements that need to be taken into account to define event-based delegation policies:

- Delegation of authority: it permits to a delegator to assign a subset of his assigned privileges to a delegatee who currently does not possess the required authorisation to execute the task. For instance, *u1:delegate* is an event that will trigger task delegation, thereby updating a policy to grant a new AC for a delegatee.
- AC enforcement: it permits dynamic policy enforcement. For instance, *u1:revoke* implies the revocation of delegated privileges where the delegator will take the control back on his assigned task and, therefore, cancel the previous policy decision.

The specified *events* define the condition to validate the policy decision effect. An event change may inquire a policy decision change. In the following, we classify delegation events and identify the relationship between delegation type, mode and

their corresponding impacts on policies decision changes (see Table 1). In Table 1, a grant delegation type using a push mode is based on events: *u1:delegate*, *u2:accept*, *u1:cancel*, *u2:execute*, *u1:validate* and *u1:revoke*. In this case, delegation policies changes when *u1:delegate*, *u2:accept* and *u1:revoke*.

Table 1 Delegation policies changes based on events

Delegation events	*Push delegation*		*Pull delegation*		*Policy decision change*
	Grant	*Transfer*	*Grant*	*Transfer*	
u1:delegate	Yes	Yes	Yes	Yes	Yes
u2:accept	Yes	Yes	No	No	Yes
u1:cancel	Yes	Yes	No	No	No
u2:execute	Yes	No	Yes	No	No
u1:validate	Yes	No	Yes	No	No
u1:revoke	Yes	No	Yes	No	Yes
U:fail	No	Yes	No	Yes	No
U:complete	No	Yes	No	Yes	No

Source: Gaaloul (2010) and Gaaloul et al. (n.d.)

From our motivation scenario, we present two examples to explain policy changes presented in Table 1.

Example 1: The *u1:revoke* event is defined in both push and pull modes. It supports grant delegation, where a delegatee needs to wait for the validation from the delegator. This event will enforce a policy change and terminates the authorisation for the delegatee. The revocation leads to the completion of the delegated task, and the revocation of the delegated privileges. For instance, Bob work is cancelled by Alice and then his delegated privilege is no more valid in the policy.

Example 2: The *u2:fail* event is defined in both push and pull modes. It supports transfer delegation, where a delegatee terminates the task by himself without validation. Defined policy will take effect until the termination of the task during transfer delegation, where no new updates are required since all the task privileges are transferred to the delegatee.

5.3 Dynamic delegation policies

At present, we can enforce delegation access rights via policy adaptation. Subsequently, we need to update the delegation relation *DR* in the workflow policy *P* once a delegation event is triggered. It consists of adding a new policy authorisation constraint for the delegated user. If this constraint changes due to a policy adaptation (e.g., a task revocation event), a new response needs to be conveyed to the delegatee dynamically (Gaaloul et al., 2009).

Definition 5.2: Let P be a global authorisation policy, *P = (target,rule,C)*, we define a delegation policy P_D = *(*$target_D$*,*$rule_D$*,*C_D*)*, where $target_D$ = *DR* the delegation relation, $rule_D \subset rule$, $C_D \subset C$ and $C_D = DC \bigcup events$ where DC the set of delegation constraints.

Returning to our example, we can observe a dynamic policy enforcement during delegation. Initially, T3 is delegated to Bob and the delegation policy for T3: $P_D = (DR, permit, \{RH, 5\ days, u1 : delegate\})$. In the meanwhile, user Alice is back to work before T3 is finished and will cancel what was performed by Bob so far. Alice is once again able to claim the task and will cancel the policy effect (permit) for

Bob. The event *revoke* will be integrated in the policy, and a notification (deny) is then conveyed back to Bob for appropriate actions. Thus, the delegation policy for T3 needs to be updated and $P_D = (DR, deny, u1 : revoke)$.

The dynamic revocation of the delegated task will enforce a new authorisation rule generation. The new rule has to be integrated in the delegation policy dynamically. In doing so, we have to compute automatically the new delegation policy within the existing policy *P*. This computation will ensure compliancy by adding valid rules for delegation in the existing policy using AC systems. The computation of delegation policies will be discussed in the next section.

6 Integrating event-based delegation policies

In this section, we propose a logical framework for reasoning about delegation policies. The framework is based on the EC formalism. We leverage the event-based TDM to control the delegation behaviour and to specify its authorisation policies dynamically. We gather relevant events that define both the task execution path and the generated delegation policies. Using EC, we develop a technique that automates delegation policies for controlling the delegation execution and increasing the compliance of all delegation changes in the existing policy.

6.1 Our approach

Securing delegation involves the definition of authorisation policies which have to be compliant with the policy of the workflow. To do so, we need to address two important issues, namely allowing task delegation to complete, and having a secure delegation. The monitoring of task delegation is an essential step to ensure delegation completion. In our TDM, delegation completion depends on events generated during the delegation life cycle. Events such as *validate* may be required, when a delegation request is issued under a certain obligation, where the delegatee has to perform specific evidence to validate the task execution. Securing delegation consists of reasoning on authorisation based on specific delegation events, and specifying them in terms of delegation policies. When one of these events changes, our access policy decision may change implying dynamic delegation policies.

Returning to our example, we consider the situation where the delegator with the role *Prosecutor* sends a delegation request for all users members of role *assistant* regarding the execution of task T3 in the delegation scenario. This defines a *push* delegation mode, where a delegatee is chosen dynamically. An acceptance of delegation implies a new AC enforcement in the existing policy, thereby adding a new authorisation rule for the delegatee under specific conditions (i.e., time) and/or obligations (i.e., evidence) agreed between the delegation principals.

The *prosecutor* may need to review all the translations done by his *assistant* for the validation based on evidence. Evidence can be related to the language of translated documents or the number of translated documents within five days. To that end, an authorisation rule, permitting the access (e.g., *read, write*) to the legal document, is constrained by an obligation allowing to investigate whether evidence were satisfactorily met. If however, evidence are not satisfied, a revoke action may be triggered including a deny result for the previous policy effect. Subsequently, another rule has to be integrated

dynamically in the policy with an effect of deny for the previous authorisation to the delegatee.

6.2 Modelling task delegation in EC

We use our TDM to monitor the delegation execution. It defines how delegation request is issued and then executed depending on delegation constraints. The idea is to offer a technique to monitor the delegation execution based on specific events. Using EC, we can foresee the delegation behaviour within a workflow.

Background and motivations. The proposed approach for the representation of task delegation model relies on the EC formalism. The choice of this formalism is motivated by the fact that given the representation of the event-based TDM, an EC reasoner can be used to reason about such events.

EC is a logic programming formalism for representing events and is being widely used for modelling different aspects such as flexible process design, monitoring and verification (Kowalski and Sergot, 1986). It comprises the following elements: $\mathcal{A}$ is the set of *events* (or actions), $\mathcal{F}$ is the set of *fluents*, $\mathcal{T}$ is the set of *time points*, and $\mathcal{X}$ is a set of objects related to the particular context. In EC, events are the core concept that triggers changes to the world. A fluent is anything whose value is subject to change over time. EC uses predicates to specify actions and their effects (see Table 2).

Table 2 EC predicates

Predicate	*Interpretation*
$Initiates(e, f, t)$	Fluent f holds after timepoint t if event e happens at t
$Terminates(e, f, t)$	Fluent f does not hold after timepoint t if event e happens at t
$Happens(e, t)$	Is true iff event e happens at timepoint t
$HoldsAt(f, t)$	Is true iff fluent f holds at timepoint t
$Initially(f)$	Fluent f holds from time 0
$Clipped(t_1, f, t_2)$	Fluent f was terminated during time interval $[t1, t2]$
$Declipped(t1, f, t2)$	Fluent f was initiated during time interval $[t1, t2]$

Source: Kowalski and Sergot (1986)

The reasoning modes provided by EC can be broadly categorised into abductive, deductive and inductive. In reference to our proposal, given a TDM and authorisation policies, it will be interested to find a plan for task delegation, that allows to identify what possible actions (policy changes) will result from the task delegation and may opt to choose the optimal plan in terms of minimal policy changes. This refers to the 'abduction reasoning'. Then, it will be also interested to find out the possible effects (including policy changes) for a given set of actions (a set of events that will allow task delegation). This leads to the choice of 'deduction reasoning' (Mueller, 2006).

EC based on TDM. The basic entities in the proposed model are tasks. In terms of discrete EC terminology, they can be considered as *sorts*, from which instances can be created. Then, each task can be in different states during the delegation execution. In reference to the TDM presented earlier (see Figure 2), the possible task states include initial, assigned, delegated, completed and others. As task states change over time, they can thus be regarded as *fluents* in the EC terminology. Further, the states change are governed by a set of actions/events and in relation to the TDM. The task state changes from initial to assigned as a result of the occurring of the *assign* event. Finally the TDM introduces a set of orderings, such as the state of a task cannot be assigned, if it is not

yet created. In reference to the event calculus model, we introduce a set of axioms to handle these dependencies. In the following, the EC model introduces the fluents, basic events and dependency axioms (see Figure 5).

Figure 5 EC-based TDM

Delegation model
sort task
fluent Initial(task), Assigned(task), Delegated(task), Started(task)...

event Create(task)
[task, time] Initiates(Create(task), Initial(task) ,time).
event Assign(task)
[task, time] Initiates(Assign(task), Assigned(task) ,time).
[task, time1] Happens(Assign(task), time1) $\rightarrow$ $\{$*time2*$\}$ *HoldsAt(Initial(task), time2)*
& $time1 > time2$

Delegation mode choice
[task, time] !Happens(Abort(task), time).
[task, time] !Happens(Start(task), time).
[task, time] !Happens(PullDelegate(task), time).

The EC model presented in Figure 5, first defines *sort* and *fluents* that mark the different task states. Then, we define an event *Create*(task), and an Initiates axiom that specifies that the fluent *Initial*(task) continues to hold after the event happens at some time. Similarly, we define the event/axiom for the assignment event and fluent. We further introduce an axiom that specifies that in order to assign some task at time1. This task must be already in initial state at time2, with time1 greater than time2. In a similar fashion, we can define events and associated *initial* axioms for the complete TDM model.

For the basic EC model above, the solutions (plans) returned by the reasoner may also include the trivial plans which does not enforce the delegation and directly *start* or *abort* the task once assigned. In order to give the user ability to choose the delegation mode once the task is assigned, we enrich the model with axioms specifying wich task does not either *Start, Abort* or requires *PullDelegation* once assigned. Thus, the only option for the reasoner is to conclude that the model requires a *PushDelegation* mode. We can similarly restrict the delegation permission (grant/transfer), once the task is in the *WaitingforCompletion* state (see the delegation mode choice in Figure 5).

6.3 Building delegation policies

Here we analyse the security requirements that need to be taken into account to define delegation policies. Additional requirements such as *pull/push* mode and *grant/transfer* type may be a source to a policy change during delegation. Using EC, we present a technique capable of computing and generating new policy rules automatically.

A policy rule may include conditions and obligations which are used to identify various conditions or cases under which a policy may become applicable. Conditions and obligations are related to delegation security constraints when defining delegation policies. Based on the result of these rules different policies may become applicable in the context of delegation.

Delegation rule: We define a delegation rule $rule_D \subseteq P_D$, $rule_D$ = *(effect, condition, obligation)*, where *effect* returns the policy decision result (permit, deny), *condition* defines the delegation mode (push, pull) and *obligation* checks evidence for the delegation type (grant, transfer).

We analyse security requirements to define delegation policy rules in a *push* mode of the TDM. We present a table that gathers specific events for *push* delegation and analyse them in terms of policy rules. Adding rules in the workflow policy will ensure the delegation of authority, thereby adding the required effect (permit, deny) to the delegation policy rules (see Table 3).

Table 3 Push delegation policy rules-based events

Delegation events	*Push delegation*		*Adding policy rule*
	Grant	*Transfer*	
u1:delegate	Yes	Yes	Add rules if accepted
u2:accept	Yes	Yes	Add rules based on execution type
u1:cancel	Yes	Yes	No add
u2:execute	Yes	No	(Permit,Push,Grant:Evidence)
u2:execute	No	Yes	(Permit,Push,Grant:NoEvidence)
u1:validate	Yes	No	No add
u1:revoke	Yes	No	(Deny,Push,Grant)
U:fail	No	Yes	No add
U:complete	No	Yes	No add

Source: Gaaloul (2010) and Gaaloul et al. (n.d.)

Returning to the example, we can observe a dynamic policy enforcement during delegation. Initially, T3 is delegated to the *Assistant* u_2 (the delegatee) and the delegation policy for T3 is defined as follows: $P_D = (DR, Permit, \{Push, 5\ days\})$ (see Table 3, *u2:execute/Grant*).

In the meanwhile, the *Prosecutor* u_1 (the delegator) is back to work before delegation is done and is not satisfied with the work progress and will revoke what was performed by his assistant so far. The *prosecutor* is once again able to claim the task and revokes the policy effect (permit) for the *assistant*. The event *revoke* is updated in the policy, and a deny rule is then inserted in the policy. Thus, the delegation policy for T3 needs to generate a new rule and the delegation policy is updated to: $P_D = (DR, Deny, \{Push, Grant\})$ (see Table 3, *u1:revoke*).

6.4 Modelling delegation policies in EC

In order to model the delegation policy rules, we introduce new sorts called *effect*, *condition*, *obligation* to the EC model. We then specify instances of each sort to be the possible effects, conditions and obligations. Possible effects include *deny* and *permit* results, and conditions define the *push* and *pull* mode. The possible instances for obligations include *grant, transfer, evidence* and *NoEvidence* which are constraints related to delegation type and mode. We further add an action *AddRule(effect, condition, obligation)* and corresponding axiom and enrich the model to specify the policy changes as a result of events (see Figure 6).

The policy change axioms presented above specify that once certain actions happen, they cause policy changes and thus we add a new rule to the existing policy. The name of actions/events depicts their invocation hierarchy, *PushDelegateAcceptExecuteGrant*

is the *execute* event with a grant permission once the PushDelegation request has been accepted by a delegatee and has to be validated before completion (see Table 3).

Figure 6 Delegation policy

```
sort task, effect, condition, obligation
effect Permit, Deny     condition Push, Pull     obligation Grant, Transfer, ..

fluent RuleAdded(effect, condition, obligation)
event AddPolicyRule(effect, condition, obligation)
[effect, condition, obligation, time]
Initiates(AddPolicyRule(effect, condition,obligation),
RuleAdded(effect, condition, obligation) ,time).

;policy change
[task, time] Happens(PushDelegateAcceptExecuteGrant(task), time) →
Happens(AddPolicyRule(Permit, Push, Evidence), time)
[task, time] Happens(PushDelegateAcceptFailTransfer(task), time) →
Happens(AddPolicyRule(Deny, Push, Transer), time)
```

6.5 Delegation automation

The formalised approach for the monitoring and securing of TDM defines the logical framework for reasoning about delegation policies. The framework is a mean to ensure the automation technique for delegation. Automation is necessary for both the task completion and the policy specification during delegation. Reasoning on delegation events using EC offers a solution to foresee the delegation execution and to increase the control and compliance of all delegation changes.

By reasoning on specific events, we are able to control the order of delegation execution which is computed automatically based on events. Events allow to distinguish between the order of execution by checking the delegation mode and type. For instance, an execution expects a validation transition if and only if we are in a *grant* delegation type. In addition, we are able to address the policy stateless issue. We can compute delegation policies from triggered events during task execution.

Moreover, delegation automation offers many benefits. Actually, it reduces efforts for users and administrators. Administrator efforts can be related to the process definition and policies specification. Besides, it increases control and compliance of all delegation changes. Subsequently, task delegation is accomplished under constraints which are compliant with the workflow's policy. For instance, time constraint has to be taken into account when granting a temporal access for delegation (see T3 deadline in Figure 1).

In order to test the performance of our technique, we develop a tool supporting delegation automation. The implementation is based on an EC reasoner. The implementation environment and its performance are presented in the next section.

7 Implementation

We use the discrete event calculus reasoner (DEC reasoner) for performing automated common sense reasoning using the EC formalism. It solves problems efficiently by

converting them into satisfiability (SAT) problems. In addition, the DEC reasoner attempts to find a solution by transforming the EC model into a SAT problem and invoking the SAT solver for the solution.

7.1 Reasoning

As discussed earlier, the reasoning can either be abductive or deductive. For the abductive reasoning, a plan is sought for the specified goal. A plan is a sequence of *EC happens clauses* that specify the temporal ordering of different event-based actions/transitions, whose execution leads to the goal.

In reference to the TDM, the goal is to have either the task in completed, cancelled or failed states. We then add the goal $[task]HoldsAt(Completed(task), 15)$ $|HoldsAt(Failed(task), 15)|HoldsAt(Cancelled(task), 15)$ to the EC model and add an instance of the delegated task T3, where 15 is the maximum range of the *timepoint* interval. The value 15 covers the possible transitions to reach the delegation goal. It presents the maximum number of transitions that may occur when delegating a task. For instance, a value less than 15 may stop our execution before the defined goal (i.e., completed, cancelled or failed).

Figure 7 Delegation plan

```
1389 variables and 7290 clauses
relsat solver
1 model
—
model 1:
0 Happens(Create(T3), 0).
1 +Initial(T3).
2 Happens(Assign(T3), 2).
3 +Assigned(T3).
4 Happens(PushDelegate(T3), 4).
5 +WaitingDelegation(T3).
6 Happens(PushDelegateAccept(T3), 6).
7 +WaitingCompletion(T3).
8 Happens(PushDelegateAcceptExecuteGrant(T3), 8).
Happens(AddPolicyRule(Permit, Push, Evidence), 8).
9 +RuleAdded(Permit, Push, Evidence).
+WaitingValidation(T3).
10 Happens(PushDelegateAcceptExecuteGrantValidate(T3), 10).
11 +Completed(T3).
—
;DECReasoner execution details
0 predicates, 0 functions, 12 fluents, 20 events, 90 axioms
encoding 0.5s - solution 0.2s - total 0.9s
```

The invocation of the EC reasoner will then give us a set of possible solutions (called plans) for achieving the goal. Let us first consider the case, when the chosen delegation mode is *PushDelegation* with the grant of permissions to the delegatee, the EC reasoner returns the plan detailed in Figure 7.

The execution plan follows the delegation of T3 described in the delegation scenario. It shows the actions that need to be considered for delegation and most importantly,

it shows the possible policy changes as a result of delegation. Steps 1 to 11 depicts the delegation process execution. Having a *push* mode as a condition, we derive the relevant rules to be added in the policy. For instance, step 8 and 9 show when a delegatee requests the task T3 for execution. Delegatee acceptance went through the 'WaitingDelegation', 'WaitingAcceptance' and 'WaitingCompletion' states (described in Figure 2). Based on these events, we deduce that an authorisation rule is added at this stage under a certain obligation (evidence for task validation). Finally, a validation of T3 completes the delegation execution (see steps 10 and 11 in Figure 7).

All the defined axioms using the DEC Reasoner language can be given to the reasoner for finding a solution (if it exists) to support policy changes, which automatically orients these axioms into delegation rules. Then, given as inputs the specification of the conditions and obligations expressed when adding rules, the generated plan by the reasoner shows that either the authorisation rules result in a permit or a deny decision.

Regarding policy changes, there are two possible scenarios. The first scenario is the adding of a new policy rule because the conjectures (conditions or obligations) are valid. The second scenario concerns cases corresponding to an overriding of this rule to a deny result.

7.2 Deployment limits

We have checked the performance of our reasoning algorithm to evaluate the proposed framework. The example presented in Figure 7 computed the time of encoding for a single delegated task. Then, we have tested the event-calculus model for sequential and parallel tasks (about 50 tasks) using the DEC reasoner tool. The tests were conducted on a MacBook Pro Core 2 Duo 2.53 Ghz and 4 GB RAM running Mac OS-X 10.6. The DEC Reasoner Version 1.0 and the SAT reasoner, relsat-2.0 were used for reasoning.

Figure 8 Performance testing for sequential and parallel tasks (see online version for colours)

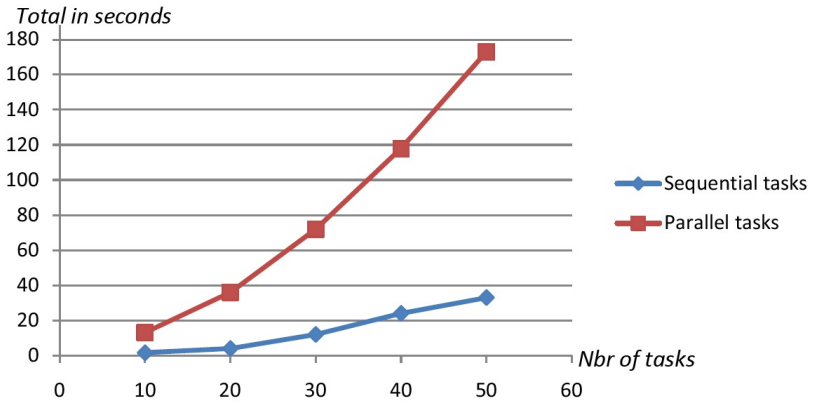


Figure 8 illustrates two curves for sequential and parallel tasks. Time is computed based on the total of time encoding (DEC reasoner) and solution computation (SAT). We can observe that the sequential task curve reflects a good performance of our model. For instance, 40 delegated tasks in sequence will take less than 25 seconds to be encoded and solved. However, parallel (concurrent) tasks are more time consuming. For the

same number of tasks, we will need about two minutes. This can be explained by the algorithm of encoding for parallel tasks. The algorithm complexity is $\Theta(n^2)$, which presents an exponential growth in the graph.

Note that the proposed framework is still under construction. This experience is limited to a user-to-user delegation (two users) where permissions and its authorisation rules are not subdivided. Cascaded delegation cross multi users (Zhang et al., 2003) will be a part of the encoding improvement in the future work.

8 Related work

We discuss different approaches that tackle task delegation in secure workflow systems. To do so, we firstly analyse delegation in workflows. We then present AC mechanisms supporting delegation and discuss their functionalities and limitations.

8.1 Delegation in workflow systems

Most of the work done in the area of workflow management systems does not treat delegation in sufficient details (Venter, 2003). There exists little related work considering task delegation behaviour within a business process. This observation is supported by research done by Russel et al. (2005) and Hung et al. (2003). They outlined that existing approaches remain static and do not support security constraints in general and task delegation modelling in particular (Bertino et al., 1999; Wainer et al., 2007).

Crampton et al. discussed delegation in the context of workflow systems using three different workflow execution models (Crampton and Khambhammettu, 2008b). The work offers a greater understanding of the effects of various delegation operations on the authorisation data structures in the context of role-based workflows. However, they did not consider task delegation constraints within authorisation policies. The integration of delegation policies into existing policy is not treated and the problem of policy compliancy is not addressed (Crampton and Khambhammettu, 2006).

Atluri et al. have extended the notion of delegation to allow conditional delegation such as time, workload and task attributes. In addition, different types of constraints come into play, which include authorisation constraints, role activation constraints and workflow dependency requirements (Atluri and Warner, 2005). Authors addressed the problem of assigning users to tasks in a consistent manner such that none of these constraints are violated. However, additional aspects related to the delegation behaviour are missed. The action of delegation remains atomic and does not investigate the internal behaviour of delegation and its impacts over the organisational process.

Russel et al. (2005) proposed an approach supporting delegation. They described the life cycle of a work item in the form of a state/transition diagram with a particular focus on the resource allocation perspective. One of the main drawbacks of this approach is that it defines a static binding of all work items associated with a task to a single resource. This approach ignores additional events (transitions) during delegation execution and does not support the aspects of users, tasks and events within a workflow to specify delegation policies.

8.2 Delegation in access control models

The discretionary access control (DAC) approach allows the creator of an object, or anyone else that is authorised to control it, to make AC decisions. These rights change dynamically at the discretion of the owner of an object. AC mechanisms that support a DAC policy include directory lists, AC lists and AC matrices. A disadvantage of this approach is that management complications arise in the event that an object is removed or renamed. Maintenance is difficult when updates are required in the directory of many users (Pfleeger and Pfleeger, 2006), which is the scope of this paper.

Role-based access control (RBAC) is recognised as an efficient AC model for large organisations. Most organisations have some business rules related to AC policy (Sandhu et al., 1996). Barka and Sandhu proposed a role-based delegation model based on the RBAC model. Their unit of delegation is a role. Authors focused also on role-based models supporting role hierarchies when studying delegation in the context of both RBAC0 model (flat roles) and RBAC1 model (hierarchical roles) of the RBAC96 family (Barka and Sandhu, 2000). However, users may want to delegate a piece of permission which is not supported in such models. This is the case when computing delegatee privileges.

The eXtensible Access Control Markup Language (XACML) was developed in order to provide a uniform way of specifying AC policies in XML (Moses, 2005). Policies comprising rules, possibly restricted by conditions, may be specified and targeted at resources, subjects and actions. Resources, subjects, actions and conditions are matched with information in an authorisation request context using attribute values and a rich set of value-matching functions. The outcome or effect of a policy evaluation may be permit, deny, not Applicable or indeterminate. Unlike other application-specific, proprietary access-control mechanisms, this standard can be specified once and deployed beyond the boundaries of organisations and countries. However, the current XACML standard does not provide explicit support for task delegation (Chadwick et al., 2006).

9 Conclusions and future work

Providing AC mechanisms to secure task delegation in workflow management systems is a non-trivial task to model and implement. In this article, we have presented problems and requirements that such a model demands, and developed a solution to model task delegation within workflows and to specify delegation policies into AC systems. The motivation of this work is based on a real world process from an e-government scenario, where a task delegation is required and may support dynamic changes when deploying a workflow.

The main contribution of this article is the development of a methodology with a logical framework to control and secure dynamic task delegation within workflows. To ease the monitoring of task delegation and the propagation of delegation authority, we have presented an event-based TDM amenable for supporting a logical framework to model, analyse and specify delegation policies. We then identified relevant events for authorisation enforcement to specify delegation policies based on a TAC model. Finally, we have applied formal methods to integrate delegation policies. Using EC, we have developed a reasoning tool to control the delegation execution and to increase the compliance of all delegation changes in the existing policy.

Our immediate priority is to complete the development of the delegation automation approach within an AC framework. We will work on implementing an event-based delegation policy component using the XACML standard. Further, we will ensure the synchronisation between the policy definition and the automation approach. Moreover, we will look at enriching our reasoning approach with historical records. Delegation history will be used to record all delegations that have been made for auditing. Auditing technique will allow us to choose the best candidate for delegation based on the delegatee's historical performance.

Acknowledgements

This paper is a revised and expanded version of a paper entitled 'Dynamic authorisation policies for event-based task delegation' presented at the 22nd International Conference on Advanced Information Systems Engineering (CAiSE '10) in Tunisia (Hammamet), June 9–11, 2010.

References

Atluri, V. and Warner, J. (2005) 'Supporting conditional delegation in secure workflow management systems', in *SACMAT '05: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, ACM, New York, NY, USA, pp.49–58.

Barka, E. and Sandhu, R. (2000) 'Framework for role-based delegation models', in *Proceedings of the 16th Annual Computer Security Applications Conference*, IEEE Computer Society, Washington, DC, USA, pp.168–176.

Bertino, E., Ferrari, E. and Atluri, V. (1999) 'The specification and enforcement of authorization constraints in workflow management systems', *ACM Trans. Inf. Syst. Secur.*, Vol. 2, No. 1, pp.65–104.

Botha, R.A. and Eloff, J.H.P. (2001) 'Separation of duties for access control enforcement in workflow environments', *IBM Systems Journal*, Vol. 40, No. 3, pp.666–682.

Chadwick, D.W., Otenko, S. and Nguyen, T-A. (2006) 'Adding support to XACML for dynamic delegation of authority in multiple domains', in *Communications and Multimedia Security, 10th IFIP TC-6 TC-11 International Conference, CMS 2006*, Heraklion, Crete, Greece, 19–21 October, pp.67–86.

Clavel, M., Silva, V., Braga, C. and Egea, M. (2008) 'Model-driven security in practice: an industrial experience', in *ECMDA-FA '08: Proceedings of the 4th European Conference on Model Driven Architecture*, Springer-Verlag, Berlin, Heidelberg, pp.326–337.

Crampton, J. and Khambhammettu, H. (2006) 'Delegation in role-based access control', in *Proceedings of the Computer Security – ESORICS 2006, 11th European Symposium on Research in Computer Security*, Hamburg, Germany, 18–20 September, Lecture Notes in Computer Science Springer, pp.174–191.

Crampton, J. and Khambhammettu, H. (2008a) 'Delegation and satisfiability in workflow systems', in *SACMAT '08: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, ACM, New York, NY, USA, pp.31–40.

Crampton, J. and Khambhammettu, H. (2008b) 'On delegation and workflow execution models', in *SAC '08: Proceedings of the 2008 ACM Symposium on Applied Computing*, ACM, New York, NY, USA, pp.2137–2144.

Freudenthal, E., Pesin, T., Port, L., Keenan E. and Karamcheti, V. (2002) 'dRBAC: distributed role-based access control for dynamic coalition environments', in *ICDCS '02: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS '02)*, IEEE Computer Society, Washington, DC, USA, p.411.

Gaaloul, K. (2010) 'A secure framework for dynamic task delegation in workflow management systems', PhD thesis, The University of Henri Poincaré, Nancy, France.

Gaaloul, K., Miseldine, P. and Charoy, F. (2009) 'Towards proactive policies supporting event-based task delegation', *SECURWARE '09: Proceedings of the 3rd International Conference on Emerging Security Information, Systems, and Technologies*, pp.99–104.

Gaaloul, K., Zahoor, E., Charoy, F. and Godart, C. (n.d.) 'Dynamic authorisation policies for event-based task delegation', in *Advanced Information Systems Engineering, 22nd International Conference, CAiSE 2010*, Hammamet, Tunisia, 7–9 June, pp.135–149.

Hagstrom, A., Jajodia, S., Parisi-Presicce, F. and Wijesekera, D. (2001) 'Revocations-A classification', in *CSFW '01: Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, IEEE Computer Society, Washington, DC, USA, p.44.

Hung, P.C.K. and Karlapalem, K. (2003) 'A secure workflow model', in *ACSW Frontiers '03: Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers*, Australian Computer Society, Inc., pp.33–41.

Kowalski, R. and Sergot, M. (1986) 'A logic-based calculus of events', *New Generation Computation Journal*, Vol. 4, No. 1, pp.67–95.

Mueller, E.T. (2006) *Commonsense Reasoning*, Morgan Kaufmann Publishers Inc., CA, USA.

Pfleeger, C.P. and Pfleeger, S.L. (2006) *Security in Computing*, 4th ed., Prentice Hall PTR, Upper Saddle River, NJ, USA.

R4eGov, Technical Annex (2006) 'Towards e-administration in the large', available at http://www.r4egov.eu/.

Russell, N., van der Aalst, W.M.P., Hofstede, A.H.M. and Edmond, D. (2005) 'Workflow resource patterns: identification, representation and tool support', in *Proceedings of the Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005*, Porto, Portugal, pp.216–232.

Sandhu, R.S., Coyne, E.J., Feinstein, H.L. and Youman, C.E. (1996) 'Role-based access control models', *IEEE Computer*, Vol. 29, No. 2, pp.38–47.

Schaad, A. (2003) 'A framework for organisational control principles', PhD thesis, The University of York, England.

Seitz, L., Rissanen, E., Sandholm, T., Firozabadi, B.S. and Mulmo, O. (2005) 'Policy administration control and delegation using XACML and delegent', in *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, IEEE Computer Society, Washington, DC, USA, pp.49–54.

Moses, T. (2005) *eXtensible Access Control Markup Language (XACML) Version 2.0*, Committee Specification, OASIS.

Venter, K. (2003) 'A model for the dynamic delegation of authorisation rights in a secure workflow management system', PhD thesis, Faculty of Science at the Rand Afrikaans University, Johannesburg, South Africa.

Wainer, J., Kumar, A. and Barthelmess, P. (2007) 'DW-RBAC: a formal security model of delegation and revocation in workflow systems', Information System, Vol. 32, No. 3, pp.365–384.

WFMC, The Workflow Management Coalition (1999) 'Workflow management coalition terminology and glossary', Document Number WFMC-TC-1011.

Zhang, L., Ahn, G-J. and Chu, B-T. (2003) 'A rule-based framework for role-based delegation and revocation', *ACM Trans. Information System Security*, Vol. 6, No. 3, pp.404–441.

Zur Muehlen, M. (2004) *Workflow-based Process Controlling. Foundation, Design, and Application of Workflow-driven Process Information Systems*, Logos Verlag Berlin.