Dynamic Authorisation Policies for Event-Based Task Delegation

Khaled Gaaloul, Ehtesham Zahoor, François Charoy, and Claude Godart

LORIA - Nancy University - UMR 7503
BP 239, F-54506 Vandœuvre-lès-Nancy Cedex, France
{kgaaloul,zahoor,charoy,godart}@loria.fr

Abstract. Task delegation presents one of the business process security leitmotifs. It defines a mechanism that bridges the gap between both workflow and access control systems. There are two important issues relating to delegation, namely allowing task delegation to complete, and having a secure delegation within a workflow. Delegation completion and authorisation enforcement are specified under specific constraints. Constraints are defined from the delegation context implying the presence of a fixed set of delegation events to control the delegation execution.
In this paper, we aim to reason about delegation events to specify delegation policies dynamically. To that end, we present an event-based task delegation model to monitor the delegation process. We then identify relevant events for authorisation enforcement to specify delegation policies. Moreover, we propose a technique that automates delegation policies using event calculus to control the delegation execution and increase the compliance of all delegation changes in the global policy.

Keywords: Workflow, task, delegation, policy, event calculus.

1 Introduction

The pace at which business is conducted has increased dramatically over recent years, forcing many companies to re-evaluate the efficiency of their business processes. The restructuring of organisational policies and methods for conducting business has been termed "Business Process Re-engineering" [1]. These refined business processes are automated in workflows that ensure the secure and efficient flow of information between activities and users that constitute the business process. Workflows aim to model and control the execution of business processes cross organisations. Typically, organisations establish a set of security policies, that regulate how the business process and resources should be managed [2].

In previous work, we argued that business processes execution are determined by a mix of ad-hoc as well as process-based interactions. 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 [3]. To address those issues, we present a task delegation approach as a mechanism supporting organisational flexibility in workflow management systems, and ensuring delegation of authority in access control systems

[4]. However, most of the work done in the area of business process security does not treat delegation in sufficient details. On one hand, Atluri *et al.* presented the Workflow Authorisation Model (WAM) that concentrates on the enforcement of authorisation flow in the domain of workflow security [5]. WAM remains static in time and poor in terms of delegation constraints within a workflow. On the other hand, existing work on access control systems do not consider dynamic enforcement of authorisation policies [6].

Moreover, a secure task delegation model has to separate various aspects of delegation within a workflow, where the interactions between workflow invariants (e.g., users, tasks and data) are triggered by delegation events. These delegation events will imply appropriate authorisation requests from access control systems. At present, responses arising from access control requests are stateless such that a response is given to an access request depending on predefined policies during the planning phase. If, however, this response changes due to a policy adaptation for delegation, no mechanism currently exists that allows the new response to be generated in the authorisation policy dynamically. Currently, when delegating a task, often the reasoning behind this is dependent on transient conditions called events. When one of these conditions changes during execution, our access policy decision may change. We do believe that delegation events define dynamic constraints for authorisation policies that should not be neglected in advanced security mechanisms supporting delegation.

The scope of the paper is to investigate the potential of delegation events to ensure a secure task delegation within a workflow. Securing delegation involves the definition of authorisation policies which are compliant with the workflow policy. In order to tackle these problems we need to address two important issues, namely allowing the delegation task to complete, and having a secure delegation within a workflow. Allowing task delegation to complete requires a task model that forms the basis of what can be analysed during the delegation process within a workflow. Secure delegation implies the controlled propagation of authority during task execution. Based on specific events, we define delegation policies in an automatic manner. In order to control the delegation behaviour and to specify its authorisation policies dynamically, we gather relevant events that will define both the task execution path and the generated policies for the delegation of authority. Using *Event Calculus* (EC), we will present a technique to monitor the delegation execution. Afterwards, we will briefly discuss the event calculus formalism to generate delegation policies, and finally, we explain how generated policies will change dynamically in response to task delegation events.

The remainder of this paper is organised as follows. Section 2 presents a workflow scenario to motivate our work. In section 3, we give an overview of our approach to reason about events to specify authorisation policies for delegation. In section 4, we present our task delegation model and explain how events will control its execution. Section 5 focuses on security requirements for delegation and its authorisation policy specifications using EC. In section 6, we motivate our technique to support delegation automation. Section 7 discusses related work. Finally, we conclude and outline several topics of potential future work.

2 Motivating Example: A Use Case Requiring Delegation Policies Integration

To understand the motivation of our research, we present a real world processes from an e-government case study requiring delegation. Mutual Legal Assistance (MLA) defines a workflow scenario involving national authorities of two European countries. 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 and activities that are part of the process are represented as tasks (see Figure 1).

In this scenario, the task "Translate Documents" T3 is originally only accessible by the user member of role *Prosecutor*, a fact defined in the workflow policy. We define a workflow policy as a level of defining access to task resources. We denote P an authorisation policy for the MLA process. This task is a long-running task and is expected to take 5 working days to complete. The *Prosecutor* 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. During delegation, the policy P is updated so that user with role *Assistant* is now allowed to complete task T3. To that end, he issues an access control request to the policy P to grant the access, and executes the task T3. As such, users with roles *Prosecutor* and *Assistant* are here the delegator and the delegatee, respectively.

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). Considering a user-to-user delegation, we motivated that such delegation is done in ad-hoc manner, thereby supporting a negotiation protocol. We consider negotiation as a fundamental step for delegation. It involves all the principals (delegator and delegatee) and negotiation specifications (e.g., time, evidence). Our intention is to envisage a wide-ranging request that gives flexibility for the delegation request. Subsequently, such specifications have to be included in the delegation policy to define specific conditions to validate the policy decision effect.

Returning to our example, the delegator *Prosecutor* sends a delegation request for all users members of role "Assistant". This defines a *push* delegation mode, where a delegatee is chosen dynamically based on the negotiation step. An acceptance of delegation inquires a new access control enforcement in the existing policy, thereby adding a new authorisation rule for the delegatee under defined 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 validation. Validation is done based on evidence defined during negotiation. Evidence can be related to the language of translated documents or the number of translated documents within 5 day. To that end, an authorisation rule permitting the access (e.g, *read, write, execute*) to the legal document in the MLA Information Service, is constrained by an obligation allowing to investigate whether evidence were satisfactorily met. If however,

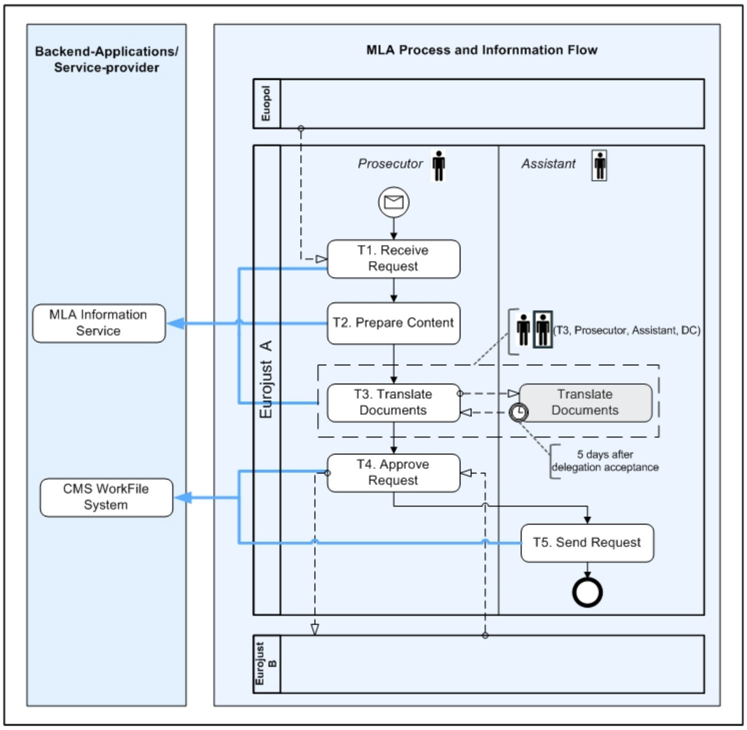

Fig. 1. MLA delegation scenario

evidence are not satisfied, a revoke action may be triggered including a deny result for the previous policy effect.

In traditional access control frameworks no mechanism exists that would support such delegation constraints. Delegation constraints are meant 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 access control rights and task executions is costly, labor intensive, and prone to errors. With an automated mechanism, when the policy changes to reflect delegation, the delegation policy will be derived automatically based on specific facts related to the delegation process. A delegation process defines a task delegation life cycle within the existing process. It is enriched with additional constraints to be compliant with the organisational policies. Organisational policies establish a set of security policies, that regulate how the business process and resources should be managed. Delegation constraints will inquire the need to support specific interactions that would be automatically captured, and specified in the delegation policies for appropriate actions. We do believe that such interactions are intermediate states in the delegation process driven by specified events to control the delegation behaviour within a workflow.

3 The Proposed Framework

The scope of the paper is to investigate the potential of delegation events to ensure a secure task delegation within a workflow. Securing delegation involves the definition of authorisation policies which are compliant with the global policy of the workflow. Therefore, these delegation events will imply appropriate authorisations on the delegatee side for further actions as well as contain specific constraints for those actions (e.g., mode, time, evidence). In order to tackle these problems we need to address two important issues, namely allowing the delegation task to complete, and having a secure delegation within a workflow. To that end, we introduce a delegation model that forms the basis of what can be analysed during the delegation process in terms of monitoring and security.

The monitoring of task delegation is an essential step to ensure delegation completion. A delegated task goes through different states to be terminated. States depends on generated events during the delegation life cycle. Events such as *revoke* or *cancel* are an integral part of the delegation behaviour. Revoking a task may be necessary when a task is outdated or an access right is abused. Moreover, additional 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. For instance, the delegation of T3 can generate evidence related to the number of translated documents within a period of 5 day. Subsequently, evidence validation will be an important step in the delegation process. Dealing with that, we came up with an event-based task delegation model (TDM) that can fulfill all these requirements. Our model aspires to offer a full defined model supporting all kind of task delegation for human oriented-interactions [3].

Additionally, we consider task delegation as an advanced security mechanism supporting policy decision. We define an approach to support dynamic delegation of authority within an access control framework. The novelty consists of reasoning on authorisation based on task 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 of authority. Existing work on access control systems remain stateless and do not consider this perspective. We propose a task delegation framework to support automated enforcement of delegation policies. Delegation policies are defined from existing policy and are specified from triggered events. For instance, T3 evidence are not satisfied and the validation will trigger the event *revoke* for the delegatee. T3 is not anymore authorised to be executed by the delegatee. In this case, another rule has to be integrated in policy with an effect of deny for the authorisation.

In order to control the delegation behaviour and to specify its authorisation policies in an automated manner, we gather specific events that will define both the task execution path and the generated policies for the delegation of authority. Using *Event Calculus*, we will present a technique to monitor the delegation execution. Afterwards, we will briefly discuss the event calculus formalism to generate delegation policies, and finally, we explain how generated policies change dynamically in response to task delegation events.

4 Task Delegation Model (TDM)

In this section, we present our task delegation model to monitor the delegation execution. Our model is based on events that covers different aspects of delegation. It defines how delegation request is issued and then executed depending on delegation constraints. The idea is to offer a technique to monitor delegation execution based on the triggered events. Using *Event Calculus*, we can foresee the delegation behaviour within its process.

4.1 Introduction to TDM

First, we present a detailed model of task execution that illustrates the delegation process. The task life cycle is based on additional events. The figure below depicts a state diagram 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 process. 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.

Our model is based on events that covers different aspects of delegation. It defines how a delegation request is issued. *Pull* mode assumes that a delegator has at his disposal a pool of delegatees to be selected to work on his behalf. *Push* mode assumes that a delegator is waiting for an acceptance from a potential delegatee [4]. Moreover, delegation of privileges can be classified into grant or transfer [7]. A *grant* delegation model allows a delegated access right (privileges) to be available for both delegator and delegatee. As such, the delegator is still having the control to *validate* or *revoke* the task, and the delegatee to *execute* it. However, in *transfer* delegation models, 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 (*complete/fail*) by the delegatee.

Each edge within the diagram is prefixed with either an S or U indicating that the transition is initiated by the workflow system or the human resource respectively, with $(u_1,u_2) \in U$ where U is a set of users, u_1 the delegator and u_2 the delegatee. In the following, we define a task delegation relation as follows:

Definition 1. We define a task delegation relation $RD = (t,u_1,u_2,DC)$, where t is the delegated task and $t \in T$ a set of tasks that composes a workflow, and DC the delegation constraints.

For instance, delegation constraints can be related to time or evidence specifications. Moreover, a role hierarchy (RH) defines the delegation relation condition in a user-to-user delegation. Returning to the example, the delegation relation (T3,Prosecutor,Assistant,(RH,5 days)) $\in RD$.

4.2 Modelling Task Delegation in Event Calculus

Background and motivations: The proposed approach for the representation of task delegation process relies on the *Event Calculus* (EC) [8]. The choice of EC is motivated by several reasons for delegation. Actually, given the representation of the task delegation model, policies and the corresponding events that trigger policy changes specified in the EC, an event calculus reasoner can be used to reason about them.

Event Calculus is a logic programming formalism for representing events and is being widely used for modeling different aspects such as flexible process design, monitoring and verification [9]. It comprises the following elements: $\mathcal{A}$ is the set of *events* (or actions), $\mathcal{F}$ is the set of fluents (fluents are *reified*[1]), $\mathcal{T}$ 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. Basic event calculus predicates used for modelling the proposed framework are:

- $Initiates(e, f, t)$ - 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 $[t1, t2]$.

The reasoning modes provided by event calculus can be broadly categorised into abductive, deductive and inductive tasks. In reference to our proposal, given a TDM and authorisation policies one may 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 leads to the "abduction reasoning". Then, one may also be 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" and using the event calculus is thus twofold.

The event calculus models discussed in this paper are presented using the discrete event calculus language [10] and we will only present the simplified models that represent the core aspects. In the models, all the variables (such as task, time) are universally quantified and in case of existential quantification, it is represent with variable name within curly brackets, {variablename}.

Event calculus based model: The basic entities in the proposed model are tasks. In terms of discrete event calculus terminology they can be considered

[1] Fluents are first-class objects which can be quantified over and can appear as the arguments to predicates.

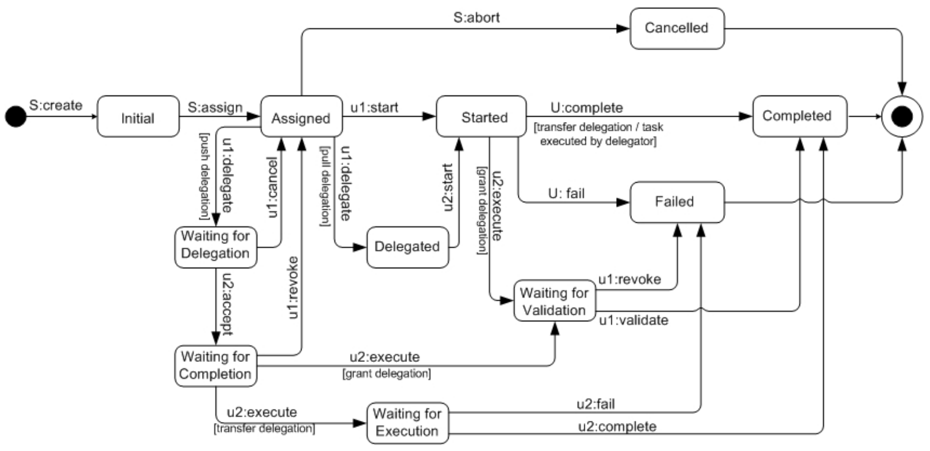

Fig. 2. Task delegation model

as sorts, of which instances can be created. Then, each task can be in different states during the delegation execution. In reference to the task delegation model 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 event calculus terminology. Further, the state change is governed by a set of actions/events and in relation to task delegation model, the task state changes from Initial to Assigned as a result of *assign* event occurring. Finally the task delegation model introduces a set of orderings, such as the state of a task cannot be assigned, if it is not created earlier. In reference to event calculus model, we will introduce a set of axioms to handle these dependencies. The event calculus model below introduces the fluents, basic events and dependency axioms:

The event calculus model presented above, first defines sort and fluents that marks 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

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*

Fig. 3. Event calculus based task delegation model

assignment event and fluent. We further introduce an axiom that specifies that in order to assign some task at time1, that task must already be created and thus in Initial state at time2, and time1 is greater than time2. In a similar fashion, we can define events and associated Initial axioms for the complete TDM model, space limitations restrict us to discuss them further.

For the basic event calculus 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 (see Figure 2), we enrich the model to include the following axioms:

```
[task, time] !Happens(Abort(task), time).
[task, time] !Happens(Start(task), time).
[task, time] !Happens(PullDelegate(task), time).
```

Fig. 4. Delegation mode choice

The event calculus model above, specifies that the task does not either Start, Abort or requires PullDelegation once assigned (and 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 WaitingCompletion state.

5 Authorisation Policies for TDM

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

5.1 Building Policies for Delegation

We define delegation transitions as events ruling delegation behaviour. The internal behaviour is based on events defined in our TDM, and may be a source to a policy change, thereby requiring the integration of additional authorisation rules.

Definition 2. We define a policy $P = (target, rule, C)$, 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 delegation policy is a policy $P_D = (target_D, rule_D, C_D)$, where $target_D = RD$, $rule_D \subseteq rule$ and $C_D \subset C$ and $C_D = DC \bigcup events$.

A policy rule may include conditions and obligations which are used to identify various conditions or cases under which a policy may become applicable. Based on

Table 1. Push delegation policy rules-based events

Delegation Events	Push Delegation		Adding Policy Rule
	Grant	Transfer	
u1:delegate	✓	✓	No add
u2:accept	✓	✓	Add rules based on execution type
u1:cancel	✓	✓	No add
u2:execute/Grant	✓		(Permit,Push,Grant:Evidence)
u2:execute/Transfer		✓	(Permit,Push,Transfer:NoEvidence)
u1:validate	✓		No add
u1:revoke	✓		(Deny,Push,Grant)
u2:fail		✓	(Deny,Push,Transfer)
u2:complete		✓	No add

the result of these rules different policies can become applicable. We define a rule as a tuple *(effect,condition,obligation)*, where effect returns the policy decision result (permit, deny), condition defines the delegation mode (push, pull) and obligation checks evidence. In the following, we analyse security requirements that need to be taken into account in a *push* mode to define delegation policy rules based on our TDM (see Figure 2). 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 1).

Returning to the example, we can observe a dynamic policy enforcement during delegation. Initially, T3 is delegated to the *Assistant* u_2 and the delegation policy for T3: P_D = *(RD,Permit,(Push,5 days))* (see Table 1/*u2:execute/Grant*). In the meanwhile, the *Prosecutor* u_1 is back to work before delegation is done and is not satisfied with the work progress and would revoke what was performed by his assistant so far. The *Prosecutor* is once again able to claim the task and will revoke the policy effect (permit) for the *Assistant*. The event *revoke* will be updated in the policy, and a deny rule is then updated in the policy. Thus, the delegation policy for T3 needs to generate a new rule and the delegation policy is updated to: P_D = *(RD,Deny,(Push,Grant))* (see Table 1/*u1:revoke*). Note that the generated rule depends on the *RD* relation to check access rights conflicts. We determined access rights granting based on the current task status and its resources requirements using a task-based access control model for delegation presented in previous work [3]. Our access control model ensures task delegation assignments and resources access to delegatees corresponding to the global policy constraints.

5.2 Modelling Delegation Policies in Event Calculus

In order to model the delegation policy rules-based events, we introduce new sorts called *effect*, *condition*, *obligation* to the event calculus model for the table defined above and specify instances of each sort to be the possible effects, conditions and obligations. Possible effects include *Deny* and *Permit* results, and

```
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)
```

Fig. 5. Delegation policy

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 5).

The policy change axioms presented above specify that once certain actions happen, they cause policy change and thus we add a new rule to the global 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 1).

6 Delegation Automation

In this section, we motivate our event-based approach supporting delegation automation. Automation is necessary for both the task completion and the policy specification. Reasoning on delegation events using *Event Calculus* offers a solution to foresee the delegation execution and increase the control and compliance of all delegation changes.

6.1 Benefits

Through this paper, we motivated the event-based approach for monitoring and securing task delegation. We observed that based on specific assumptions we are able to control any delegation request. We defined different valid orders of executions for delegation. The order of executions are computed automatically based on events. Events can 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.

Additionally, we defined a technique to specify delegation policies automatically. By reasoning on specific events, we are able to address the policy stateless issue. We can compute delegation policies from triggered events during task execution. Policy automation offers many benefits. Actually, it reduces efforts for users and administrators. Administrator efforts can be related to processes definition and policies specification. Moreover, it increases control and compliance of all delegation changes. Subsequently, a delegation request is accomplished under constraints which are compliant to the global policy. For instance, time constraint has to be taken into account when granting a temporal access for delegation (i.e.,T3 deadline in Figure 2).

6.2 Reasoning

In our study, we utilise the Discrete Event Calculus Reasoner (DECReasoner[2]) for performing automated commonsense reasoning using the event calculus, a comprehensive and highly usable logic-based formalism. It solves problems efficiently by converting them into satisfiability (SAT) problems.

The event calculus based on the task delegation model and policy specifications can be used to reason about the delegation process. As we discussed earlier, the reasoning task can either be abductive or deductive. For the abductive reasoning, a plan is sought for the specified goal. In reference to our proposal, the goal is to have either the task in completed, cancelled or failed state, so we add the goal *[task] HoldsAt(Completed(task),15) | HoldsAt(Failed(task),15) | HoldsAt(Cancelled(task),15)* to the event calculus model presented above and add an instance of the delegated task T3. The invocation of the event calculus 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 event calculus reasoner returns the plan as follows.

The execution plan follows the delegation of T3 described in the use case. It shows the action that need to be taken for delegation and most importantly, it shows the possible policy changes as a result of delegation. Steps from 1 to 11 depicts the delegation process execution. Having a *push* mode as a condition, we derive the relevant rules to add in the policy. For instance, step 8 and 9 show when a delegatee request the task T3 for execution. Delegatee acceptance went through the "WaitingDelegation", "WaitingAcceptance" and "WaitingCompletion" states (see Figure 2). Based on those events, we deduce that an authorisation rule is added at this stage under a certain obligation (evidence validation), and finally a task validation complete the delegation execution (see steps 10 and 11 in Figure 6).

All the defined axioms using the DECReasoner language can be given to the reasoner for finding a solution (if 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

[2] For more details: http://decreasoner.sourceforge.net/

```
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 3.1s - solution 0.7s - total 5.8s
```

Fig. 6. Delegation plan

(using EC), the generated plan by the reasoner shows that either the authorisation rules result in a permit or a deny decision[3].

In concrete policy changes, there are two possible scenarios. The first scenario is the integration of a new authorisation policy because the conjectures (conditions or obligations) are valid. The second one concerns cases corresponding to an overriding of this rule to a deny result. Note that, we can leverage the trace of DECReasoner to give all necessary information (events, fluents and timepoints) to help designer (policy administrator) to detect policies problems in the deployed process. Finally, the event calculus model can further be enriched to ensure minimal policy changes in the execution plans using auditing techniques. Space limitations restrict us to discuss them further.

7 Related Work

The Workflow Authorisation Model (WAM) presents a conceptual, logical and execution model that concentrates on the enforcement of authorisation flows in task dependency and transaction processing [5]. In addition, Atluri *et al.* discussed the specification of temporal constraints in a static approach, which is

[3] For space reasons, verification results and encoding details can be found at http://webloria.loria.fr/∼kgaaloul/caise2010/DelegReasoner.rar

not sufficient to support workflow security in general and task delegation in particular. This is due to workflows needing a more dynamic approach to synchronise the flow of authorisations during the workflow execution. The Workflow Authorisation Model does not discuss the order of operation flow such as our task delegation process. In a workflow, we need to investigate the delegation control in different aspects such as tasks, events and data by leveraging delegation constraints to support authorisation policies.

Role-based Access Control (RBAC) is recognised as an efficient access control model for large organisations. Most organisations have some business rules related to access control policy [11]. In [12,13], authors extend the RBAC96 model by defining some delegation rules. They proposed a flexible delegation model named Permission-based Delegation Model (PBDM), where users may want to delegate a piece of permission from a role [13]. However, neither RBAC nor PBDM support offer a suitable solution to support task delegation constraints. We do believe that constraints such as *Push/Pull* mode or *Grant/Transfer* privileges are essential for delegation and have an impact on the security requirements during policies specification [3].

The eXtensible Access Control Markup Language (XACML) is an XML-based, declarative access control policy language that lets policy editors specify rules about who can do what and when. Policies comprising rules, possibly restricted by conditions, may be specified and targeted at subjects, resources and actions. Subjects, resources, 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, NotApplicable or Indeterminate. In [14], Seitz and Firozabadi added new structured data-types to express chains of delegation and constraints on delegation using XACML. The main result of their research is an administrative approach that does not support ad-hoc delegation and lacks of explicit support for task delegation constraints.

8 Conclusion and Future Work

Providing access control mechanisms to support dynamic delegation of authority in workflow systems, is a non-trivial task to model and engineer. In this paper we have presented problems and requirements that such a model demands, and proposed a technique for delegation to specify delegation policies automatically. The motivation of this direction is based on a real world process from an e-government case study, where a task delegation may support changes during execution. Delegation policies may change according to specific events. We defined the nature of events based on task delegation constraints, and described their interactions with policies decisions. When relevant events occur, we define how delegation will behave and how policy rules change dynamically in response to this change. Using *Event Calculus* formalism, we implemented our technique and deployed a use case scenario for task delegation ensuring the required authorisation policy changes.

The next stage of our work is the implementation of our framework using XACML standard. We propose an extension supporting task delegation constraints with regards to the XACML conditions and obligations specifications. Future work will look also at enriching our approach with additional delegation constraints supporting historical records. Delegation history will be used to record delegation that have been made to address administrative requirements such as auditing.

References

1. Venter, K., Olivier, M.S.: The delegation authorization model: A model for the dynamic delegation of authorization rights in a secure workflow management system. In: CCITT Recommendation X.420, Blue Book (2002)
2. Vijayalakshmi, A., Janice, W.: Supporting conditional delegation in secure workflow management systems. In: SACMAT 2005: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies, pp. 49–58. ACM, New York (2005)
3. Gaaloul, K., Charoy, F.: Task delegation based access control models for workflow systems. In: I3E 2009: Proceedings of Software Services for e-Business and e-Society, 9th IFIP WG 6.1 Conference on e-Business, e-Services and e-Society, Nancy, France, September 23-25. IFIP, vol. 305. Springer, Heidelberg (2009)
4. Gaaloul, K., Miseldine, P., Charoy, F.: Towards proactive policies supporting event-based task delegation. In: The International Conference on Emerging Security Information, Systems, and Technologies, pp. 99–104 (2009)
5. Atluri, V., Huang, W., Bertino, E.: An execution model for multilevel seccure workflows. In: Proceedings of the IFIP WG11.3 Eleventh International Conference on Database Security, pp. 151–165. Chapman & Hall, Ltd., London (1998)
6. Bertino, E., Castano, S., Ferrari, E., Mesiti, M.: Specifying and enforcing access control policies for xml document sources. World Wide Web 3(3), 139–151 (2000)
7. Crampton, J., Khambhammettu, H.: Delegation in role-based access control. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 174–191. Springer, Heidelberg (2006)
8. Kowalski, R.A., Sergot, M.J.: A logic-based calculus of events. New Generation Comput. 4(1), 67–95 (1986)
9. Zahoor, E., Perrin, O., Godart, C.: A declarative approach to timed-properties aware Web services composition, INRIA internal report 00455405 (February 2010)
10. Mueller, E.T.: Commonsense Reasoning. Morgan Kaufmann Publishers Inc., USA (2006)
11. Sandhu, R., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer 29(2), 38–47 (1996)
12. Barka, E., Sandhu, R.: Framework for role-based delegation models. In: ACSAC 2000: Proceedings of the 16th Annual Computer Security Applications Conference, Washington, DC, USA, p. 168. IEEE Computer Society, Los Alamitos (2000)
13. Zhang, X., Oh, S., Sandhu, R.: PBDM: a flexible delegation model in RBAC. In: SACMAT 2003: Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies, pp. 149–157. ACM Press, New York (2003)
14. Seitz, L., Rissanen, E., Sandholm, T., Firozabadi, B., Mulmo, O.: Policy administration control and delegation using xacml and delegent. In: Proceedings of 6th IEEE/ACM International Conference on Grid Computing (GRID 2005), Seattle, Washington, USA, November 13-14, pp. 49–54 (2005)