

Delegation and Satisfiability in Workflow Systems

Jason Crampton
Information Security Group
Royal Holloway, University of London
jason.crampton@rhul.ac.uk

Hemanth Khambhammettu
Information Security Group
Royal Holloway, University of London
h.khambhammettu@rhul.ac.uk

ABSTRACT

Supporting delegation mechanisms in workflow systems is receiving increasing interest from the research community. An important requirement of a constrained workflow is to guarantee the satisfiability of the workflow, which requires that some set of authorized users can complete a workflow. Typically, any mechanism that is used to establish the satisfiability of a workflow is based on the workflow specification and the user authorization information. The effect of a successful user delegation request is to change the user authorization information, thereby affecting the satisfiability of the workflow.

Existing work on delegation in workflows does not consider the satisfiability of the workflow. In this paper, we address the satisfiability problem of workflows, while supporting user delegation mechanisms, in the context of three different workflow execution models. We consider delegation of concrete tasks, abstract tasks and roles. We present algorithms for evaluating various delegation requests in each workflow execution model.

Categories and Subject Descriptors

H.4.1 [Information Systems Applications]: Office Automation—*Workflow management*; K.6.5 [Management of Computing and Information Systems]: Security and Protection

General Terms

Security

Keywords

Delegation, Satisfiability, Workflow management systems

1. INTRODUCTION

A workflow comprises various activities that are involved in an organizational or a business process. A *computer-*

ized workflow system automates the management and coordination of such processes. In a computerized workflow *activities* that are part of a process are represented as *tasks*. Authorization information is given which authorizes users to perform tasks. Such authorization information may be specified using a simple access control list or more complex role-based structures. Users may perform tasks for which they are authorized.

In traditional access control, *user delegation* is a mechanism that permits a user to assign a subset of his assigned authorizations to other users who currently do not possess the authorization. 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’. A user may perform a *grant* or a *transfer* delegation operation. A successful grant delegation operation allows a delegated access right to be available to both the delegator and delegatee. However, following a transfer delegation operation, 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.

In the context of a workflow system, users may perform *abstract* task delegations or *concrete* task delegations. An abstract task delegation authorizes the delegatee to perform the delegated task in any instance of the workflow, whereas a concrete task delegation authorizes the delegatee to perform the delegated task *only* in the specified workflow instance.

A *constrained workflow* is a workflow specification that specifies constraints on the execution of tasks. For example, we may specify a simple constrained workflow with two tasks t and t' and a constraint that states no user can perform both t and t' . An important requirement of a constrained workflow specification is to guarantee the *satisfiability* of the workflow, which requires that some set of authorized users *can complete* a workflow. Typically, any mechanism that is used to establish the satisfiability of a workflow is based on the constrained workflow specification and the user authorization information.

The effect of a successful user delegation request is to change the user authorization information, thereby affecting the satisfiability of the workflow. Hence, it is important to ensure that permitting a delegation request does not render the workflow unsatisfiable. A reference monitor must consider the following six questions while supporting user delegation operations in workflow management systems (WfMS): Is user u authorized to delegate task t ? Is user v authorized to receive delegation of task t ? Are all authorization constraints satisfied if task t is delegated to v ? Can

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SACMAT'08, June 11–13, 2008, Estes Park, Colorado, USA.
Copyright 2008 ACM 978-1-60558-129-3/08/06 ...\$5.00.

the workflow instance complete if u delegates an instance of task t to v ? Does the workflow authorization schema remain satisfiable if u delegates an authorization of t to v ? Can all workflow instances complete if u delegates an authorization of t to v ?

Existing work on delegation in the context of workflow systems has considered the first three questions [1, 2, 5, 9, 10]. To our knowledge, however, no work has addressed the last three questions. We previously discussed the effect of successful delegation operations in three different workflow execution models [6]. In this paper, we study the satisfiability problem of workflows, while supporting user delegation operations, in the context of these execution models. The following are the three main contributions of this paper.

- We consider the satisfiability problem of workflow while delegating concrete tasks, abstract tasks and roles in three different workflow execution models.
- We present algorithms for delegation operations in each workflow execution model, which guarantee that permitting a delegation request does not prevent the completion of workflow instances and/or render the workflow authorization schema unsatisfiable.
- We determine the computational time complexity of the algorithms for evaluating various delegation requests.

The rest of the paper is organized as follows. In the next section, we discuss relevant work of the literature and motivate our work. In Section 3, we discuss the background for this paper. Section 4 describes workflow satisfiability and relevant algorithms. We discuss delegation of tasks and how they affect the completion of workflow instances and the satisfiability of the workflow in Section 5. In particular, we discuss delegation of task instances in Section 5.1 and delegation of abstract tasks in Section 5.2. We discuss delegation of roles in Section 6. Section 7 concludes this work and presents future work.

2. RELATED WORK AND MOTIVATION

Atluri *et al* studied delegation in workflow systems [1]. In this work, a user initiates a delegation request of an abstract task t and the delegation request is permitted on satisfaction of certain conditions. Such conditions ensure that no authorization constraint, such as separation of duty and binding of duty constraints, is violated. Subsequently, the WfMS assigns an instance of the delegated task t to the delegatee if no (dynamic) authorization constraint is violated. This work has further been extended to provide support for conditional delegation in workflow systems [2]. Such delegation conditions are based on time, workload and task attributes.

Wainer *et al* proposed a framework for delegation in workflow systems [10]. The framework provides support for delegating both concrete tasks and abstract tasks; however, the framework does not include *transfer* delegation of abstract tasks. A delegation request is only permitted if certain conditions are satisfied. Such conditions require that the delegator is authorized to perform a delegation, the delegatee is authorized to receive the delegation and no authorization constraint is violated.

Venter and Olivier also consider delegation in workflow systems [9]. Their delegation model ensures that the dele-

gated authorization t is available for the delegatee only during the specified “authorization time” intervals during which t can be executed.

We previously discussed the effect of successful delegation operations in three different workflow execution models [6]. However, none of the above works require that permitting a delegation request does not prevent the completion of workflow instances and/or render the workflow authorization schema unsatisfiable.

We illustrate the satisfiability problem by considering delegation of an abstract task in a constrained workflow. Consider, for example, a simple workflow W with two tasks t and t' , two users u and v such that u is authorized to perform t and v is authorized to perform t' and a constraint that states no user can perform both t and t' . Clearly, such a workflow is satisfiable since we can assign tasks to users such that the above constraint is not violated. Assume that user u successfully performs a grant delegation of t to v . Following the success of this grant delegation, both u and v are authorized to perform t . We can still assign all tasks in W to users in an instance of $W : [(t, u), (t', v)]$, such that the above constraint is not violated, so W remains satisfiable.

However, a successful transfer delegation of t to v by u , would mean that the delegator u is no longer authorized to perform t and the delegatee v is authorized to perform both tasks t and t' . It can easily be seen that if we assign t to v in a workflow instance, then we can not assign task t' to a user such that the above constraint is not violated. In other words, permitting the above transfer delegation will result in a workflow that cannot be executed by authorized users while simultaneously enforcing the constraints specified on task execution. Hence, the transfer delegation must be denied in order to ensure the satisfiability of W .

3. PRELIMINARIES

In this section, we summarize a model for constrained workflows [3], describe three different workflow execution models [6] and classify delegation operations.

Let U be a set of users and let $Rel(U)$ denote the set of all binary relations on U . In other words, $Rel(U)$ is the powerset of $U \times U$.

3.1 Constrained workflows

DEFINITION 1. A workflow specification W is a partially ordered set of tasks (T, \leq) , where $T = \{t_1, \dots, t_n\}$. A workflow authorization schema is a pair (T, A) , where $A \subseteq T \times U$; u is authorized to perform (or execute) t iff $(t, u) \in A$.

If $t < t'$ then t must be performed before t' in any instance of the workflow. We assume that the user authorization information A will be inferred from access control data structures, such as user-role assignment and task-role assignment relations [8], or access control lists.

DEFINITION 2. An entailment constraint has the form $(D, (t, t'), \rho)$, where $D \subseteq U$, $\rho \in Rel(U)$ and $t \not\geq t'$. D is the domain of the constraint; t is the antecedent task; and t' is the consequent task.

An entailment constraint places some restriction on the users who can perform t' given that $u \in D$ has performed t . Hence a separation of duty constraint that prevents any user from performing both t and t' can be expressed as

$(U, (t, t'), \neq)$, and a binding of duty constraint that requires t and t' be performed by the same user can be expressed as $(U, (t, t'), =)$.

The relation \preceq will be used to denote an ordering on the set of users, which may be derived, depending on context, from role information, organizational information or the user groups to which users belong. In particular, \preceq can be defined in terms of A in the following way: $u \preceq u'$ iff $\{t : (t, u) \in A\} \subseteq \{t : (t, u') \in A\}$. Notice that \preceq is a pre-order (not a partial order, since we may have two users $u \neq u'$ with $u \preceq u'$ and $u \succcurlyeq u'$). We define $u \simeq u'$ iff $u \preceq u'$ and $u \succcurlyeq u'$; we say u' is more senior than u and write $u \prec u'$ if $u \preceq u'$ and $u \not\simeq u'$. A constraint of the form $(U, (t, t'), \prec)$ requires that t' is performed by a user more senior than the one who performed t .

DEFINITION 3. *If users u and u' perform t and t' , respectively, and $u \in D$, then constraint $(D, (t, t'), \rho)$ is satisfied iff $(u, u') \in \rho$.*

DEFINITION 4. *A constrained workflow authorization schema is a triple (T, A, C) , where C is a set of entailment constraints.*

Crampton previously noted that linear extensions are important in the context of workflows because of their ability to “linearize” a partially ordered set of tasks [3]. In particular, a linear extension of T represents a possible sequence of execution of the tasks in a workflow. We write L to denote a linear extension of T . We denote the set of linear extensions of T by $\mathcal{L}(T)$.

DEFINITION 5. *Let $W = (T, A, C)$ be a constrained workflow authorization schema. An execution schedule for W is a pair (L, α) , where $L \in \mathcal{L}(T)$ and $\alpha : T \rightarrow U$ is an assignment of tasks to users. We say (L, α) is a valid execution schedule iff for all $t \in T$, $(t, \alpha(t)) \in A$ and for all $(D, (t, t'), \rho) \in C$, if $u \in D$ then $(\alpha(t), \alpha(t')) \in \rho$.*

In other words, an execution schedule (L, α) is valid if $L \in \mathcal{L}(T)$ and $\alpha : T \rightarrow U$ is an assignment of tasks to users such that no constraint is violated.

We write W to denote an instance of the workflow $W = (T, A, C)$. A concrete task $t \in W$ is an instance of an abstract task $t \in T$.

DEFINITION 6. *An instance W of a workflow specification $W = (T, A, C)$ is a list of task-user pairs $[(t_1, u_1), \dots, (t_k, u_k)]$, where $[t_1, \dots, t_k]$ is a prefix of some linear extension of T and denotes the set of tasks that have been completed in W .*

An instance $[(t_1, u_1), \dots, (t_k, u_k)]$ means that user u_i has successfully performed task t_i . Implicitly, this means that $(t_i, u_i) \in A$ and that all constraints in C are satisfied.

At any given time point, there are zero or more instances of the workflow schema W . We write W_i to denote a particular instance of W . Generally, we simply write W when it is clear from the context. We write $\mathcal{W} = \{W_1, \dots, W_m\}$ to denote the set of partially completed instances of W .

3.2 Workflow execution models

A workflow may be executed in (at least) two ways: either tasks that are to be performed may be assigned to users by the WfMS or users may initiate requests to perform tasks. We briefly describe the three workflow execution models that we previously discussed in [6].

DEFINITION 7. *A static workflow-driven execution model (WDEM-S) is a workflow model that pre-determines the set of users who must perform tasks $\{t_1, \dots, t_n\}$, whenever W is instantiated.*

That is, when the WfMS instantiates a workflow specification $W = (T, A, C)$, a static tasklist $TL = \{(t_1, u_1), \dots, (t_n, u_n)\}$ is created, such that $\{u_1, \dots, u_n\} \subseteq U$, $(t_i, u_i) \in A$, $1 \leq i \leq n$ and no constraint $c \in C$ is violated. In other words, a static tasklist TL is a valid execution schedule for W .

We write TL_i to denote the unique tasklist that is associated with a particular workflow instance $W_i \in \mathcal{W}$. We simply write TL whenever it is clear from the context. We write $\mathcal{TL} = \{TL_1, \dots, TL_m\}$ to denote the set of all tasklists.

Recall that a workflow instance $W = \{(t_1, u_1), \dots, (t_k, u_k)\}$, $1 \leq k \leq n$, specifies the set of tasks in W that have been completed by the assigned user (according to the tasklist). We define $E_i = TL_i \setminus W_i = \{(t_{k+1}, u_{k+1}), \dots, (t_n, u_n)\}$ that specifies the set of valid execution assignments for all uncompleted tasks in the workflow instance W .

DEFINITION 8. *A dynamic workflow-driven execution model (WDEM-D) is a workflow model that, during runtime, determines the next task t to be performed in an instance and selects a user u to perform the selected task t .*

In a WDEM-D, the WfMS instantiates a process definition but does not create a tasklist for the workflow instance. Instead, the WfMS assigns each task that is ready to be performed to a user during runtime.

While assigning a task t to a user u , the WfMS ensures that u is authorized to perform t , no constraint is violated as a result of the execution assignment (t, u) and the execution assignment (t, u) will not prevent the completion of the workflow instance W . In other words, a dynamic tasklist TL is a *satisfiable instance* of W . (As before, we define $E_i = TL_i \setminus W_i$.)

DEFINITION 9. *A user-driven workflow execution model (UDEM) is a workflow model that is responsible only for instantiating and managing workflow instances.*

In a UDEM, users initiate requests to perform tasks in a workflow instance. An access control mechanism determines whether to permit a user u to perform a task t . Typically, such a request is permitted if $(t, u) \in A$, no constraint is violated in the workflow instance, and the workflow instance can complete, and denied otherwise.

3.3 Delegation in workflows

Informally, delegation in a workflow system is the act of (temporarily) authorizing a user to perform a task. The semantics of a delegation operation are based on three factors: the execution model, whether it is the delegation of a task in a workflow specification (an “abstract” task) or a task in a particular workflow instance (a “concrete” task), and whether it is a grant or transfer of authorization. A transfer operation authorizes the delegatee to perform a task, but the delegator may no longer perform the task; a grant operation authorizes the delegatee to perform a task, but the delegator may still perform the task.

Table 1 summarizes these delegation operations and the execution models to which they apply. We assume that a

Table 1: A summary of delegation operations in WfMS

Execution model	Concrete task		Abstract task		
	Grant	Transfer	Grant	Non-cascading transfer	Cascading transfer
WDEM	n/a	Yes	Yes	Yes	Yes
UDEM	n/a	n/a	Yes	Yes	n/a

concrete task is performed by a single user. In other words, no more than one user can be assigned to a concrete task. Hence, we do not consider a grant delegation of a concrete task in a WDEM. In a UDEM, users select tasks to execute. This means that tasklists are not part of such systems. Hence, we do not need to consider delegation of concrete tasks in a UDEM.

In a WDEM, while performing an abstract task transfer delegation of t , we need to consider the effect of existing assignments of t (defined in relevant tasklists) on the delegator and delegatee. In particular, what should happen to any instances of t that are currently assigned to the delegator? A *non-cascading* transfer means that the delegator must perform all instances of t that are currently assigned to him; a *cascading* transfer means that the delegatee must perform all such task assignments. As before, in a UDEM, the lack of tasklists mean that we do not need to consider cascading transfer delegations in such systems.

4. WORKFLOW SATISFIABILITY

Given a constrained workflow authorization schema $W = (T, A, C)$, the following three questions are of interest to us: Is an execution schedule (L, α) valid? Does there exist a valid execution schedule for W ? Can a workflow instance successfully complete?

Establishing whether an execution schedule is valid is done by referring to the list of task-user assignment pairs and workflow specification. Figure 1(a) illustrates the algorithm `isValidSched()`, which takes a list of task-user assignment pairs and the set of constraints as input parameters and returns either *true* or *false*. The algorithm simply checks whether each constraint is satisfied (steps 01-02). The total number of constraints is bounded by $|T|^2$ [3]; hence, step 01 is executed at most $|T|^2$ times. The complexity of step 02 depends on the constraints that are specified on the execution of tasks. In particular, if we specify only separation of duty and binding of duty constraints then step 02 is a simple comparison. However, if we allow the specification of seniority constraints (or arbitrary binary relations on U) then step 02 requires $\mathcal{O}(|U|^2)$ computations. Hence, the overall time complexity of the algorithm `isValidSched()` is $\mathcal{O}(|U|^2|T|^2)$ in the general case; and is $\mathcal{O}(|T|^2)$ if we only allow the specification of separation of duty and binding of duty constraints.

Determining whether there exists a valid execution schedule for $W = (T, A, C)$ is more costly. Figure 1(b) illustrates the algorithm `isSchemaSatisfiable()` that determines if there exists a valid execution schedule for the workflow schema W . The algorithm takes the constrained workflow authorization schema $W = (T, A, C)$ as input, and returns either *true* or *false*. Essentially, the algorithm computes all possible execution schedules for W and checks if there exists at least one execution schedule that is valid. Note that the number of possible execution schedules for W is bounded by

$|U|^{|T|}$ (step 01). The algorithm uses `isValidSched()` (Figure 1(a)) as a sub-routine to check if an execution schedule is valid (step 02). The overall time complexity of the algorithm `isSchemaSatisfiable()` is $\mathcal{O}(|U|^{|T|})$.

Crampton previously observed that a partially completed workflow instance can be regarded as a workflow specification where exactly one user is authorized for each completed task [3]. Hence, we can determine whether a workflow instance can complete by modifying the user authorization information A to obtain a new constrained workflow authorization schema. Specifically, given a partially completed workflow instance $W = [(t_1, u_1), \dots, (t_k, u_k)]$, $1 \leq k < n$, we set $A' = \{(t_1, u_1), \dots, (t_k, u_k)\} \cup \{(t', u) \in A : t' \notin \{t_1, \dots, t_k\}\}$ and test whether the workflow authorization schema is satisfiable (using A').

Figure 1(c) illustrates the algorithm `isInstanceSatisfiable()` that determines whether a workflow instance can complete. The algorithm takes the workflow authorization schema $W = (T, A, C)$ and a partially completed workflow instance W as inputs, and returns either *true* or *false*. In step 01, the algorithm updates the user authorization information. In step 02, the algorithm uses `isSchemaSatisfiable()` (Figure 1(b)) as a sub-routine to check the satisfiability of the workflow schema using the updated authorization information. The algorithm returns the result of algorithm `isSchemaSatisfiable()`. If k tasks have been completed, then there are a maximum of $|U|^{|T|-k}$ possible execution schedules to check. Hence, the overall running time of the algorithm `isInstanceSatisfiable()` is $\mathcal{O}(|U|^{|T|-k})$.

Crampton observed that it is sufficient to consider no more than m^{m-1} execution schedules to determine schema satisfiability, where $m \leq |T|$, in the particular case that all constraints are separation of duty constraints. Typically, m will be considerably smaller than $|T|$: its value is related to the number of users authorized for each task; in particular, $m = 0$ if there are at least $|T|$ authorized users for each task [4].

More formally, Wang and Li have shown recently that workflow satisfiability is fixed parameter tractable [7] if $rel \in \{\neq, =\}$ [11]. Hence, for workflow schemata that only use separation of duty constraints (by far the most commonly used type of constraint), the performance of `isSchemaSatisfiable()` (and hence `isInstanceSatisfiable()`) will be acceptable in practice.

5. TASK DELEGATION

In this section, we discuss delegation of both concrete tasks and abstract tasks in the context of three different workflow execution models.

5.1 Delegating concrete tasks

As mentioned earlier, concrete task delegations occur only in WDEMs. In a WDEM, a user may delegate a concrete task that has been assigned to him to another user. Sup-

Inputs: execution schedule $[(t_1, u_1), \dots, (t_n, u_n)]$
 set of constraints C

```

01 for each constraint  $(D, \{t_i, t_j\}, rel) \in C$ 
02   if  $(u_i, u_j) \notin rel$  then return false
03 return true

```

(a) Algorithm `isValidSched()`

Inputs: set of tasks T
 user authorization information A
 set of constraints C

```

01 for each execution schedule  $(L, \alpha)$ 
02   if  $(isValidSched((L, \alpha), C))$  then return true
03 return false

```

(b) Algorithm `isSchemaSatisfiable()`

Inputs: set of tasks T
 user authorization information A
 set of constraints C
 workflow instance $W = \{(t_1, u_1), \dots, (t_k, u_k)\}$

```

01 let  $A' = \{(t_1, u_1), \dots, (t_k, u_k)\} \cup \{(t', u) \in A : t' \notin \{t_1, \dots, t_k\}\}$ 
02 return  $isSchemaSatisfiable(T, A', C)$ 

```

(c) Algorithm `isInstanceSatisfiable()`

Figure 1: Algorithms for determining workflow satisfiability

pose, for example, a user u wishes to delegate a concrete task t to another user v . (We may assume that u is authorized to perform t since u has been assigned the task t by the WfMS.) After a successful delegation, the delegatee v is required to perform task t (and u may no longer perform t in the particular workflow instance).

5.1.1 WDEM-S

A successful delegation of a concrete task t to v by u results in the assignment of t to the delegatee v . Hence, $TL' = (TL \cup \{(t, v)\}) \setminus \{(t, u)\}$. (That is, we simply update the tasklist TL by replacing the execution assignment (t, u) with (t, v) .) However, we require that permitting a delegation of task t does not prevent the completion of the workflow instance. In order to evaluate a request initiated by user u to delegate t to v , we update the tasklist appropriately and check if the updated tasklist TL' is a valid execution schedule by running the algorithm `isValidSched()`. Hence, the delegation is permitted if `isValidSched(TL', C)` returns *true*, and denied otherwise.

Consider, for example, the constrained workflow authorization schema illustrated in Figure 2. Figure 2(a) depicts a workflow W in which a solid edge (t, t') indicates that $t < t'$; the broken edge indicates the existence of a constraint between t_2 and t_3 . Labelled edges indicate a constraint exists on the execution of the two tasks; for example in Figure 2(a) t_5 must be performed by a user who is more senior than the user that performed t_3 . Figure 2(b), Figure 2(c) and Figure 2(d) show the role hierarchy, task-role assignments and user-role assignments, respectively.

Let the tasklist $TL = \{(t_1, b), (t_2, a), (t_3, c), (t_4, a), (t_5, b)\}$ and assume that user c initiates a request to delegate task t_3 to user a . In order to evaluate the above delegation, we set $TL' = \{(t_1, b), (t_2, a), (t_3, a), (t_4, a), (t_5, b)\}$ and run the algorithm `isValidSched(TL', C)`. We find that the con-

straint $(U, (t_2, t_3), \neq)$ is violated. Hence, the request to delegate t_3 to a will be denied. However, if c delegates t_3 to d then $TL' = \{(t_1, b), (t_2, a), (t_3, d), (t_4, a), (t_5, b)\}$ and all constraints are satisfied. Hence, a delegation of t_3 to a will fail, whereas a delegation of t_3 to d will succeed.

Recall that, in a WDEM-S, the tasklist is always a valid execution schedule for W . Hence, in order to determine whether the updated tasklist TL' is a valid execution schedule, we only need to consider constraints that apply to the task that is being delegated. In the above example, in order to evaluate the delegation of t_3 , we only need to consider constraints $\{(U, (t_2, t_3), \neq), (U, (t_3, t_5), <)\}$ and the task assignments for the tasks in those constraints. In other words, the size of the inputs to `isValidSched()` will, generally, be less than $|C|$ and $|T|$.

5.1.2 WDEM-D

Recall that in a WDEM-D the WfMS determines the set of tasks that are ready to be performed during runtime and appends execution assignments to the tasklist. Recall also that the WfMS ensures the completion of workflow instance W while assigning tasks to users. Consider, for example, the workflow in Figure 2, after t_1 has successfully been completed, task t_2, t_3 or t_4 may be performed. Hence, once t_1 has successfully been performed the WfMS appends valid execution assignments for t_2, t_3 and t_4 to the tasklist. For example, in Figure 2 if user b successfully performed t_1 then the WfMS can not assign t_3 to user a because a is the only user authorized to perform t_2 and there exists a constraint $(U, (t_2, t_3), \neq)$. However, the WfMS may assign tasks t_3 and t_4 to users c and a , respectively, and the tasklist $TL = \{(t_1, b), (t_2, a), (t_3, c), (t_4, a)\}$ is a satisfiable instance.

The effect of a successful concrete task delegation in WDEM-D is the same as in WDEM-S. That is, a successful delegation of t to v by u results in (t, u) being replaced

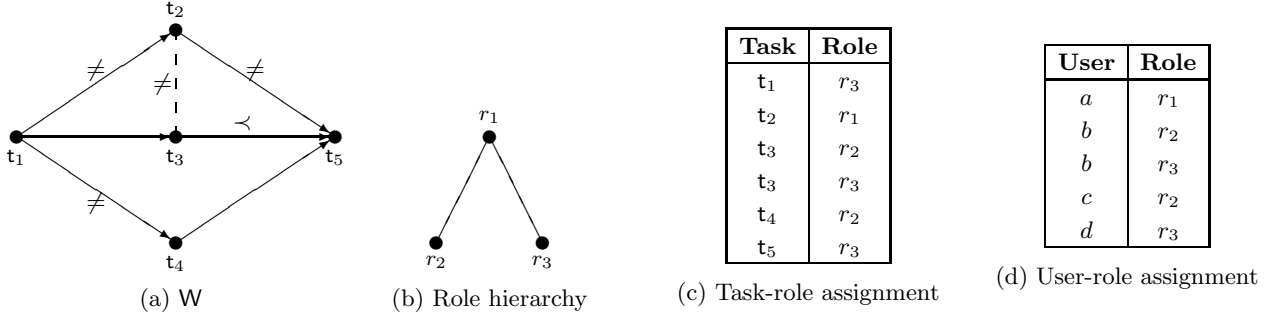


Figure 2: An example of a constrained workflow

by (t, v) in the tasklist. As before, we require that permitting a concrete task delegation request does not prevent the completion of W . That is, a concrete task delegation of t to v by u is only permitted if W can complete using the execution assignments defined in the updated tasklist TL' . Specifically, we set $TL' = (TL \cup \{(t, v)\}) \setminus \{(t, u)\}$ and run the algorithm $\text{isInstanceSatisfiable}(T, A, C, TL')$ (Figure 1(c)). The delegation request is permitted if $\text{isInstanceSatisfiable}(T, A, C, TL')$ returns *true*, and denied otherwise.

Consider, for example, the workflow in Figure 2 and let $TL = \{(t_1, b), (t_2, a), (t_3, c), (t_4, a)\}$. Assume that user c wishes to delegate concrete task t_3 to user b . We set $TL' = \{(t_1, b), (t_2, a), (t_3, b), (t_4, a)\}$ and run the algorithm $\text{isInstanceSatisfiable}(T, A, C, TL')$. Then we need to consider all execution schedules that include $[(t_1, b), (t_2, a), (t_3, b), (t_4, a)]$. The only users authorized for task t_5 are holders of roles r_1 and r_3 ; that is, users a, b and d . Hence, we need to consider the following execution schedules:

$$\begin{aligned} & \{(t_1, b), (t_2, a), (t_3, b), (t_4, a), (t_5, a)\}, \\ & \{(t_1, b), (t_2, a), (t_3, b), (t_4, a), (t_5, b)\}, \\ & \{(t_1, b), (t_2, a), (t_3, b), (t_4, a), (t_5, d)\}. \end{aligned}$$

Then, we check if at least one of the above execution schedules is valid. Note that $c < b < a$ and $d < b < a$, so at least one constraint is violated in each of the above execution schedules. That is, should the delegation of t_3 to b succeed then we can not find an authorized user to perform t_5 such that all constraints are satisfied. Hence, the delegation of t_3 to b must be denied. However, a delegation of t_3 to d will be permitted since there exists a satisfiable instance for W using TL' , such that all constraints are satisfied, for example $\{(t_1, b), (t_2, a), (t_3, d), (t_4, a), (t_5, b)\}$.

5.2 Delegating abstract tasks

A successful grant delegation of an abstract task t monotonically increases the set of available authorizations. In particular, a successful grant delegation of t to v by u does not alter any (task-user) execution assignments in the tasklist; but will change the authorization information by including an authorization that permits v to execute t .¹ Hence, monotonic operations, such as a *grant* delegation, do not affect the satisfiability of the workflow schema. Hence, for the

¹Hereafter, the delegator is assumed to be authorized for the task to be delegated.

purposes of our discussion, grant delegations are less interesting.

However, a successful transfer delegation of an abstract task t changes the authorization information that permits the delegatee to execute t , but prohibits the delegator from executing t . Furthermore, in WDEMs, a successful cascading transfer operation of t will also change relevant execution assignments in appropriate tasklists. Hence, we must ensure that permitting a transfer delegation request of an abstract task t does not prevent the completion of workflow instances and that the workflow authorization schema remains satisfiable. In this section, we consider both non-cascading and cascading transfer delegation of abstract tasks.

5.2.1 WDEM-S

A successful non-cascading transfer delegation of t to v by u will authorize v to execute t , but u may no longer execute t . Hence, $A' = (A \cup \{(t, v)\}) \setminus \{(t, u)\}$. Recall that a successful non-cascading transfer delegation of t will not alter any tasklists and will only change user authorization information. Hence, a transfer delegation of t is only permitted if we can guarantee the satisfiability of the workflow authorization schema $W = (T, A', C)$. Specifically, a transfer delegation of t to v by u is permitted if $\text{isSchemaSatisfiable}(T, A', C)$ (Figure 1(b)) returns *true*, and denied otherwise. Hence, the time complexity of evaluating a non-cascading transfer delegation of t is $\mathcal{O}(|U|^{|\mathbb{T}|})$.

A successful cascading transfer of an abstract task t to v by u will transfer both u 's authorization to execute t to v and u 's existing execution assignments of t to the delegatee v . Hence, a cascading transfer of an abstract task t is permitted only if we can guarantee both the satisfiability of the workflow schema W and the completion of all instances of W . Figure 3 illustrates an algorithm that determines whether to permit a cascading transfer of an abstract task t . The algorithm takes the workflow schema $W = (T, A, C)$, the set of tasklists, the set of workflow instances and the delegation request parameters as inputs, and returns either *true* or *false*. Essentially, the algorithm checks for the satisfiability of all partially completed workflow instances (steps 01-06) and the satisfiability of the workflow schema (steps 07-08).

Note that the cascading effect of the transfer delegation on concrete task assignments is only applied on uncompleted tasks. Hence, we only need to consider the completion of workflow instances in which the delegation task has not yet been performed. The algorithm updates the tasklist(s) associated with such a workflow instance(s) and checks whether the updated tasklist is a valid execution schedule, using the

```

Inputs: set of tasks T
        user authorization information A
        set of constraints C
        set of tasklists  $\mathcal{TL} = \{TL_1, \dots, TL_m\}$ 
        set of workflow instances  $\mathcal{W} = \{W_1, \dots, W_m\}$ 
        delegation task  $t$ , delegator  $u$ , delegatee  $v$ 

01  for i = 1 to m
02      let  $E_i = TL_i \setminus W_i$ 
03      if  $((t, u) \in E_i)$  then
04          let  $E_i = (E_i \cup (t, v)) \setminus \{(t, u)\}$ 
05          let  $TL'_i = W_i \cup E_i$ 
06          if  $(\text{isValidSched}(TL'_i, C))$  then return false
07  let  $A' = (A \cup (t, v)) \setminus \{(t, u)\}$ 
08  return  $\text{isSchemaSatisfiable}(T, A', C)$ 

```

Figure 3: An algorithm for evaluating an abstract task cascading transfer in WDEM-S

subroutine `isValidSched()` (Figure 1(a)) (steps 02-06). The algorithm returns *false* if permitting the cascading transfer prevents the completion of at least one of the workflow instances (step 06). Finally, in steps 07-08, the algorithm updates the user authorization information and checks whether the workflow authorization schema remains satisfiable using the sub-routine `isSchemaSatisfiable()` (Figure 1(b)). Note that the time complexity of step 08 dominates steps 01-06. Hence, the overall time complexity of evaluating a cascading transfer delegation of t is $\mathcal{O}(|U|^{|T|})$.

5.2.2 WDEM-D

The semantics of a non-cascading transfer operation of an abstract task t in WDEM-D is the same as in WDEM-S. That is, a successful transfer delegation of t will not alter any tasklists and will only change user authorization information. However, permitting such an abstract task transfer delegation may prevent the completion of some workflow instances.

Consider, for example, the workflow in Figure 2(a) and assume a partially completed workflow instance $W = \{(t_1, b), (t_2, a), (t_3, d), (t_4, c)\}$. Note that the workflow instance W is a satisfiable instance of W since the WfMS can assign user b to perform task t_5 . Assume that user a initiates a request to perform a non-cascading transfer of abstract task t_4 to user d . In order to check whether the workflow authorization schema is satisfiable, we set $A' = (A \cup \{(t_4, d)\}) \setminus \{(t_4, a)\}$ and run the algorithm `isSchemaSatisfiable(T, A', C)` (Figure 1(b)).

We obtain the following ordering on the set of users from A' : $c \prec b$ and $c \prec d$. The updated workflow schema $W = (T, A', C)$ is satisfiable since we can assign all tasks to users such that no constraint is violated, for example $\{(t_1, d), (t_2, a), (t_3, c), (t_4, c), (t_5, b)\}$. However, should we permit the above transfer delegation request, the workflow instance W can not complete since user b is *no longer* senior to d . Hence, a non-cascading transfer delegation of an abstract task t is permitted only if *all existing tasklists continue to be satisfiable instances* and the *updated workflow authorization schema* $W = (T, A', C)$ *is satisfiable*.

Figure 4 illustrates an algorithm that determines whether to permit a non-cascading transfer of an abstract task t . The algorithm takes the workflow schema $W = (T, A, C)$, the set of tasklists and the delegation request parameters as inputs, and returns either *true* or *false*. Firstly, in step 01, the algorithm updates the user authorization information. Then,

the algorithm checks whether all tasklists are satisfiable instances of the updated workflow authorization schema using the sub-routine `isInstanceSatisfiable()` (Figure 1(c)) (steps 02-03). Should there exist at least one workflow instance that can not complete, the algorithm returns *false* and terminates prematurely (step 03). If all workflow instances can complete, then the algorithm checks whether the workflow authorization schema remains satisfiable (step 04). The algorithm returns *false* if permitting the transfer delegation renders the workflow schema unsatisfiable, and *true* otherwise. Note that the complexity of step 04 dominates steps 02-03. Hence, the overall time complexity of evaluating a non-cascading transfer delegation of an abstract task t is $\mathcal{O}(|U|^{|T|})$.

As before, a successful cascading transfer of t to v by u will have the effect of transferring u 's authorization to execute t and u 's existing execution assignments of t in all workflow instances to the delegatee v . Hence, we require that the updated tasklists are satisfiable instances and that the updated workflow authorization schema is satisfiable.

However, a cascading transfer of t may potentially prevent the completion of some workflow instances, whose tasklists are not updated. Consider, for example, the workflow in Figure 2(a). Let us assume that there are two tasklists $TL_1 = \{(t_1, b), (t_2, a), (t_3, d), (t_4, c)\}$ and $TL_2 = \{(t_1, b), (t_2, a), (t_3, c), (t_4, a)\}$. (Note that both TL_1 and TL_2 are satisfiable instances of W since the WfMS can assign user b to perform task t_5 in TL_1 and TL_2 .) Now suppose that user a initiates a request to perform a cascading transfer of abstract task t_4 to user d . Note that if this delegation were to succeed then TL_2 would change but TL_1 would not. (Specifically, $TL_2 = \{(t_1, b), (t_2, a), (t_3, c), (t_4, d)\}$.) Note also that TL_2 is still satisfiable (since t_5 can be performed by b) and that the schema is still satisfiable. However, TL_1 is no longer a satisfiable instance, because b is no longer senior to d (who performed t_3 in this instance). Hence, a cascading transfer delegation of an abstract task t is permitted only if *all tasklists remain satisfiable* (whether they are updated by the transfer or not) and the *updated workflow authorization schema* $W = (T, A', C)$ *is satisfiable*.

The algorithm for determining whether a cascading transfer delegation of an abstract task in WDEM-D is a simple combination of the algorithms given in Figures 3 and 4. Specifically, we need to update all those tasklists affected by the cascading effect of the delegation (as in lines 02-05 of Figure 3) and update the authorization information arising

```

Inputs: set of tasks T
        user authorization information A
        set of constraints C
        set of tasklists  $\mathcal{TL} = \{TL_1, \dots, TL_m\}$ 
        delegation task t, delegator u, delegatee v

01  let A' = (A  $\cup$  (t, v))  $\setminus$  {(t, u)}
02  for i = 1 to m
03      if (!isInstanceSatisfiable(T, A', C, TLi)) then return false
04  return isSchemaSatisfiable(T, A', C)

```

Figure 4: An algorithm for evaluating an abstract task non-cascading transfer in WDEM-D

from the abstract task delegation (as in line 01 in Figure 4). Then, we test for satisfiability of all tasklists and workflow schema satisfiability (as in lines 02–03 and line 04 in Figure 4, respectively). We do not present this algorithm in detail because of its obvious similarities to those algorithms already presented.

5.2.3 UDEM

In a UDEM, the lack of tasklists mean that a successful transfer of an abstract task t will only change the user authorization information. That is, following a successful transfer delegation of t to v by u , $A \leftarrow (A \cup \{(t, v)\}) \setminus \{(t, u)\}$. However, an abstract task transfer delegation may also prevent the completion of some workflow instances.

Consider, for example, a binding of duty constraint that states the same user must perform both tasks t and t' in a workflow instance. Assume that user u initiates a delegation request to transfer an abstract task t' to v . If the delegator u already successfully performed the task t , but not t' , in a partially completed workflow instance W , then permitting the above transfer delegation of t' by u will prevent the completion of W (at least until the delegator u regains authorization for t'). Hence, in a UDEM, we require that a transfer delegation of t is permitted only if we can guarantee both the completion of all partially completed instances of W and the satisfiability of the workflow schema W .

Note that we can use the algorithm illustrated in Figure 4 to determine whether to permit a transfer delegation of t . Essentially, instead of the set of tasklists, the algorithm takes the workflow schema, the set of workflow instances and delegation parameters as inputs, and returns either *true* or *false*. Now in step 03, instead of checking that all tasklists are satisfiable instances, the algorithm checks that all workflow instances are satisfiable instances. Hence, the overall time complexity of evaluating a transfer delegation of t is $\mathcal{O}(|U|^{|\mathcal{T}|})$.

5.3 Discussion

In this section, we have examined how delegation of concrete tasks and abstract tasks affect the satisfiability of a workflow in three different workflow execution models. Table 2 summarizes the checks required for evaluating various delegation requests in different workflow execution models.

The mechanism to evaluate a concrete task delegation request is different in WDEM-S and WDEM-D. In particular, in a WDEM-S, we check that the updated tasklist is a *valid execution schedule*. However, in a WDEM-D, we check that the updated tasklist is a *satisfiable instance*. In a UDEM, users initiate requests to perform tasks in a workflow instance. This means that concrete tasks are not assigned to

users in such systems. Hence, we do not consider delegation of concrete tasks in a UDEM.

The effect of a successful non-cascading transfer delegation of an abstract task t is to change the user authorization information, and is the same in all three workflow execution models. However, the mechanism to determine whether to permit such a transfer delegation is different. In particular, in a WDEM-S we only need to check the satisfiability of the updated workflow authorization schema, in a WDEM-D we need to check that all tasklists remain satisfiable instances and that the updated workflow authorization schema is satisfiable, and in a UDEM we need to check that all workflow instances are satisfiable instances and that the updated workflow authorization schema is satisfiable.

The effect of a successful cascading transfer delegation of an abstract task t is to change the user authorization information and the task-user assignments in relevant tasklists. In a WDEM-S, a cascading transfer of an abstract task t is permitted if we can guarantee that *all updated tasklists are valid execution schedules* and that the workflow authorization schema is satisfiable. In a WDEM-D, we need to guarantee that *all tasklists are satisfiable instances* and that the workflow authorization schema is satisfiable. The lack of tasklists, in a UDEM, means that cascading transfer delegations can not occur in such systems.

It is important to note that in a WDEM-D although the checks required to evaluate a ‘non-cascading’ transfer and a ‘cascading’ transfer are the same, the mechanism to evaluate such delegation operations are different. In particular, a non-cascading transfer delegation of t requires checking for the completion of all workflow instances without updating the tasklists. However, a cascading transfer delegation of t requires the update of relevant tasklists and then checking for the completion of all workflow instances.

6. ROLE DELEGATION

In a role-based workflow system, a user may delegate a role for which he is authorized. Specifically, a user u may delegate a role r if $(u, r') \in UA$ such that $r \leq r'$. We assume that a successful delegation of a role r to user v by u authorizes v for all tasks $\{t \in \mathcal{T} : (t, r'') \in TA, r'' \leq r\}$. As before, a user may perform a grant or transfer delegation.

Recall that a successful grant delegation operation does not affect the completion of workflow instances or the satisfiability of the workflow schema. Hence, as before, we only consider transfer delegation operations.

A successful non-cascading transfer of a role r to v by u authorizes the delegatee v for r , but the delegator u is no longer authorized for r . In other words, u is no longer authorized for tasks $t \in \mathcal{T}$ such that $(t, r') \in TA$ and $r' \leq r$. Consider,

Table 2: Checks required for evaluating delegation requests in WfMS

Execution model	Concrete task	Abstract task	
	Transfer	Non-cascading transfer	Cascading transfer
WDEM-S	Is the updated tasklist a valid execution schedule?	Is the workflow authorization schema satisfiable?	Are all updated tasklists valid execution schedules? Is the workflow authorization schema satisfiable?
WDEM-D	Is the updated tasklist a satisfiable instance?	Do all tasklists remain satisfiable instances? Is the workflow authorization schema satisfiable?	Do all tasklists remain satisfiable instances? Is the workflow authorization schema satisfiable?
UDEM	n/a	Can all workflow instances complete? Is the workflow authorization schema satisfiable?	n/a

for example, the task-role and user-role assignments in Figure 2(c) and Figure 2(d), respectively, and assume that user b wishes to perform a non-cascading transfer delegation of role r_2 to user d . Following the success of the above transfer operation $A \leftarrow (A \cup \{(t_3, d), (t_4, d)\}) \setminus \{(t_3, b), (t_4, b)\}$. However, a successful cascading transfer of a role r to v by u also transfers u 's concrete task assignments of $t \in T$ such that $(t, r') \in TA$ and $r' \leq r$ to v . In the above example, user b 's concrete task assignments of t_3 and t_4 , if any, are also transferred to delegatee d .

In summary, the effect of a successful transfer delegation of a role is to change the task-user authorizations and/or task-user execution assignments. Specifically, a successful non-cascading transfer of role r only changes the user authorization information A , whereas a successful cascading transfer of role r also changes appropriate task-user execution assignments in relevant tasklists.

6.1 Non-cascading transfer

The effect of a non-cascading transfer delegation of role r by u to v is to transfer u 's authorization for all tasks $\{t \in T : (t, r') \in TA, r' \leq r\}$ to the delegatee v , and is the same in all three workflow execution models. However, as described earlier in Section 5.2, the mechanism to evaluate a non-cascading transfer of an abstract task t is different in all three workflow execution models.

Recall that in a WDEM-S determining whether to permit a non-cascading transfer of an abstract task t requires us to check only that the updated workflow authorization schema is satisfiable. Hence, a non-cascading transfer delegation of role r by u to v is permitted if $\text{isSchemaSatisfiable}(T, A', C)$ (Figure 1(b)) returns *true*, where $A' = (A \cup \{(t, v) : (t, r') \in TA, r' \leq r\}) \setminus \{(t, u) : (t, r') \in TA, r' \leq r\}$.

Recall that in a WDEM-D determining whether to permit a non-cascading transfer of an abstract task t requires us to check that all tasklists remain satisfiable instances and that the updated workflow authorization schema is satisfiable. Hence, we set $A' = (A \cup \{(t, v) : (t, r') \in TA, r' \leq r\}) \setminus \{(t, u) : (t, r') \in TA, r' \leq r\}$ and check that all tasklists remain satisfiable instances and that the updated workflow authorization schema is satisfiable.

Recall that in a UDEM transferring an abstract task may also prevent the completion of some workflow instances. Hence, in order to determine whether to permit a transfer of a role r in a UDEM, we update the user authorization information, and check that all workflow instances can complete and that the updated workflow authorization schema

is satisfiable. Specifically, we set $A' = (A \cup \{(t, v) : (t, r') \in TA, r' \leq r\}) \setminus \{(t, u) : (t, r') \in TA, r' \leq r\}$ and check for the completion of workflow instances using A' and satisfiability of the workflow authorization schema $W = (T, A', C)$.

Note that, in all three workflow execution models, determining whether to permit a non-cascading transfer of a role r , generally, requires us to check the satisfiability of the updated workflow authorization schema. Hence, the overall time complexity of evaluating a cascading transfer of a role r in all three workflow execution models is $\mathcal{O}(|U|^{|T|})$.

6.2 Cascading transfer

The effect of a cascading transfer delegation of role r by u to v is to transfer both u 's concrete task assignments and abstract task authorizations of all tasks $\{t \in T : (t, r') \in TA, r' \leq r\}$ to the delegatee v . Hence, a cascading transfer of a role r is permitted only if we can guarantee both the completion of all workflow instances and the satisfiability of the updated workflow authorization schema. (As before, the lack of tasklists in a UDEM means that we do not need to consider cascading transfer delegation in such systems.)

In a WDEM-S, a cascading transfer of a role r is permitted only if *all updated tasklists are valid execution schedules* and the updated workflow authorization schema is satisfiable. However, in a WDEM-D, a cascading transfer of a role r is permitted only if *all tasklists are satisfiable instances* and the updated workflow authorization schema is satisfiable.

As before, in both WDEM-S and WDEM-D, deciding whether to permit a cascading transfer of a role r , generally, requires us to check the satisfiability of the updated workflow authorization schema. Hence, the overall time complexity of evaluating a cascading transfer of a role r , in both WDEM-S and WDEM-D, is $\mathcal{O}(|U|^{|T|})$.

7. CONCLUSION

We considered the satisfiability of workflows while delegating concrete tasks, abstract tasks and roles in the context of three different workflow execution models. We presented algorithms for authorizing various delegation requests in each workflow execution model. In particular, our algorithms guarantee that permitting a delegation request does not prevent the completion of workflow instances and does not render the workflow authorization schema unsatisfiable. This is the first work in the literature to consider satisfiability of workflows while supporting user delegation operations.

Note that considering satisfiability when deciding delegation requests generally requires us to run isSchemaSatis-

`fiable()`. The only exception being a concrete task delegation request in WDEM-S, which requires us to only run `isValidSched()`. However, if we only allow the specification of separation of duty and binding of duty constraints, then determining whether to permit a transfer delegation request is fixed parameter tractable.

We hope to extend the work presented in this paper in several directions. We intend to introduce a more fine-grained treatment of tasks. In particular, we would like to include the notion of state for tasks: typical states might include “initialized”, “assigned” and “complete”, for example. In this way we can consider the delegation of concrete tasks in UDEM and also consider more complex workflow patterns.

We previously discussed the notion of “weak transfer” delegation of roles in role-based workflows [6]. Informally, following a weak transfer delegation of a role r , the delegator u retains the authorization of role $x \in R$, such that $x \leq r$, if there is an alternative path from x to a role to which u is assigned. We will investigate the affect of permitting such delegation operations on the satisfiability of the workflow.

Revocation is a mechanism that is used to reverse delegation operations. The effect of a successful revocation operation is to change the user authorization information, thereby affecting the satisfiability of the workflow. We will also examine whether the algorithms presented in this paper can be used, and if necessary develop new algorithms, for deciding revocation requests in different workflow execution models.

Acknowledgements.

The authors would like to thank the anonymous referees for their comments, which have helped us to improve the paper considerably.

8. REFERENCES

- [1] ATLURI, V., BERTINO, E., FERRARI, E., AND MAZZOLENI, P. Supporting delegation in secure workflow management systems. In *Proceedings of 17th Annual IFIP WG 11.3 Working Conference on Data and Applications Security* (2003), pp. 190–202.
- [2] ATLURI, V., AND WAINER, J. Supporting conditional delegation in secure workflow management systems. In *Proceedings of 10th ACM Symposium on Access Control Models and Technologies* (2005), pp. 49–58.
- [3] CRAMPTON, J. A reference monitor for workflow systems with constrained task execution. In *Proceedings of 10th ACM Symposium on Access Control Models and Technologies* (2005), pp. 38–47.
- [4] CRAMPTON, J. Personal communication to Ninghui Li, November 2006.
- [5] CRAMPTON, J., AND KHAMBHAMMETTU, H. Delegation in role-based access control. *International Journal of Information Security* 7, 2 (2008), 123–136.
- [6] CRAMPTON, J., AND KHAMBHAMMETTU, H. On delegation and workflow execution models. In *Proceedings of 23rd ACM Symposium on Applied Computing* (2008), pp. 2137–2144.
- [7] DOWNEY, R., AND FELLOWS, M. *Parameterized Complexity*. Springer, 1999.
- [8] KANDALA, S., AND SANDHU, R. Secure role-based workflow models. *Database Security XV: Status and Prospects* (2002), 45–58.
- [9] VENTER, K., AND OLIVIER, M. The delegation authorization model: A model for the dynamic delegation of authorization rights in a secure workflow management system. In *Proceedings of Information Security South Africa* (2002). Published electronically. Available at <http://icsa.cs.up.ac.za/issa/2002/proceedings/A021.pdf>.
- [10] WAINER, J., KUMAR, A., AND BARTHELMESS, P. DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Information Systems* 32, 3 (2007), 365–384.
- [11] WANG, Q., AND LI, N. Satisfiability and resiliency in workflow systems. In *Proceedings of 12th European Symposium On Research In Computer Security* (2007), vol. 1146 of *LNCS*, pp. 90–115.