A Reference Monitor for Workflow Systems with Constrained Task Execution

Jason Crampton
Information Security Group, Royal Holloway, University of London

ABSTRACT

We describe a model, independent of any underlying access control paradigm, for specifying authorization constraints such as separation of duty and cardinality constraints in workflow systems. We present a number of results enabling us to simplify the set of authorization constraints. These results form the theoretical foundation for an algorithm that can be used to determine whether a given constrained workflow can be satisfied: that is, does there exist an assignment of authorized users to workflow tasks that satisfies the authorization constraints? We show that this algorithm can be incorporated into a workflow reference monitor that guarantees that every workflow instance can complete. We derive the computational complexity of our algorithm and compare its performance to comparable work in the literature.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Security, integrity and protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Theory

Keywords

workflow system, reference monitor, entailment constraint, authorization constraint

1. INTRODUCTION

A workflow is a representation of an organizational or business process and is typically specified as a set of tasks and a set of dependencies between the tasks. The dependencies fall into two broad categories: those determined by the application logic of the workflow such as the order of execution of the tasks [9], and those determined by security requirements. In this paper we concentrate on the second of these categories, which includes authorization constraints, such as separation of duty requirements, where two different users must execute two different tasks, and binding of duty constraints, in which the same user is required to perform two different tasks. Cardinality constraints, which require that certain tasks are performed a certain number of times, have also been considered in the context of authorization [2]. There are at least four questions that are of interest when considering workflow systems with authorization constraints: How do we specify constraints? How do we ensure that a workflow specification with constraints is satisfiable? How do we design a reference monitor that ensures the constraints are satisfied? How do we ensure that granting a request to a particular user does not prevent a workflow instance from completing? There exist several specification schemes in the literature for authorization constraints [1, 2, 3, 4, 8, 11] in computerized workflow systems, but less work has been done on satisfiability [2, 10].

Arguably the most sophisticated approach to these questions is due to Bertino, Ferrari and Atluri [2], henceforth referred to as the BFA model, which describes the specification and enforcement of authorization constraints in workflow management systems. The BFA model defines a *workflow role specification* to be a list of *task role specifications*. A task role specification identifies a task, specifies the roles authorized to perform the task, and the maximum number of activations of the task that are permitted in an instance of the workflow.

Let us consider a simple workflow forming part of a purchase ordering and financial system. There are six tasks involved in ordering and paying for goods:

- the creation of a purchase order requesting goods from a supplier (`crtPO`);
- the approval of the purchase order prior to despatch to the supplier (`apprPO`);
- the acknowledgement of delivery of the goods by signing a goods received note (`signGRN`);
- the acknowledgement of delivery by countersigning the goods received note (`ctrsignGRN`);
- the creation of a payment file on receipt of the supplier's invoice for the goods (`crtPay`);
- the approval of the payment to the supplier (subject to receipt of goods) (`apprPay`).

SACMAT'05, June 1–3, 2005, Stockholm, Sweden.

By defining a workflow role specification to be a list, the BFA model imposes a unique order on the execution of the tasks in the workflow. We believe this is a limitation of the BFA model. In the example above, for example, it is not necessary for the `crtPay` task to follow the `signGRN` task. Indeed, in many cases, the goods and invoice are dispatched separately by the supplier; the finance department processes the invoice, while the recipient of the goods handles the goods received note (GRN). Nevertheless, the payment should not be approved until the goods have been signed for. Hence we obtain a workflow in which the order of execution of certain tasks is not fixed, as depicted in Figure 1(a). An edge (t, t') in the diagram indicates that task t should be performed before t' in each instance of the workflow. This ordering on the execution of tasks defines what are sometimes called *control-flow* dependencies [9]. In the diagram, `crtPO` must precede `apprPO`, for example, but the order of execution of `signGRN` and `crtPay`, for example, is not pre-determined.

We also note that neither the BFA model, nor any other workflow authorization model to our knowledge, models the conditional execution of workflow tasks. In the purchase order system, for example, we may only require that the purchase order is approved if the order value exceeds $10000, or that a GRN only requires a countersignature if the order value exceeds $10000.[1] Finally, we observe that the BFA model and other similar workflow authorization models [1, 3, 4, 8, 11] confine their attention to separation of duty constraints, binding of duty constraints and constraints on the number of executions of a task. However, we might, for example, additionally require that a purchase order is approved by someone more senior than and belonging to the same department as the user who created the purchase order. To our knowledge, existing constraint specification schemes do not include such constraints.

Most authorization schemes for workflows are based on a particular computational model: examples include logic programs [2, 11], active databases [4] and petri nets [1]. We introduced a simple specification scheme for authorization constraints that is independent of an underlying computational model and showed that it could be used to articulate *inter alia* separation of duty constraints and binding of duty constraints [6]. In this model, the execution of tasks in a workflow instance may complete in a variety of different ways (unlike the BFA model), providing the partial ordering on the tasks is respected. However, this model did not include cardinality constraints. In this paper we present some new results that provide the theoretical basis for a reference monitor with considerably lower complexity than other proposals in the literature. By treating a workflow as an acyclic, labelled directed graph, we extend our model to include cardinality constraints on the execution of tasks and conditional execution of tasks.

In the next section we review our original model for specifying authorization constraints in workflow systems [6]. In Section 3 we describe an algorithm that can establish whether a constrained workflow specification is satisfiable. We show that the algorithm can also be used as the basis for a reference monitor in workflow management systems. In Section 4 we extend our model to include cardinality constraints and conditional execution of tasks, and present a modified algorithm for this extended model. In the penultimate section we compare our approach to the BFA model and explain why we do not include role-based constraints in our model. Finally, we summarize our contribution and discuss future work.

[1]There exist workflow models that consider such behaviour, often called *value dependencies* [9], but such models do not consider authorization constraints.

2. A SIMPLE MODEL FOR CONSTRAINED WORKFLOW SYSTEMS

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 $U \times U$. Given $\rho \in Rel(U)$, let $\widetilde{\rho} = \{(v, u) : (u, v) \in \rho\}$. Note that if ρ is symmetric, then $\rho = \widetilde{\rho}$. If $\langle X, \leqslant \rangle$ is a partially ordered set, then we write $x \parallel y$ if $x \not\leqslant y$ and $y \not\leqslant x$. We may write $x \geqslant y$ whenever $y \leqslant x$.

Definition 1. *A* workflow specification *is a partially ordered set of tasks* T. *A* workflow authorization schema *is a pair* (T, A), *where* $A \subseteq \mathsf{T} \times U$; u *is* authorized *to* perform *(or* execute*)* $\mathtt{t}$ *iff* $(\mathtt{t}, u) \in A$.

If $\mathtt{t} < \mathtt{t}'$ then $\mathtt{t}$ must be performed before $\mathtt{t}'$ in any instance of the workflow. Generally, task-user pairs will not be explicitly defined. Instead, A will be inferred from other access control data structures, such as user-role assignment and task-role assignment relations [7], or access control lists. The second approach is analogous to associating a list of users with transformation procedures (tasks) in the Clark-Wilson framework [5]. We believe that separating the constraint model from the access control mechanism lends our model greater generality and potentially wider applicability. In Section 3 we give an example in which role-based access control data structures are used to generate A.

Definition 2. *An* entailment constraint *has the form* $(D, (\mathtt{t}, \mathtt{t}'), \rho)$, *where* $D \subseteq U$, $\rho \in Rel(U)$ *and* $t \not\geqslant t'$. D *is the* domain *of the constraint;* $\mathtt{t}$ *is the* antecedent *task; and* $\mathtt{t}'$ *is the* consequent *task. If users* u *and* u' *perform* $\mathtt{t}$ *and* $\mathtt{t}'$, *respectively, and* $u \in D$, *then constraint* $(D, (\mathtt{t}, \mathtt{t}'), \rho)$ *is* satisfied *iff* $(u, u') \in \rho$.

Intuitively, an entailment constraint places some restriction on the users who can perform $\mathtt{t}'$ given that $u \in D$ has performed $\mathtt{t}$. Hence a separation of duty constraint that prevents any user from performing both $\mathtt{t}$ and $\mathtt{t}'$ can be expressed as $(U, (\mathtt{t}, \mathtt{t}'), \neq)$, and a binding of duty constraint that requires $\mathtt{t}$ and $\mathtt{t}'$ be performed by the same user can be expressed as $(U, (\mathtt{t}, \mathtt{t}'), =)$. We specify a domain in order to handle "weak" separation of duty constraints of the form "if user `bob` performed task $\mathtt{t}_1$, then `bob` is not permitted to perform task $\mathtt{t}_2$" [2]. Such a constraint can be expressed as $(\{\mathtt{bob}\}, (\mathtt{t}_1, \mathtt{t}_2), \neq)$. D can also have the form $U(r)$, which is interpreted as the set of users assigned to role r.

In fact, any binary relation between users can be used (including those that can be derived from contextual information). Hence it is possible to articulate constraints of the form "tasks $\mathtt{t}$ and $\mathtt{t}'$ must be performed by two different users in the same department". If we assume the existence of group-based or role-based authorization structures, then it is possible to induce an ordering (binary relation) on the set of users determined by the relative seniority of the roles to which each user is assigned. We anticipate that this sort of relation will prove particularly important, because it is

natural to implement access control in workflow systems using role-based techniques [7].

The relation $\preccurlyeq$ will be used to denote an order relation 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, $\preccurlyeq$ can be defined in terms of A in the following way: $u \preccurlyeq u'$ iff $\{\mathtt{t} : (\mathtt{t}, u) \in A\} \subseteq \{\mathtt{t} : (\mathtt{t}, u') \in A\}$. Notice that $\preccurlyeq$ is a pre-order (not a partial order, since we may have two users $u \neq u'$ with $u \preccurlyeq u'$ and $u \succcurlyeq u'$). We define $u \simeq u'$ iff $u \preccurlyeq u'$ and $u \succcurlyeq u'$; we say u' is more senior than u and write $u \prec u'$ if $u \preccurlyeq u'$ and $u \not\simeq u'$. A constraint of the form $(U, (\mathtt{t}, \mathtt{t}'), \prec)$ requires that $\mathtt{t}'$ is performed by a user more senior than the one who performed $\mathtt{t}$. Figure 1(b) gives some examples of entailment constraints that could be defined for the purchase order workflow. Note that a logical consequence of c_2 and c_3 is that the same person cannot execute the `signGRN` and `crtPay` tasks.

We can also express more complicated security requirements within our scheme. Consider the following security requirement in our purchase order system (assuming the use of a role-based access control mechanism): "a purchase order must be approved by a user who is more senior than the user that created the purchase order, except when the purchase order is raised by someone who is assigned to the `poOfficer` (Purchasing Officer) role, in which case someone assigned to the `finOfficer` (Financial Officer) role must approve the order". Let UA be the user-role assignment relation. Then we can express this requirement as the entailment constraint $(U, (\mathtt{crtPO}, \mathtt{apprPO}), \rho)$, where

$$\rho = \{(u, v) : u \prec v\} \cup \{(u, v) : (u, \mathtt{poOfficer}), (v, \mathtt{finOfficer}) \in UA)\}.$$

Definition 3. *A* constrained workflow authorization schema *is a triple* (T, A, C)*, where* C *is a set of entailment constraints. A constrained workflow authorization schema* (T, A, C) *is* well-formed *if* $((\mathtt{t}_i, \mathtt{t}_j), \rho) \in C$ *and* $\mathtt{t}_i \parallel \mathtt{t}_j$ *implies* $((\mathtt{t}_j, \mathtt{t}_i), \widetilde{\rho}) \in C$.

Note that if $\mathtt{t}_i \parallel \mathtt{t}_j$, then the order of execution of $\mathtt{t}$ and $\mathtt{t}'$ is not fixed. Hence we would expect that for a constraint of the form $(D, (\mathtt{t}_i, \mathtt{t}_j), \rho)$, the relation ρ will be symmetric, in which case $\rho = \widetilde{\rho}$. For example, ρ could be $\neq$ (to specify separation of duty) or $=$ (to specify binding of duty). However, we would not expect ρ to be $\prec$, for example, as the asymmetry of the relation $\prec$ implies some ordering on $\mathtt{t}_i$ and $\mathtt{t}_j$. (Of course, the definition of a well-formed workflow authorization schema does not prohibit the use of $\prec$ when $\mathtt{t}_i \parallel \mathtt{t}_j$, but if we were to use it, we must also define the constraint $(D, (\mathtt{t}_j, \mathtt{t}_i), \succ)$.)

2.1 Linear extensions and execution assignments

Let $\langle X, \leqslant \rangle$ be a partially ordered set. A *linear extension* of X is a total ordering of the elements of X that respects the ordering of the elements in X. In other words, a linear extension of $\langle X, \leqslant \rangle$ can be interpreted as a list $[x_1, \ldots, x_n]$, where $X = \{x_1, \ldots, x_n\}$ and $x_i \leqslant x_j$ in X implies that x_i precedes x_j in the list. We denote the set of linear extensions of X by $\mathcal{L}(X)$.

Linear extensions are important in the context of workflows because they "linearize" a partially ordered set of tasks.[2] In other words, a linear extension of T represents a possible sequence of execution of the tasks in a workflow. The linear extensions of the workflow in Figure 1(a) are

$[\mathtt{crtPO}, \mathtt{apprPO}, \mathtt{crtPay}, \mathtt{signGRN}, \mathtt{ctrsignGRN}, \mathtt{apprPay}]$,

$[\mathtt{crtPO}, \mathtt{apprPO}, \mathtt{signGRN}, \mathtt{crtPay}, \mathtt{ctrsignGRN}, \mathtt{apprPay}]$,

$[\mathtt{crtPO}, \mathtt{apprPO}, \mathtt{signGRN}, \mathtt{ctrsignGRN}, \mathtt{crtPay}, \mathtt{apprPay}]$.

Definition 4. *Let* $\mathsf{W} = (\mathsf{T}, A, C)$ *be a constrained workflow authorization schema. An* execution assignment *for* W *is a pair* (L, α)*, where* $L \in \mathcal{L}(\mathsf{T})$ *and* $\alpha : \mathsf{T} \to U$ *is an assignment of tasks to users. We say* (L, α) *is a* valid execution assignment *iff for all* $\mathtt{t} \in \mathsf{T}$*,* $(\mathtt{t}, \alpha(\mathtt{t})) \in A$ *and for all* $(D, (\mathtt{t}, \mathtt{t}'), \rho) \in C$*, if* $u \in D$ *then* $(\alpha(\mathtt{t}), \alpha(\mathtt{t}')) \in \rho$.

Generally we will prefer to write L as a list of tasks $[\mathtt{t}_1, \ldots, \mathtt{t}_n]$ and an execution assignment as a list of task-user pairs $[(\mathtt{t}_1, u_1), \ldots, (\mathtt{t}_n, u_n)]$ with the understanding that task $\mathtt{t}_i$ is performed by u_i. In other words, $[(\mathtt{t}_1, u_1), \ldots, (\mathtt{t}_n, u_n)]$ is a valid execution assignment if $(\mathtt{t}_i, u_i) \in A$, and $(u_i, u_j) \in \rho$ whenever $((\mathtt{t}_i, \mathtt{t}_j), \rho) \in C$ and $u \in D$.

2.2 Manipulating entailment constraints

The following three results are all derived using basic set operations and first appeared in our earlier paper [6].

Proposition 5. *Let* $\mathsf{W} = (\mathsf{T}, A, \{(D_1, (\mathtt{t}, \mathtt{t}'), \rho), (D_2, (\mathtt{t}, \mathtt{t}'), \rho)\})$ *be a constrained workflow authorization schema. Then* (L, α) *is a valid execution assignment for* W *iff* (L, α) *is a valid execution assignment for* $(\mathsf{T}, A, \{D_1 \cup D_2, (\mathtt{t}, \mathtt{t}'), \rho\})$.

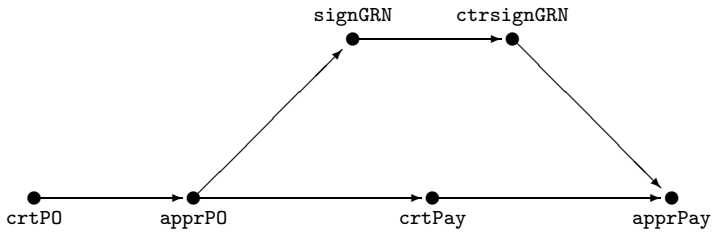
Proposition 6. *Let* $\mathsf{W} = (\mathsf{T}, A, \{(D, (\mathtt{t}, \mathtt{t}'), \rho)\})$ *be a constrained workflow authorization schema and define* $\sigma = (U \setminus D) \times U$*. Then* (L, α) *is a valid execution assignment for* W *iff* (L, α) *is a valid execution assignment for* $(\mathsf{T}, A, \{(U, (\mathtt{t}, \mathtt{t}'), \rho \cup \sigma)\})$.

Proposition 7. *Let* $\mathsf{W} = (\mathsf{T}, A, \{((\mathtt{t}, \mathtt{t}'), \rho_1), ((\mathtt{t}, \mathtt{t}'), \rho_2)\})$ *be a constrained workflow authorization schema. Then* (L, α) *is a valid execution assignment for* W *iff* (L, α) *is a valid execution assignment for* $(\mathsf{T}, A, \{((\mathtt{t}, \mathtt{t}'), \rho_1 \cap \rho_2)\})$.

In other words, we can assume that the domain of every constraint is U (by Proposition 6), and that for each pair of tasks $(\mathtt{t}, \mathtt{t}')$ there is a single constraint (by Proposition 7). Henceforth, therefore, we will write entailment constraints in the form $((\mathtt{t}, \mathtt{t}'), \rho)$.

At the moment we only consider entailment constraints to be specifications of security policy requirements such as separation of duty. However, we can view the authorization information as a set of entailment constraints on the execution of tasks. In particular, let $\mathsf{T} = \{\mathtt{t}_1, \ldots, \mathtt{t}_n\}$ and define $a_{ij} = U(\mathtt{t}_i) \times U(\mathtt{t}_j)$, where $U(\mathtt{t}) = \{u \in U : (\mathtt{t}, u) \in A\}$. Then the entailment constraint $((\mathtt{t}_i, \mathtt{t}_j), a_{ij})$, is only satisfied if two appropriately authorized users perform tasks $\mathtt{t}_i$ and $\mathtt{t}_j$. If there exists an entailment constraint of the form $((\mathtt{t}_i, \mathtt{t}_j), \rho_{ij})$ then we form the new constraint $((\mathtt{t}_i, \mathtt{t}_j), \rho_{ij} \cap a_{ij})$. More formally, we have the following result. The proof of this result follows immediately from the definition of a valid execution assignment and is omitted.

[2]We note that in certain circumstances, it will be possible for certain tasks in a workflow to execute in parallel. Specifically, if $\mathtt{t}$ and $\mathtt{t}'$ are tasks with $\mathtt{t} \parallel \mathtt{t}'$ and neither $\mathtt{t}$ nor $\mathtt{t}'$ appears in any constraint, then they may be executed in parallel. Such situations are outside the scope of this paper.



(a) A purchase order workflow specification

Constraint	Informal semantics
$c_1 = (U, (\texttt{crtPO}, \texttt{apprPO}), \prec)$	The user that approves a purchase order must be more senior than the user that creates it
$c_2 = (U, (\texttt{crtPO}, \texttt{signGRN}), =)$	The user that creates a purchase order must sign for the goods
$c_3 = (U, (\texttt{signGRN}, \texttt{ctrsignGRN}), \neq)$	The user that countersigns the GRN must be different from the user that signed the GRN
$c_4 = (U, (\texttt{crtPO}, \texttt{crtPay}), \neq)$	The user that creates the purchase order cannot create the payment for those goods
$c_5 = (U, (\texttt{crtPay}, \texttt{apprPay}), \prec)$	The user that approves the payment must be more senior than the user that creates the payment

(b) Entailment constraints

Figure 1: A constrained workflow system for purchase order processing

PROPOSITION 8. *Let* $\mathsf{W} = (\mathsf{T}, A, \{((\mathsf{t}, \mathsf{t}'), \rho)\})$ *be a constrained workflow authorization schema. Then* (L, α) *is a valid execution assignment for* W *iff* (L, α) *is a valid execution assignment for* $(\mathsf{T}, \mathsf{T} \times U, \{((\mathsf{t}, \mathsf{t}'), \rho \cap a)\})$*, where* $a = \{(u, u') : (\mathsf{t}, u), (\mathsf{t}', u') \in A\}$.

In other words, we can express all the information required to make an authorization decision in terms of entailment constraints. Hence it is sufficient from a theoretical point of view to consider workflow schemata of the form (T, C), although, from a practical perspective, it is clearly more natural to include authorization information. In fact, this result was the inspiration for the algorithm described in the next section that forms the basis for our reference monitor.

3. CONSTRUCTING A REFERENCE MONITOR

Determining whether a request to execute a workflow task should proceed is typically made by considering the identity of the requester and the permissions associated with that person. However, in the case of constrained workflows, it is important that granting the request should not violate any constraints. Furthermore, we wish to guarantee that granting the request does not prevent some other subsequent task from completing because certain constraints have been rendered unsatisfiable. As a trivial example, consider a workflow with only two tasks t_1 and t_2, in which t_2 can only be performed by a particular user u, and we have the requirement that the same user must not execute both tasks. Then any request by u to execute t_1 should be denied (even if she is authorized to execute the task) because no authorized user can subsequently perform task t_2. The BFA model includes a method for determining whether requests should be permitted. Unfortunately, the procedure is exponential in the number of tasks. We now show how to create a reference monitor which runs in time polynomial in the number of tasks and users.

In general, given a request by u to execute task t in a partially completed instance of a workflow, there are three questions a reference monitor could consider:

- Is u authorized to perform t?
- Are all constraints in which t is the consequent task satisfied for this instance if u performs t?
- Can the workflow complete if u performs t?

The reference monitor must certainly guarantee that the answers to the first two questions are yes. It is up to the designers of the reference monitor to decide whether the third question should always have an affirmative answer. Indeed, researchers have considered overriding (that is, not enforcing) constraints in the event that a workflow cannot complete because of previous task executions and the existence of constraints [11]. We say a reference monitor is *enforcement compliant* if it guarantees (for all requests) that the answers to the first two questions are yes and *completion compliant* if it guarantees that the answer to each of the three questions is yes. We say a constrained workflow authorization schema is *satisfiable* if there exists a valid execution assignment (and *unsatisfiable* otherwise).

LEMMA 9. *Let* $\mathsf{W} = (\mathsf{T}, A, C)$ *be a well-formed constrained workflow authorization schema, and let* $[(\mathsf{t}_1, u_1), \ldots, (\mathsf{t}_n, u_n)]$ *be a valid execution assignment for* W*. Then for any permutation* π *of the integers* $\{1, \ldots, n\}$ *such that* $[\mathsf{t}_{\pi(1)}, \ldots, \mathsf{t}_{\pi(n)}]$ *is a linear extension*

of T, $[(\mathtt{t}_{\pi(1)}, u_{\pi(1)}), \ldots, (\mathtt{t}_{\pi(n)}, u_{\pi(n)})]$ *is a valid execution assignment for* W.

Proof. Consider $u_{\pi(i)}$ and $u_{\pi(j)}$ and assume without loss of generality that $\pi(i) < \pi(j)$. There are two cases:

- $i < j$

  In this case the ordering of the tasks $\mathtt{t}_i$ and $\mathtt{t}_j$ within L are preserved in L'. Hence $(u_{\pi(i)}, u_{\pi(j)}) \in \rho_{\pi(i)\pi(j)}$ since $(u_i, u_j) \in \rho_{ij}$.

- $i > j$

  In this case the ordering of the tasks $\mathtt{t}_i$ and $\mathtt{t}_j$ within L is not preserved in L'. Since both L and L' are linear extensions, we have $\mathtt{t}_i \parallel \mathtt{t}_j$. Since $\mathtt{t}_j$ precedes $\mathtt{t}_i$ in L, we have $(u_j, u_i) \in \rho_{ji}$. Moreover, W is well-formed; by definition, $(u_i, u_j) \in \rho_{ij}$. Hence $(u_{\pi(i)}, u_{\pi(j)}) \in \rho_{\pi(i)\pi(j)}$. ■

3.1 Establishing workflow satisfiability

A constrained workflow authorization schema is satisfiable if there exists a valid execution assignment. The result of Lemma 9 means that we only need to consider a single linear extension to check whether a workflow specification is satisfiable. This enables us to improve on the algorithm presented in [6]. In particular, given a constrained workflow authorization schema $\mathsf{W} = (\mathsf{T}, A, C)$, we specify a single linear extension $[\mathtt{t}_1, \ldots, \mathtt{t}_n]$, called the *canonical* linear extension of T.

Figure 2 illustrates an algorithm (written in pseudo-code) that determines whether a workflow specification has a valid execution assignment. The algorithm computes $V(i, j)$ for each pair $(\mathtt{t}_i, \mathtt{t}_j)$, where $V(i, j)$ is the set of users that can execute $\mathtt{t}_i$ and $\mathtt{t}_j$ (in that order) given the authorization information and the entailment constraints in the schema that apply to $\mathtt{t}_i$ and $\mathtt{t}_j$. It assumes the existence of a canonical linear extension in which the tasks are indexed by an integer between 1 and $n = |\mathsf{T}|$. The basic strategy is to initialize each $V(\mathtt{t})$ to the set of users that are authorized to perform $\mathtt{t}$ (line 02) and to apply all possible constraints defined for each pair of tasks, including those derived from authorization information (lines 06–07). Essentially, the algorithm is applying Proposition 8 and computing a new relation for each entailment constraint. If one of these relations is empty, then the algorithm terminates prematurely (line 07), since there does not exist a pair of authorized users that comply with the entailment constraints, and by Lemma 9 there cannot exist a valid execution assignment for the workflow. Finally, for each task $\mathtt{t}$ we (re-)compute the set of users that can perform $\mathtt{t}$ (lines 09–10).

The overall time complexity of the algorithm is $\mathcal{O}(|T|^2|U|^4)$, since the number of constraints is $\mathcal{O}(|T|^2)$ and the comparison in line 06 is $\mathcal{O}(|U|^4)$ in the worst case. Note that the computational complexity of the comparison in line 06 dominates the complexity of the computations required in lines 09 and 10, which are $\mathcal{O}(|U|^2)$. Note also that if `rel` is $\neq$ or $=$, then the computation in line 06 is a simple comparison of $V(i)$ and $V(j)$ and hence has time complexity $\mathcal{O}(|U|^2)$. In other words, if we restrict our attention to separation of duty and binding of duty constraints, then the overall complexity reduces to $\mathcal{O}(|\mathsf{T}|^2|U|^2)$.

3.2 A completion compliant reference monitor

We will write W to denote an instance of the workflow $\mathsf{W} = (\mathsf{T}, A, C)$ and t to denote an instance of task $\mathtt{t} \in \mathsf{T}$. An instance W is defined by a list of task-user pairs $[(t_1, u_1), \ldots, (t_k, u_k)]$, where $[\mathtt{t}_1, \ldots, \mathtt{t}_k]$ is a prefix of some linear extension of T and denotes the tasks that have been completed in W.

We now make the crucial observation that a partially completed instance of a workflow can be regarded as a workflow specification in which completed tasks are assigned to a single user, namely the user that executed the task. Hence the algorithm in Figure 2 can be incorporated into a completion compliant reference monitor to test whether allowing a request to complete would render the workflow instance unsatisfiable. For example, given a request to execute the first task $\mathtt{t}_1$ by user u, we set $V(1) = \{u\}$ and run the algorithm to determine whether there exists a valid execution schedule for the remaining tasks.

In general, given an instance of the workflow $W = [(t_1, u_1), \ldots, (t_k, u_k)]$ and a request from user u to execute task t, we run the algorithm on the workflow (T, A', C), where $A' = \{(\mathtt{t}, u), (\mathtt{t}_1, u_1), \ldots, (\mathtt{t}_k, u_k)\} \cup \{(\mathtt{t}', u) \in A : \mathtt{t}' \notin \{\mathtt{t}, \mathtt{t}_1, \ldots, \mathtt{t}_k\}\}$.

Consider the workflow W shown in Figure 3(a). A solid edge $(\mathtt{t}, \mathtt{t}')$ indicates that $\mathtt{t} < \mathtt{t}'$; the broken edge indicates the existence of a constraint between $\mathtt{t}_2$ and $\mathtt{t}_3$. Labelled edges indicate a constraint exists on the execution of the two tasks; $\mathtt{t}_5$ must be performed by a user who is more senior than the user that performed $\mathtt{t}_3$, for example. In this example we assume the use of role-based access control and assign users and tasks to roles. In other words, the set of authorization information A is not explicitly stored and has to be derived from other access control data structures. In order to compute $V(\mathtt{t}_i)$, the set of users authorized to execute $\mathtt{t}$, we execute a simple select query on the join of the task-role assignment and user-role assignment relations. Recall that we define $u \preccurlyeq u'$ if $\{\mathtt{t} : (\mathtt{t}, u) \in A\} \subseteq \{\mathtt{t} : (\mathtt{t}, u') \in A\}$. Hence we can deduce that $d \prec b \prec a$ and $c \prec b \prec a$ from the user-role assignment relation and the role hierarchy, which can be expressed as the set $\{(d, b), (d, a), (c, b), (c, a), (b, a)\}$.

We now illustrate how the algorithm in Figure 2 can be used in a completion compliant reference monitor. Suppose that a attempts to execute $\mathtt{t}_1$. We set $V(1) = \{a\}$ and initialize $V(i)$, $2 \leqslant i \leqslant 5$. Now $V(1, 2) = \emptyset$, since $V(2) = \{a\}$, $V(1) \times V(2) = \{(a, a)\}$ and $((\mathtt{t}_1, \mathtt{t}_2), \neq) \in C$. Hence the workflow cannot complete if a performs $\mathtt{t}_1$. (This is obvious from a cursory examination of the workflow because a is the only user authorized to perform $\mathtt{t}_2$.)

Now suppose that d has performed $\mathtt{t}_1$ and consider an attempt by b to execute $\mathtt{t}_3$. Note that any one of $\mathtt{t}_2$, $\mathtt{t}_3$ and $\mathtt{t}_4$ could be performed after $\mathtt{t}_1$. In this case, $V(1) = \{d\}$, since the task has already been performed, and $V(3)$ is set to $\{b\}$. Then running the algorithm we obtain

$$
\begin{aligned}
&V(1, 2) = \{(d, a)\},\\
&V(1, 3) = \{(d, b)\},\\
&V(1, 4) = \{(d, a), (d, b)\},\\
&V(2, 3) = \{(a, b)\},\\
&V(2, 5) = \{(a, b), (a, c), (a, d)\},\\
&V(3, 5) = \emptyset \text{ (algorithm terminates).}
\end{aligned}
$$

The problem here is the requirement that $\mathtt{t}_5$ be performed by

```
Inputs: canonical linear extension [t1,...,tn]
        user authorization information
        set of constraints C
01    for i = 1 to n
02        let V(i) = set of users authorized to perform task i
03    for i = 1 to n
04        for j = i to n
05            if ((i,j),rel) ∈ C
06                let V(i,j) = (V(i) ×  V(j)) ∩ rel
07                if V(i,j)  =  ∅ then return false
08                let V(i) = set of users in first position of V(i,j)
09                let V(j) = set of users in second position of V(i,j)
10    return true
```

Figure 2: An algorithm for determining whether a valid execution assignment exists

a more senior user that the one who performed t_3, coupled with the fact that there is only one user assigned to the most senior role. (In actual fact, given the user-role assignment relation in Figure 3(d), t_3 can never be performed by either a or b, irrespective of who performs t_1.) However, if we assign a second user e to r_1, then a is permitted to perform t_1 (since e can perform t_2), and b is permitted to perform t_3 (since both a and e are more senior than b). All valid execution assignments for W are shown in Fig 3(e).

In certain cases, the algorithm will deduce that a particular task can only be completed by a certain user or a small number of users. It would make sense to cache this information to facilitate quicker processing of requests to perform that task. Given a user request, the reference monitor first checks the cache to see whether any information is available for the task. If not, the algorithm is used to determine whether the request should be granted.

4. EXTENDING THE MODEL

We replace the partially ordered set of tasks with a directed graph in which an edge between t and t' denotes that t' must follow t in every instance of the workflow. We also label each task with a set of integers, which defines the number of occurrences of the task that are permitted in an instance of the workflow. We extend the definitions of entailment constraint, workflow authorization schema and constrained workflow authorization schema in the natural way.

More formally, a workflow specification is an acyclic, labelled directed graph (T, E, ϕ), where T is a set of tasks, $E \subseteq \mathsf{T} \times \mathsf{T}$ is a set of edges and $\phi : \mathsf{T} \to 2^{\mathbb{Z}}$. Let $E^* \subseteq \mathsf{T} \times \mathsf{T}$ denote the transitive closure of E. In other words, $(\mathsf{t}, \mathsf{t}') \in E^*$ iff there exists a path between t and t' in the graph (T, E). We require that if $(\mathsf{t}, \mathsf{t}') \in E^*$, then t must be executed before t' in every instance of the workflow. To complete the specification of the workflow, we define a function $\phi : \mathsf{T} \to 2^{\mathbb{Z}}$, where $\phi(\mathsf{t})$ is either of the form $[m, n] = \{m, m+1, \ldots, n-1, n\}$ or $[m, \infty) = \{m, m+1, \ldots\}$ and $m \geqslant 0$. We require that if t is performed k times in an instance of the workflow, then $k \in \phi(\mathsf{t})$. If, for example, $\phi(\mathsf{t}) = [0, 1]$, then t must be performed at most once in every workflow instance. Alternatively, if $\phi(\mathsf{t}) = [1, \infty)$, then t must be performed at least once and can be repeated arbitrarily often.

Clearly, the simple model described in Section 2 is a special case of the model described above in which $\phi(\mathsf{t}) = [1, 1]$ for all $\mathsf{t} \in \mathsf{T}$. In addition, we can model cardinality constraints of the form "task t must be performed twice" by setting $\phi(\mathsf{t}) = [2, 2]$. We can also model constraints of the form "task t must be performed twice by different users" [2] by additionally specifying the entailment constraint $((\mathsf{t}, \mathsf{t}), \neq)$.

We can modify our running example so that `signGRN` may be performed several times, catering for the possibility that the entire order may not be received in a single delivery. Similarly, the company may issue several invoices for the same order, so the `crtPay` task may be performed more than once. We also have the possibility that one or more of the tasks `apprPO` and `ctrsignGRN` are not executed in an instance of the workflow. One possible application scenario is that `apprPO` is not executed if the order value is less than some threshold value. In other words, we define $\phi(\texttt{signGRN}) = \phi(\texttt{crtPay}) = [1, \infty)$ and $\phi(\texttt{apprPO}) = \phi(\texttt{ctrsignGRN}) = [0, 1]$.

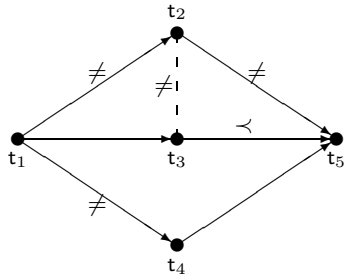
DEFINITION 10. *A* workflow authorization schema *is a* 4*-tuple* (T, E, ϕ, A)*, where* $A \subseteq \mathsf{T} \times U$ *and* $(\mathsf{t}, u) \in A$ *means that* u *is* authorized to perform *(or* execute*)* t*. An* entailment constraint *has the form* $((\mathsf{t}, \mathsf{t}'), \rho)$*, where* $\rho \in Rel(U)$ *and* $(\mathsf{t}', \mathsf{t}) \notin E^*$*. A* constrained workflow authorization schema *is a* 5*-tuple* $(\mathsf{T}, E, \phi, A, C)$*, where* C *is a set of entailment constraints.*

For convenience, we will write $\lceil \mathsf{t} \rceil$ for the maximum number of permitted occurrences of t in any workflow instance and $\lfloor \mathsf{t} \rfloor$ for the minimum number. We write $\mathsf{t}^{(i)}$ to denote the ith execution of task t. We will generally write t rather than $\mathsf{t}^{(1)}$ if t is only executed once. We will also write $k(\mathsf{t})$ to denote the number of occurrences of t in a workflow instance.

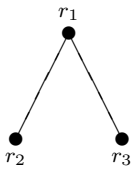
DEFINITION 11. *Let* $\mathsf{W} = (\mathsf{T}, E, \phi, A, C)$ *be a constrained workflow authorization schema. An* execution schedule *for* W *is a sequence of tasks* $L = [\mathsf{t}_1, \ldots, \mathsf{t}_n]$ *such that for all* $i < j$*,* $(\mathsf{t}_j, \mathsf{t}_i) \notin E^*$*, and* $\lfloor \mathsf{t} \rfloor \leqslant k(\mathsf{t}) \leqslant \lceil \mathsf{t} \rceil$*.*

In other words, the execution schedule does not violate any paths in the workflow specification, and the number of times each task is executed lies within the bounds defined for the task.

DEFINITION 12. *An* execution assignment *for* W *is a pair* (L, α)*, where* L *is an execution schedule for* W *and* $\alpha : L \to U$ *is an assignment of tasks to users. We say* (L, α) *is a* valid execution assignment *for* W *if* $(\mathsf{t}, \alpha(\mathsf{t})) \in A$ *and for all* $((\mathsf{t}, \mathsf{t}'), \rho) \in C$*,* $(\alpha(\mathsf{t}^{(i)}), \alpha(t'^{(j)})) \in \rho$*,* $\lfloor \mathsf{t} \rfloor \leqslant i \leqslant \lceil \mathsf{t} \rceil$*,* $\lfloor \mathsf{t}' \rfloor \leqslant j \leqslant \lceil \mathsf{t}' \rceil$*.*



(a) W



(b) Role hierarchy

Task	Role
t_1	r_3
t_2	r_1
t_3	r_2
t_3	r_3
t_4	r_2
t_5	r_2
t_5	r_3

(c) Task-role assignment

User	Role
a	r_1
b	r_2
b	r_3
c	r_2
d	r_3

(d) User-role assignment

t_1	t_2	t_3	t_4	t_5
b	a	c	a	b
d	a	c	a	b
d	a	c	b	b
b	a	c	c	b
d	a	c	c	b
b	a	d	a	b
d	a	d	a	b
d	a	d	b	b
b	a	d	c	b
d	a	d	c	b

(e) Valid execution assignments

Figure 3: An example of a constrained workflow

4.1 A reference monitor

DEFINITION 13. *Let* $\mathsf{W} = (\mathsf{T}, E, \phi, A, C)$ *be a constrained workflow authorization schema. A* minimal *execution schedule is an execution schedule such that* $k(\mathsf{t}) = \max\{1, \lfloor \mathsf{t} \rfloor\}$. *A* canonical *execution schedule is a minimal execution schedule* L *such that for all* $\mathsf{t} \in \mathsf{T}$, $\mathsf{t}^{(1)}, \ldots, \mathsf{t}^{(k(\mathsf{t}))}$ *is a sub-list of* L.

In other words, a minimal execution schedule includes every task at least once and no task occurs more than the minimum number of required times, and all executions of a particular task occur consecutively in a canonical execution schedule. For illustrative purposes, we simplify our running example by omitting the `ctrsignGRN` task, setting $\phi(\texttt{signGRN}) = [2, 2]$, and introducing the constraint $((\texttt{signGRN}, \texttt{signGRN}), \prec)$. In other words, for any purchase order, we assume there is a single GRN that must be signed twice, the second time by a more senior user. Then we obtain the following canonical execution schedules:

$$[\texttt{apprPO}, \texttt{crtPO}, \texttt{signGRN}^{(1)}, \texttt{signGRN}^{(2)}, \texttt{crtPay}, \texttt{apprPay}],$$

$$[\texttt{apprPO}, \texttt{crtPO}, \texttt{crtPay}, \texttt{signGRN}^{(1)}, \texttt{signGRN}^{(2)}, \texttt{apprPay}].$$

Of course, there may exist several execution schedules in which the occurrences of a particular task are not consecutive. Nevertheless, using Lemma 9, it is sufficient to consider a canonical execution schedule when checking satisfiability.

Figure 4 shows a revised algorithm for determining whether there exists a valid execution assignment for a workflow specification containing both entailment constraints and cardinality constraints. Notice that lines `09` and `11` replace the check in our previous algorithm that established whether there exists a user authorized to perform each task. Instead, we check that there exists a sufficient number of authorized users to perform the minimum number of executions of each task. Clearly, the algorithm has the same computational complexity as our original algorithm: that is to say $\mathcal{O}(|\mathsf{T}|^2|U|^4)$ in the general case, and $\mathcal{O}(|\mathsf{T}|^2|U|^2)$ if we only consider separation of duty and binding of duty constraints.

Let $k \in \mathbb{Z}$ and t be a task. Define

$$\phi(\mathsf{t}){-}k = \begin{cases} [0, n-k] & \text{if } \phi(\mathsf{t}) = [m, n] \text{ and } m \leqslant k < n, \\ [m-k, n-k] & \text{if } \phi(\mathsf{t}) = [m, n] \text{ and } m > k, \\ [0, \infty) & \text{if } \phi(\mathsf{t}) = [m, \infty) \text{ and } m \leqslant k, \\ [m-k, \infty) & \text{if } \phi(\mathsf{t}) = [m, \infty) \text{ and } m > k, \\ \emptyset & \text{otherwise.} \end{cases}$$

To construct a completion compliant reference monitor, we select a particular canonical execution schedule for each workflow specification. Then, given an instance $[(t_1, u_1), \ldots, (t_j, u_j)]$ of the workflow $(\mathsf{T}, E, \phi, A, C)$ and a request from user u to execute task t, we run the algorithm on the workflow $(\mathsf{T}, E, \phi', A', C)$, where

$$\phi'(x) = \begin{cases} \phi(x) - k(x) - 1 & \text{if } x = \mathsf{t}, \\ \phi(x) - k(x) & \text{otherwise;} \end{cases}$$

$$A' = \{(\mathsf{t}, u), (\mathsf{t}_1, u_1), \ldots, (\mathsf{t}_j, u_j)\} \cup \{(\mathsf{t}', u) \in A : \phi'(\mathsf{t}') \neq \emptyset\}.$$

Note that we make the assumption that the workflow management system will check that it is valid to try to execute the task. More specifically, we assume that if an optional task t is not executed, then any attempt to subsequently

```
Inputs: canonical linear extension [t1,...,tn]
        user-authorization information
        set of constraints C
        function φ
01    for i = 1 to n
02        let V(i) = set of users authorized to perform task i
03    for i = 1 to n
04        for j = i to n
05            if ((i,j),rel) ∈ C
06                let V(i,j) = (V(i) ×  V(j)) ∩ rel
07                let V(i) = set of users in first position of V(i,j)
08                    if |V(i)| < ⌊i⌋ then return false
09                let V(j) = set of users in second position of V(i,j)
10                    if |V(j)| < ⌊j⌋ then return false
11    return true
```

Figure 4: An algorithm for determining whether a valid execution assignment exists

execute t will be prevented by the workflow management system if a task $\mathtt{t}'$ such that $(\mathtt{t},\mathtt{t}') \in E^*$ has already been executed. In our example, if we perform the `crtPay` task, we do not subsequently allow an attempt to execute the `apprPO` task. Note that this assumption does not affect the correct operation of our algorithm, since $\lfloor \mathtt{t} \rfloor = 0$ for any optional task t.

5. DISCUSSION

5.1 Computational complexity

We have presented a model for specifying constraints and given an algorithm that will establish both the satisfiability of a workflow specification and form the basis for the implementation of completion compliant reference monitor. Such a reference monitor has computational complexity $\mathcal{O}(|\mathsf{T}|^2|U|^4)$ in the general case, and $\mathcal{O}(|\mathsf{T}|^2|U|^2)$ if we only allow separation of duty and binding of duty constraints. In contrast, Bertino *et al* describe a reference monitor for the BFA model that works by computing all possible assignments of roles to tasks and all possible assignments of users to tasks. Essentially, all valid execution assignments are encoded in the user-assignment graph. Naturally, these two procedures are exponential in the number of tasks in the workflow.[3] In short, the lower complexity of our algorithm is attributable to the different emphasis of the respective algorithms: the BFA model pre-computes *all* valid execution assignments, whereas our algorithm essentially tests for the existence of a valid execution assignment.

Table 1 illustrates how the number of valid execution assignments varies with the number of users and the number of constraints defined on the workflow. The figures were computed for the workflow specification in Figure 3(a). We successively added constraints $((\mathtt{t}_1,\mathtt{t}_2),\neq)$, $((\mathtt{t}_2,\mathtt{t}_3),\neq)$, $((\mathtt{t}_1,\mathtt{t}_4),\neq)$, $((\mathtt{t}_2,\mathtt{t}_5),\neq)$ and $((\mathtt{t}_3,\mathtt{t}_5),\prec)$. In other words, when the number of constraints is 3, for example, the set of constraints is $\{((\mathtt{t}_1,\mathtt{t}_2),\neq),((\mathtt{t}_2,\mathtt{t}_3),\neq),((\mathtt{t}_1,\mathtt{t}_4),\neq)\}$. We successively doubled the number of users simply by replicating (and then renaming) the existing user population and their respective user-role assignments. Hence, 8 of the 32 users are assigned to role r_1, for example.

[3]The authors also describe a heuristic that can be used to reduce the number of users that need to be considered in the user-assignment graph, but the algorithm remains exponential. We believe that similar heuristics could be used to reduce the running time of our algorithm.

The table illustrates the exponential growth of the number of valid execution assignments (VEAs), as expected, but also shows that the ratio of valid VEAs to execution assignments (EAs) increases as the user population increases. This is also to be expected, as more users provide more ways of satisfying a separation of duty constraint. Nevertheless, it reinforces the belief that computing all VEAs is unlikely to be a scalable solution, and that our approach of establishing whether the workflow specification determined by a partially completed instance is satisfiable is likely to be more useful in practice. Notice also the dramatic effect of constraints such as $((\mathtt{t}_3,\mathtt{t}_5),\prec)$ on the number of valid execution assignments.

Users	Constraints	VEAs	EAs	%VEAs
4	1	96	144	66.67
	2	72		50.00
	3	60		41.67
	4	45		31.25
	5	10		6.94
8	1	3840	4608	83.33
	2	3360		72.92
	3	3024		65.63
	4	2646		57.42
	5	756		16.41
16	1	135168	147456	91.67
	2	126720		85.94
	3	120000		81.38
	4	112500		76.29
	5	34000		23.06
32	1	4521984	4718592	95.83
	2	4380672		92.84
	3	4261632		90.32
	4	4128456		87.49
	5	1271616		26.95

Table 1: How the number of valid execution assignments varies with users and constraints

We also note that the BFA model stores and updates a copy of the user-assignment graph for each instance of the workflow. The legitimacy of a request to execute a task is determined by querying this graph. Clearly, this requires

considerable storage and computational effort. In contrast, our algorithm only requires information about which users have already executed tasks in the workflow instance, user authorization information and a canonical linear extension. The last two of these inputs are fixed for each instance of a workflow.

5.2 Role-based constraints

Two aspects of our specification scheme deserve further comment. Unlike most other specification schemes, our scheme does not rely on the use of role-based access control, nor do we use role-based entailment constraints.

We have deliberately avoided linking our specification and enforcement model to a particular access control mechanism. Provided we can generate the set of users that are assigned to a task, a straightforward exercise using role-based techniques or access control lists, our algorithm for checking satisfiability will work. This means our algorithm for checking completion compliance can be incorporated into a variety of different access control mechanisms.

Constraints of the form "$\mathtt{t}_2$ must be performed by a role that is more senior than the role that performed $\mathtt{t}_1$", for example, have received attention in the literature [1, 2]. It seems superficially attractive to extend the specification scheme for entailment constraints to include role-based ones of the form $(S, \{\mathtt{t}, \mathtt{t}'\}, \rho)$, where $S \subseteq R$. However, we believe that such constraints are inappropriate in the wider context of role-based access control. More specifically, it is difficult to interpret the phrase "must be performed by a role". We note that tasks are performed by *users* acting in roles: in order for a user u to perform task t, u must be assigned to some role r and the task must be assigned to some role r' such that $r' \leqslant r$. Hence, there are two possible answers to the question "which role performed task $\mathtt{t}$": either r because this role implicitly assigns u to r', the role that is assigned to t; or r' because it is the role to which t is explicitly assigned. Nevertheless, both of these interpretations have their problems. In the first case, what happens if u is assigned to two roles r_1 and r_2 that are both greater than r' and r_1 and r_2 are not comparable in the role hierarchy? Which of r_1 and r_2 is considered to be the role that performed t? In the second case, what happens if t is assigned to two roles r_1' and r_2' that are both less than r? Which of r_1' and r_2' is considered to be the role that performed t? In order to address these problems, we might consider two simplifying assumptions.

- *Every task is assigned to precisely one role*

  Therefore, we can assume that the role that performed the task is the role to which it is explicitly assigned. This means that any role-based constraints can be checked *statically*. For example, to enforce an entailment constraint of the form $(R, \{\mathtt{t}, \mathtt{t}'\}, <)$, it is sufficient to check that $\mathtt{t}$ and $\mathtt{t}'$ are assigned to roles r and r', respectively, with $r < r'$.

- *Every user is assigned to precisely one role*

  We then assume that the role that performed a task is the role to which the user who performed the task is assigned. In this case, it is not possible to check role-based constraints statically, but the rich role-based access control paradigm is reduced to one that is no more powerful than Unix groups.

Moreover, we believe that constraints of the form "$\mathtt{t}_2$ must be performed by a role that is more senior than the role that performed $\mathtt{t}_1$" should either be expressed as constraints on the relative seniority of users or within the authorization schema itself. This constraint, for example, could be reformulated as "for every role to which task $\mathtt{t}_1$ is assigned, there must be a more senior role to which $\mathtt{t}_2$ is assigned". An advantage of this approach is that such constraints can be encoded in the authorization schema, rather than enforced in each instance of the workflow.

In short, we do not believe it is necessary to assume the underlying access control mechanism is role-based, nor is it desirable, because of the aforementioned problems of interpretation, to specify and dynamically check role-based entailment constraints.

5.3 A limitation of our model

A consequence of the definition of constraint satisfaction in our extended model is that an entailment constraint $((\mathtt{t}, \mathtt{t}'), \rho)$ is enforced on each pair of users that perform $\mathtt{t}$ and $\mathtt{t}'$. We now discuss some practical consequences of this definition.

We note that in our earlier example, every instance of the `signGRN` task is performed by the same user (namely the user that raised the corresponding purchase order). This means that there is no ambiguity in the specification of the constraint $((\mathtt{signGRN}, \mathtt{ctrsignGRN}), \prec)$. However, if we remove the constraint $((\mathtt{crtPO}, \mathtt{signGRN}), =)$, then any user can sign for the goods. In our example, the `ctrsignGRN` task is performed at most once and has the effect of countersigning any GRNs that were signed in the execution one or more `signGRN` tasks. Realistically, however, the business requirements would be that each particular GRN is countersigned by a more senior user. This cannot be modelled within our scheme (although we do not believe that it is not possible in any other workflow authorization scheme that we are aware of). This is because our interpretation of constraints is that any user who performs `ctrsignGRN` must be more senior than *any* user who performed `signGRN`. One possible solution is to admit the possibility of nested workflow specifications. In particular, the purchase order workflow might itself contain a simple workflow comprising the tasks `signGRN` and `ctrsignGRN` with a single constraint $((\mathtt{signGRN}, \mathtt{ctrsignGRN}), \prec)$; an instance of this nested workflow is created when `signGRN` is executed. Hence each instance of `signGRN` is associated with a single `ctrsignGRN` task. This is outside the scope of our work in this paper, but would be certainly interesting to pursue in future work.

6. CONCLUDING REMARKS

We have provided a simple yet expressive method for specifying authorization constraints in workflow systems. In particular, we can define separation of duty constraints, weak separation of duty constraints, binding of duty constraints, constraints on the relative seniority of users who perform different tasks, and constraints determined by contextual user-based information. To the best of our knowledge, this provides a far greater range of constraints than any existing model. Our model is independent of any computational model or access control mechanism.

Moreover, we have provided an algorithm that can establish the satisfiability of a constrained workflow authorization schema. This algorithm can be used to determine whether

granting a request in a given instance of a workflow schema would prevent the workflow from completing. Assuming this is a desirable property, we can incorporate this algorithm into a reference monitor for workflow management systems. Since our model is independent of any computational or access control paradigm, we expect that it could be employed in a variety of different contexts. The algorithm runs in time polynomial in the number of users and tasks, unlike the equivalent procedure in the BFA model.

Nevertheless, there are opportunities for further research, one of which was discussed in the last section. Another important question is how to handle changes to authorization data over the lifetime of a workflow instance. Changes to such data may have unintended side effects on the behaviour of the proposed reference monitor. Obviously, a simple solution is to take a "snapshot" of the authorization data when the workflow instance is created and to use that data for that instance of the workflow. Nevertheless, more elegant methods might be available. The work of Bertino *et al* on temporal models for database authorization and role-based access control may provide some guidance here.

Our approach, in common with related work on specifying and enforcing authorization constraints in workflow systems, adopts a relatively simple model for workflows. A task is regarded as an atomic component within these models: that is, a task has either been executed or it has not. Researchers in information systems have defined more complex workflow models in which tasks are associated with one or more primitive operations, such as *begin*, *abort*, and *commit*, and each task is associated with a state that is dependent on the primitive operations that have been performed on that task [9]. The focus in such work has been on enforcing dependencies between tasks defined in terms of their respective states and determined by the business logic, rather than access control and the enforcement of a security policy defined by a set of authorization constraints. We believe it is important to develop a unified workflow model that incorporates both task dependencies, such as control flow dependencies and value dependencies [9], and security dependencies such as separation of duty constraints and other entailment constraints. We anticipate that, under the simplifying assumption that all primitive operations within a task are executed by the same user, such a model is achievable.

Acknowledgements. The author would like to thank the anonymous reviewers for their constructive comments and suggestions.

7. REFERENCES

[1] Atluri, V., and Huang, W. An authorization model for workflows. In *Proceedings of the 4th European Symposium on Research in Computer Security* (1996), pp. 44–64.

[2] Bertino, E., Ferrari, E., and Atluri, V. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security 2*, 1 (1999), 65–104.

[3] Botha, R., and Eloff, J. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal 40*, 3 (2001), 666–682.

[4] Casati, F., Castano, S., and Fugini, M. Managing workflow authorization constraints through active database technology. *Information Systems Frontiers 3*, 3 (2001), 319–338. Also available as Technical Report HPL-2000-156, Hewlett Packard Laboratories.

[5] Clark, D., and Wilson, D. A comparison of commercial and military computer security policies. In *Proceedings of 1987 IEEE Symposium on Security and Privacy* (1987), pp. 184–194.

[6] Crampton, J. An algebraic approach to the analysis of constrained workflow systems. In *Proceedings of 3rd Workshop on Foundations of Computer Security* (2004), pp. 61–74.

[7] Kandala, S., and Sandhu, R. Secure role-based workflow models. In *Database Security XV: Status and Prospects* (2002), pp. 45–58.

[8] Knorr, K., and Stormer, H. Modeling and analyzing separation of duties in workflow environments. In *Trusted Information: The New Decade Challenge, IFIP TC11 Sixteenth Annual Working Conference on Information Security* (2001), pp. 199–212.

[9] Rusinkiewicz, M., and Sheth, A. Specification and execution of transactional workflows. In *Modern Database Systems: The Object Model, Interoperability, and Beyond*. Addison-Wesley, 1995, pp. 592–620.

[10] Tan, K., Crampton, J., and Gunter, C. The consistency of task-based authorization constraints in workflow systems. In *Proceedings of 17th IEEE Computer Security Foundations Workshop* (2004), pp. 155–169.

[11] Wainer, J., Barthelmess, P., and Kumar, A. W-RBAC – A workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems 12*, 4 (2003), 455–486.