

On Delegation and Workflow Execution Models

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

Workflow systems have long been of interest to computer science researchers due to their practical relevance. Supporting delegation mechanisms in workflow systems is receiving increasing research interest. In this paper, we conduct a comprehensive study of user delegation operations in computerized workflow systems.

In a workflow system, the semantics of a delegation operation are largely based on three factors: the underlying *workflow execution* model, *task* type and *delegation* type. We describe three different workflow execution models and examine the effect of various delegation operations in each workflow execution model. We then extend our workflow execution models to examine the effect of various delegation operations in different *role-based* workflow execution models.

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, Workflow management systems

1. INTRODUCTION

A workflow identifies various activities that are involved in an organizational or a business process. A *computerized* workflow system automates such processes. In a computerized workflow, typically, *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.

A workflow may be executed in (at least) two ways: either tasks that are to be performed may be assigned (forwarded) to users or users may initiate requests to perform tasks. The Workflow Management Coalition (WfMC) recommends a “workflow reference model” for modeling computerized workflows [6]. In the WfMC workflow reference model, the workflow management system (WfMS) assigns tasks that are to be performed to users. Subsequently, the user performs the task that has been assigned. We refer to such a workflow execution model as a *WfMS-driven execution model* (WDEM). Recently, Crampton proposed an alternative workflow model where users initiate requests to perform tasks [4]. Subsequently, a reference monitor evaluates such user requests and permits the request if the user is authorized to perform the task. We refer to such a workflow execution model as a *user-driven execution model* (UDEM).

User delegation, in traditional access control, is a mechanism that permits a user to assign a subset of her 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’. Broadly, delegation of privileges may be classified into (at least) two kinds: *grant* and *transfer*. A *grant delegation* model, following a successful delegation operation, allows a delegated access right to be available to both the delegator and delegatee. However, in *transfer delegation* models, following a successful 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 (or issue) *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. However, a concrete task delegation authorizes the delegatee to perform the delegated task *only* in the specified workflow instance.

In a workflow system, the semantics of a delegation operation are largely based on three factors: the underlying *workflow execution* model, *task* type and *delegation* type. If we were to consider delegation in workflow systems, in general, then there are eight different possible ways that must (are to) be considered. Consider, for example, a WDEM in which a user wishes to perform a delegation. Such a user delegation operation may occur

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.

SAC’08 March 16-20, 2008, Fortaleza, Ceará, Brazil
Copyright 2008 ACM 978-1-59593-753-7/08/0003 ...\$5.00.

in one of the following ways: `<abstract-task,grant>`, `<concrete-task,grant>`, `<abstract-task,transfer>` and `<concrete-task,transfer>`. Similarly, we can also identify the above four delegation operations in a UDEM.

Several researchers have considered delegation in workflow systems [1, 2, 10, 11]. However, while supporting delegation in workflows, none of the above works differentiate between the underlying execution models, namely the WDEM and UDEM. Furthermore, none of the above works differentiate between grant and transfer delegations in workflow systems, except [11]. We identify ambiguities and suggest solutions, for existing task-user assignment, that arise following a successful `<abstract-task,transfer>` delegation operation. In this paper, we conduct a thorough examination of various delegations that may occur in different workflow execution models. This is the first attempt in the literature to properly consider all possible delegation operations and compare their enforcement in different workflow execution models.

The rest of the paper is organized as follows. In the next section, we introduce preliminary concepts for this paper. Section 3 discusses delegation in three different workflow execution models. We introduce role-based workflows and discuss delegation in the context of role-based workflows in Section 4. We compare our work with relevant work in the literature in Section 5. We conclude this work and discuss future work in Section 6.

2. PRELIMINARIES

DEFINITION 1. A workflow specification W is a partially ordered set of tasks (T, \preceq) , where $T = \{t_1, \dots, t_n\}$.

A workflow management system instantiates a workflow specification, which is referred to as an *instance* of the workflow, in order to execute a workflow specification. A workflow instance W is an instantiation of a workflow specification W . A concrete task $t \in W$ is an instance of an abstract task $t \in T$. If $t \prec t'$ then t must be performed before t' in any instance of the workflow.

A *workflow authorization model* authorizes users to perform tasks in a workflow.

DEFINITION 2. A workflow authorization schema is a pair (T, A) , where $A \subseteq T \times U$; u is authorized to perform (or execute) t iff $(t, u) \in A$.

We assume the presence of a mechanism that evaluates delegation requests. Successful delegation operations need to be recorded in order to guarantee that delegation operations are correctly enforced. We adopt two relations, the *delegation history* relation and *temporary assignment* relation, from [5] to record successful delegations.

The delegation history relation DH records all successful delegations that have been made. The DH relation is used by the delegators and administrative users for administrative purposes, such as auditing and revoking delegations.

The *tempTA* relation records temporary task-user authorizations that arise from successful delegation operations. *tempTA* contains tuples of the form (i, u, o, s) : i identifies a unique tuple in the DH relation; u is a user; o is the delegated object (which can either be an abstract task t , a concrete task t or a set of roles); and $s \in \{-, +\}$. The tuple $(i, v, t, +)$ means that v is allowed to use t in any workflow instance; such a tuple would arise as a result of a grant or

transfer delegation of t to v . A tuple of the form $(i, u, t, -)$ means that u is prohibited from performing task t in any instance of the workflow; such a tuple only arises when u transfers t to v .

3. DELEGATION IN WORKFLOW MANAGEMENT SYSTEMS

In a workflow system, the semantics of a delegation operation are largely based on three factors: the underlying *workflow execution* model, *task* type and *delegation* type. In this section, we consider three different workflow execution models and discuss delegation in each workflow execution model.

In a WDEM, there exists a process definition, which defines the set of tasks and conditions on the execution of tasks. Such a process definition is also referred to as a *workflow specification*. Authorization information is given which authorizes a user to perform a task.

Typically, in a WDEM system, the WfMS is also responsible for selecting a user u such that $(t, u) \in A$ to perform a task t in a process instance.¹ The WfMS includes the (t, u) pair in a *tasklist*. A handler application then notifies the users about the set of tasks that have been assigned to each of them; users are required to perform tasks that are forwarded to them by the handler application.²

Let L be the (unique) tasklist that is associated with a partially completed workflow instance W . In other words, L is a list of task-user assignments $[(t_1, u_1), \dots, (t_n, u_n)]$. We write $\mathcal{L}(W)$ to denote the set of tasklists associated with instances of the schema W .

We now consider two alternatives for tasklist construction and examine the effect this has on delegation requests.

3.1 Static tasklists (WDEM-S)

In a simple WDEM, when the WfMS instantiates a process definition $W : \{t_1, \dots, t_n\}$, the WfMS pre-determines the set of users who *must* perform the tasks $\{t_1, \dots, t_n\}$.³ Such `<task,user>` pairs are included in the tasklist associated with the instance. In other words, when a process definition $W : \{t_1, \dots, t_n\}$ is instantiated, a tasklist $L : [(t_1, u_1), \dots, (t_n, u_n)]$ is also created, for some $\{u_1, \dots, u_n\} \subseteq U$, such that $(t_i, u_i) \in A; 1 \leq i \leq n$. We refer to such an execution model as a *static* workflow-driven execution model (WDEM-S).

During runtime, the WfMS refers to the process definition and instance state, and decides which tasks are ready to be performed. The WfMS notifies the handler about such tasks

¹The actual mechanism that is used by the WfMS to select a user to perform a task may depend on several factors such as workload balancing, availability of the user, etc. Such user selection mechanisms are ancillary to our discussion and are omitted.

²Note that, in a WDEM, a request initiated by a user u to perform a concrete task t will be permitted if $(t, u) \in L$, and denied otherwise. In particular, in order to evaluate the request, the WfMS will not refer to the user authorization information A .

³A task-user assignment $(t, u) \in L$ means that user u is *required* to perform task t . Hence, we may regard $(t, u) \in L$ as an *obligation* defined for user u (to perform task t). Delegating a concrete task assignment t means that a user u is delegating her current obligation to perform t to another user v .

and the handler notifies the users about the tasks that have been assigned to each of them.

3.1.1 Delegating concrete tasks

Users may wish to delegate concrete tasks that have been assigned to them to other users. Suppose, for example, a user u wishes to delegate a concrete task authorization t , such that $(t, u) \in L$, to another user v . After a successful delegation, the delegatee v is required to perform the task t .

In order to enforce the above delegation, we simply update the tasklist L by replacing the task-user assignment (t, u) with (t, v) . Note that no data is added to the $tempTA$ relation. However, we record the delegation operation in the DH relation for administrative purposes.

A successful delegation of a concrete task t will always result in (re-)assigning the delegated task t to the delegatee. Note that in a WDEM each task in a workflow instance is assigned to a single user (via the tasklist for that instance). This implies that the grant delegation operation has no meaning in this context.

3.1.2 Delegating abstract tasks

A user u may wish to delegate an abstract task authorization \mathbf{t} to another user v . Following a successful delegation, the delegatee v acquires the right to perform task t in any instance of the workflow.

In order to enforce an abstract task delegation, generally, we update the authorization information by including an authorization for the delegatee with the delegated task. Specifically, we record the successful delegation operation in the DH relation and include appropriate tuples in the $tempTA$ relation. However, the effect of such a delegation on the $tempTA$ relation depends on the delegation type.

Recall that the delegator u may perform a grant or a transfer delegation. If u performs a grant delegation of \mathbf{t} to v , then the delegatee v gains an authorization to perform t in any instance of the workflow. In order to enforce the above grant delegation operation, we include tuples in the $tempTA$ relation that authorize the delegatee for \mathbf{t} . More formally, $tempTA \leftarrow tempTA \cup \{(i, v, \mathbf{t}, +)\}$. Note that the enforcement of a `<abstract-task,grant>` delegation does not alter any tasklist.

However, if u performs a transfer delegation of \mathbf{t} to v , then v gains an authorization to perform t in any instance of the workflow and the delegator u is no longer authorized to perform t . Hence, following a successful transfer delegation of \mathbf{t} by u to v , $tempTA \leftarrow tempTA \cup \{(i, v, \mathbf{t}, +), (i, u, \mathbf{t}, -)\}$.

An important question that arises in the context of `<abstract-task,transfer>` delegations is: how do we deal with any concrete task assignments of delegator u ? There are at least two options that can be considered: (i) existing task assignments of u are transferred to v (ii) u retains existing task assignments. We refer to the former case as a *cascading transfer* and the latter case as a *non-cascading transfer* (of \mathbf{t} to v).

Following a successful cascading transfer delegation of \mathbf{t} by u to v , the delegator u transfers existing task assignments of \mathbf{t} , in all workflow instances, to the delegatee v . Now, the delegatee v is required to perform the tasks that are transferred by the delegator u . In order to enforce the cascading transfer delegation of \mathbf{t} , besides updating the $tempTA$ relation as discussed above, we replace all instances of (t, u) with (t, v) for all $L \in \mathcal{L}(W)$.

However, following a successful non-cascading transfer delegation of \mathbf{t} by u to v , the delegator u will retain any existing assignments of t but will no longer receive new assignments of t . While u is expected to perform her current assignments of t , u will not receive any new assignments of t . The delegatee v will not receive any concrete task assignments of t from u , but the WfMS may assign t to v in new workflow instances. Hence, in order to enforce the above non-cascading transfer delegation, $tempTA \leftarrow tempTA \cup \{(i, v, \mathbf{t}, +), (i, u, \mathbf{t}, -)\}$. Note that the non-cascading transfer of \mathbf{t} will not alter any tasklists.

We may also consider a third case where the delegator may wish to transfer all current concrete task assignments (obligations); but is willing to receive new concrete task assignments from the WfMS. We refer to such a delegation as an *obligation-only* transfer operation. In order to enforce an obligation-only transfer delegation of \mathbf{t} by u to v , we replace all instances of (t, u) with (t, v) for all $L \in \mathcal{L}(W)$. Note that an obligation-only transfer of \mathbf{t} will not alter the $tempTA$ relation. Note that an obligation-only transfer is equivalent to transferring each concrete task assignment separately. Table 1 summarizes the above discussed delegation operations.

3.2 Dynamic tasklists (WDEM-D)

Alternatively, in a more complex setting, the WfMS instantiates a process definition but does not create a tasklist for the workflow instance. Instead, when the WfMS selects the next task(s) to be performed in the instance, the WfMS refers to the user authorization information to select a user u to perform the selected task t , such that $(\mathbf{t}, u) \in A$. The WfMS then appends (t, u) to the tasklist. The handler reads the tasklist and notifies the users about the tasks that are assigned to them. Finally, users perform tasks that have been assigned to them. We refer to such an execution model as a *dynamic* workflow-driven execution model (WDEM-D).

3.2.1 Delegating concrete tasks

Concrete task delegations may occur in two scenarios: a user u may wish to delegate a concrete task t that is already assigned to u or may wish to delegate a concrete task t that has not yet been assigned to u . Although the latter case may be of interest, it is unlikely that the process of task assignment is transparent to users, making it more difficult for users to anticipate tasks that may be assigned to them. Further research is needed to establish a genuine case for considering such scenarios. Hence, in our model, we only consider the former case.

The semantics of a concrete delegation with dynamic tasklists is the same as in static tasklists. That is, a concrete task delegation always results in (re-)assigning the delegated task t to the delegatee. As before, we do not need to consider `<concrete-task,grant>` delegations in WDEMs.

A user u may delegate a concrete task t to v only if there exists an assignment $(t, u) \in L$. Following a successful delegation of t , the delegatee v must perform the task t , and the delegator u may no longer perform t , only in that workflow instance. In order to enforce the above delegation, we update the tasklist L by replacing the task-user assignment (t, u) with (t, v) .

3.2.2 Delegating abstract tasks

As before, a user may perform a grant, cascading transfer or non-cascading transfer delegation of an abstract task

Table 1: The effects of abstract task delegation in WDEM-S

Delegation operation	After abstract task delegation	
	Delegator may receive t	Delegator retains t
Grant	✓	✓
Cascading transfer	✗	✗
Non-cascading transfer	✗	✓
Obligation-only transfer	✓	✗

t . The effect of such delegations are same the as in static tasklists. Following a successful `<abstract-task,grant>` delegation of t by u to v , $tempTA \leftarrow tempTA \cup \{(i, v, t, +)\}$.

If u performs a cascading transfer delegation of t , then $tempTA \leftarrow tempTA \cup \{(i, v, t, +), (i, u, t, -)\}$ and we replace all instances of (t, u) with (t, v) for all $L \in \mathcal{L}(W)$. If u performs a non-cascading transfer delegation of t to v , then $tempTA \leftarrow tempTA \cup \{(i, v, t, +)\}$.

Note that following a successful `<abstract-task,grant>` delegation of t by u to v , the WfMS may assign t to u . However, following a successful `<abstract-task,transfer>` delegation of t by u to v , the WfMS will no longer assign (the delegated task) t to u .

3.3 User-driven execution model (UDEM)

In a user-driven workflow execution model, there exists a workflow specification $W : \{t_1, \dots, t_n\}$ and user authorization information $A \subseteq T \times U$. The WfMS is responsible for instantiating and managing workflow instances. 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 will be permitted if $(t, u) \in A$ and denied otherwise.

3.3.1 Delegating concrete tasks

In a UDEM, concrete tasks are not assigned to users. The WfMS maintains a list of concrete tasks that are ready to be performed and users have access to such a list. Subsequently, users initiate requests to perform concrete tasks. The lack of tasklists mean that delegation of concrete tasks is not relevant. Delegation in a UDEM means that the delegator issues an authorization for an abstract task t to the delegatee.

3.3.2 Delegating abstract tasks

A user u may wish to delegate an abstract task t to v . Typically, such a request will be permitted if $(t, u) \in A$. Following a successful abstract task delegation of t , the delegatee v gains an authorization to perform t in any instance of the workflow. As before, u may wish to perform a grant or transfer delegation. Hence, in order to enforce a grant delegation of t , we update the $tempTA$ relation by including an authorization for the delegatee with the delegation task. That is, $tempTA \leftarrow tempTA \cup \{(i, v, t, +)\}$.

However, if u performs a transfer delegation of t then, the delegatee v gains an authorization to perform t in any instance of the workflow and u may no longer be authorized for t . Hence, $tempTA \leftarrow tempTA \cup \{(i, v, t, +), (i, u, t, -)\}$.

3.4 Discussion

We have described two execution models for a WDEM, namely WDEM-S and WDEM-D, and discussed delegation

in both the models. In particular, we have shown that, although, the implementation of the WDEM-S and WDEM-D models are different, the effect of various delegations on the underlying data structures, in both the models, are the same. We have also considered delegation in an execution model in which users select tasks to perform (UDEM).

Table 2 summarizes the effect of various delegation operations in different execution models. This is the first work in the literature to consider delegation in both the WDEM-S and WDEM-D execution models and show that the enforcement effects of delegation operations are the same in both the models.

However, there exist differences between both the models when we consider more complex access control requirements, such as authorization constraints. Our original intention was to study the interaction of delegation and authorization constraints, such as separation of duty. It soon became clear that a careful study of different execution models was required before such a study could be undertaken. It is this preparatory work that we discuss in this paper.

Consider, for example, a simple dynamic separation of duty constraint that states no user may perform both tasks t and t' in the same workflow instance. Assume that a user u initiates a request to delegate a concrete task t , such that $(t, u) \in L$, to another user v . In order to evaluate the above delegation request, in WDEM-S, we simply need to ensure that $(t', v) \notin L$.

Note that the enforcement of a *dynamic* separation of duty constraint in WDEM-S is similar to a *static* separation of duty in RBAC96 [8]. Consider, for example, a simple static separation of duty constraint that states no user may be assigned to both roles r and r' . A request to assign role r to u will only succeed if r' is not assigned to u . (Alternatively, if u is already assigned to r then r' can no longer be assigned to u .) Similarly, in WDEM-S with a dynamic constraint as in the above example, in order to permit a delegation of t to v , we need to ensure that t' is not assigned to v .

However, evaluating the above delegation request in WDEM-D is more complex. Should the above delegation request succeed, then t' cannot be assigned to v in the current instance. Furthermore, if v is the only user authorized for t' and delegation of t to v is successful, then we may not have an authorized user to perform t' . The requirement is that, should we allow the above delegation request, we *always* need to ensure that v is not the only user authorized to perform t' for the current instance. The problem is, since tasks are dynamically assigned to users, the set of users authorized to perform a task $t' \in T$ also changes dynamically. Similar considerations are relevant in UDEMs. Due to space limitations we cannot discuss such scenarios any further in this paper. However, such scenarios will certainly be an immediate priority in our future work.

Table 2: The effect of delegation operations in different workflow execution models

		WDEM-S/D		UDEM	
		Abstract t	Concrete t	Abstract t	Concrete t
Grant	u	retains t	N/A	retains t	N/A
	v	gains t	N/A	gains t	N/A
Cascading transfer	u	loses t ; transfers all t	transfers t	loses t	N/A
	v	gains t ; receives all t	receives t	gains t	N/A
Non-cascading transfer	u	loses t ; retains all t	N/A	N/A	N/A
	v	gains t ; does not receive t	N/A	N/A	N/A
Obligation-only transfer	u	retains t ; transfers all t	transfers t	N/A	N/A
	v	does not gain t ; receives all t	receives t	N/A	N/A

4. EXTENDING THE MODEL TO RBAC

Recent research has considered the use of role-based access control as an authorization model for workflows [3, 7]. A role-based workflow authorization model defines a set of roles R , a set of tasks T , a set of users U , a task-role assignment relation $TA \subseteq T \times R$, a user-role assignment relation $UA \subseteq U \times R$ and a role hierarchy relation $RH \subseteq R \times R$. The graph of the relation RH is acyclic and the transitive reflexive closure of RH defines a partial order on R . We write $r \leq r'$ in preference to $(r, r') \in RH^*$ (the transitive reflexive closure of RH). We may also write $r' \geq r$ whenever $r \leq r'$. We write $\downarrow r$ to denote the set $\{r' \in R : r' \leq r\}$ and $\downarrow R'$ to denote $\bigcup_{r' \in R'} \downarrow r'$. We write $\uparrow r$ to denote the set $\{r' \in R : r' \geq r\}$ and $\uparrow R'$ to denote $\bigcup_{r' \in R'} \uparrow r'$.

Users may perform tasks for which they are authorized based on the roles that are assigned to them. Specifically, a user $u \in U$ is authorized to perform an instance of a task $t \in T$ if there exist $r, r' \in R$ such that $(t, r) \in TA$, $(u, r') \in UA$ and $r \leq r'$.

Grant delegation operations in RBAC are well understood in the literature. However, the semantics of a transfer delegation are more complex in role-based systems. Crampton and Khambhammettu have previously described various semantics for transfer delegation operations in role-based access control [5]. Specifically, following a successful transfer delegation operation of a role $r \in R$, they address the problem of when a delegator loses or retains roles in $\downarrow r$ in complex role hierarchies. We adopt their approach in this paper to discuss transfer delegations of roles and abstract tasks in role-based workflows. In particular, we use the *strong transfer* and (*static*) *weak transfer* delegation operations of a role $r \in R$ from [5].

Following a successful ‘strong’ transfer of a role r , the delegator u may no longer use roles in $\downarrow r$ and the delegatee may use all roles in $\downarrow r$. However, following a successful ‘weak’ transfer of a role r , the delegator may use $x \in \downarrow r$ if there exist roles $x = x_1, x_2, \dots, x_k$ such that $x_i \neq r$, $x_i < x_{i+1}$ and $(u, x_k) \in UA$. (Informally, u retains the use of a role x if there is an alternative path from x to a role to which u is assigned.) In other words, should there exist an alternate path from $x \in \downarrow r$ to a role (say x_k) to which u is assigned, u loses all roles in $R' = \downarrow r \setminus \downarrow x_k \subseteq \downarrow r$.

Consider, for example, the role hierarchy depicted in Fig. 1. We assume that some user u is assigned to roles b and f . In the diagram, unfilled nodes represent roles that are available to u (and filled nodes represent nodes that are not available). Each diagram represents different views of

the role hierarchy: Fig. 1(a) illustrates the whole hierarchy; the remaining figures illustrate the effect of different types of delegation.

The user delegates role d , represented by the square node, to another user. Figure 1(b) illustrates the roles that are denied to the user in the event that strong transfer is used to delegate the role. If we are using weak transfer, the delegator retains role h , as depicted in Fig. 1(c).

4.1 Delegation in role-based WDEMs

In a role-based WDEM, users may delegate a role or a task for which they are authorized. As before, a user u may perform a grant or transfer delegation.

4.1.1 Delegating roles

A user u may delegate a role $r \in R$ if there exists $r' \in R$ such that $(u, r') \in UA$ and $r \leq r'$. We now consider various role delegations that may occur in a role-based WDEM.

grantR(u, v, r): The delegator u performs a grant delegation of a role r to the delegatee v . The delegatee v is authorized for all roles in $\downarrow r$. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, r, +)\}$. Note that, following a successful grant delegation of r , the delegator u retains all existing concrete task assignments that have been assigned through r .

xferRstrongcasc(u, v, r): The delegator u performs a **<strong, cascading>** transfer of a role r to the delegatee v . Since the transfer operation is ‘strong’, u may no longer use roles in $\downarrow r$. The delegatee v gains the use of $\downarrow r$. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, r, +), (i, u, R', -)\}$ where $R' = \downarrow r$. Note that because u is no longer authorized for any role in $\downarrow r$, the WfMS will not assign instances of all tasks $t \in T$ such that $(t, x) \in TA$ and $x \in \downarrow r$ to u .

Furthermore, since the transfer operation is ‘cascading’, u ’s current assignments of t , such that $(t, x) \in TA$ and $x \in \downarrow r$, are transferred to the delegatee v . Hence, for all $t \in T$, such that $(t, x) \in TA$ and $x \in \downarrow r$, we replace all instances of (t, u) with (t, v) for all $L \in \mathcal{L}(W)$.

xferRstrongNcasc(u, v, r): The delegator u performs a **<strong, non-cascading>** transfer of role r to the delegatee v . As before, u may not use roles in $\downarrow r$ and the delegatee v is authorized for all roles in $\downarrow r$. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, r, +), (i, u, R', -)\}$ where $R' = \downarrow r$; and u will no longer receive any assignments

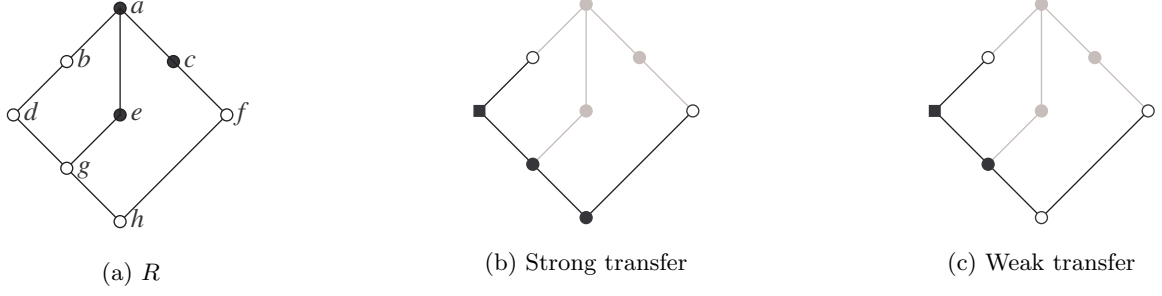


Figure 1: Transfer delegation patterns for role d

of any $t \in T$ such that $(t, x) \in TA$ and $x \in \downarrow r$. However, since the transfer operation is ‘non-cascading’, the delegator u will retain all existing task assignments.

xferRweakcasc(u, v, r): The delegator u performs a **<weak, cascading>** transfer of r to the delegatee v . The delegator may use $x \in \downarrow r$ if there is an alternative path from x to a role (say x_k) to which u is assigned. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, r, +), (i, u, R', -)\}$, where $R' \subseteq \downarrow r \setminus \downarrow x_k \subseteq \downarrow r$.

However, because the transfer operation is ‘cascading’, u ’s existing assignments of t such that $(t, y) \in TA$ and $y \in \downarrow r \setminus \downarrow x_k$ are transferred to v . Hence, for all $t \in T$ such that $(t, y) \in TA$ and $y \in \downarrow r \setminus \downarrow x_k$, we replace all instances of (t, u) with (t, v) for all $L \in \mathcal{L}(W)$.

xferRweakNcasc(u, v, r): The delegator u performs a **<weak, non-cascading>** transfer of r to the delegatee v . As before, u may use a role $x \in \downarrow r$ if there exists an alternative path from x to a role (say x_k) to which u is assigned. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, r, +), (i, u, R', -)\}$, where $R' \subseteq \downarrow r \setminus \downarrow x_k \subseteq \downarrow r$. However, since the transfer operation is ‘non-cascading’, the delegator u will retain all existing task assignments.

4.1.2 Delegating tasks

A user u may delegate a concrete task t that is assigned to u or an abstract task t such that $(t, r) \in TA$, $(u, r') \in UA$ and $r \leq r'$. In both WDEM-S and WDEM-D, delegating a concrete task t will always result in updating (t, u) with (t, v) in the tasklist (where u is the delegator and v is the delegatee). That is, the effect of a successful concrete task delegation is same as in the original model. As before, we do not consider **<concrete-task, grant>** delegations in WDEMs.

The effect of a successful grant delegation of an abstract task t by u to v is the same as in the original model. That is, $tempTA \leftarrow tempTA \cup \{(i, v, t, +)\}$. In a role-based WDEM, a user u may perform a transfer delegation of t to v in one of the following ways: **<strong, cascading>**, **<strong, non-cascading>**, **<weak, cascading>** and **<weak, non-cascading>**.

xferTstrongcasc(u, v, t): The delegator u performs a **<strong, cascading>** transfer of an abstract task t to the delegatee v . Since the transfer operation is

‘strong’, u may no longer use t . Hence, $tempTA \leftarrow tempTA \cup \{(i, v, t, +), (i, u, t, -)\}$. Note also that because u is no longer authorized for t , the WfMS will no longer assign t to u .

Furthermore, since the transfer operation is ‘cascading’, all existing task assignments of t of u will be transferred to the delegatee v . Hence, we replace all instances of (t, u) with (t, v) for all $L \in \mathcal{L}(W)$.

xferTstrongNcasc(u, v, t): The delegator u performs a **<strong, non-cascading>** transfer of an abstract task t to the delegatee v . As before, u may no longer use t . Hence, $tempTA \leftarrow tempTA \cup \{(i, v, t, +), (i, u, t, -)\}$ and u will no longer receive any assignments of t .

However, since the transfer operation is ‘non-cascading’, the delegator u will retain existing task assignments of t . The WfMS may assign new instances of t to the delegatee v .

xferTweakcasc(u, v, t): The delegator u performs a **<weak, cascading>** transfer of an abstract task t to the delegatee v . The delegator may use t , such that $(t, r) \in TA$, if there is an alternative path from the role r to a role (say x_k) to which u is assigned. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, t, +)\}$.

However, because the transfer operation is ‘cascading’, u ’s existing assignments of t are transferred to the delegatee v . Hence, we replace all instances of (t, u) with (t, v) for all $L \in \mathcal{L}(W)$.

xferTweakNcasc(u, v, t): The delegator u performs a **<weak, non-cascading>** transfer of an abstract task t to the delegatee v . As before, u may use t , such that $(t, r) \in TA$, if there exists an alternative path from r to a role x_k to which u is assigned. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, t, +)\}$. However, since the transfer operation is ‘non-cascading’, the delegator u will retain existing task assignments of t .

Table 3 summarizes the effect of various **<abstract-task, transfer>** delegation operations in role-based workflows.

4.2 Delegation in role-based UDEM

A user u may delegate a role $r \in R$, such that $(u, r') \in UA$ and $r \leq r'$ or an abstract task $t \in T$ such that $(t, r) \in TA$,

Table 3: The effect of abstract task delegation in role-based workflows

Delegation operation	After abstract task delegation	
	Delegator may receive t	Delegator retains t
Strong cascading	✗	✗
Strong non-cascading	✗	✓
Weak cascading	✓	✗
Weak non-cascading	✓	✓

$(u, r') \in UA$ and $r \leq r'$.⁴ The lack of tasklists makes it easier to model delegation operations in a role-based UDEM.

4.2.1 Delegating roles

A user u may perform a grant or transfer delegation of a role r . Following a successful grant delegation of r by u to v , the delegatee v acquires the right to use all roles in $\downarrow r$ and the delegator may continue to use all roles in $\downarrow r$. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, t, +)\}$. As before, a user u may perform a ‘strong’ or ‘weak’ transfer of role r .

xferRstrong(u, v, r): The delegator u performs a *strong transfer* of role r to delegatee v . The delegator may not use any role in $\downarrow r$. The delegatee acquires the right to use all roles in $\downarrow r$. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, r, +), (i, u, R', -)\}$, where $R' = \downarrow r$.

xferRweak(u, v, r): The delegator u performs a *weak transfer* of r to the delegatee v . As before, the delegator may use $x \in \downarrow r$ if there is an alternative path from x to a role (say x_k) to which u is assigned. In other words, should there exist an alternative path from x to a role x_k to which u is assigned, u loses all roles in $R' = \downarrow r \setminus \downarrow x_k$. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, r, +), (i, u, R', -)\}$, where $R' = \downarrow r \setminus \downarrow x_k \subseteq \downarrow r$.

4.2.2 Delegating tasks

As in the original model, we do not need to consider delegation of concrete tasks in role-based UDEM (because of the lack of task-user assignments). A user u may delegate an abstract-task t to v if there exists $r, r' \in R$ such that $(t, r) \in TA$, $(u, r') \in UA$ and $r \leq r'$. As before, a successful grant delegation operation of an abstract task t authorizes the delegatee for t . Hence, $tempTA \leftarrow tempTA \cup \{(i, v, t, +)\}$.

A user u may perform a *strong* or *weak* transfer of an abstract task t . Following a successful strong transfer of t , the delegator u may no longer use t and the delegatee v may use t . Hence, $tempTA \leftarrow tempTA \cup \{(i, v, t, +), (i, u, t, -)\}$. However, following a successful weak transfer of t , the delegator may use t such that $(t, r) \in TA$ if there is an alternative path from r to a role (say x_k) to which u is assigned. Hence, $tempTA \leftarrow tempTA \cup \{(i, v, t, +)\}$.

5. RELATED WORK

Atluri *et al* have studied delegation in workflow systems [1]. The WfMS selects a user u to perform a concrete task t and assigns t to u . Users perform tasks that are assigned to them. Such a workflow execution model is similar to our WDEM. Furthermore, a successful delegation of an

abstract task t is *only* enforced when the WfMS selects the delegator u to perform a task t .⁵ Specifically, should the WfMS select the delegator u to perform t , then the WfMS will actually assign t to delegatee v . Note also that, following a successful delegation of t , u will no longer receive further assignments of t . However, u will retain any existing task assignments of t . The enforcement effect of such a delegation mechanism is similar to the ‘‘weak transfer’’ of an abstract task t (WDEM) described in our work. While Atluri *et al* consider only a single delegation operation in a specific workflow execution model, our work includes an analysis of both grant and transfer delegation operations, for abstract tasks, concrete tasks and roles, in three different workflow execution models.

More recently, Wainer *et al* have proposed a framework for delegation in workflow systems [11]. This work adopts a more general approach, but does not differentiate between underlying workflow execution models. This work includes three delegation operations: `delegate(G,D,R)`, `delegate(G,D,R,C)`, and `transfer(G,D,T,C)`. The `delegate(G,D,R)` operation is similar to our `<abstract-task,grant>` delegation operation and the `transfer(G,D,T,C)` operation is similar to our `<concrete-task,transfer>` delegation operation. However, this work does not consider `<abstract-task,transfer>` delegations.

Unlike, the work of Wainer *et al*, we *explicitly* consider, and compare, various delegation operations in three different execution models. The approach that we adopt in our work helps us to clearly differentiate between grant and transfer delegations that may occur in WDEM-S, WDEM-D and UDEM. Furthermore, our approach clarifies the effects of delegation operations in different workflow execution models. Note also that it is easier in our model to determine when the delegator must be considered to be an authorized user and when the delegator will be able to retain existing task assignments.

We have extended our model to discuss delegation in role-based workflows. In particular, we have introduced the notion of ‘‘strong’’ and ‘‘weak’’ transfer for ‘role’ delegations and ‘abstract task’ delegations in role-based workflows. We then discussed four different transfer delegation operations – `<strong,cascading>`, `<strong,non-cascading>`, `<weak,cascading>` and `<weak,non-cascading>` – for both roles and abstract tasks. To our knowledge, no-one has considered such transfer delegations for roles and abstract tasks in role-based workflows.

Schaad considered delegation of obligations [9, Chapter 7]. Our work is different from Schaad’s work in that we do not consider an obligation to be an *explicit* right that a user

⁴As in the original UDEM, we do not need to consider concrete task delegations in role-based UDEM.

⁵Note that, following the delegation of t , the delegator u is still considered to be authorized for t .

might possess. In particular, we do not consider “generic obligations” in our work. Rather, if $(t, u) \in A$ and $(t, u) \in L$ then we regard the delegation of a concrete task t by u to v as the delegation of an obligation. However, similarities may be observed between delegation of a “generic obligation” in Schaad’s work and `<abstract-task,transfer>` delegation in our work. In particular, in Schaad’s work, following the successful delegation of a generic obligation `obl`, the delegator may either retain or transfer any assigned instances of `obl` to the delegatee; this corresponds to the distinction between ‘non-cascading’ transfer and ‘cascading’ transfer of abstract task delegations in our model. However, Schaad’s work does not include a thorough examination of `<strong,cascading>`, `<strong,non-cascading>`, `<weak,cascading>` and `<weak,non-cascading>` transfer delegations of roles and abstract tasks in role hierarchies.

Venter and Olivier also consider delegation in workflow systems [10]. Their delegation model ensures that the delegated authorization t is available for the delegatee only during the specified “authorization time” intervals during which the task t has to be performed. The work of Atluri *et al* presented in [1] has further been extended to provide support for conditional delegation in workflow systems [2]. Such delegation conditions may be based on time, workload and task attributes. We believe that such delegation conditions may easily be included in our work. In fact, the work that we present in this paper is part of a larger work that aims to study delegation in constrained workflows. We believe that the work that we present in this paper lays a robust foundation for our future work, where we address authorization constraints, such as *separation* and *binding* of duty constraints, and ensuring the completion of workflow instances while supporting delegation in constrained workflows.

6. CONCLUSIONS

We have discussed delegation in the context of workflow systems using three different workflow execution models. In particular, we have considered the notion of ‘grant’ and ‘transfer’ delegations for both ‘abstract’ and ‘concrete’ tasks in all three execution models. This is the first attempt in the literature to consider, and compare the effect of, various delegation operations in different workflow execution models.

Our work offers a greater understanding of the effects of various delegation operations on the authorization data structures. Specifically, we have shown that although the WDEM-S and WDEM-D have different execution semantics, the effect of delegation operations on the underlying data structures remain the same. We have also considered both grant and transfer delegations of abstract tasks in UDEM.

We extended our original model to consider delegation in the context of role-based workflows. In particular, we have introduced the notions of ‘strong’ and ‘weak’ transfer of abstract task and role delegations in both role-based WDEM and role-based UDEM. This is the first work in the literature to consider such transfer delegations in role-based workflows.

As mentioned in Section 3.4, the work that we present in this paper is part of a larger work that aims to study delegation in constrained workflow systems, in which authorization constraints introduce additional problems for delegation. Hence, an immediate priority in future work is to examine the interaction between delegation and autho-

rization constraints, such as separation and binding of duty constraints. Revocation is a mechanism that is used to reverse delegation operations. We also intend to examine the effect(s) of revoking successful delegations on the completion of workflow instances.

7. REFERENCES

- [1] ATLURI, V., BERTINO, E., FERRARI, E., AND MAZZOLENI, P. Supporting delegation in secure workflow management systems. In *Proceedings of Seventeenth 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 Tenth ACM Symposium on Access Control Models and Technologies (SACMAT’05)* (2005), pp. 49–58.
- [3] 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 (TISSEC)* 2, 1 (1999), 65–104.
- [4] CRAMPTON, J. A reference monitor for workflow systems with constrained task execution. In *Proceedings of Tenth ACM Symposium on Access Control Models and Technologies (SACMAT’05)* (2005), pp. 38–47.
- [5] CRAMPTON, J., AND KHAMBHAMMETTU, H. Delegation in role-based access control. In *Proceedings of Eleventh European Symposium On Research In Computer Security (ESORICS’06)* (2006), vol. 4189 of *LNCS*, pp. 174–191.
- [6] HOLLINGSWORTH, D. Workflow management coalition: The workflow reference model, 1995. Document Number TC00-1003, Document Status- Issue 1.1, Available at <http://www.wfmc.org/standards/docs/tc003v11.pdf>.
- [7] KANDALA, S., AND SANDHU, R. Secure role-based workflow models. *Database Security XV: Status and Prospects* (2002), 45–58.
- [8] SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. Role-based access control models. *IEEE Computer* 29, 2 (1996), 38–47.
- [9] SCHAAD, A. *A Framework for Organisational Control Principles*. PhD thesis, The University of York, York, England, 2003.
- [10] 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 (ISSA’02)* (2002). Published electronically. Available at <http://icsa.cs.up.ac.za/issa/2002/proceedings/A021.pdf>.
- [11] 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.