A Delegation Framework for Access Control in WfMS based on Tasks and Roles

P., Jian, H.-J., Hsu, and F.-J. Wang
Department of Computer Science
National Chiao-Tung University, Taiwan
{pjian, hjhsu, fjwang}@cs.nctu.edu.tw

Abstract

Access control is important for protecting the information integrities in WfMS's. Compared to conventional access control models such as discretionary, mandatory, and role-based access control models, an access-control model based on both tasks and roles meets more requirements for modern enterprise environments. However, there are no discussions on delegation mechanisms for such a model. In this paper, we propose a new delegation framework to support the access control associated with the model. Among various delegations, two typical cases and their solution algorithms are presented to indicate the usability of the framework.

Keywords: Workflow Management System (WfMS), task, role-based access control, delegation

1. Introduction

Workflow Management Systems (WfMSs) coordinate resources and business processes systematically for modern enterprises [8]. During operation of business processes, activities performed by employees are regulated by access control systems. Role-based access control (RBAC) model groups users with similar permissions into roles and is known as a suitable access control mechanism for enterprise organizations [1][2]. However, business processes operate based on not only roles but also tasks. Instead of RBAC, TRBAC [1] manages access control with both tasks and roles and thoroughly meets the requirements of modern enterprises. However, the delegation mechanisms for TRBAC are still left un-discussed, and the frameworks for such mechanisms would be constructed in this paper.

When an employee meets accidents on business or in privacy, the tasks operated by him might be halt or be delegated. In case a task is delegated, the mechanism(s) for delegation are required for the access control system to protect important information and to assure the process operation normally. Delegation approaches are often built based on access control. For example, RBDM0 [10], RBDM1 [4], and the methods in [5], [6] and [13] work for all delegation models constructed based on RBAC96 [2]. Therefore, it is also necessary to construct delegation approaches for TRBAC.

In this paper, based on the TRBAC model, we present a delegation framework which makes delegations to systematically operate in WfMS's. To indicate the usability of the framework, we discuss two typical cases of delegation, and their solution.

The rest part of this paper is organized as follows. RBDM's and TRBAC models are introduced in section 2. Our delegation framework for tasks and roles is described in section 3. In section 4, two typical delegation cases under the framework and their solution approaches are presented. Finally, the conclusions and future works are presented in section 5.

2 Backgrounds

2.1 RBAC-based Delegation Approaches

Based on RBAC96, RBDM0 [10] provides a flexible way for granting and revoking permissions between roles. RBDM1 [4], an extension of RBDM0, is more realistic since it organizes the roles with hierarchy instead of the latter's flat model. On the other hand, by identifying "can-delegate" relationships between roles, both techniques are focused on role-to-role delegations.

In [5], the user-to-user delegation is considered. The essence of this delegation model is that a user

1071-0485/08 $25.00 © 2008 IEEE
DOI 10.1109/FTDCS.2008.13

delegates a particular right to another user. Nevertheless, unlike RBDM1, this model allows partial delegations, i.e., a user might delegate part of his permissions to another user rather than the whole role. Depending on the delegator's knowledge and experiences, this model may prevent unnecessary authorization from happening.

Osborn separates users in organization, role hierarchies, and relationships among privileges into different graph models in [6][7]. The role graph model gives a visualization presentation about the permission and role assignments. The delegation adopted in [6][7] shows a simple way to delegate privileges to users by creating a delegatee role. This special role provides a convenient way to delegate total or partial permissions of a role.

In RBAC based delegation approaches summarized above, privileges are delegated among users, and it is important for users to acquire appropriate permissions to execute the delegated work. Security problems may arise if the delegatee acquires too much permission; on the contrary, the delegated work may not be accomplished without enough privileges. The approaches may allow delegator to decide how much authority is required for the delegatee; however, the decision which reduces troubles is based on the delegator's wisdom.

2.2 Task-role based access control (TRBAC)

Task-Role-Based Access Control Model (T-RBAC) [1] is proposed by adapting RBAC96 to modern enterprise environments. TRBAC binds permissions on tasks and groups users operating the same tasks into roles. In a WfMS, tasks are the fundamental units of business processes. Restricting the access rights of business objects on tasks helps permission management and reduces risks in inappropriate permission authority made by users. For example, the project budget data are not allowed to be accessed by an engineer. However, an engineer may get these data through the budget request task. This scenario expresses a security fraud in RBAC model, where a user can access an unauthorized datum through the authorized task. TRBAC eases these conditions by binding permissions on tasks.

In TRBAC, there are responsible tasks for each role and the permissions of business objects are bound with specific tasks. In other words, TRBAC supports the active access control through binding the permissions on tasks and sustains the passive access control by grouping users into roles.

In TRBAC model [1], a business process can be viewed as an executing sequence of tasks and the tasks within business processes are offered or allocated to users according to the process schema. Therefore, the tasks coordinated by business processes are under active access control. On the contrary, there are also tasks not belonging to any business processes. These tasks are routine works irrelative to business processes such as private tasks or supervision tasks, and therefore the tasks are classified as under passive access control.

Organization structure gives a view of authority hierarchies. Authorities assigned to descendent job positions or business roles might be inherited by its ancestor roles. In other words, besides the tasks assigned to the corresponding roles, a user may obtain tasks inherited from its descendent roles. Thus, the tasks in TRBAC model can be categorized into four classes in Table 1:

Table 1 Classes of tasks in TRBAC model

	Non-inheritable	Inheritable
Passive access	P (private)	S (supervision)
Active access	W (workflow)	A (approval)

The tasks in class P and S are personal and not related to any business processes and are not delegated. Therefore, in this paper, only are the delegations of the tasks in class W and A considered in our framework.

3 The Delegation Framework for Tasks and Roles

In this section, our delegation framework, based on both tasks and roles, is constructed. Then, the components and their relationships in the framework are introduced.

Conventionally, the one who delegates his privileges or tasks to others is called the delegator, and the one who accepts the delegated objects is called the delegatee. There might be multiple delegatees in one delegation; such a condition is defined as the multiple delegations in [4]. On the other hand, the level of a delegation is also defined to restrict the times a work can be further delegated. These definitions are used in RBDM related researches such as [5] and [10], and our works also.

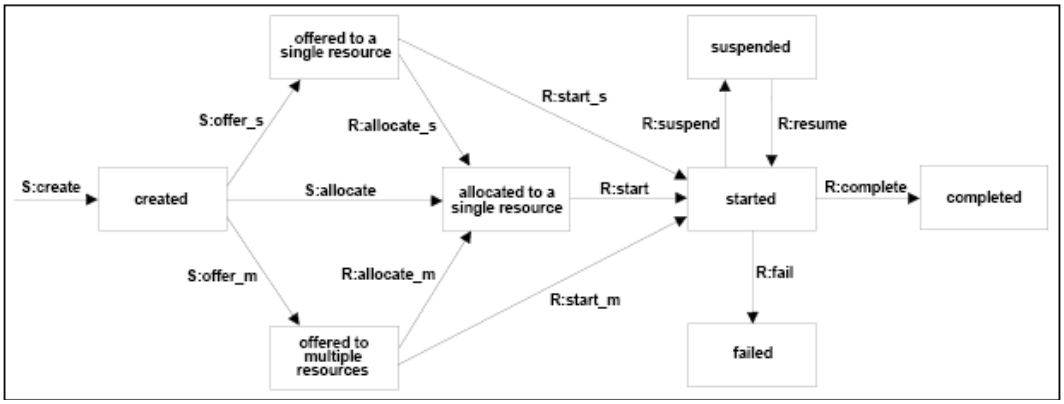

Figure 1 The life cycle of a work item

Tasks define how business works are accomplished, and work items are the run-time instances of tasks. A work item is generally executed by a user playing one of the roles assigned to the instantiation task. Figure 1 [11] illustrates the life cycle of a work item from initiation, offering, allocation, operation to completion. The work items currently executed by a user are recorded in his/her work list. Figure 2 shows the relationships between components in our frameworks. Roles are assigned to tasks and played by users. Work items are instantiated from tasks and executed by users. Permissions constraining access to business objects are bound on tasks and followed by the corresponding work items instantiated. Role hierarchies indicate inheritance relationships between roles to reflect organization lines of authority or responsibility [2].

Similar to an activity defined in [14], a task is the basic component which describes a piece of work forming a logical step within a process. However, in TRBAC model, a task is assigned to roles and describes the required permissions. Each role can be viewed as a collection of users with similar responsibilities, i.e., a collection of users who can execute the same kind of tasks. A user can play multiple roles for business, and a role can also be played by multiple users. On the other hand, the Permission records the access constraints for tasks to business objects.

Since a task can be assigned to multiple roles, a work item is active with one of the roles assigned to its instantiation task, and is executed by the user playing the active role. Therefore the active role of a work item is defined as the role played by the user when he/she executes the work item.

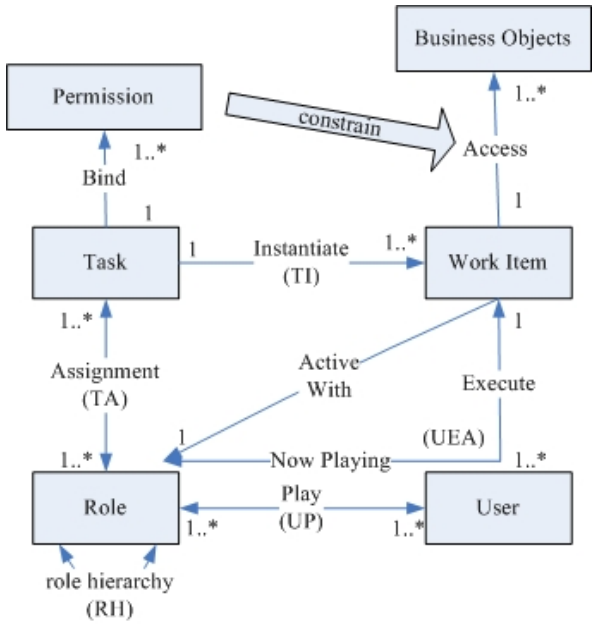

Figure 2: The relationships between components

Let T, R, U, I and P be the sets for tasks, roles, users, work items, and permissions. The relationships between the components are defined in Definition 1.

Definition 1 (Relationships between the components)

1. $TA \subseteq T \times R$ is a set of many-to-many relationships for assigning tasks to roles.
2. $TI \subseteq T \times I$ is a set of one-to-many relationships for instantiating a task to work items.
3. $UP \subseteq U \times R$ is a set of many-to-many relationships for users playing roles.
4. $UEA \subseteq U \times I \times R$ is a set containing elements

indicating that a user u who is now playing a role r on executing a work item i. $\forall (u, i, r) \in$ UEA, $(u, r) \in$ UP

5. $TP \subseteq P \times T$ is a set of many-to-many relationships assigning permissions to tasks.
6. $IP \subseteq P \times I$ is a set of many-to-many relationships assigning permissions to work items. The permissions assigned to any work item must be the same as its corresponding relationship in TI.
7. $RH \subseteq R \times R$ is a set of role hierarchies. $\forall (r_1, r_2) \in$ RH, (r_1, r_2) shows a partial order that all inheritable tasks assigned to r_1 are also assigned to r_2.

As the relationships between components are described, the attributes within each component such as status and timing features are described in Definition 2.

Definition 2 (Attributes of components)

1. $\forall t \in$ T, $t.d$ and $t.D$ represent the minimum and the maximum predicted working duration of the task t. $t.type = \{T_S, T_W, T_P, T_A\}$ is defined as in TRBAC model [11].
2. $\forall i \in$ I, $i.st$ shows the instantiated time of the work item. $i.status = \{S_I, S_O, S_A, S_E, S_C, S_S\}$ shows the status of the work item that is initiated, offered, allocated, being executed, completed, or suspended [11]. $i.ar = \{r \mid r \in R, (t, i) \in TI, (t, r) \in TA\}$ is the active role of i.
3. $\forall u \in$ U, $u.wl = \{i \mid (u, i) \in UE, i.status = \{S_A, S_E, S_S\}\}$ is the work list of the user. $u.status = \{U_R, U_U\}$ shows the status of the user is ready or unavailable.

The role hierarchies describe role-to-role relationships as described in Definition 1. To facilitate our discussion, the relationships between roles can be defined in more detail with the role hierarchies indicated in Definition 3.

Definition 3 (Relationships in role hierarchies)

$\forall r, r' \in R$, $r' >_r r$ or $r <_r r'$ if and only if there exists roles $r_1, \ldots r_k$ that $(r, r_1), (r_1, r_2), \ldots, (r_k, r') \in$ RH. DisRH() is defined as a function showing the distances between roles in role hierarchies:

1) IF $r' >_r r$ DisRH $(r', r) = k+1$, and DisRH $(r, r') = -(k+1)$
2) If $(r, r') \in$ RH, DisRH $(r', r) = 1$ and DisRH $(r', r) = -1$
3) If neither $r' >_r r$ nor $r' <_r r$, DisRH (r, r') and DisRH (r', r) are undefined
4) Otherwise, DisRH$(r, r) = 0$

4 Delegation Approaches in the Framework

In this section, the approaches for delegations and revocations are constructed based on the framework. The properties for a delegation are discussed, and the algorithms are described.

The rest of this section is organized as follows: Section 4.1 describes the algorithm for committing a delegation in our framework. On the other hand, the algorithm for revoking a delegation is discussed in section 4.2.

4.1 The properties and the algorithm for delegation in the framework

Information for delegations is basically defined in tasks and roles; however, work items, the instances of tasks, are actually delegated during run-time. Therefore, in our discussion, the delegator is the user who delegates a work item to another user, and the delegatee of the delegation is the user who accepts the delegated work item. The active role of the delegated work item is called the delegation role.

The relationship for delegation in our framework is formally defined in Definition 4.

Definition 4 (The relationship for delegation)

$UDA \subseteq U \times U \times I$ is a set containing elements showing the user delegation assignment, where the former user, the delegator, delegates a work item to the latter one, the delegatee.

There are various approaches to decide the delegatee for each case of delegation. Assume that the delegatee is decided through a delegatee selection function. To facilitate revocation and other constraint checking, a temporary role is defined. When a delegation occurs, a temporary role is created and assigned to the corresponding task of the delegated work item. With the delegatee selection function, a proper delegatee is selected to play the temporary role, than the delegated work item is re-allocated to the delegatee.

In Definition 15, a work item, the instance of a task, is defined to be executed by one user; thus the case of multiple delegations cannot occur. However, delegation with multiple levels is allowed. Whenever a delegated work item is further delegated, a new delegatee is selected to replace the original delegatee to play the temporary role. The replaced delegatee who is now a delegator is added to the history delegator list recorded in the temporary role. Not only records the information about delegation with multiple levels, the history delegator list also prevents loop assignment from happening.

Definition 5 (Temporary role)

Let TR be the set of temporary roles. $\forall$ *tr* $\in$ TR, *tr.i* $\in$ I is the delegated work item, *tr.u* is the current delegatee, *tr.ar* is the original active role of *tr.i*, and *tr.hul* is the user list recording past delegators of the delegated work item.

For (*t*, *tr.i*) $\in$ TI and (*t*, *tr*) $\in$ TA, such that (*tr.u*, *tr*) $\in$ UP and (*tr.u*, *tr.i*, *tr*) $\in$ UEA.

Let the size of tr.hul *be* k, *(*u_k, tr.u, i*)* $\in$ *UDA, and if* k>*1*, $\forall$ *1*<j<k, *(*u_j, u_{j+1}, i*)* $\in$ *UDA*

Figure 3 shows a sample case of delegation in our framework. Task1 is originally assigned to Role1 for execution. Originally, WorkItem1 is instantiated from Task1, and assigned to User1 who plays Role1. When a delegation for WorkItem1 is requested, TemporaryRole1 is created to accept Task1. Let User2 be decided as the delegatee, the WorkItem1 is re-allocated to User2. In case, User2 requests a delegation again, delegation in multiple levels happens. If User3 is selected as the new delegatee, User3 replaces User2 to play TemporaryRole1. WorkItem1 is than re-allocated to User3, and the relationship between User2 and TemporaryRole1 is destroyed.

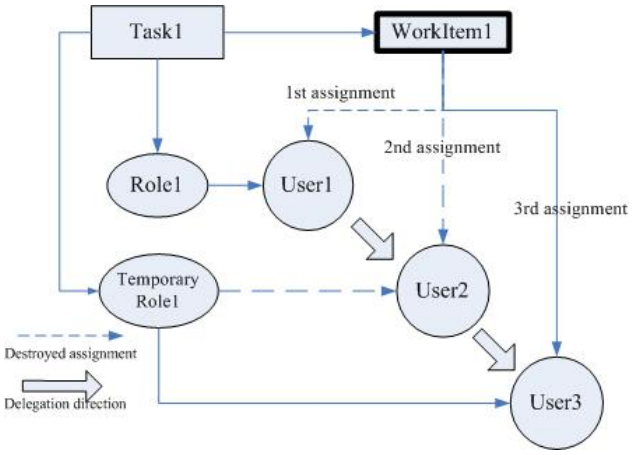

Figure 3: A sample delegation

Algorithm 1 indicates how to a delegation operating in our approach. In this paper, the framework for delegation in TRBAC is discussed, and therefore the functions for delegatee selection are left for the future works. In the following algorithm, we assume that there exist delegatee selection functions for different enterprises.

Algorithm 1 (Delegation)

Input: the delegating work item *i*, the original executor *u*, and the active role *r*. (*u*, *i*, *r*) $\in$ UEA

Output: a boolean indicating whether the delegation is successful

Begin

// too see if it is delegation with multiple levels

if ($\forall$ *tr* $\in$ TR, $\exists$ *tr.i* == *i*) *ttr* = *tr*

else {

create new temporary role *ttr* and add *ttr* to TR

ttr.i = *i*, *ttr.u* = ϕ , *ttr.ar* = *r*, *ttr.hul* = ϕ

for (*t*, *i*) $\in$ TI, add (*t*, *ttr*) to TA

}

set *ttr.u* to the user u_d

u_d is decided by some delegatee selection function

if (*ttr.u* == NULL) return false

else {

// reallocate the work item to the delegatee

remove (*u*, *i*, *r*) from UEA

add (*ttr.u*, *i, ttr*) to UEA

remove *i* from *u.wl* and add *i* to *ttr.u.wl*

add (*u*, *ttr.u*, *i*) to UDA , add *u* to *ttr.hul*

i.status = S_A

```
  }
  return true
End
```

Algorithm 1 first checks whether the delegation is an existing one. If not, a new temporary role for the new created delegation is initiated. With the presumed delegatee selection function, a user would be designated to execute the delegated work item. The work item is re-allocated to the selected delegatee, and the original executor is added to the history user list recorded in the temporary role.

4.2 The algorithm for revocation of a delegation

When the user is available for a work item previously delegated by him, he may also revoke the delegated work item back. The revocation of a delegation can be discussed in several aspects. First, a revocation may be requested by the original owner of the work item or by one of the middle delegatees in multiple levels delegation. In the former situation, the work item is re-allocated to the original executer and the temporary role is eliminated. In the latter situation, the work item is re-allocated to the middle delegatee who requests the revocation, and the other delegatees after him in the history delegatee list are removed.

Second, the timing of revocation is considered. A revocation may happen before or during the operation of the delegated work item. In the latter circumstances, the work item already started might be rolled back before it is transferred to the requestor of revocation, i.e. the work already done by the delegatee is discarded and the work is restarted. On the other hand, the work item may also be revoked directly and the requestor of revocation continues the work unfinished by the delegatee.

Algorithm 2 describes how to make a revocation in our framework..

Algorithm 2 (Revocation)
Input: $tr \in$ TR, tr is the temporary role related to the revoked work item
$u \in$ U, u is the user who requires the revocation
Output: a boolean indicating if the revocation is successful
Begin
If ($u \notin tr.hul$) return false
Else {
let $tr.hul = \{u_1, \ldots, u_k\}$, and $u == u_i$
for $j = i..k-1$ {
remove (u_j, u_{j+1}, i) from UDA
remove u_j from $tr.hul$
}
remove $(u_k, tr.u, tr.i)$ from UDA
if (roll back is required)
for $(t, tr.i) \in$ TI, $tr.i$ = new instance of t
remove i from $tr.u.wl$ and add i to $u_i.wl$
$tr.i.status$ = A
if ($u_i == u_1$) {
remove (tr, i) from TI
remove tr from TR, and destroy tr
remove $(tr.u, i, tr)$ from UEA
add $(u_i, i, tr.ar)$ to UEA
}
else {
remove $(tr.u, i, tr)$ from UEA
add (u_i, i, tr) to UEA
$tr.u = u_i$
}
}
return true
End

Only can the user who delegates the work item revoke it back, and therefore, Algorithm 2 first checks this constraint. If the user asking for revocation of the work item is not recorded in the history delegator list of the corresponding temporary role, the revocation fails. Nevertheless, once the revoked work item was delegated in multiple levels, the delegatees after the revoking user are released from the delegation. Before the work item is re-assigned to the revoking user, roll back is invoked if necessary.

5. Conclusion and Future Work

In this paper, by observing the behaviors of delegation, a framework for delegation based on tasks and roles is proposed. The components for the delegation are described, and the methodologies for both delegation and revocation are constructed.

In the future, delegatee selection functions needs to be constructed for usage of this framework. Besides the approach described in [12], the delegatee might be selected by delegator in the user-authorized style, or be determined dynamically by a WfMS during run-time.

Constraints in delegation such as timing constraints and SoD constraints can be discussed further and the methods for detection need be constructed if the candidates violating the constraints.

6 References

[1] Sejong Oh, Seog Park, "Task-role-based access control model," Information Systems, Volume 28, Issue 6, pp. 533-562, September 2003

[2] R. S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, "Role-Based Access Control Models," IEEE Computer 29(2): 38-47, IEEE Press, 1996.

[3] Simon, R.T.; Zurko, M.E., "Separation of duty in role-based environments," 10th Computer Security Foundations Workshop, pp.183-195, 1997.

[4] Ezedin Barka, Ravi Sandhu, "Role-Based Delegation Model/Hierarchical Roles (RBDM1)," 20th Computer Security Applications Conference, pp. 396-404, Dec. 2004

[5] Jacques Wainer, Akhil Kumar, "A Fine-grained, Controllable, User-to-User Delegation Method in RBAC," ACM symposium on Access control models and technologies (SACMAT), pp. 59-65, 2005

[6] He Wang, Sylvia L. Osborn, "Delegation in the Role Graph Model," ACM symposium on Access control models and technologies, pp. 91-100, 2006

[7] Matunda Nyanchama, Sylvia Osborn, "The Role Graph Model and Conflict of Interest,"ACM Transactions on Information and System Security, Vol. 2, No. 1, pp. 3-33 1999

[8] Workflow Management Coalition Terminology & Glossary, WFMC-TC-1011, 1994

[9] R. A. Botha, J. H. P. Eloff, "Separation of duties for access control enforcement in workflow environments," IBM System Journal, Vol. 40, No.3, pp. 666-683, 2001.

[10] E. Barka and R. Sandhu. "A role-based delegation model and some extensions," In 23rd National Information Systems Security Conference, Baltimore, MD, October 2000

[11] N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst, "Workflow Resource Patterns," BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004.

[12] J.W. Wang, C.H Chang, and F.J. Wang, "An Analysis of Delegation Mechanism in Workflow Management System," in proceedings of 2003 National Computer Symposium.

[13] J. B. D. Joshi, and E. Bertino, "Fine-grained Role-based Delegation in Presence of the Hybrid Role Hierarchy," in Proceedings of the 11th ACM symposium on Access control model and technologies, pp. 81-90, 2006

[14] Workflow Management Coalition, "Workflow Management Coalition, Terminology & Glossary," Document Number WFMC0TC-1011, Document Status – Issue 3.0, Feb. 99'