國立交通大學

網路工程研究所

碩士論文

一個在工作流程系統管理系統中基於Task-Role-Based Access Control Model的代理程序框架

A Delegation Framework Based on the Task-Role-Based Access Control Model for Workflow Management Systems

研究生：簡　璞

指導教授：王豐堅　教授

中華民國　九十六　年八月

一個在工作流程系統管理系統中基於 Task-Role-Based Access Control Model 的代理程序框架

A Delegation Framework Based on the Task-Role-Based Access Control Model for Workflow Management Systems

研 究 生：簡　璞　　Student：Pu Jian

指導教授：王豐堅　　Advisor：Feng-Jian Wang

國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年八月

一個在工作流程系統管理系統中基於 Task-Role-Based Access Control Model 的代理程序框架

研究生：簡璞　　指導教授：王豐堅 博士

國立交通大學網路工程研究所

新竹市大學路 1001 號

摘要

企業使用工作流程管理系統來實行許多的商業流程，當有公司員工無法執行他們的工作時，工作流程管理系統將這些工作代理給適合的人選，Task-role-based access control (T-RBAC) model 減少系統管理者在管理工作流程系統上的成本。然而，在工作流程管理系統中以 T-RBAC 為基礎的代理機制是需要被探討的。本篇論文提出一個在工作流程系統管理系統中基於 T-RBAC Model 的代理程序框架，藉由觀察代理程序的行為，本篇論文將代理程序分成三類，並在這個框架中提出使用者指派迴圈、責任分散以及組織角色衝突等三個問題，並且根據這三個問題提出相關的分析及解決方法。

關鍵字： 以工作角色為基礎的存取控制模型、工作流程管理系統、代理程序

A Delegation Framework based on the Task-Role-Based Access Control Model for Workflow Management Systems

Student: Pu Jian Advisor: Feng-Jian Wang

Institute of Network Engineering National Chiao Tung University

1001 Ta Hsueh Road, Hsinchu, Taiwan, ROC

Abstract

The workflow management system (WfMS) is widely used in performing business processes among enterprises. When an employee is unavailable to execute his/her tasks, WfMS delegates the tasks to appropriate users. The task-role-based access control (T-RBAC) model reduces the administration costs for WfMS. However, the delegation mechanism in T-RBAC for WfMS is necessary to be discussed. In this paper, a delegation framework based on the T-RBAC for WfMS is proposed. By observing the delegation behaviors, three types of delegations are described respectively. Based on the framework, the issues about user assignment loop, separation of duty, and organization role conflict are proposed, and their corresponding analysis methods are also presented.

Keywords: task-role-based access control (T-RBAC) model, delegation, workflow management system (WfMS).

誌謝

本篇論文的完成，首先要先謝謝我的指導教授王豐堅教授，在這兩年的諄諄指教，讓我不論在知識以及觀念上有所精進，對於軟體工程以及工作流程系統也有相當程度的了解。另外，感謝口試委員朱志平博士、梅興博士以及留忠賢博士給我的建議，提供了不一樣的觀點及意見，來補足我論文的缺失。

再者，我要謝謝實驗室的學長姐及同學，博士班的懷中學長不辭辛勞的與我討論，並且修改我的論文，在口試前一天仍然陪我將論文修飾到隔天早上，非常感謝，另外還有博士班的靜慧學姊對於我口式技巧的指點，讓我在口試時得以流暢地對答，對於這些幫助我的學長姐，心中的感激實在難以言語。同學間的鼓勵，也讓我在困頓的時候，拉了我一把，也謝謝他們。

最後，我要感謝我的家人，不論在經濟上以及心理上，都提供我不少的支持，讓我得以無後顧之憂做我的研究。謝謝你們一路陪我走來，陪我做研究的歲月。

Table of Contents

List of Figures

List of Algorithms

List of Tables

1. Introduction

In modern enterprise, company employees perform tasks in several processes. Usually, these employees may be sent to abroad to have meetings or technical supporting. Once a user is not internal, his jobs are still needed to be executed. Therefore, these tasks are assigned to some other people to execute. This re-assignment action is called delegation. The delegation concept is very common in the real world. When a user cannot perform his task, he asks someone to give him a hand to do this task. After the helper agrees the request, this task is passed to him. This is an example of task delegation.

The workflow management system gives the companies a systematic way to realize business processes. It replaces the traditional paper-work business processes by using e-forms and the internet. It increases the efficiency of executing processes. However, like the real world, the tasks in the system process are still performed by company employees. These task performers might sometimes unavailable. Therefore, a systematic delegation is needed to implement as the real world.

Although there are lots of delegation approaches, most of them bases on the Role-Based Access Control model (RBAC96) [2]. The RBAC model defines the relationship between the users and permissions through roles. It gives the administrator a convenient way to management the permissions. Nevertheless, it only defines relationship the between users and data, not for the task and data. This model is not totally suitable for the task-based workflow management systems. Moreover, the subjects of those delegation approaches are permissions. For the workflow process, the atomic elements are tasks. Delegating permission cannot totally fit to the workflow process in delegations. Therefore, a task-based viewpoint is needed.

This paper proposes a system view of delegation. By observing the delegation behavior between the system and users, we define three types of delegations. The observation leads us to develop a task-based delegation framework for the workflow management system. This framework in introduced in Section3. Section2 presents the fundamental approaches of our framework. In Section4, some problems in this framework are stated, and the analysis methods of these problems are introduced.

2. Background

This section introduces backgrounds about access control models and delegation models. In section 2.1, role-based access control model and related delegation model are introduced. Section 2.2 presents the task-role-based access control model.

2.1. Role-Based Access Control Model

Role-based access control (RBAC96) [2] presents a new relationship between users and access rights. In this model, permissions are associated with roles, and roles are assigned to appropriate users. The relationship between user and role is called “User Assignment (UA)”, and the other one between role and permission is called “Permission Assignment (PA).” Fig.2.1 displays RBAC96 model. In RBAC, users get permissions through roles. UA and PA are added constraints to restrict users to gain permissions.

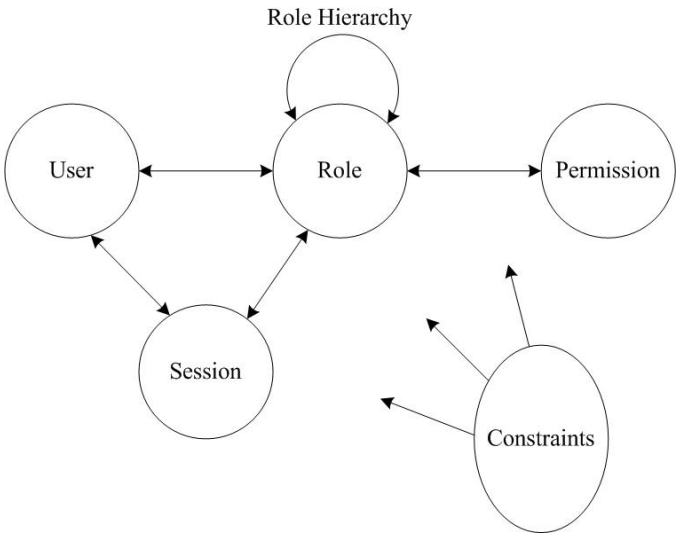


Fig.2.1 RBAC96

The primary concept of RBAC96 is to assign roles to users instead of directly assigning permissions to users. In this model, a role is viewed as a set of permissions. It is convenient to give administrators to manage permissions. A number of products

support some form of RBAC directly, and others support closely related concepts, such as user groups, that can be utilized to implement roles. There are lots of approaches to extend RBAC96 describing below.

Although role is a powerful concept for simplifying access control, the implementation is normally restricted to single systems and applications. Enterprise Role-Based Access Control (ERBAC) [4] enhances RBAC96 to implement the role concept on different systems and applications by defining Enterprise Roles. Using this enhanced role with parameters reduces the number of roles dramatically, thereby minimizing administration and role maintenance costs on different systems and applications.

Generalized Temporal Role-Based Access Control Model (GTRBAC) [6, 7] adds time concept on roles. In practical, users may be restricted to perform roles at predefined time periods. Moreover, roles may only be invoked on pre-specified time intervals when certain actions are permitted. In this model, the duration constraints are added on roles, user-role assignments, and role-permission assignments. The roles are activated only at the limit period. The role is assigned to the user at predefined time. Furthermore, the permissions are allocated to the role at specific time.

2.2. Task-Role-Based Access Control Model and Workflow Management System

Task-Role-Based Access Control Model (T-RBAC) [1] modified RBAC96 to adapt modern enterprise environment. Fig.2.2 [1] shows the comparison of these two approaches. In RBAC, users get permissions through roles, but users in T-RBAC get permissions through tasks and perform tasks through roles. The difference between the two models is that there is a new notion named task in T-RBAC model. The task

is a fundamental unit of business work or business process. In order to reduce company costs and increase working efficiency, business processes are emulated by workflow process in workflow management system. A workflow is organized by tasks. Therefore, the T-RBAC is suitable to model workflow processes.

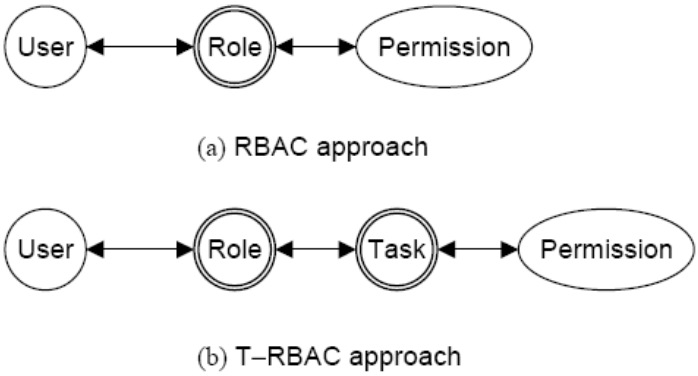


Fig.2.2 Comparison of RBAC and T-RBAC

One disadvantage in RBAC model is that RBAC didn't define the permissions between task and data. For example, the project budget data is not allowed for an engineer role to access. But someone performing the engineer role may get this data through the budget request task. This scenario expresses a security fraud in RBAC model which a role can access an unauthorized data through the authorized task.

T-RBAC simplifies the permission between tasks and data by binding permissions on the tasks. The role cannot directly access the data. Only when the role has the task can it access the data. For example, the access right of budget data is on the task of budget request, not on the engineer role. Therefore, an engineer role cannot access the budget data directly if the budget request task is not assigned to him.

2.3. Delegation approaches in RBAC

The other disadvantage of RBAC is that it only defines a fixed assignment between permissions, roles and users. If a user wants to access an unauthorized data, he must ask the system administrator to grant the permission to him manually. This

increases the maintenance cost of the administrator. Delegation provides a flexible way to grant permissions and roles systematically. When the user wants to access the unauthorized data, the system follows some delegation rules to grant the access rights to him. By using delegations, the administration cost is decreased. There are lost of approaches of delegation based on RBAC.

RBDM1 [10] is a role-based delegation model. It bases on the RBAC model and extends the RBDM0, which was a delegation model using flat roles. This model considers the hierarchical role. By identifying different semantics of ***can-delegate*** relation, it presents the role to role delegation.

A user to user delegation is presented in [11]. The essence of this delegation model is that a user delegates a particular right to another user. Unlike RBDM1, this model not only delegates roles to users but also delegates the single permission to users. Therefore, users get some particular permission without receiving a whole role. This model also gives an algorithm for accepting the delegation.

The role graph model [16] gives the visualization of permission and role assignments. The delegation in role graph [13] shows a simple way to delegate privileges to users by creating a delegatee role. This special role provides a convenient way to delegate a whole role or some particular permissions.

However, all these delegation approaches delegate the privileges. In workflow management system, tasks are the basic elements of the workflow process. Therefore, delegating rights to user is insufficient to support the task-based workflow. In our framework, the idea of delegation is added to provide the task delegation. And the T-RBAC gives the fundamental basis of the framework.

3. A Delegation Framework Based on Task-Role-Based Access Control Model

This section presents a delegation framework in the workflow management systems. The three types of delegations are described in Section 3.1. Section 3.2 shows the basic delegation framework. Revocation process is presented in section 3.3. Section 3.4 describes the advantage of this framework.

3.1. Delegation

In modern enterprises, company employees usually have some business travels. When they go abroad to have meetings, their jobs still need to be done. Thus, their jobs have to be delegated to others properly. This scenario can be corresponded to workflow management system. In a workflow process, if a user is unable to do his task, he can delegate his task to someone who can perform it. Moreover, if a user is accidentally unavailable to his task, the workflow engine may perform such delegation actively.

The user who delegates his task out is called the ***delegator user,*** and the user who is delegated the task is called the ***delegatee user.*** By observing the delegation behaviors, delegations can be divided into three types: ***user-authorized delegation, fixed delegation,*** and ***dynamic delegation***. Table 3.1 shows this observation.

Delegation Requestor	Delegatee User Selection	Types of Delegation
Human or System	Human	User-Authorized Delegation
	Selected by the system through a predefined delegatee user list	Fixed Delegation

	Selected by the system from a delegatee user set constructed by a collection algorithm	Dynamic-Selection Delegation

Table 3.1 Three types of delegations

Users delegate their tasks to others manually are called user-authorized delegations. Fixed delegations mean that tasks are delegated by the system following the predefined delegation delegatee user list. A dynamic-selection delegation is triggered under some unexpected circumstances. For example, a manager is accidentally unable to attend an important meeting. The system needs to automatically select an appropriate user to the meeting in time. However, such an accident is unpredictable, and the replaced person cannot be chosen through a pre-defined delegatee user list. Thus, an algorithm must be defined for each dynamical selection of appropriate replaced users.

3.2. Task-based delegation framework

Fig.3.1 presents the task-based delegation framework. The ***process role repository*** stores the information of process roles in one process, and the ***organization role repository*** stores the organization role information. When a dynamic-selection delegation starts, the workflow engine executes the ***delegatee user collecting algorithm*** to collect users from the process role repository. The set of collected users is called the ***candidate user set***. The collecting algorithm is described in Section 3.2.4.2. After generating the candidate user set, the ***analysis methods*** are executed to prevent some problems in delegations. These methods are described in Chapter4. After the analysis, a ***delegatee user set*** is constructed, and the delegatee user is selected from this set following the ***delegatee user decision rules***. However, there

are some special tasks that cannot be delegated by using the collecting algorithm. The monitor users are introduced to deal with this kind of tasks. The monitor users are stored in the ***monitor user repository***. These special users are defined by the process designers and described in Section 3.2.6.

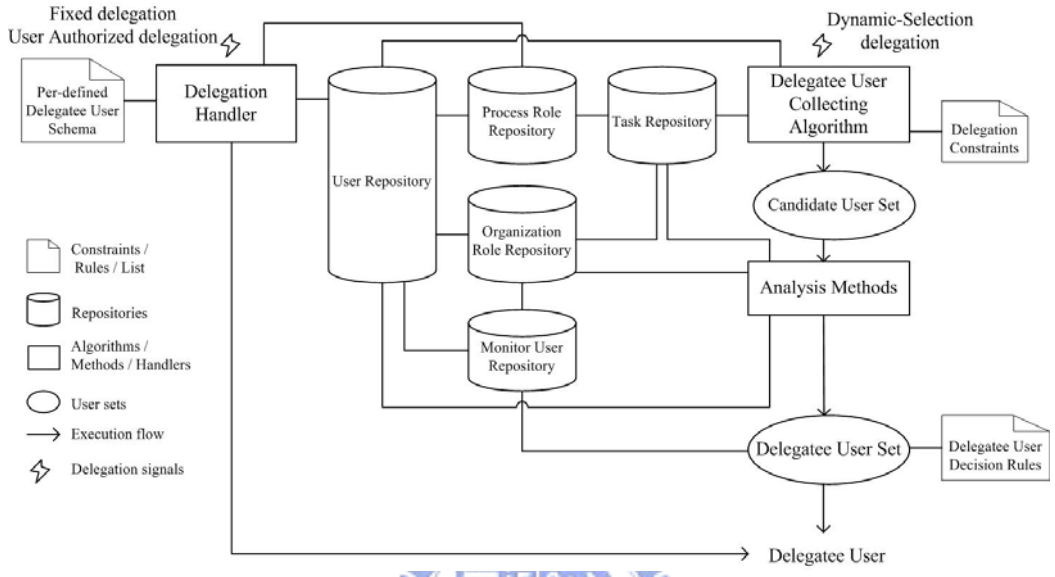


Fig.3.1 Task-based delegation framework

The ***delegation handler*** takes care of fixed delegation and user-authorized delegation. When a fixed delegation happens, the handler chooses a delegatee user from the ***predefined delegatee user schema***. The delegatee user of user-authorized delegation is decided by the delegator user. Thus, the function of the delegation handler in user-authorized delegation is to pass the delegated task to the delegatee user.

3.2.1. User and Task repositories

The user and task attributes are stored in the user and task repositories module respectively. Table 3.2 lists the attributes of tasks. The contents of attributes are defined by process designers at design phase. The attributes in Table3.1 are essential. The `p_id` represents the id of the process this task belongs to, and the `type` shows the type of the task. The `p_role` field stores the process role to which to this task is

assigned. The d_role field stores the id of the delegatee role which is created in delegation. Prior is the priority of a task in a process. The primary use of priority is that if there are lots of tasks in a user's task queue, he executes them from high priority task to low one. The m_flag is a boolean variable. If m_flag is true, it means that this task needs a monitor user to decide the delegatee user. Otherwise, if the task is delegated, the delegatee user is automatically chosen by the system from the delegatee role constructed by the selection algorithm. The state shows the current state of this task. There are three states of one task: ready, running, and submit. When a process is initiated, all the tasks are set to ready. The running state reflects the task is executing. After the completion of the task, the state is set to submit. There are three constraint fields needed by the system. The sod_flag is used to label if this task needs the separation of duty principle. There are three settings of this attribute: Stong_SoD, Weak_SoD, and none. The org_conflict_flag is a boolean variable used to mark the allowance of the organization role conflict. The max_d_cnt defines the maximum amount of delegating this task.

Task attributes
1. id: task id 2. p_id: process id 3. p_role: process role id 4. d_role: delegatee role id 5. type: type of this task 6. state: the task state 7. prior: priority 8. m_flag: monitor user flag 9. sod_flag: separation of duty constraint 10. org_conflict_flag: organization role constraint 11. max_d_cnt:the maximum amount of delegation

Table 3.2 Task attributes

The user attributes defined here are listed in Table 3.3. The information of user performing tasks is stored in the process role repository. The `org_role` is used for recording the user's job position in this enterprise. The `id` and `p_role` represent user's id and process role. The `role_cnt` stores the amount of assigned role(s). The `w_cnt` attribute increases one when a task is added into user's task queue. There are two user constraints. The `max_load` field contains the user's restriction of maximum workload, and the `max_role_assign` field gives the maximum amount of role assignment.

User attributes
1. `id`: user id 2. `p_role`: process roles 3. `org_role`: organization roles 4. `role_cnt`: the amount of assigned role(s) 5. `w_cnt`: current work count 6. `max_load`: maximum workload 7. `max_role_assign`: maximum amount role assignment

Table 3.3 User attributes

3.2.2. Organization role and process role repositories

As implied by the name, the organization role repository stores the structure of organization role, including job position, levels of organization roles…etc. Table 3.4 lists the attributes of roles. The `id` is the id of the role, and the `pf` field stores the performers of the role. The process and organization role attributes extend the role attributes. The `o_t` field in process role defines all the tasks belonging to this process role. The `org_level` field stores the level of the organization role.

Role attributes
1. `id`: the id of the role 2. `pf`: performers

Process role attributes
1. `o_t`: own tasks
Organization role attributes
1. `org_level`: the level of this role in organization

Table 3.4 Role/Process role/Organization role attributes

Normally, the organization role structure is represents as a tree structure. By using tree structure, organization role hierarchy is easily presented, and the relationships between roles are described clearly. The relation in tree structure is classed into two types: vertical and horizontal. The vertical relation gives a level-view of organization roles. The horizontal relation gives an up-and-down concept of organization roles. These relations between organization roles are defined below.

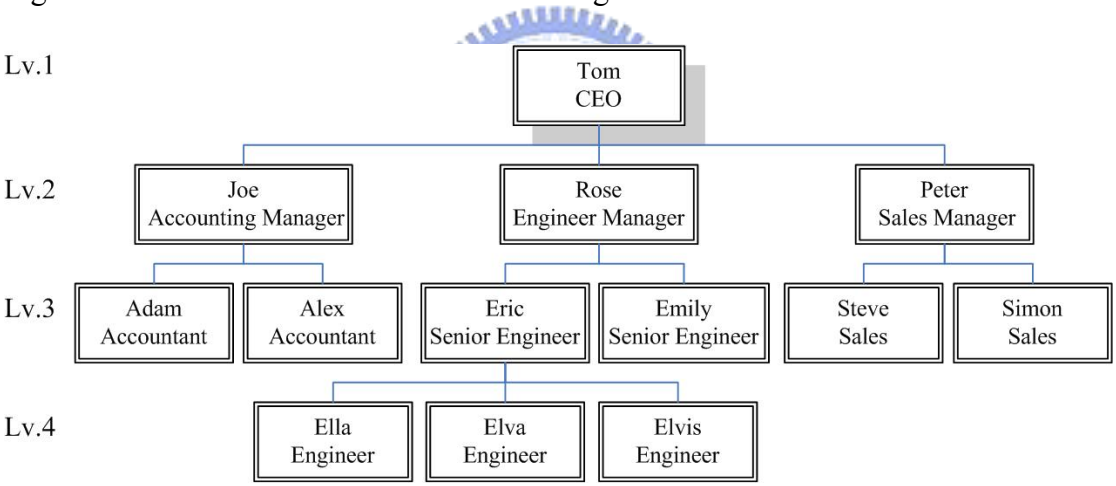


Fig.3.2 An example of the organization structure

The organization role repository stores the information of organization structures. Fig.3.2 gives an example of organization structure. It contains job position, the rank of each employee, and the hierarchies of organization roles…etc. The "***lower than***" relationship between two organization roles is defined in Definition 3.1. For example, Tom plays CEO role in company, and Joe is one of his successors. It says that Joe is "lower than" Tom and denotes "Joe $\angle$ Tom." Similarly, Ella is Rose's grand successor, and Ella $\angle$ Rose.

Definition 3.1 (lower than): A is B's successor, and B is on the A's path to the root of organization tree. It says that A's organization role is **lower than** B's. This relationship is denoted by "$\angle$".

Definition 3.2 defines another vertical relation in tree structure. If the level of one organization role is larger than another with different path to the root of organization tree, the relation between these two is called "***cross-lower than***." For example, Elva's level is lager than Emily, but Emily is not on Elva's path to the root. Therefore, Elva's organization role is cross-lower than Emily's, and it is denoted by "Elva $\measuredangle$ Emily".

Definition 3.2 (cross-lower than): A's level is lager than B's, and their path to the root of organization tree are not totally the same. It is said that A's organization role is **cross-lower than** B's. This relationship is denoted by "$\measuredangle$".

Two organization roles of two users have the same parents is called "***equal to***" defined in Definition 3.3. For instance, Joe and Rose are both managers. In the organization role tree, they have the same parent, CEO Tom. It says that Joe is "equal to" Rose and is denoted by "Joe $\equiv$ Rose."

Definition 3.3 (equal to): A and B have the same parent iff that A's organization role is **equal to** B's. This relationship is denoted by "$\equiv$".

The other horizontal relation "***cross- equal to***" is defined in Definition 3.4. The difference of "equal to" relation is that the "equal to" relation has the same parent, but the cross-equal to not. For example, Eric and Steve are at the same level, level 3, but their parents are different. This is called Eric is "cross-equal to" Steve. And it is denoted by "Eric $\fallingdotseq$ Steve".

Definition 3.4 (cross-equal to): A's level and B's level are the same with different parent iff that A's organization role is **cross-equal to** B's. This

relationship is denoted by "≒".

3.2.3. User-authorized delegation

Delegating the task by user himself is called a user-authorized delegation. When a user wants to delegate his task to a designate user, he asks the workflow engine to create a temporal process role called the ***delegatee role*** in the beginning. After creating the delegatee role, the system assigns this task to this temporal role. Then the designate user is allocated, and the delegatee role is assigned to him. Fig.3.3 is an example.

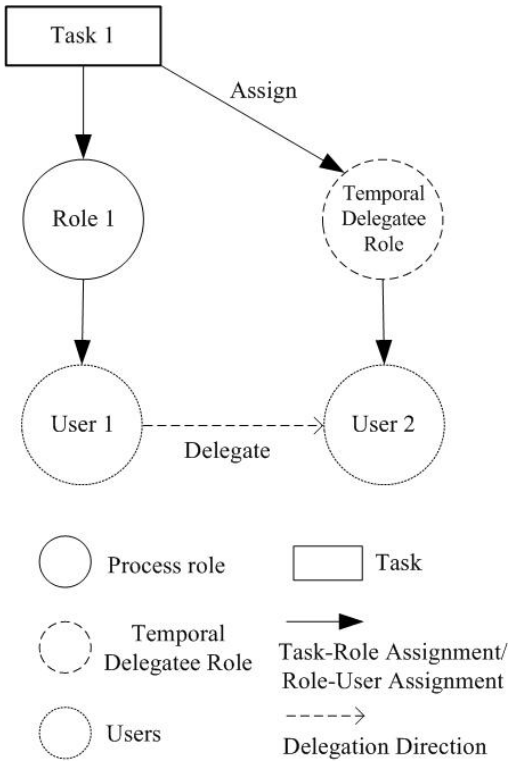


Fig.3.3 User-authorized delegation

Once User1 is too busy to perform Task1, he requires a delegation for Task1 from the system. When the workflow engine accepts such a request, it creates a temporal delegatee role immediately and assigns Task1 to this role. Then, the system assigns temporal delegatee role to User2 authorized by User1 directly. After these three steps, Task1 is delegated from User1 to User2. Obviously, this is a **user-authorized delegation**.

3.2.4. Fixed delegation

In case of interrupting the execution of a process, the process designer defines some delegatee users for each task to prevent the user unavailable problem. If someone cannot perform his task when executing this process, the system delegates this task to the predefined delegatee user. This kind of delegation is called **fixed delegation**. Fig.3.4 is an example.

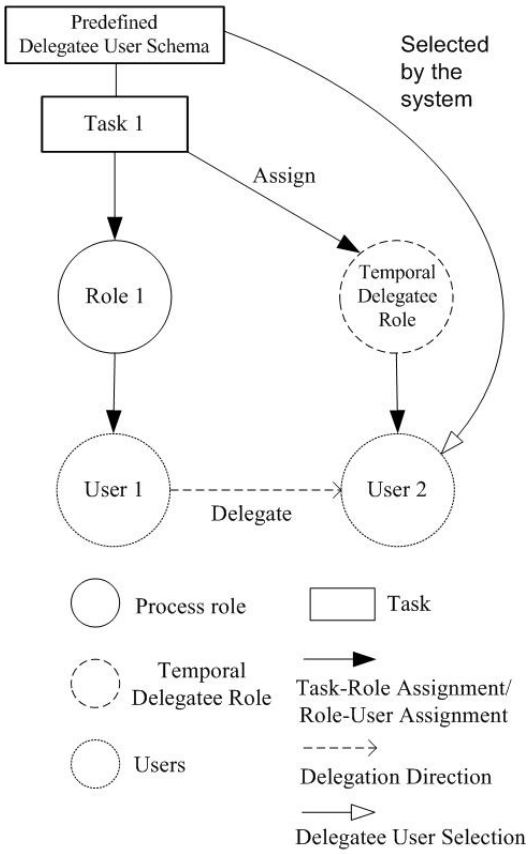


Fig.3.4 Fix delegation

In Fig.3.4, the delegatee user list is defined by the process designer for Task1. When User1 is unavailable to perform Task1, User2 is woken up in the delegatee user schema by the system to perform Task1. After waking User2 up, the temporal delegatee role is created, and Task1 is assigned to this temporal role. At last step, the temporal delegatee role is assigned to Uses2.

3.2.5. Dynamic-selection delegation

There are lots of unexpected events when executing workflow processes. For example, let the defined delegatee user in fixed delegation be accidentally absent from an important meeting. The system has to choose another delegatee user to do this task automatically. However, the process designer didn't assign the delegatee user for this meeting. Therefore, some rules or algorithms are needed to select an appropriate user as a substitute for the meeting dynamically. This kind of delegation is called the dynamic-selection delegation. Fig.3.5 shows the dynamic-selection delegation.

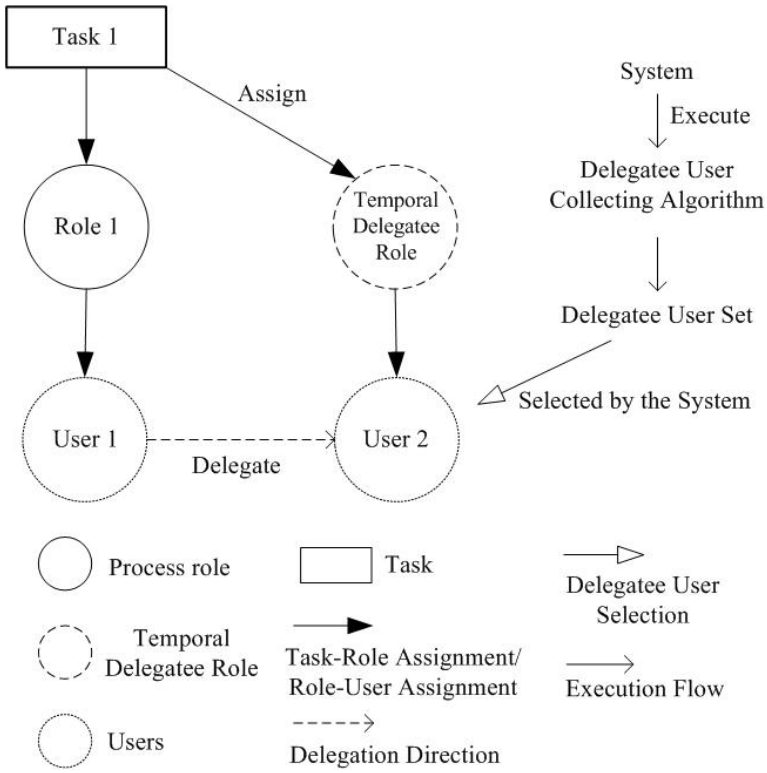


Fig.3.5 Dynamic-selection delegation

When a dynamic-selection delegation starts, the temporal delegatee role is created, and Task1 is assigned to this role. Then, the algorithm for collecting delegatee users is executed to generate the delegatee user set. After constructing the delegatee user set, the system follows the delegatee user decision rules to select the delegatee user. This approach is named as **dynamic-selection delegation**.

3.2.5.1. Delegatee user collecting algorithm

Delegatee user assignment is the most important part in delegation process. In some cases, numbers of user are collected by the system and one of them is selected by the monitor role. This case is called ***dynamic-selection delegation with monitor user*** described in Section 3.2.6. For the rest, the delegatee users are automatically chosen by the workflow engine. The delegatee user is selected from a list called ***delegatee user set*** shown in Fig.3.1. The delegatee user list is constructed in accordance with a collecting algorithm. This algorithm is executed by the workflow engine to select users into delegatee user set. Some user and task attributes are provided to determine whom to choose. These attributes are shown in Table 3.1 and Table 3.2. The collecting algorithm is presented in Algorithm 3.1.

Algorithm 3.1 (Delegatee user collecting algorithm)

Input:

The process role repository `P`

The delegated task `t`

Output:

A candidate user set `C`

DUCA:

```
01  Begin
02    For each process role p∈P, do
03      For each user u∈p.o_t do
04        If u.w_cnt < u.max_load ||
                  u.role_cnt < u.max_role_assign
05          Insert u to C
06        EndIf
```

```
07          EndFor
08        EndFor
09        If t.prior = HIGH
10          For each user u' ∈ C, do
11            For each task t'∈u'.p_role.o_t, do
12              If t'.prior = HIGH
13                Remove u' from C
14                Break
15              Endif
16            Endfor
17          Endfor
18        Endif
19    End
```

Delegatee users are selected from the process role repository. The workflow engine checks user's attributes to collect appropriate users as a list of delegatee users. In Step1 (Line02-08) of Algorithm3.1, the engine chooses the users whose current work counts are less than their maximum workload. Second, it selects the users whose assignment numbers are less than their limitation respectively. The function of the second constraint is to limit the numbers of delegations for each user.

The Step2 (Line09-18) checks the attributes, in Table3.1, of the delegated task. If the priority of delegated task is marked as "HIGH", the system selects users with no high priority tasks in task queue first from the result of Step1. The priority of task is defined by process designers. There are two levels of priority: "HIGH" and "NORMAL". This is used to give the high execution order to high priority tasks. If the priority is set to "NORMAL", there is no change for Step1.

3.2.6. Dynamic-selection delegation with monitor user

Sometimes delegation process invokes a special role type called a ***monitor role***. The primary job of a monitor role is to decide the final delegatee user. The user assigned the monitor role is called a ***monitor user***. Fig.3.6 shows how a monitor user participates in a dynamic delegation process.

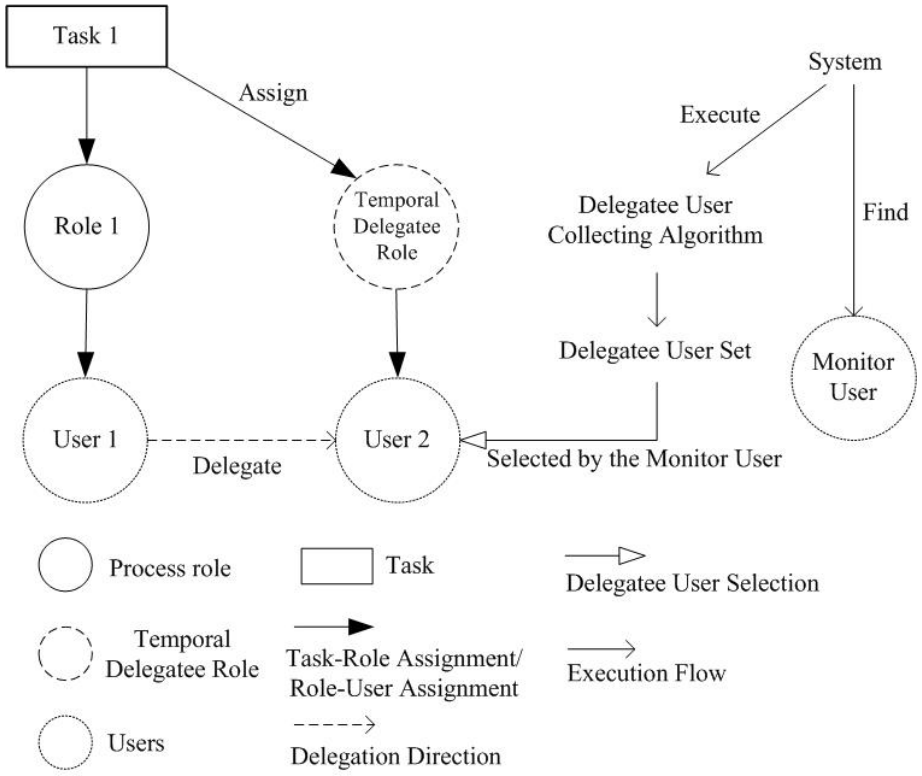


Fig.3.6 Dynamic-selection delegation with monitor user

When a dynamic-selection delegation starts, the workflow engine first checks the attributes of the task. If the monitor flag is labeled true, the engine creates a monitor role and assigns it to an appropriate user based on a pre-defined monitor role assignment policy. When the monitor user is decided, the selection is then executed to generate the delegatee user set. Unlike dynamic delegation without monitor user, the final delegatee user is picked from the delegatee user set by the monitor user.

The other function of monitor user is to appoint the delegatee user when no users are picked into the delegatee user set. During execution, an existing collecting

algorithm might find no user for the delegatee user set. When this situation happens, the system asks the monitor user to select the final delegatee user manually.

3.2.6.1. Monitor user constraint

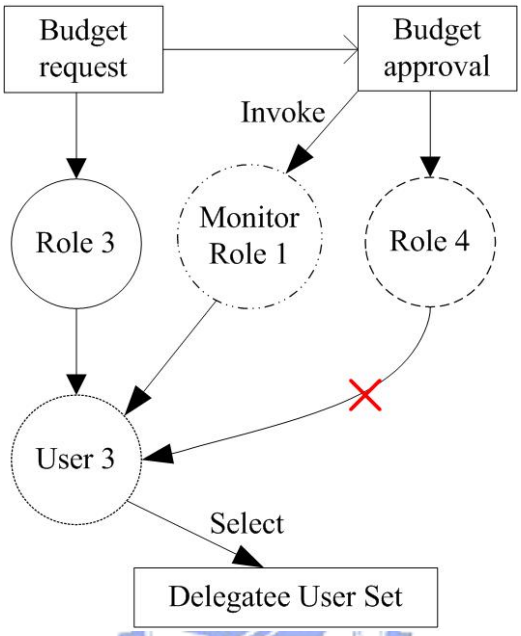


Fig.3.7 Monitor role SoD problem

There is a constraint for the monitor role assignment. The monitor user of the delegated task cannot be selected into the delegatee user set. This is used to prevent the ***separation of duty*** problem. Fig.3.7 presents an example of separation of duties. In the example, the budget request task is assigned to User3. Now a monitor role, Monitor Role1, is invoked when the budget approval task is delegated. And a temporal role, Role4, is created by the workflow engine for delegation. When the monitor role is assigned, User3 is expected to select a delegatee user from delegatee user set to play Role4. If User3 is in the delegatee user set at the same time, he can select himself to do this task. Thus, User3 can request the budget and approve it. This kind of problem is likely to be prevented in enterprises.

3.2.7. Repeated delegation

The repeated delegation is a phenomenon of the delegation. When the delegatee user is unable to perform the task delegating to him, this task must be delegated to another person. The delegation from one delegatee user to another is called a **repeated delegation.**

Fig.3.8 gives an example of a repeated delegation with the fixed delegation. After User1 delegates Task1 to User2, User2 is accidentally unavailable. Thus, he requests a fixed delegation to delegate Task1 out. When the system receives this request, it selects User3 from the predefined delegatee user schema as the delegatee user of User2 and assigns the temporal delegatee role to User3. This multiple delegation is called repeated delegation.

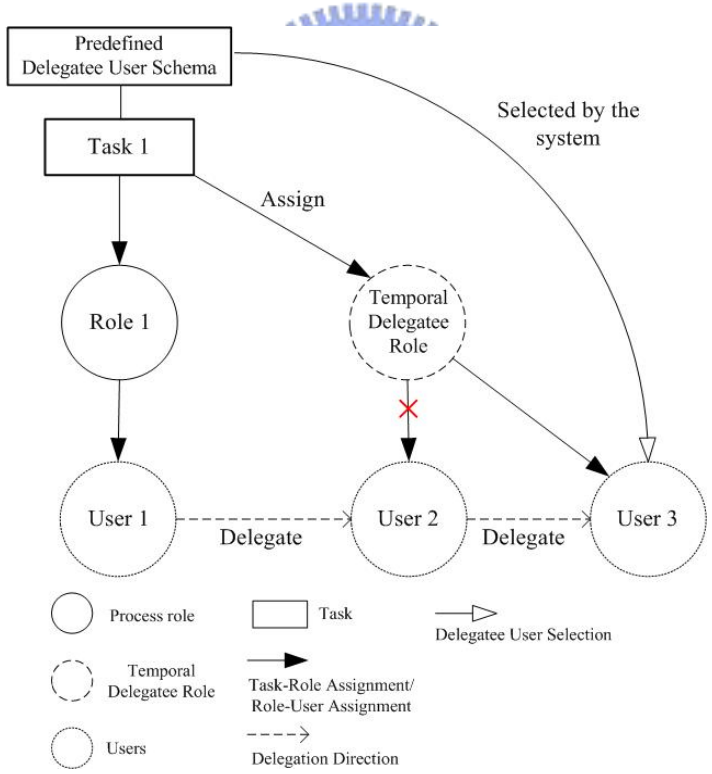


Fig.3.8 Repeated delegation with a predefined delegatee user schema

3.3. Revocation

With the temporal role in the delegation process, the revocation process only

requires only three steps. First, the engine deletes the role assignment between the delegatee role and the delegatee user. Second, task assignment between the delegated task and delegatee role is deleted. The temporal role is deleted at the last step, and the delegated task can be revoked back to the original user. For example, in Fig.3.1 the workflow engine first deletes the role assignment between the temporal delegatee role and User2. Then, it deletes the assignment between Task1 and temporal delegatee role. Last, the temporal delegatee role is delegated, and the revocation process is finished.

However, there is a problem in revocation: the delegated task is asked to be revoked while a delegatee user is performing it. Here presents a solution. According to the state of the delegated task, the result of the `submitted` state task is remained. And the result of the task in `running` state is discarded. The `ready` state task is revoked immediately after the revocation request is sent. The advantage of this solution is that the workflow engine doesn't have to halt the delegatee user while he/she is performing the delegated task. Instead, it only has to re-assign the task to delegator and re-sends the information before delegating to him.

3.4. Some discussions of the task-based delegation framework

There are some discussions in our framework. Section 3.4.1 describes the benefit of using two kinds of role repositories. The advantages of using a temporal delegatee role are mentioned in Section 3.4.2.

3.4.1. Process role repository and organization role repository

The primary reason to separate the roles into two types is to clearly distinguish roles in the processes and organization. When a new project is started, new processes are created for this new project. Some roles, such as project managers,

quality assurance engineers, and programmers…etc., are created for this project only. If using the organization role as the performers in the processes, the relationships between the new project-oriented roles and the existing organization roles must be defined. Moreover, these project-oriented roles and the relationships must be deleted after finishing the project. In this scenario, the system administrators may have to rapidly define and delete the relationships between project roles and organization roles when new projects are created and finished. It costs some maintenance efforts for the administrators.

Separating roles into the process and organization roles makes the administrators maintain these project-oriented roles easier. When each new process is created in a new project, a process role repository is allocated to this process. The administrators focus on managing the relationships of the new created roles in the process role repository. He doesn't have to define the relationship between the new roles and the organization roles. It saves some efforts to manage the roles in the processes and organization.

3.4.2. The advantages of the temporal delegatee role

The other distinguished feature of our framework is that using a temporal delegatee role instead of directly delegating the delegator user's role to the replaced person. The work can decrease some administrator's maintenance efforts too.

In our framework, users perform tasks through the process roles. A process role may be assigned multiple tasks. If delegating the delegator role to the delegatee user directly, the delegatee user can perform all the tasks assigned to the delegator role. Therefore, in other approaches without the temporal delegatee roles, the administrators have to define extra delegation constraints between the delegator role and all the tasks belonging to the delegator roles. In our framework, the

administrator manages the assignment between the delegated task and the temporal delegatee role. He/she doesn't have to define the extra delegation constraints between the delegator role and all the tasks belonging to the delegator role. In comparison to previous approaches in managing the delegations, our framework saves some management efforts of defining the extra delegation constraints.

4. Analysis Methods for Task-Based Delegation Framework

Our framework supports delegation handling in workflow systems. Several issues are derived from our framework. The issues and corresponding solutions are discussed in this Chapter. First, user assignment loop problem may occur in the repeated delegation, and this problem is presented in Section 4.1 In section 4.2, the separation of duty principle in our framework is introduced. Organization role conflict problem in Section 4.3 gives a new issue between organization role and process role.

4.1. User Assignment Loop

If a delegated task is delegated to its delegator user, user assignment loop problem happens. Normally, a user doesn't delegate his task to himself. Therefore, the user assignment loop only occurs in a repeated delegation.

Fig.4.1 presents an example of user assignment loop problem. Task1 is assigned to User1 through Role1. Assume that User1 delegates his task, Task1, to User2, and delegates Task1 to User3 due to an emergency task. Furthermore, User3 is accidentally unavailable, after he accepts the delegated task. If Task1 is automatically now assigned to User1 by the workflow engine, the user assignment loop problem takes place under this scenario.

To prevent this problem, the delegator users are needed to be recorded. There are two ways to record the delegator users of a delegated task. One is ***marking method***, and the other one is ***tracing method***. The two methods are introduced in following sections.

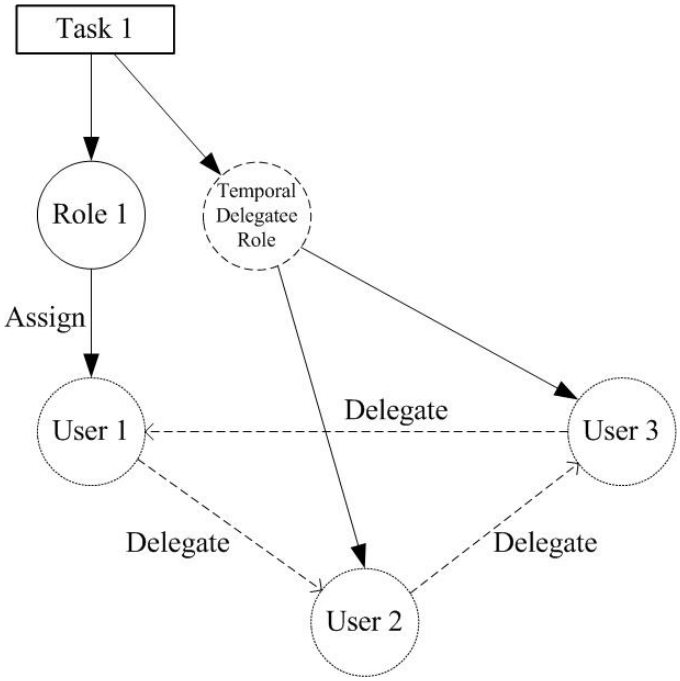


Fig.4.1 User assignment loop problem

4.1.1. The Marking Method for User Assignment Loop Problem

In marking method, the id of the delegated task is stored in a table named ***d_task*** which is associated with the delegator user as an extended attribute. When a delegation happens, the delegated task id is put into the d_task table associated with the delegator user. Moreover, while the repeated delegation occurs, the delegated task id is recorded in all delegator users' d_task tables respectively. As described in Chapter 3, when the original user revokes his delegated task or the task completes, the delegated task id is kick off from those user who has it in his/her d_task.

Algorithm 4.1 AALMm (Analysis of Assignment Loop with Marking Method) describes the marking method. This algorithm removes users who might cause the user assignment loop from the user set generated by the collecting algorithm and analyzed by other analysis methods. Each candidate user's d_tasks table is checked if there is a recorded task id equaled to the delegated one. In case it is true, this user is deleted from the delegatee user set.

Algorithm 4.1 (Analysis of Assignment Loop with Marking Method)

Input:

A user set C

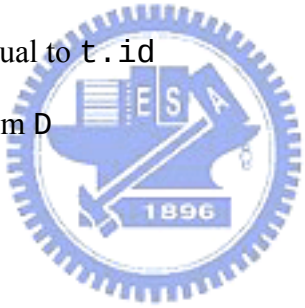
The delegated task t

Output:

A delegatee user set D without the user assignment loop problem

AALMm

```
01  Begin
02     D = C
03     For each user u∈D, do
04        For each task id d_id∈u.d_task do
05           If the d_id is equal to t.id
06              Remove u from D
07              Break
08           EndIf
09        EndFor
10     EndFor
11  End
```



4.1.2. Tracing Method for Use Assignment Loop Problem

In tracing method, the id of the delegator user is stored in a table, named ***r_task,*** which is associated with the temporal delegatee role. When a delegation happens, the delegator user's id is put into the r_task table associated with the temporal delegatee role. Furthermore, while the repeated delegation occurs, all ids of the delegator users are recorded in the r_task table. As described in Chapter 3, when the original user revokes his delegated task or the task completes, the temporal

delegatee role is removed, and the r_task table on this role is also deleted.

Algorithm 4.1 AALTm (Analysis of Assignment Loop with Tracing Method) describes the tracing method. The purpose of this algorithm is the same with the algorithm AALMm. The r_tasks table binding on the temporal delegatee role related to the delegated task is checked.

Algorithm 4.2 (Analysis of Assignment Loop with Tracing Method)

Input:

A user set `C`

The delegated task `t`

Output:

A delegatee user set `D` without the user assignment loop problem

AALTm

```
01  Begin
02     D = C – t.d_role.r_task
03  End
```

Algorithm 4.2 Tracing method algorithm for user assignment loop problem

The difference between the two methods is where to log the information of delegator users. The marking method records the information of delegator users in the user attribute, but the tracing method uses the delegatee role to store the information of delegator users. Generally speaking, the tracing method is faster than marking method in checking user assignment loop for one delegated task. This is because that tracing method only verifies users in the delegator user table. But marking method has to go through all the users in the delegatee user set to check whether the delegated task ID is in their d_tasks attributes. Normally, delegator users in multi-delegations are not more than five people, but the numbers of users in delegatee user set are. Thus, executing tracing method is usually faster than marking

method in checking loop problem for one task.

The advantage of marking method is that it is easy to know what tasks are delegated by users. The marking method is better used before the selection algorithm to help the system to collect delegatee user set. When picking up delegatee users, the engine checks the d_tasks in users. If the delegated task ID is appeared in this attribute, this user is not chosen into delegatee user set. The tracing method can also be used before executing selection algorithm. But in comparison of executing time, the marking method is faster then tracing. The reason is that tracing method traces delegator user table every time before a user is picked. The marking method only has to compare the delegated ID with the d_tasks attribute. Normally, the size of d_tasks attribute is smaller than the delegator user table. Thus, the marking method is better than the tracing method to put at the beginning of selection algorithm.

4.2. Separation of Duty (SoD)

The separation of duty (SoD) is a security principle. When two or more tasks in the same process are performed by one user, SoD problem might occur.

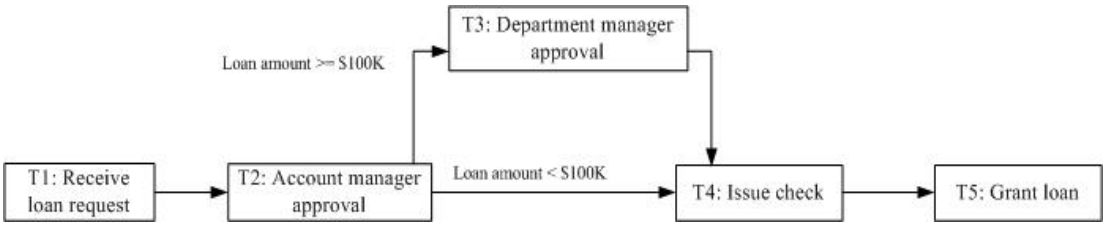


Fig.4.2 A loan request process

Fig.4.3 presents a loan process. All the tasks are assigned to different users through different roles. Suppose that T2 is assigned to User2 through a process role named account manager. T3 is assigned to User3 through another process role named department manager. Now User3 is unexpectedly unavailable, and the task approval task is needed to be executed. If T3 is automatically delegated by the

engine and is assigned to User2, thus, User2 can easily approve any amount of loans.

To state the SoD principle in detail, tasks are divided into two types: ***decision task*** and ***general task***. The definition is described below.

> **Definition 4.1 (Decision Task):** A task within the workflow where a single thread of control makes a decision upon which branch to take when encountered with at least one alternative workflow branches. This kind of task is called decision task.

In definition 4.1, decision tasks decide the execution path of a process. Using Fig 3.5 as an example, T2 and T3 are loan approval tasks. If managers in T2 or T3 disagree with loan requests, these loan requests are dropped. Obviously, T2 and T3 can effect the execution of process. Therefore, T2 and T3 are decision tasks. Check points and decision making tasks in processes are usually classed as decision tasks. From workflow viewpoint, XOR-split nodes are classed as decision tasks. T1 and T5 in Fig.4.3 are general tasks. These kinds of tasks are routines or those that never change process execution path.

> **Definition 4.2 (General Task):** All Tasks which are not classed as decision tasks are general tasks.

When constructing the delegatee user set, the workflow engine shall check the picked users' SoD constraints. If some ones violate these constraints, they cannot be selected into delegatee user set. There are two types of separation of duty constraints: ***strong SoD*** and ***weak SoD***.

> **Definition 4.3 (Strong SoD):** Any two tasks in one process, including decision tasks and general tasks, are not performed by the same person is called Strong Separation of Duty.

That the delegations both decision and general tasks cannot violate separation of duty constraints is called strong SoD. Limiting decision tasks not to violate separation of duty constraints only is called weak SoD. The benefit of strong SoD constraint is to prevent that a task in highly secure processes, such as military processes, are performed by the same user. Usually, the enterprise processes only require weak SoD constraints.

Definition 4.4 (Weak SoD): That the decision tasks in one process only are restricted to be performed by different users is called Weak Separation of Duty.

4.2.1. Analysis method of Weak SoD

The weak SoD principle does not allow two decision tasks to be executed by one person, since one can easily change the execution path or result when holding two or more decision tasks in a process. Algorithm 4.3 gives a method filtering the candidate user set to assure that the delegatee user set violates no weak SoD principle.

Algorithm 4.3 (Analysis of Weak SoD Principle)

Input:

The Task repository `T`

A user set `C`

Output:

A user set `D`

AWSDP

00 `D = C`

01 For each task `t` ∈ `T`, do

02 If `t.type` = decision

```
03        If t.state = submit
04          If  t.d_role != NULL
05            Remove t.d_role.pf from D
06          Else
07            Remove t.p_role.pf from D
08          Endif
09        Else
10          Remove t.d_role.pf from D
11          Remove t.p_role.pf from D
12        Endif
13      Endif
14  Endfor
```

Algorithm 4.3, AWSDP (Analysis of Weak SoD Principle), is executed when the delegated tasks is typed "decision" and under the constraint of weak SoD principle. In AWSDP, all the users which potentially perform any decision tasks in the process are removed from the result set. To achieve the goal, the execution states of tasks are considered. For a submitted task, if the task is not delegated, the original performer is considered, otherwise only the delegatee user is considered. To a ready or a running task, because revocation might take place, both the original performer and the delegatee user (if any) must be considered.

4.2.2. Analysis method of Strong SoD

Strong SoD restrict that all tasks in one process must be performed by different users. Since the collecting algorithm collects users only from the process role repository, the collecting algorithm find no delegatee users under strong SoD. For this reason, when delegating one task with strong SoD constraint, a new collecting

algorithm is brought up for fitting strong SoD by using organization role repository.

When one task is set to strong SoD constraint, Algorithm 3.1 is not used to select the delegatee users of this task in order not to against this constraint. A new selection algorithm is used to collect candidate users from the organization role repository. Algorithm 4.4 presents this method.

Algorithm 4.4 (Strong SoD User Collecting Algorithm)

Input:

The User repository U

The Process role repository P

The delegator user d

The delegated task t

Output:

The delegatee user set D

SSUCA

```
00  Begin
01  For each user u∈U, do
02    If d.org_role ∠ u.org_role || d.org_role ≡ u.org_role
03      Insert u to D
04    Endif
05  Endfor
06  For each user u'∈D, do
07    If u'.w_cnt+1 > u'.max_load || u'∈P
08      Remove u' from D
09    Endif
10  Endfor
```

```
11   D = AUALTm(D, t)
12   End
```

First, the delegator user's supervisors in organization are collected. Moreover, users with the equal organization level to the delegator user are picked. Second, the system deletes users whose work count is out of their maximum workload or process role appeared in the process repository of delegated task. This action is used to achieve the main concept of separation of duty. Third, the user assignment loop checking algorithm in Section 4.1 is executed to prevent the loop problem. After executing this algorithm, a delegatee user set based on organization role is constructed without against the strong SoD principle.

4.3. Organization Role Conflict

The delegatee role may be assigned to a user whose organization role is lower than the delegator user. A problem is raised that some decision tasks of a supervisor may be performed by his subordinate after delegation. Fig.4.5 presents an example of organization role assignment and process role assignment. In this organization, Tom is Rose's supervisor, and Elvis is Rose's subordinate. In one project of this organization, the project manager role is played by Tom, and the QA and RD roles are respectively assigned to Rose and Elvis. It is noticed that the roles are hierarchical in organization role structure, but those in process roles are not totally hierarchical. The project manager, QA, and RD roles are project-oriented role. Therefore, they belong to process roles.

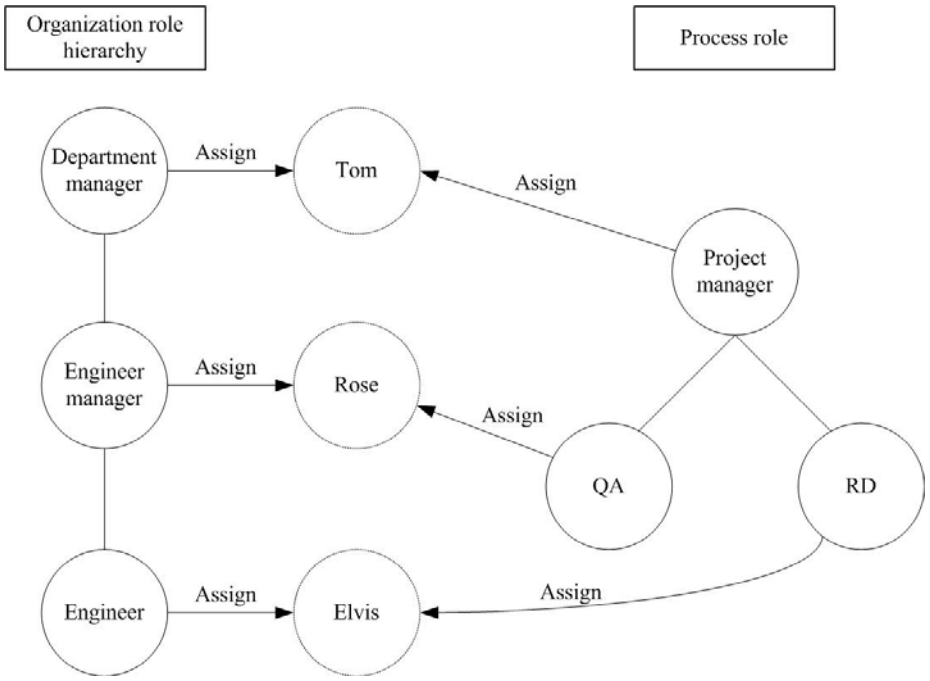


Fig.4.3 Organization role assignment and process role assignment

Fig.4.6 shows a process of asking leave of absence. If someone in this project cannot attend scheduled meetings, he starts this process. He sends the form of asking leave of absence to the project manager to approve this request. In this scenario, approving absence requests task is assigned to the project manager, Tom. Assume that someday Tom is accidentally unavailable, and the workflow engine automatically delegates the approval task to Elvis. A strange situation is brought about that Rose's absence request is approved by Elvis, i.e. the supervisor's absence request is approved by his subordinate.

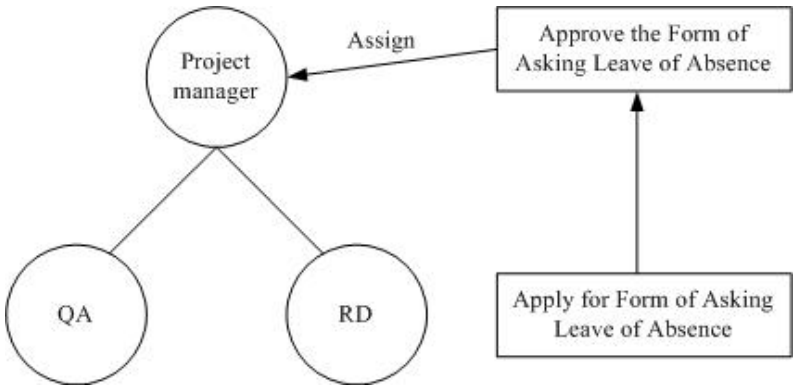


Fig.4.4 Task assignment with process role

The primary reason to cause organization conflict is that tasks are assigned to the

process role instead of being assigned to the organization role. Section 3 has described the advantages of separating role into these two types. However, the implicit problem is coming up with this special assignment of role, tasks, and users. From company ethics viewpoint, an engineer cannot approve the manager's absence request. For not to violate such company ethics problem, the delegatee user's organization role is needed to be checked when collecting the candidate users. Considering the type of task, the execution path and result is not changed when executing general task. This means that no organization role conflict in delegating general tasks. But the decision type task may change. Thus, in order not to cause the organization role conflict problem, decision tasks are not allowed to be delegated to users with lower organization roles. Algorithm 4.5 gives this solution of checking the organization role conflict.

Algorithm 4.5 (Organization Role Conflict Checking Algorithm)

Input:

A user set C

The delegator user d

Output:

A user set D

ORCCA

00 Begin

01 D = C

02 For each $u \in C$, do

03 If $u.org\_role \angle d.org\_role \parallel u.org\_role \triangleleft d.org\_role$

04 Remove u from D

05 Endif

06 Endfor
07 End

The users whose organization roles are lower then the delegator user are deleted. This step ensures no subordinate of the delegator user is collected into the delegatee user set. It prevents the primary reason to cause organization role conflicts.

There are lots of departments in modern enterprises. And the process in this environment usually requires inter-department cooperation. When delegations happen in such environment, the organization role conflict problem may apparent much easier. This conflict is called ***inter-department organization role conflict***. Using Fig.4.3 as an example, Rose is the supervisor of engineer department, and Peter is sales department manager. There is a project to be carried out by the two departments to carry out, but one collaborate process is created. In this project, Rose is the project leader, and Peter is a member of this project. In process view, Peter's absence request is approved by Rose. Rose is accidentally unavailable one day, and project leader role is delegated to Eric. Peter's absence request is approved by Eric in this scenario. This breaks the company ethics, and inter-department organization role conflict occurrs. This problem is viewed as a special case in organization role conflict. Decision type tasks bring up this conflict. The solution of this problem is similar to organization conflict. In order not to let the decision type task delegate to the subordinate of other departments, the checking condition is modified to check the level of organization role. Algorithm 4.6 presents inter-department organization role conflict checking algorithm.

Algorithm 4.6 (Inter-Department Organization Role Conflict Checking Algorithm)
Input:

A user set C

The delegator user d

Output:

A user set D

IDORCCA

```
00  Begin
01  For each u∈C, do
02    If u.org_role.org_level ≦ d.org_role.org_level
03      Remove u from C
04    Endif
05  Endfor
06  End
```

Algorithm 4.6 Inter-department organization role conflict checking algorithm

Line02 changes the checking condition of Algorithm 4.5. In Algorithm 4.5, the checking condition is organization role. If the delegatee user's organization role is lower than the delegator user's, he is deleted from delegatee user set. In Algorithm 4.6, the checking condition is the organization level. If the level of delegatee user is smaller than the delegator user's, he is erased from the delegatee user set.

4.4. Integration of the analysis methods

After describing the analysis methods, we use a flow to integrate these methods. Fig.4.5 shows the relationships between these three analysis methods.

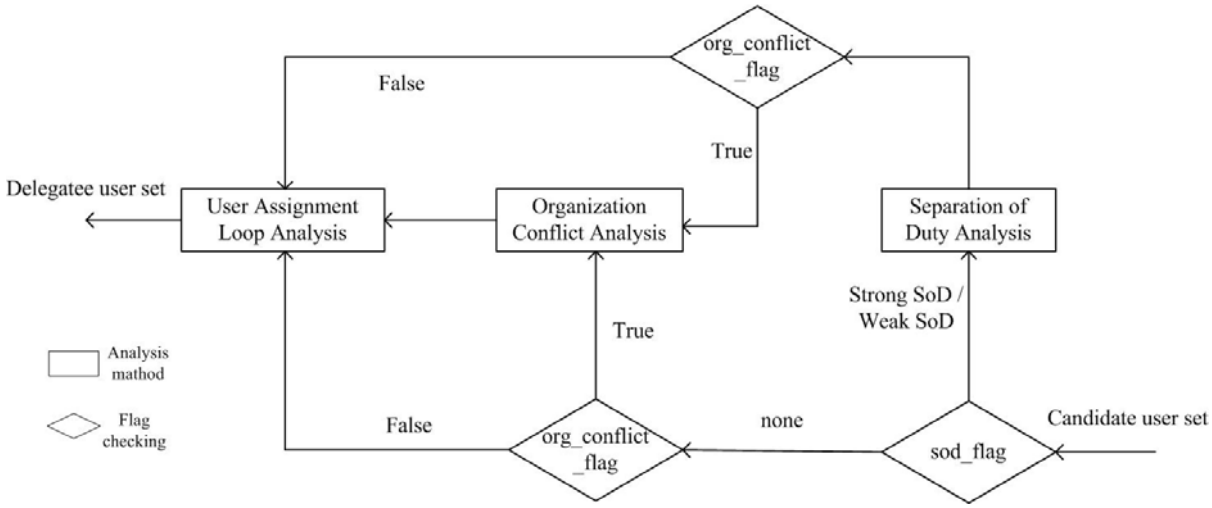


Fig.4.5 The integration of analysis methods

In Fig.4.5, after generating the candidate user set, the system checks the `sod_flag` of the delegated task. If this flag is labeled `Strong_SoD` or `Weak_SoD`, this means that the task must not violate the separation of duty principle. Therefore, the candidate user set is passed into the analysis methods of SoD. Otherwise, this set is passed to the organization conflict analysis stage. When the user set from SoD analysis or the collecting algorithm is passed out, the system checks the `org_conflict_flag` to see whether the delegated task must not violate the organization role conflict. If this flag is true, the candidate users are delivered to the organization conflict analysis method. After checking the two flags of the delegated task, the candidate user set may be filtered by the SoD and organization role conflict analysis methods. Finally, the user assignment loop problem must be checked and the delegatee user set is constructed. This is how the three analysis methods to be integrated.

5. Conclusion and future works

In this paper, we introduce three types of delegation by observing the behaviors of delegation. The delegatee user is selected by delegator user in the user-authorized delegation. The system chooses the delegatee user following the predefine delegatee user schema in fixed delegation. A delegatee user collecting algorithm is executed by the system to construct the delegatee user set for deciding the delegatee user in the dynamic-selection delegation. Two extensions of the delegations are also introduced. The monitor user selects the delegatee user from the delegatee user set constructed by the dynamic-selection delegation. The repeated delegation is composed of at least two delegations. A task-based delegation framework is proposed to handle the delegation described above.

With three analysis methods, the problems and conflicts are given appropriate solutions respectively. In the future, one of our works is to implement this framework to existing workflow management systems.

Reference

[1] Sejong Oh, Seog Park, “Task-role-based access control model,” Information Systems, Volume 28, Issue 6, 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,” Computer Security Foundations Workshop, 1997.

[4] Axel Kern, Andreas Schaad, Jonathan Moffett, “Advanced Features for Enterprise-Wide Role-Based Access Control,” Computer Security Applications Conference, 2002

[5] Yang, L. Ege, R. K.Ezenwoye, O. Kharma, Q., “A Role-Based Access Control Model for Information Mediation,” Information Reuse and Integration, 2004.

[6] Basit Shafiq, Arjmand Samuel, Halima Ghafoor, “A GTRBAC Based System for Dynamic Workflow Composition and Management,” Object-Oriented Real-Time Distributed Computing, 2005

[7] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor, “Generalized Temporal Role-Based Access Control Model,” IEEE Transaction on Knowledge and Data Engineering, Vol. 17, No. 1, January 2005, pages. 4 - 23.

[8] Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N., “Modeling security requirements through ownership, permission and delegation,” Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference

[9] Gail-Joon Ahn, Badrinath Mohan, “Secure Information Sharing Using Role-based Delegation,” Information Technology: Coding and Computing, 2004

[10] Ezedin Barka, Ravi Sandhu, “Role-Based Delegation Model/Hierarchical Roles

(RBDM1),” Computer Security Applications Conference, 2004.

[11] Jacques Wainer, Akhil Kumar, “A Fine-grained, Controllable, User-to-User Delegation Method in RBAC,” ACM symposium on Access control models and technologies SACMAT’05

[12] Jason Crampton,“A Reference Monitor for Workflow Systems with Constrained Task Execution,” ACM symposium on Access control models and technologies SACMAT’05.

[13] He Wang, Sylvia L. Osborn, “Delegation in the Role Graph Model,” ACM symposium on Access control models and technologies SACMAT’06

[14] Geethakumari, G. Negi, A. Sastry, V.N.,” Dynamic Delegation Approach for Access Control in Grids,” e-Science and Grid Computing, 2005

[15] James B. D. Joshi, Elisa Bertino, “Fine-grained role-based delegation in presence of the hybrid role hierarchy,” ACM symposium on Access control models and technologies SACMAT '06.

[16] Matunda Nyanchama, Sylvia Osborn, “The Role Graph Model and Conflict of Interest, “ACM Transactions on Information and System Security, Vol. 2, No. 1, 1999

[17] Vijayalakshmi Atluri, Janice Warner, “Supporting conditional delegation in secure workflow management systems,” ACM symposium on Access control models and technologies SACMAT '05

[18] Shih-Chien Chou, “Dynamic adaptation to object state change in an information flow control model,” Information and Software Technology, 2004

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

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

[21] Simon, R.T.; Zurko, M.E., Separation of duty in role-based environments, IEEE Computer Security Foundations Workshop, 1997.