Available online at www.sciencedirect.com

The Journal of Systems and Software 73 (2004) 375–387

www.elsevier.com/locate/jss

Role-based authorizations for workflow systems in support of task-based separation of duty

Duen-Ren Liu *, Mei-Yu Wu, Shu-Teng Lee

Institute of Information Management, National Chiao Tung University, 1001 Ta Hseuh Road, Hsinchu 300, Taiwan

Received 23 July 2002; received in revised form 15 April 2003; accepted 23 April 2003
Available online 5 December 2003

Abstract

Role-based authorizations for assigning tasks of workflows to roles/users are crucial to security management in workflow management systems. The authorizations must enforce separation of duty (SoD) constraints to prevent fraud and errors. This work analyzes and defines several duty-conflict relationships among tasks, and designs authorization rules to enforce SoD constraints based on the analysis. A novel authorization model that incorporates authorization rules is then proposed to support the planning of assigning tasks to roles/users, and the run-time activation of tasks. Different from existing work, the proposed authorization model considers the AND/XOR split structures of workflows and execution dependency among tasks to enforce separation of duties in assigning tasks to roles/users. A prototype system is developed to realize the effectiveness of the proposed authorization model.
© 2003 Elsevier Inc. All rights reserved.

Keywords: Workflow system; Separation of duty; Role-based access control; Authorization management

1. Introduction

As effective process management tools, workflow management systems (WfMS) enable a business to analyze, simulate, design, enact, control and monitor its business processes (Cichocki et al., 1998; Georgakopoulos et al., 1995; Kappel et al., 1995; WfMC, 1995). A workflow consists of a set of tasks, and the ordering of tasks, or control flow dependencies, that specify the order of execution among the tasks. Workflows commonly process sensitive business information. Adequate access control mechanisms are needed to protect workflow-related sensitive information from insecure access. Consequently, security policies with appropriate authorization mechanisms are required to ensure that tasks are performed by authorized users.

Role-based access control (RBAC) (Barkley, 1995; Ferraiolo et al., 1995; Sandhu et al., 1996) has become a widely accepted access control mechanism for security management. The authorizations for granting access permissions are based on the concept of roles. A role represents a duty or a job position (title) with the authority and responsibility to perform certain job functions within an organization. In RBAC, access permission is associated with roles, and users are assigned to appropriate roles. By assigning users to roles, rather than permission, RBAC reduces the complexity of the access control.

Moreover, authorization constraints or policies must be defined to enforce the legal assignment of access privileges to roles and roles to users, and thereby ensure secure access. Separation of duty (SoD) (Ferraiolo et al., 1995; Gligor et al., 1998; Nash and Poland, 1990; Sandhu et al., 1996; Simon and Zurko, 1997) is a security principle to spread the responsibility and authority for a complex action or task over different users or roles, to prevent fraud and errors. Under this principle, conflicting (mutually exclusive) tasks are executed by different roles/users.

Schier (1998) presented a role and task-based security model. Although authorization rules for SoD have been designed, they are merely derived from SoD in RBAC. Moreover, Lee et al. (2001) implemented RBAC in computer supported collaborative writing (CSCWriting).

* Corresponding author. Tel.: +886-35712121x57405; fax: +886-35723792.
E-mail address: dliu@iim.nctu.edu.tw (D.-R. Liu).

They focused on integrating distributed version management and RBAC for CSCWriting environments. Accordingly, they did not consider the issues of role-based authorizations and SoD.

Several researchers (Ahn et al., 2002; Atluri et al., 2001; Atluri and Huang, 1996; Bertino et al., 1999; Botha and Eloff, 2001; Huang and Atluri, 1999; Kang et al., 1999; Kang et al., 2001) have addressed RBAC and authorization management in workflow systems. The major issues concern the design of role-based authorization mechanisms in support of SoD. Bertino et al. (1999) elucidated a flexible model to specify and enforce authorization constraints in WfMS. Ahn et al. (2002) developed an system architecture to enforce RBAC in Web-based WfMS. Atluri and Huang (1996) proposed a workflow authorization model (WAM). Huang and Atluri (1999) presented a secure Web-based WfMS. Botha and Eloff (2001) considered conflicting tasks, conflicting users and the access history of documents to support the SoD requirements for workflow systems. Furthermore, as the demand for the globalization of business increases, access control mechanisms and security models of inter-organizational workflows are presented (Atluri et al., 2001; Kang et al., 2001).

Most of the above literatures have addressed SoD constraints in role-based authorizations of workflows. However, they do not consider SoD constraints in relation to various duty-conflict relationships and execution dependencies among tasks. Moreover, they do not consider authorization planning for assigning users/roles to workflow tasks. Although Bertino et al. (1999) proposed algorithms for authorization planning to assign roles/users to workflow tasks, their algorithms mainly determine valid assignments by consistency-checking authorization constraints expressed in logical language, and making deductive inferences. They do not consider the variations of SoD that arise from various duty relationships among tasks.

This work presents a novel analysis and defines various duty-conflict relationships among tasks from the aspect of how enterprises set up tasks. Moreover, this work designs authorization rules for SoD, based on the defined duty-conflict relationships and execution dependencies of workflow tasks. An authorization model, including a planning phase and a run-time phase, of assigning roles/users to workflow tasks is developed. The proposed model enforces SoD by taking into account the AND/XOR split structure of workflows and execution dependency among tasks. Furthermore, a prototype system that can conduct authorization management in task-based workflow environments is implemented and evaluated. An analyzed procurement process is deployed in the system to demonstrate the proposed authorization management of workflow tasks.

The rest of this paper is organized as follows. Section 2 describes the basic concepts. Section 3 analyzes and defines various duty-conflict relationships among tasks, and presents authorization rules for SoD. Section 4 illustrates the proposed algorithms for planning role-task and user-role-task assignments. Section 5 elucidates the implementation and evaluation of the system. Section 6 compares related work with our proposed work. Conclusions and areas for future research are finally made in Section 7.

2. Basic concepts

A workflow consists of a set of tasks, and their order of execution, according to control flow dependencies (Georgakopoulos et al., 1995; WfMC, 1995). The various tasks in a workflow are typically performed by multiple collaborating users/roles in an organization. A role implicitly defines a job position and its corresponding authority to perform a set of tasks. Each task is assigned to one or more roles. Users are assigned appropriate roles based on their capabilities. A role can be assigned to one or several users. Moreover, roles are partially ordered by organizational position within the organization. For two roles R_i and R_j, $R_i > R_j$, if the position of R_i is higher than the position of R_j in the organization.

WfMS generally include process definition tools and workflow engines (WfMC, 1995). A process definition tool supports facilities that define a workflow (process definition) during the period of design, while a workflow engine governs the run-time enactment of workflow according to the process definition. Tasks are executed according to the control flow dependencies in a workflow. During run-time, a single execution of a workflow (process) is called a *workflow (process) instance*, while the execution of a task (activity) within a workflow instance is called a *task instance* (WfMC, 1995). Each instance represents a separate execution of the process, and has its own associated process instance data. Here, an execution/activation of a workflow/task represents an enactment of a workflow/task instance.

[Execution dependency] Two tasks T_i and T_j are execution-dependent tasks, denoted as $T_i \sim T_j$, if they are correlated, such that the execution (processing) of one task (T_i) depends on the execution (processing) of the other task (T_j). $T_i \times T_j$ indicates that tasks T_i and T_j are not execution-dependent.

Execution dependency among tasks can generally be derived from the accessed data objects. If task T_i accesses data objects that are created/modified by task T_j, then the execution of T_i depends on the execution of T_j. For instance, a simple procurement workflow includes two tasks, "purchase" and "verify" which are execution-dependent in the process of purchasing items (workflow instance). The authorization of the "verify" task on a computer (purchased item) must consider the authorization of the "purchase" task on a computer.

SoD (Ferraiolo et al., 1995; Gligor et al., 1998; Nash and Poland, 1990; Sandhu et al., 1996; Simon and Zurko, 1997) is a security principle that spreads the responsibility and authority for a complex action or task over various users or roles, to prevent fraud or error. In general, two strategies can be used to enforce the SoD. Static SoD prevents conflicting (mutually exclusive) roles or operations from being assigned to the same user. Dynamic SoD provides flexibility by allowing conflicting roles or operations be assigned to a user, but the user must not activate them at the same time. Notably, static SoD constraints are imposed during design time, while dynamic SoD constraints are imposed during run-time. Although SoD has been addressed, most work does not consider SoD with respect to various duty-conflict relationships and execution dependencies among tasks.

3. Task-based separation of duty

3.1. Analysis of duty-conflict relationships

A task defines a set of task-related privileges to be assigned to roles or users. Assigning a task to a role or a user gives the role or the user the duty to perform the task; the duty is then called a task-duty. The planning of tasks not only defines task-privileges, but also implicitly defines task-duties of roles/users. Some duty relationships such as duty-balancing and duty-supervising are enforced on tasks to ensure the correctness of the work and to support auditing. The duty relationship between two tasks is called a duty-conflict relationship, as if assigning the two tasks to the same user or role results in fraud. We have defined several duty relationships, including duty-conflict, duty-balancing, duty-supervising, duty-coordinating and non-proprietary duty. For clarity, this work presents duty-conflict, duty-balancing, and duty-supervising relationships.

[Duty-conflict tasks] Two tasks T_i and T_j are duty-conflict tasks, denoted as $T_i \oplus T_j$, if they have duty-conflict relationships; that is, their implicit task-duties conflict.

Duty-conflict relationships can be further distinguished into duty-balancing and duty-supervising relationships.

[Duty-balancing tasks] Two tasks T_i and T_j are duty-balancing tasks, denoted as $T_i \equiv T_j$, if the implicit task-duty of T_i (T_j) is to review task T_j (T_i). T_i and T_j have an equal level of task duty.

[Duty-supervising tasks] Task T_i supervises task T_j, denoted as $T_i \succ T_j$, if the implicit task-duty of T_i is to supervise task T_j. T_i has a higher task-duty than T_j.

Tasks that have duty-conflict (duty-balancing) relationships should not be assigned to the same role/user, to ensure SoD. If T_i has a higher task-duty than T_j, T_i must be performed by a role with a higher position than the role that performs T_j. The duty-balancing relationship is commutable; $T_i \equiv T_j$ implies $T_j \equiv T_i$. The duty-balancing relationship is also a kind of duty-conflict relationship; that is, $T_i \equiv T_j \Rightarrow T_i \oplus T_j$. The duty-supervising relationship is not commutative; $T_i \succ T_j$ does not imply $T_j \succ T_i$. The duty-supervising relationship is also a kind of duty-conflict relationship; $T_i \succ T_j \Rightarrow T_i \oplus T_j$.

3.2. Authorization rules for SoD

Several novel authorization rules, including duty-supervising and execution-dependent rules, have been designed to impose the SoD in task-based access control environments, such as WfMS. The designed authorization rules can enforce the principle of SoD in the assignment and activation of roles or tasks. For clarity, we only present some of the proposed authorization rules. Table 1 lists the defined functions used in the authorization rules. In this paper, T denotes a task, R a role and S a user/subject. Table 1 also shows the implicit meanings of functions used in authorization rules. For

Table 1
The meaning of functions contained in authorization rules

Functions	Meaning
authorized_tasks(R)	The set of tasks that were assigned to role R
authorized_roles(S)	The set of roles that were assigned to user S
authorized_tasks(S)	The set of tasks that were assigned to user S
active_tasks(R)	The set of tasks that are activated by role R
active_roles(S)	The set of roles that are activated by user S
active_tasks(S,R)	The set of tasks that are executed by user S as role R
activated_roles(S)	The set of roles that were activated by user S
executed_tasks(S,R)	The set of tasks that were executed by user S in role R
	Implied meaning
$T \in$ active_tasks(R)	$T \in$ authorized_tasks(R)
$R \in$ active_roles(S)	$R \in$ authorized_roles(S)
$T \in$ active_tasks(S,R)	$R \in$ active_roles(S); $T \in$ authorized_tasks(R)
$R \in$ activated_roles(S)	$R \in$ authorized_roles(S)
$T \in$ executed_tasks(S,R)	$R \in$ authorized_roles(S); $T \in$ authorized_tasks(R)

example, "$T \in$ active_tasks(R)" implies "$T \in$ authorized_tasks(R)". "Role R can activate task T", implies that role R is authorized to perform task T.

SoD variations are either static or dynamic, as described below.

Static SoD: Static SoD requires that two duty-conflict tasks cannot be assigned to the same role or user. The authorization constraints on user-role/role-task assignments are validated during the design phase to enforce SoD.

Rule 1: [Static SoD—role and task]

$\forall T_i, T_j \in$ TaskSet, $R \in$ RoleSet
$T_i \in$ authorized_tasks(R) and $(T_i \oplus T_j) \Rightarrow T_j \notin$ authorized_tasks(R)

If task T_i (for example, preparing a check) and task T_j (for example, auditing a check) have a duty-conflict relationship, and role R was authorized to perform task T_i, then role R cannot also be authorized to perform task T_j.

Rule 2: [Static SoD—user, role and task]

$\forall T_i, T_j \in$ TaskSet, $R_x, R_y \in$ RoleSet, $S \in$ SubjectSet and $x \neq y$, $i \neq j$
$T_i \in$ authorized_tasks(R_x) and $T_j \in$ authorized_tasks (R_y) and $R_x \in$ authorized_roles(S) and $(T_i \oplus T_j)$
$\Rightarrow R_y \notin$ authorized_roles(S)

If tasks T_i and T_j have a duty-conflict relationship; roles R_x and R_y are authorized to perform tasks T_i and T_j, respectively; also, if role R_x was assigned to user S, then role R_y can not also be assigned to user S.

Dynamic SoD variations: Static SoD is too strict to describe a real-world security principle. The constraints of dynamic SoD variations are weaker than those of static SoD. Dynamic SoD variations provide flexibility by allowing two duty-conflict tasks to be assigned to different roles and then to the same user. The authorization constraints on role/task/object activation are then validated during the run-time phase to enforce SoD. For example, dynamic SoD enforces that a user cannot activate different duty-conflict roles simultaneously. Notably, dynamic SoD variations may also strictly require duty-conflict tasks cannot be assigned to the same role, as in Rule 1.

The authorization rules for dynamic SoD variations include authorization rules for dynamic SoD and execution-dependent SoD. Authorization rules for dynamic SoD specify whether a user (subject) may activate several roles, execute several tasks and/or access several objects simultaneously. The authorization rules that govern execution-dependent SoD mainly enforce SoD during the activation of execution-dependent tasks. Two tasks are execution-dependent if they are correlated and the execution (process) of one task (B) depends on the execution (process) of the other task (A), as defined in Section 2. Defining authorization rules requires that execution dependency among tasks be considered to enforce SoD.

SoD can be enforced on various levels, including the role-level (role activation), task-level (task execution) or the object-level (object access). The authorization rules for SoD on the task-level are presented below. The authorization rules for SoD on other levels are similar, and are therefore omitted for clarity.

A user/subject may activate two duty-conflict roles simultaneously but cannot activate the roles to execute duty-conflict tasks, as described in Rule 3. Tasks T_i and T_j have a duty-conflict relationship. If role R_y was authorized to perform task T_j; role R_x was authorized to perform task T_i, and user S has activated role R_x, then user S can activate role R_y but can not execute task T_j in role R_y. Formally, "a user activates a role" or "a role executes a task" implies that the role has been assigned to the user and that the task has been assigned to the role.

Rule 3: [Dynamic SoD]

$\forall T_i, T_j \in$ TaskSet, $R_x, R_y \in$ RoleSet, $S \in$ SubjectSet, and $x \neq y$, $i \neq j$
$(T_i \oplus T_j)$ and $T_i \in$ active_tasks(S, R_x) and $R_y \in$ active_roles(S) and $T_j \in$ authorized_tasks(R_y) $\Rightarrow T_j \notin$ active_ tasks(S, R_y)

A session denotes a particular instance of a connection of a user to the system. At any moment, a user may establish several sessions. Dynamic SoD focuses on enforcing SoD within the user's current active sessions, while execution-dependent SoD enforces SoD across current active sessions and previous (historical) sessions. The SoD is enforced beyond the user's active sessions via execution dependency among tasks, as described in Rule 4. Tasks T_i and T_j are duty-conflict and execution-dependent tasks ($T_i \oplus T_j$ and $T_i \sim T_j$). User S executed task T_i in role R_x, and role R_y is authorized to perform task T_j. User S can activate R_y, but user S cannot execute task T_j in role R_y.

Rule 4: [Execution-dependent SoD]

$\forall T_i, T_j \in$ TaskSet, $R_x, R_y \in$ RoleSet, $S \in$ SubjectSet, and $x \neq y$, $i \neq j$
$(T_i \oplus T_j)$ and $(T_i \sim T_j)$ and $T_i \in$ executed_tasks(S, R_x) and $R_y \in$ active_roles(S) and $T_j \in$ authorized_tasks (R_y) $\Rightarrow T_j \notin$ active_tasks(S, R_y)

Notably, the execution-dependent relationship does not imply a duty-conflict relationship. Two tasks may have execution dependency without a duty-conflict relationship. A stricter rule can be defined as follows.

Tasks T_i and T_j are duty-conflict and execution-dependent tasks. Roles R_x and R_y are authorized to perform tasks T_i and T_j, respectively. If user S activated role R_x, then S cannot activate R_y.

The above illustrates the authorization rules for duty-conflict tasks. Those rules also apply to duty-balancing tasks. The duty-supervising relationship is also a kind of duty-conflict relationship; that is, $T_i \succ T_j \Rightarrow T_i \oplus T_j$. Accordingly, dynamic SoD for duty-supervising tasks must follow the dynamic SoD for duty-conflict tasks. Furthermore, additional authorization rules are required for duty-supervising tasks, as described in Rule 5. Users S_A and S_B activate roles R_x and R_y to execute task T_i and T_j, respectively. If Tasks T_i and T_j have a duty-supervising relationship, $T_i \succ T_j$, role R_x must have a higher position than role R_y.

Rule 5: [Dynamic SoD for duty-supervising tasks]

$\forall T_i, T_j \in \text{TaskSet}$, $R_x, R_y \in \text{RoleSet}$, $S_A, S_B \in \text{SubjectSet}>$ and $x \neq y$, $i \neq j$
$T_j \in \text{active\_tasks}(S_B, R_y)$ and $T_i \in \text{active\_tasks}(S_A, R_x)$ and $(T_i \succ T_j) \Rightarrow R_x > R_y$

SoD for duty-supervising tasks can also be enforced across sessions via execution-dependent relationships. Execution-dependent SoD for duty-supervising tasks must follow the authorization rules for execution-dependent SoD for duty-conflict tasks. Additionally, Rule 6 is applied. If user S_B has executed task T_j in role R_y; S_A activates R_x to execute task T_i; tasks T_i and T_j are duty-supervising and execution-dependent tasks, then role R_x must have a higher position than R_y.

Rule 6: [Execution-dependent SoD for duty-supervising tasks]

$\forall T_i, T_j \in \text{TaskSet}$, $R_x, R_y \in \text{RoleSet}$, $S_A, S_B \in \text{SubjectSet}$ and $x \neq y$, $i \neq j$, $A \neq B$
$T_j \in \text{executed\_tasks}(S_B, R_y)$ and $T_i \in \text{active\_tasks}(S_A, R_x)$ and $(T_i \succ T_j)$ and $(T_i \sim T_j) \Rightarrow R_x > R_y$

4. Authorization model for workflows

The proposed authorization model handles the assignment of workflow tasks to roles/users. The assignments must satisfy the authorization constraints for SoD defined in Section 3. The proposed authorization model includes the planning phase and the run-time phase. The planning phase generates initial workflow activation plans in advance. These plans assign tasks to a set of valid roles/users, to satisfy the constraints of SoD. The planning phase is carried out before the workflow execution starts, while the run-time phase is executed upon the actual activation of each task during the execution of the workflow. The enactment of a workflow decides the current task to be activated. According to the selected role/user activation plan (current plan) generated by the planning phase, the run-time phase identifies the user authorized to activate the current task in a certain role. Dynamic SoD variations are more realistic security policies. Both planning and run-time phases assign tasks to roles/users to satisfy dynamic SoD variations. The authorization must also satisfy the constraints of execution-dependent SoD, specified in Section 3, since workflow tasks generally have execution-dependent relationships. The planning phase includes two algorithms, for role-task planning and user-role-task planning, to determine valid role-task assignments and user-role-task assignments, respectively. The planning begins with role-task planning, and then assigns users to tasks in user-role-task planning. Sections 4.1 and 4.2 detail the algorithms.

Notably, the current activation plan may need to be modified during the run-time phase for the following reasons. The planned user, authorized to perform the current task according to the current plan, may not be available. Furthermore, the activation of the current task by the planned user may violate the constraints (authorization rules) of dynamic SoD variations since the activation plan is generated in the planning phase before the workflow is executed. The planning phase can consider only the assignment of those tasks of the workflow that are being planned, and cannot verify run-time activation of tasks when a user activates several tasks from more than one workflow execution. Consequently, the activation of the current task by the planned user must be verified to ensure that constraints of dynamic SoD variations are not violated. If the authorization check fails, the current activation plan must be modified. The run-time phase identifies an available user authorized to activate the current task, and generates a new activation plan from the current activation plan. Section 4.3 presents the plan-adjust algorithm to determine a new valid activation plan in the run-time phase.

4.1. Role-task planning algorithm

Fig. 1 depicts the role-task planning algorithm. The algorithm first invokes the *GenExecDependency( )* function to generate the execution dependency among tasks of the workflow W. The algorithm then invokes the recursive function *RoleAssignment( )* to generate the set of all valid role-task assignments in the input workflow. If no valid role-task assignment is found, then the algorithm returns a failure; otherwise, it returns success. The tasks of the input workflow are recorded in a list, *Tlist*, ordered with a topological order according to the ordering dependency in the workflow. The algorithm uses *RTplan* to record a valid role-task activation plan, that is, a list of role-task assignments, (R_i, T_i), for

```
Algorithm The role-task planning algorithm
Input: 1) workflow $W$
       2) $capable\_roles(T_i)$: the set of roles which have the capability to execute task $T_i$
       3) $Tlist$: a list of tasks with topological order in a workflow $W$
Output: Fail if no valid role-task assignment; Success, otherwise.
$assigned\_role(T)$: the role assigned to execute task $T$ in an activation plan;
$valid\_roles(T)$: the set of roles that are valid (authorized) to execute task $T$;
$RTplan$: a valid role-task plan, i.e., a list of role-task assignment, $<R_i, T_i>$, of tasks in $Tlist$;
$AllRTplans$: a set of all valid $RTplans$;
begin
   $GenExecDependency(Tlist, W)$
   $T_1$ = the first task of $Tlist$; $AllRTplans = \{\}$; $RTplan = \{\}$;
   for each task $T_i \in Tlist$ do $assigned(T_i)$ = False;
   return $RoleAssignment(T_1)$;
end;

function $RoleAssignment(T_i$: task) : Boolean
begin
   $valid\_roles(T_i) = \{r \mid r \in capable\_roles(T_i)\}$;
   for each task $T_j$ where $T_j \in Tlist$, $T_j \sim T_i$ and $assigned(T_j)$ == True do
      if $T_j \oplus T_i$ then $valid\_roles(T_i) = \{r \mid r \in valid\_roles(T_i)$ and $r \neq assigned\_role(T_j)\}$;
      if $T_i \succ T_j$ then $valid\_roles(T_i) = \{r \mid r \in valid\_roles(T_i)$ and $r > assigned\_role(T_j)\}$;
      if $T_j \succ T_i$ then $valid\_roles(T_i) = \{r \mid r \in valid\_roles(T_i)$ and $r < assigned\_role(T_j)\}$;
   endfor;
   $result$ = Fail;
   $T_k = NextTaskFromTaskList(Tlist, T_i)$;
   $R = ChooseNextRole(valid\_roles(T_i))$;
   while $R$ is not Null do
      $assigned\_role(T_i) = R$;
      $assigned(T_i)$ = True;
      if $T_k$ == Null then
         { $RTplan = CreateNewRTplan(Tlist)$;
           $AddRoleTaskPlans(AllRTplans, RTplan)$;
           $result$ = Success; }
      else if $RoleAssignement(T_k)$ == Success then
              $result$ = Success;
      $R = ChooseNextRole(valid\_roles(T_i))$;
   endwhile;
   $assigned(T_i)$ = False;
   return $result$;
end
```

Fig. 1. The role-task planning algorithm.

each task T_i in *Tlist*. *AllRTplans* records the set of all valid assignments generated by the algorithm.

The *RoleAssignment( )* function finds valid role assignments for current task by recursively finding role assignments for the next task in *Tlist*. Notably, the assignment of valid roles to current tasks must satisfy the constraints of SoD. The SoD verification must satisfy the constraints of execution-dependent SoD as specified in Section 3, since workflow tasks have execution-dependent relationships. When planning the activation of current task, the algorithm conducts SoD verification based on the duty-relationships among the current task and all previously assigned tasks. Notably, the assigned tasks are those activated before the current task during workflow execution, since tasks are planned (assigned) in topological order, according to the ordering dependency (control flow) of the workflow.

Initially, $valid\_roles(T_i)$ is the set of roles that are capable to execute the current task T_i. According to the roles previously assigned to tasks and the SoD constraints, roles that are not valid to activate T_i are excluded from $valid\_roles(T_i)$. Considering each previously assigned task T_j, where T_j and T_i have execution dependency ($T_j \sim T_i$), valid roles for task T_i must exclude the roles assigned to T_j that has a duty-conflict relationship with task T_i ($T_j \oplus T_i$). Moreover, if T_i and T_j have a duty-supervising relationship, $T_i \succ T_j$, then valid roles of task T_i must have a higher position than the role assigned to T_j.

After the SoD verification has been conducted, $valid\_roles(T_i)$ is the set of roles that can validly activate the current task T_i. The while-loop considers each role in $valid\_roles(T_i)$ as a seed to explore possible role-task activation plans by recursively finding role assignments for the subsequent task T_k in *Tlist*. If the current task is the last task in *Tlist*, then a valid role-task assignment of the workflow has been found. A new *RTplan*, an $\langle \text{assigned\_role}(T_j), T_j \rangle$ list, is created to record the role assignments according to the $assigned\_role(T_j)$, for each task T_j in *Tlist*. The *RTplan* is added to the *AllRTplans*. The algorithm finds all valid role-task plans of a workflow. If only one role-task plan is needed, then the statement "result = Success" in the while-loop can be changed to "return Success". The algorithm then returns only one valid role-task plan.

4.1.1. Execution dependency considering the AND/XOR split structure

The *GenExecDependency( )* function can be implemented by simply assigning execution dependency to all tasks of the workflow W, without considering the AND/XOR split structure of the workflow. The AND-SPLIT structure splits the workflow execution into multiple parallel paths (tasks) that are all executed, while the XOR-SPLIT structure splits the workflow execution into multiple mutually exclusive alternative paths (XOR-paths), only one of which is executed. Tasks in different XOR-paths are not executed in the same workflow instance. Thus, no execution dependency exists among tasks in different XOR-paths. Further checking can be conducted to remove the execution dependency from tasks in different XOR-paths. Our current implementation checks the AND/XOR split structure to determine execution dependency. Moreover, as described in Section 2, execution dependency among tasks can be derived from the accessed data objects. If task T_i accesses data objects that are created/modified by task T_j, then the execution of T_i depends on the execution of T_j. The *GenExecDependency( )* function can also be further implemented by checking the accessed data objects to determine the execution dependency among tasks.

4.2. User-role-task planning algorithm

Fig. 2 shows the user-role-task planning algorithm. The algorithm primarily invokes the recursive function *UserAssignment( )* to find a valid user-role-task assignment of the input workflow based on the valid *RTplan* generated by the role-task planning algorithm. If no valid user-role-task assignment is found, the algorithm returns failure; otherwise, it returns success. The *RTplan* is a list of valid role-task assignments, $\langle R_i, T_i \rangle$, of tasks in *Tlist*. The algorithm uses *URTplan* to record a valid user-role-task activation plan; that is, a list of user-role-task assignments, $\langle U_i, R_i, T_i \rangle$, for each task T_i in *Tlist*. The *UserAssignment( )* function determines valid user assignments for current ⟨role, task⟩ pair by recursively finding user assignments for subsequent ⟨role, task⟩ in *RTplan*. The assignments must satisfy the SoD constraint that no duty-conflict (duty-supervising) tasks are assigned to the same user. Notably, duty-supervising and duty-balancing relationships imply duty-conflict relationships. The user-role-task planning algorithm is similar to the role-task planning algorithm depicted in Fig. 1, and a detailed explanation of the algorithm is thus omitted. Notably, the algorithm can be easily modified to find the set of all valid user-role-task assignments, *AllURTplans*.

4.3. Plan-adjust algorithm

The activation plans generated by the role-task planning and the user-role-task planning algorithms can be utilized in various ways. For instance, the workflow engine may store some *RTplans* in advance, without storing a *URTplan*. The user assignments are then determined in run-time to enact each task, using the user-role-task planning algorithm. Notably, the assignment must satisfy the constraints (authorization rules) of dynamic SoD variations. Moreover, a minimal workload policy may be implemented to choose the user with the lowest workload from all users that satisfy the SoD constraints.

An alternative approach is to store some *RTplans* and some *URTplans* in advance. The enactment of a workflow is based on the chosen activation plan. The run-time phase executes the plan-adjust algorithm to find an available user for the current task and to generate a new activation plan based on the actual activation of tasks, as described below. The plan-adjust algorithm is invoked when the planned user, assigned to activate the current task, is not available, or when the run-time activation of the current task by the planned user violates the constraints of dynamic SoD variations.

Assume that the workflow W involves n tasks, $T_1, T_2, \dots, T_n$, where $T_1, T_2, \dots, T_{k-1}$ have been activated and T_k is the current task to be activated. Let $\langle AU_i, AR_i, T_i \rangle$ represent the actual activation of task T_i by user AU_i in role AR_i, for $i = 1$ to $k-1$. The activation plan of the current task is $\langle PU_k, PR_k, T_k \rangle$. However, PU_k is not available, or the constraints of dynamic SoD variations are violated. *RTplanSet* and *URTplanSet* contain some *RTplans* and some *URTplans*, respectively. The *URTplan-adjust( )* algorithm, depicted in Fig. 3, finds another valid *URTplan*, called *NUplan*, from *URTplanSet*, where *NUplan* follows the actual activation of task T_i, $\langle AU_i, AR_i, T_i \rangle$, for $i = 1$ to $k-1$, such that the activation of current task T_k, $\langle U_k, R_k, T_k \rangle$, satisfies the constraints of dynamic SoD variations. The *RTplan-adjust( )* algorithm finds another valid *RTplan*, called

```
Algorithm The user-role-task planning algorithm
Input: 1) capable_users($R_i$): the set of users which have the capability to activate role $R_i$
       2) Tlist: a list of tasks with topological order in a workflow
       3) RTplan: a valid role-task plan, i.e., a list of role-task assignment, $<R_i, T_i>$, of tasks in Tlist
Output: Fail if no valid user-role-task assignment; Success, otherwise.
assigned_user(R, T): the user assigned to execute task T as role R in an activation plan;
valid_users(R, T): the set of users that are valid (authorized) to execute task T as role R;
URTplan: a valid user-role-task plan, i.e., a list of user-role-task assignment, $<U_i, R_i, T_i>$, of tasks in Tlist;
begin
  $<R_1, T_1>$ = the first role-task assignment of RTplan; URTplan = {};
  for each role-task assignment $<R_i, T_i> \in$ RTplan do assigned($R_i$, $T_i$) = False;
  return UserAssignment(RTplan, $R_1$, $T_1$);
end;

function UserAssignment(RTplan: role-task assignment, $R_i$: role, $T_i$: task) : Boolean
begin
  valid_users($R_i$, $T_i$) = { $u \mid u \in$ capable_users($R_i$)};
  for each task $T_j$ where $T_j \in$ Tlist, $T_j \sim T_i$ and assigned($R_j$, $T_j$) == True do
    if $T_j \oplus T_i$ then valid_users($R_i$, $T_i$) = { $u \mid u \in$ valid_users($R_i$, $T_i$) and $u \neq$ assigned_user($R_j$, $T_j$)};
  endfor;
  $<R_k, T_k>$ = NextRoleTaskFromRTPlan(RTPlan, $<R_i, T_i>$);
  U = ChooseNextUser(valid_users($R_i$, $T_i$));
  while U is not Null do
    assigned_user($R_i$, $T_i$) = U;
    assigned($R_i$, $T_i$) = True
    if $<R_k, T_k>$ == Null then
      { URTplan = CreateNewURTplan(RTplan)
        return Success; }
    else if UserAssignement(RTplan, $R_k$, $T_k$) == Success then return Success;
    U = ChooseNextUser(valid_users($R_i$, $T_i$));
  endwhile;
  assigned($R_i$, $T_i$) = False;
  return Fail;
end
```

Fig. 2. The user-role-task assignment algorithm.

NRplan, from *RTplanSet*, where *NRplan* follows the actual activation of task T_i, $\langle AR_i, T_i\rangle$, for $i = 1$ to $k-1$. The algorithm then finds a valid *URTplan*, called *NUplan*, based on *NRplan* and the actual activation of task T_i, $\langle AU_i, AR_i, T_i\rangle$, for $i = 1$ to $k-1$, by invoking the $UserAssignment(NRplan, R_k, T_k)$ algorithm. The activation of T_k, $\langle U_k, R_k, T_k\rangle$, in *NUplan*, must satisfy the constraints of dynamic SoD variations.

5. System implementation and evaluation

5.1. System implementation

Various authorization rules are incorporated into the system to achieve SoD in the assignment of tasks to roles and users. A graphical interface is also supported to enable security managers to specify tasks, roles and users, and impose appropriate authorization rules. Fig. 4 depicts the system architecture which integrates a WfMS. The system contains databases and four modules, including a client application module, a user authentication module, an authorization controller, and an activation controller. The databases store information required to identify users, authorization rules for SoD, user/role/task assignments and the historical record of role/task activations. The client application provides a graphical user interface (GUI) between the user and the system. The user authentication module supports user authentication to validate the user's identity. The system ensures that only authorized users can conduct operations such as activating roles, executing tasks, and accessing objects. The authorization controller governs role-task and user-role assignments. Role activations and task executions conducted by users are verified to ensure that authorization constraints for SoD are not violated.

The WfMS supports the design and enactment of workflows. The design module assists a workflow designer to specify a workflow in the planning phase. The design module supports the assignment of roles/users to each task in a workflow. The assignment must be validated by interaction with the authorization controller to ensure that no authorization constraints for SoD are violated. Furthermore, the design module implements the user/role-task planning algorithms, as illustrated in Section 4, to generate initial workflow activation plans. The run-time module is responsible for the enactment of

```
Case 1: Some RTplans and URTplans for the workflow W are stored in advance.
URTplan-adjust ($W$, $T_k$)
begin
  Find another valid URTplan, NUplan, from URTplanSet, where NUplan contains <$U_i$, $R_i$, $T_i$>, for $i$ = 1 to $n$, and
     NUplan satisfies the following conditions.
     (a) $U_i == AU_i$ and $R_i == AR_i$, for $i$ = 1 to $k$-1.
     (b)The activation of $T_k$, <$U_k$, $R_k$, $T_k$>, satisfies the constraints of dynamic SoD variations.
     (c) $U_k$ has the lowest workload among users satisfying (a) and (b).
  if no valid URTplan can be found, then invoke the RTplan-adjust($W$, $T_k$) to find a valid URTplan.
  if no valid URTplan can be found, then the enactment of workflow W aborts and fails.
end

Case 2: Some RTplans for the workflow W are stored in advance.
RTplan-adjust ($W$, $T_k$)
begin
  repeat
    Find another valid RTplan, NRplan, from RTplanSet, where NRplan contains <$R_i$, $T_i$>, for $i$ = 1 to $n$, and
      NRplan satisfies the following conditions.
      (a) $R_i == AR_i$, for $i$ = 1 to $k$-1.
      (b) The activation of $T_k$, <$R_k$, $T_k$>, satisfies the constraints of dynamic SoD variations.
    if a valid RTplan, NRplan, has been found then
      begin
        Set assigned_users($R_i$, $T_i$) = $AU_i$, according to the actual activations of task $T_i$, <$AU_i$, $AR_i$, $T_i$>, for $i$ = 1 to
          $k$-1.
        Set assigned_users($R_j$, $T_j$) = False, for $j$ = $k$ to $n$.
        Invoke the UserAssignment(NRplan, $R_k$, $T_k$) algorithm to find a valid URTplan, NUplan, where the acti-
          vation of $T_k$, <$U_k$, $R_k$, $T_k$>, in NUplan satisfies constraints of dynamic SoD variations.
      endif
  until no more valid RTplan exists or a valid URTplan has been found
  if no valid URTplan can be found, then the enactment of workflow W aborts and fails.
end
```

Fig. 3. Plan adjust algorithm in run-time phase.

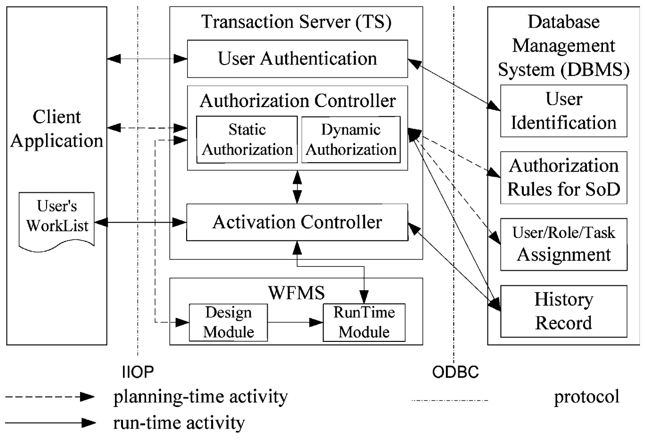


Fig. 4. The system architecture.

workflows in the run-time phase, including the execution and scheduling of tasks. The run-time module controls the task execution flow and assigns a user to perform the current task. The assignment must also be validated by interaction with the authorization controller to verify authorization constraints of SoD. Moreover, the run-time module executes the plan-adjust algorithm described in Section 4.3, to find a valid user for the current task, if the planned user cannot activate the current task.

The activation controller manages role/task activation and interaction with users and WfMS. The run-time module of WfMS provides each user with his/her work-list via the activation controller. Furthermore, the activation controller handles users' requests for role/task activation and issues an authorization request to the authorization controller to verify the satisfaction of authorization constraints for SoD. During the run-time phase, a user may issue a request to activate a role or execute certain tasks in his/her work-list. First, his/her identity must be verified by the authentication module. Then the client application sends the user's request to the activation controller. The activation controller communicates with the authorization controller to manage the authorization. The activation controller examines the related authorization rules, assignments and historical records to authorize the role/task activation. If the request for role/task activation is confirmed as legal and adequate, then the activation controller sends the user's request to the WfMS. Finally, the WfMS allows the user to execute the authorized task.

Sybase's Enterprise Application Server (EAS) is used to develop the system. EAS is an integrated development tool, including a front-end tool, an object-based development tool, a transaction server (TS), and a database managed by Sybase SQLAnywhere database management system (DBMS). The prototype system is a three-tier architecture with the TS as the middle tier between the client application and the DBMS server. The DBMS stores authorization data and process/task data. Notably, the application program on the client side communicates with the TS using IIOP (Internet Inter-ORB Protocol). The client application program serves the user by calling the business objects supported in the authorization controller, the activation controller and the TS. The TS retrieves the required data from the DBMS server via ODBC.

5.2. System demonstration

This section describes a procurement process to demonstrate the application of the system to managing authorization for business processes. The system provides the security administrator with a GUI to manage the authorization, involving enacting authorization rules, duty-conflict relationships, objects (documents, tasks, roles, users) and their relationships, as shown in Fig. 5. Fig. 5 reveals that "Approving item-request" supervises "Issuing item-request". An initial workflow activation plan generated in the planning phase is as follows. Mary/Clerk is assigned to "Issuing item-request", and John/Assistant-Manager is assigned to "Approving item-request". During the run-time phase, the activation controller communicates with the authorization controller to verify the users' requests on activating a role/task. Once the request is verified to satisfy the dynamic constraints for SoD, the user is allowed to play the role or execute the task.

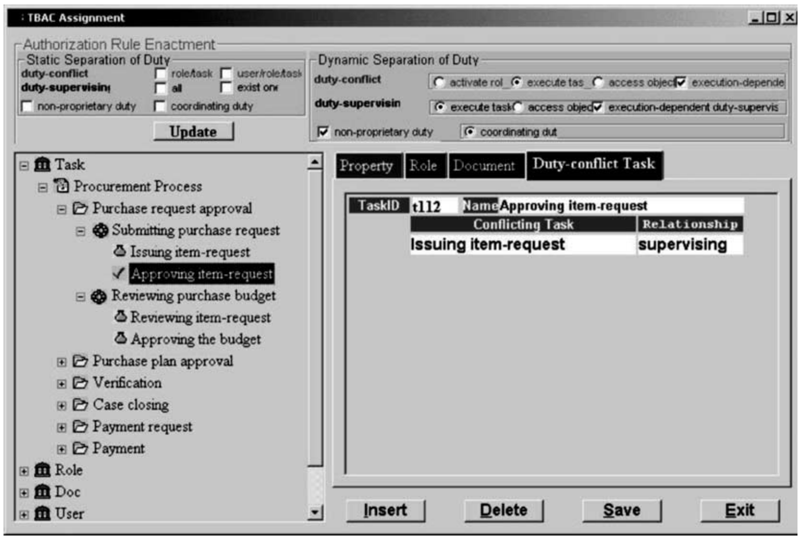
Assume that Mary is unavailable to perform the task "Issuing item-request" in the current workflow instance (workflow instance no. 135). The plan-adjust algorithm finds that John is a valid user to activate the task "Issuing item-request" in the current workflow instance (wf-no. 135). Fig. 6 shows that even though John is authorized as an assistant manager, he cannot activate the task "Approving item-request" to approve the request to purchase a computer (CPU P4 1.4 GHz, workflow instance no. 135), since the request was issued by John, himself. Such approval would violate the SoD constraints. Notably, in other workflow instances, John can play the role of the assistant manager to activate the "Approving item-request" task to approve requests issued by other clerks.

5.3. System evaluation

A case organization, i.e., an information department of the Army data management center, was invited to evaluate and test run our prototype system. A procurement process is deployed to evaluate the system. Notably, only some necessary roles and users participated in the procurement-process were considered for evaluation purposes. Detailed comments provided in the evaluation responses by those who had used the proposed prototype system are as follows.

(1) A preliminary planning phase is required to specify the capabilities of the roles and users as well as the duty-conflict relationships among tasks. The case organization needs to specify, for each task, the set of capable roles that can execute the task, and, for each role, the set of capable users that can activate the role. However, the operational procedure of the case organization does not include such a planning phase. Accordingly, the case organization needs to redefine its operational procedure to accommodate such a planning phase, increasing, however, the workload of the case organization.

(2) The workflow designer in the case organization directly assigns users/roles to tasks, according to the



Static Separation of Duty
duty-conflict ☐ role/task ☐ user/role/task
duty-supervising ☐ all ☐ exist one
☐ non-proprietary duty ☐ coordinating duty

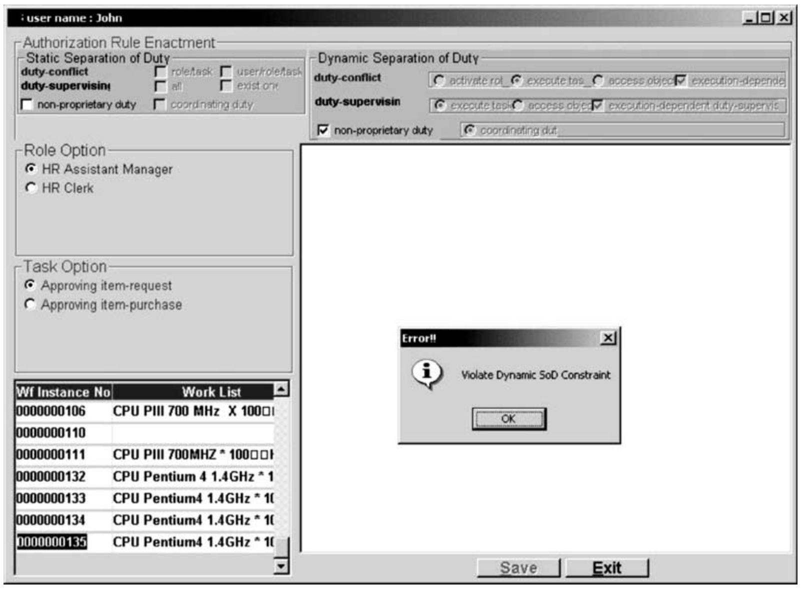
Dynamic Separation of Duty
duty-conflict: activate rol [✓ execute tas] access objec ✓ execution-depende
duty-supervisin: [✓ execute task] access objec ✓ execution-dependent duty-supervis
✓ non-proprietary duty [✓ coordinating dut]

Task
- Procurement Process
  - Purchase request approval
    - Submitting purchase request
      - Issuing item-request
      - ✓ Approving item-request
    - Reviewing purchase budget
      - Reviewing item-request
      - Approving the budget
  - Purchase plan approval
  - Verification
  - Case closing
  - Payment request
  - Payment
Role
Doc
User

TaskID t112 Name Approving item-request

Conflicting Task	Relationship
Issuing item-request	supervising

Fig. 5. Enactment of data for authorization control.


: user name : John

Authorization Rule Enactment

Static Separation of Duty
duty-conflict ☐ role/task ☐ user/role/task
duty-supervising ☐ all ☐ exist one
☐ non-proprietary duty ☐ coordinating duty

Dynamic Separation of Duty
duty-conflict ○ activate rol ● execute tas ○ access objec ✓ execution-depende
duty-supervisin ● execute task ○ access objec ✓ execution-dependent duty-supervis
✓ non-proprietary duty ● coordinating dut

Role Option
● HR Assistant Manager
○ HR Clerk

Task Option
● Approving item-request
○ Approving item-purchase

Wf Instance No	Work List
0000000106	CPU PIII 700 MHz X 100□
0000000110	
0000000111	CPU PIII 700MHZ * 100□□
0000000132	CPU Pentium 4 1.4GHz * 1
0000000133	CPU Pentium4 1.4GHz * 1(
0000000134	CPU Pentium4 1.4GHz * 1(
0000000135	CPU Pentium4 1.4GHz * 1(

Error!!
Violate Dynamic SoD Constraint
OK

Save Exit

Fig. 6. Verifying SoD in the activation of "Approving item-request".

organization's understanding of the duties associated with roles/users. There is no system means and verification to ensure that the principle of SoD is not violated. The proposed system provides a graphical interface in which the security manager can specify tasks, roles and users, and implement appropriate authorization rules to maintain separation of duties. The response of the case organization is very positive regarding the aid provided by our system for verifying the principle of SoD.

(3) The case organization uses primarily user ids and passwords for security control. The security control of our system is a more complex process that requires tasks/roles/users to be specified and separation of duties to be verified. Such a complex process is inconvenient when the case organization seeks flexibly to adjust manpower when executing workflows. For example, an unplanned role/user may need to perform an unauthorized task due to workflow exceptions or emergent organizational needs. However, the case organization agreed that our system is helpful in providing a more secure mechanism for controlling the execution of tasks/workflows.

6. Discussion

This section surveys some related work in details, and compares them with our proposed work. Bertino et al. (1999) proposed a flexible model for the specification and enforcement of authorization constraints in WfMS. The comparison of their work with ours can be elucidated as follows. First, they considered neither the execution dependency nor the variations of SoD that arises from different duty-relationships among tasks. On the contrary, we have defined several authorization rules for SoD based on various duty-conflict and execution-dependent relationships. The execution-dependent SoD supports the enforcement of SoD across users' active sessions and historical sessions. Second, the authorization planning algorithms proposed by Bertino et al. mainly find valid assignments by consistency checking with deductive inference on authorization constraints expressed in logic language. Different from their work, our approach finds valid assignments by verifying SoD constraints based on various duty-conflict relationships among tasks, and in particular, the execution dependency among tasks in workflow instances. Moreover, we have considered the AND/XOR split structure of a workflow to explore the execution dependency.

With the rapid growth of Internet usage for business applications, conducting workflow management on the Internet is an inevitable trend for business commerce. Ahn et al. (2002) developed a system architecture for enforcing RBAC in Web-based WfMS. The architecture mainly consists of a role server for maintaining user-role assignments and issuing certificates with client's role information. The proposed role-based authorization is still based on the simple RBAC96 model. Atluri and Huang (1996) proposed a WAM for workflows. The model associates each task with authorization templates to support the authorization of granting a subject to execute the task. Huang and Atluri (1999) also presented a secure Web-based WfMS based on the WAM. A workflow authorization server is employed to support the specification and enforcement of security policies based on RBAC and SoD. Botha and Eloff (2001) presented a context-sensitive access control model to protect unauthorized access to documents used in workflow systems. The model considers conflicting tasks, conflicting users and access history of document in supporting dynamic SoD requirements.

Moreover, access control mechanisms and security models have been proposed for inter-organizational workflows (Atluri et al., 2001; Kang et al., 2001). Kang et al. (2001) proposed a notion of role domain, instead of an organization's role structure, to specify the data access policy associated with each task of the inter-organizational workflow. A context-based access control model is proposed to enforce data access according to the capability of each task. Atluri et al. (2001) considered the issues of conflict-of-interest among competing organizations of inter-organizational workflows in decentralized workflow environments. A variation of Chinese wall security model is proposed to prevent sensitive dependency information of a task leaking to another task agent (organization) with conflict-of-interest.

The comparisons of our work with above literatures (Ahn et al., 2002; Atluri et al., 2001; Atluri and Huang, 1996; Botha and Eloff, 2001; Huang and Atluri, 1999; Kang et al., 2001) are illustrated as follows. First, they did not consider authorization planning for assigning workflow tasks to roles/users. In contrast, we have developed the user/role/task planning algorithms in planning-time phase and the plan-adjust algorithm in run-time phase, respectively. Second, they considered neither the execution dependency nor the variations of SoD that arises from different duty-relationships among tasks. On the contrary, we have defined several authorization rules for SoD based on various duty-conflict and execution-dependent relationships. Finally, some researchers have addressed access control mechanisms for inter-organizational workflows. Our current work does not focus on inter-organizational environments, though some of the proposed work can still be applied in such environments. Further investigation is required to extend our work to inter-organization workflows, and thus is proposed as future work.

7. Conclusions and future work

Authorization management and access control are essential in supporting secure WfMS. This work presents a novel system that facilitates the effective authorization management of workflows to assign tasks to roles or users, while enforcing task-based SoD. The proposed system was evaluated by a case organization. The evaluation results have the following implications. First, the proposed system requires that enterprises plan the capabilities of roles and users in advance. However, enterprises may not have clearly identified the capabilities of roles and users. The policy of SoD may not be specified in organizations. To implement the proposed system in enterprises, a re-engineering process is required to adjust organizations' operational procedures for specifying roles, users and security policy. Accordingly, the proposed system is more complex and increases the workload required for security validation. Second, the proposed system enforces a strict security control. In practice, an enterprise may require flexibility to adapt to dynamically changing business environments. Strict authorization enforcement can achieve SoD and thus prevent fraud, by sacrificing flexibility and convenience.

Our future work will address three themes. First, duty-conflict relationships are essential to design SoD constraints. Further work is necessary to explore more kinds of duty-conflict relationships. Second, the execution dependency is proposed and employed to support the enforcement of SoD across various users' sessions. The concept of execution dependency can be similarly applied to tasks of different workflows. However, deriving such execution dependency across different workflows requires further study. Third, inter-organization workflows are gaining importance in B-to-B commerce. Although some works have addressed access control in this aspect, they disregard the coordination behavior in inter-organizational workflows (Koetsier et al., 2000; Shen and Liu, 2001). Future research will be to investigate the authorizations and access control in inter-organizational workflows.

Acknowledgement

The authors would like to thank the National Science Council of the Republic of China for financially supporting this research under contract no. NSC 89-2416-H-009-041 and NSC 91-2623-7-009-018.

References

Ahn, G.-J., Sandhu, R., Kang, M., Park, J., 2002. Injecting RBAC to secure a Web-based workflow system. In: Proceedings of 5th ACM Workshop on Role-Based Access Control.

Atluri, V., Huang, W.-K., 1996. An authorization model for workflows. In: Proceedings of the fifth European Symposium on Research in Computer Security, Rome, Italy, pp. 44–64.

Atluri, V., Chun, S.A., Mazzoleni, P., 2001. A Chinese wall security model for decentralized workflow systems. In: Proceedings of the 8th ACM Conference on Computer and Communications Security.

Barkley, J., 1995. Implementing role based access control using object technology. In: First ACM Workshop on Role Based Access Control, November.

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

Botha, R.A., Eloff, J.H.P., 2001. Access control in document-centric workflow systems—an agent-based approach. Computers and Security 20 (6), 525–532.

Cichocki, A., Helal, A., Rusinkiewicz, M., Woelk, D., 1998. Workflow and Process Automation: Concepts and Technology. Kluwer Academic Publishers.

Ferraiolo, D.F., Cugini, J., Kuhn, R., 1995. Role-based access control (RBAC): features and motivations. In: Proceedings of 11th Annual Computer Security Application Conference. IEEE Computer Society Press, pp. 241–248. December.

Georgakopoulos, D., Hornick, M., Sheth, A., 1995. An overview of workflow management: from process modeling to workflow automation infrastructure. Distributed and Parallel Databases, 119–153.

Gligor, V.D., Gavrila, S.I., Ferraiolo, D., 1998. On the formal definition of separation-of-duty policies and their composition. In: Proceedings of IEEE Symposium on Security and Privacy. IEEE Computer Society.

Huang, W.-K., Atluri, V., 1999. SecureFlow: a secure web-based workflow management system. In: Proceedings of 4th ACM Workshop on Role-Based Access Control, Fairfax, VA, pp. 83–94, October.

Kang, M.H., Froscher, J.N., Sheth, A.P., Kochut, K.J., Miller, J.A., 1999. A multilevel secure workflow management system. In: Proceedings of the 11th Conference on Advanced Information Systems Engineering (CAiSE'99), Heidelberg, Germany, June, pp. 271–285.

Kang, M.H., Park, J.S., Froscher, J.N., 2001. Access control mechanisms for inter-organizational workflow. In: Sixth ACM Symposium on Access Control Models and Technologies (SACMAT 2001), May 3–4.

Kappel, G., Lang, P., Rausch-Schott, S., Retschitzegger, W., 1995. Workflow management based on objects, rules, and roles. IEEE Bulletin of the Technical Committee on Data Engineering 18/1, 11–17.

Koetsier, M., Grefen, P., Vonk, J., 2000. Contracts for cross-organizational workflow management. In: Proceedings of the 1st International Conference on Electronic Commerce and Web Technologies, London, UK, pp. 110–121.

Lee, B.G., Narayanan, N.H., Chang, K.H., 2001. An integrated approach to distributed version management and role-based access control in computer supported collaborative writing. The Journal of Systems and Software 59, 119–134.

Nash, M.J., Poland, K.R., 1990. Some conundrums concerning separation of duty. In: Proceedings of IEEE Computer Society Symposium on Security and Privacy. IEEE Computer Society Press. May.

Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E., 1996. Role-based access control models. IEEE Computer 29 (2), 38–47.

Schier, K., 1998. Multifunctional smartcards for electronic commerce—application of the role and task based security model. In: 14th Annual Computer Security Applications Conference, December.

Shen, M., Liu, D.R., 2001. Coordinating interorganizational workflows based on process-views. In: Proceedings of the DEXA 2001 12th International Conference on Database and Expert Systems Applications, Munich, Germany, September, LNCS 2113. Springer-Verlag, Berlin, Heidelberg, pp. 274–283.

Simon, R.T., Zurko, M.E., 1997. Separation of duty in role-based environments. In: 10th Computer Security Foundations Workshop, June 10–12.

Workflow Management Coalition (WfMC), 1995, Workflow management coalition: workflow reference model. Document number TC00-1003. Available from <http://www.wfmc.org/standards/docs/tc003v11.pdf>.