國 立 交 通 大 學

資訊管理研究所

博 士 論 文

以角色與工作為基礎的工作流程授權管理

Role and Task Based Authorization Management for Workflows

研 究 生：吳 美 玉

指導教授：劉 敦 仁

中 華 民 國 九 十 四 年 一 月

國 立 交 通 大 學

資訊管理研究所

博 士 論 文

以角色與工作為基礎的工作流程授權管理

Role and Task Based Authorization Management for Workflows

研 究 生：吳美玉

研究指導委員會：黃景彰　博士
李瑞庭　博士
曾文貴　博士
羅濟群　博士
楊　千　博士

指導教授：劉敦仁　博士

中 華 民 國 九 十 四 年 一 月

以角色與工作為基礎的工作流程授權管理

Role and Task Based Authorization Management for Workflows

研 究 生：吳美玉　　Student：Mei-Yu Wu

指導教授：劉敦仁　　Advisor：Dr. Duen-Ren Liu

國 立 交 通 大 學

資 訊 管 理 研 究 所

博 士 論 文

A Dissertation

Submitted to Institute of Information Management

College of Management

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Information Management

January 2005

Taipei, Taiwan, Republic of China

中華民國九十四年一月

Role and Task Based Authorization Management for Workflows

Student: Mei-Yu Wu Advisor: Dr. Duen-Ren Liu

Institute of Information Management
National Chiao Tung University

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. Moreover, this work discusses the authorization management of organizational roles in a process-view. A process-view, an abstracted process derived from a base process, can provide adaptable task granularity to suit different needs of workflows participants. Authorization mechanisms are proposed to derive a role's permissions on virtual activities based on the role's permissions on base activities. The proposed authorization mechanisms consider duty-conflict relationships among base activities to enforce SoD. A prototype system is developed to realize the effectiveness of the proposed authorization model.

Keywords : Role-based access control, workflow, process-view, authorization management, separation of duty

Acknowledgement

終於完成了博士學程，首先要感謝我的恩師 劉敦仁教授，謝謝老師從碩士班開始的諄諄教誨與悉心指導，使得我在研究所期間，習得學術工作者應有的倫理規範與社會責任，同時也讓我的寫作進步許多，老師嚴謹的治學研究態度，更將是我一輩子的榜樣。

同時還要感謝口試委員 黃景彰教授、李瑞庭教授、曾文貴教授、羅濟群教授以及楊千教授，謝謝您們在百忙之中，於論文口試期間，仍細審論文並給予指正與建議，使得本論文能夠更臻完善，在此深表感激。此外，還要感謝在投稿過程中的匿名審查者所給予的各項指正。

另外，還要謝謝所有關心我的朋友，謝謝許許多多在生活上與研究上無可取代的歷屆學長姐、實驗室伙伴與室友，其中尤其要感謝淑惠，總是貼心的幫助我完成許多行政與生活上的事務，感謝她體貼的接送與適時的鼓舞，分擔了我許多的煩憂。

最後，我得感謝我的先生志坤，感謝他永遠的支持與關懷，感謝他毫無怨言地當我的出氣筒，陪伴我走過許多快樂與悲傷的時光，感謝所有家人的支持，來自家人的愛與關懷，是我最溫暖而厚實的依靠。謝謝你們。

美玉 2005/01

Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1. Motivation

As effective process management tools, workflow management systems (W*f*MSs) enable a business to analyze, simulate, design, enact, control and monitor its business processes[7][11][17][30][32]. A workflow (process) consists of a set of tasks (activity), and the ordering of tasks, or control flow dependencies, that specify the order of execution among the tasks. Workflows commonly process sensitive business information. The important information should be controlled securely to avoid attack by outsider or unauthorized access by insiders. Therefore, 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, [4][9][10][22][28][31]) has become a widely accepted access control mechanism for security management. 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. Separation of duty (SoD) [9][12][14][20][21][22][27] 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.

Thomas et al. [29] presented task-based authorization control to manage the execution of tasks by controlling the run-time execution status of tasks. They considered neither SoD nor authorizations among tasks, roles and users. Schier [23] also 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. [19] implemented role-based access control in computer supported collaborative writing (CSCWriting). They focused on integrating distributed version management and RBAC for CSCWriting environments. Accordingly, they did not consider the issues of role-base authorizations and SoD.

A novel virtual workflow process, i.e., a process-view, in a WfMS is proposed by Shen and Liu [26]. A process-view, i.e., an abstracted process derived from an implemented base process, is employed to provide aggregate abstraction of a process. They focused on develop view mechanism in workflow management systems. They did not discuss the aggregation of virtual activity and permissions for a role in a process-view.

1.2. Goals

According to the motivations, this dissertation lists major goals as follows:

- Analyze and define several duty-conflict relationships among tasks
- Design authorization rules to enforce separation of duty constraints
- Propose an authorization model
- To derive the aggregation of virtual activity
- Analyze and define privilege principle for a role in a Process-view
- Implement a prototype system
- To evaluate the system

1.3. Contributions

This work analyzes and defines various duty-conflict relationships among tasks from the aspect of how enterprises set up tasks. 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. Some duty-relationships 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. Moreover, this work designs authorization rules for SoD, based on the defined duty-conflict relationships and execution dependencies of workflow tasks. 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 model enforces SoD by taking into account the AND/XOR split structure of workflows and execution dependency among tasks in assigning tasks to roles/users.

Besides, this work also discusses the authorization management in process-views.

Process-views allow a workflow management system to provide various aggregated views of a process for different levels or departments in an organization or for different organizations in a supply chain. An aggregation of a set of base activities called a virtual activity. This work discusses the authorization management of organizational roles in a process-view. The derivation of a virtual process (process-view) involves grouping the base activities in a base process and aggregation functions into virtual activities. This work defines several permissions for a role on base activities. The permissions for a role on a virtual activity are according to the permissions of aggregated base activities under strict or lenient privilege principle. An algorithm is proposed to derive permissions for a role on the virtual activity in general cases. The derivation also considers the duty-conflict relationships among base activities. Furthermore, a prototype system that can conduct authorization management in task-based workflow environments is designed and implemented. An analyzed procurement process is deployed in the system to demonstrate the proposed authorization management of workflow tasks.

1.4. Organization

The rest of this paper is organized as follows. Chapter 2 presents the related work on role-based access control, workflow management and process view. Chapter 3 analyzes and defines various duty-conflict relationships among tasks, and presents authorization rules for SoD. Chapter 4 illustrates the proposed algorithms for planning role-task and user-role-task assignments and plan-adjust algorithm in run-time. Chapter 5 discusses the authorized permissions for a role in a process-view. Chapter 6 elucidates the architecture and implementation of the system. Chapter 7 compares related work with our proposed work. Conclusions and future works are finally made in Chapter 8.

Chapter 2. Related Work

In this chapter, we will discuss the related work with our topic including role-based access control, workflow management, and process-view.

2.1. Role-based Access Control

Role-based access control (RBAC, [4][9][10][22]) has become a widely accepted access control mechanism for security management. Ferraiolo proposes the concept of role-based access control (RBAC) and elucidate RBAC is a non-discretionary access control [10]. In role-based access control model [9][10], a user can play several roles and a role can be assigned to several users. The permission assignments are not assigned to users but to roles. The permissions are enacted with roles and roles are authorized to users. Permissions assign to roles will reduce the complex of management and easy to change access control rules when employee left or change to another position in the organization.

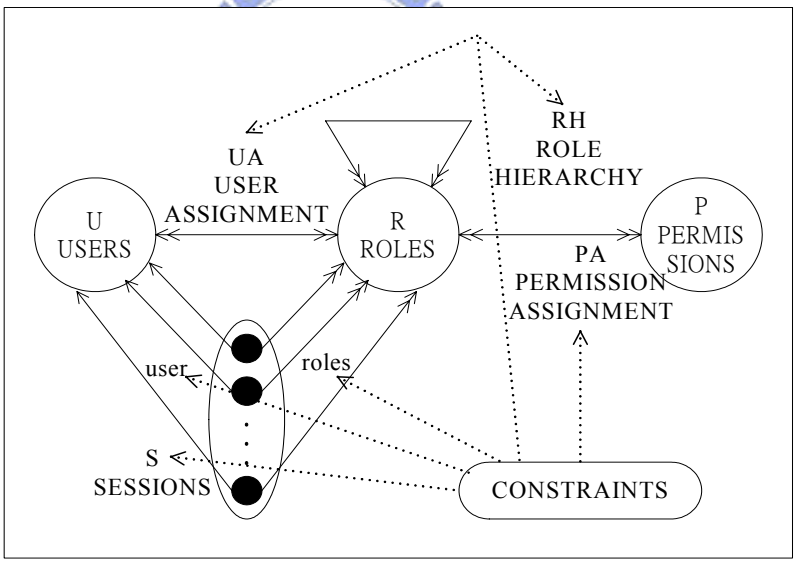

Figure 1. Role-Based Access Control Model
Reference: [22]

The role-based access control model, proposed by Sandhu etc. illustrated in Figure1, contains users, roles, permissions and sessions.[22] Users are people in real world or executable program as subject. Roles are the position of the organization and in other words, the collection of the permissions. For example in bank environment, “teller” and “accountant” are all a kind of role. Permissions are the privilege to access the objects in

the systems. Sessions are the collection of activating users and users can change roles dynamically to achieve the purpose of least privilege.

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.

2.1.1. Separation of duty

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) [9][12][21][22][27] is the most important feature of role-based access control model. SoD 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) roles are authorized to different users. For example, "purchaser" and "cashier" are mutually exclusive role therefore they should not be authorized to the same user. There are two kinds of separation of duty, i.e. strong exclusion, namely static separation of duty (SSoD), and the weak exclusion, namely dynamic separation of duty (DSoD). SSoD restricts that a user cannot own two mutual exclusive roles while DSoD allows that a user owns two mutual exclusive roles but cannot activate them at the same time.

Thomas et al. [29] presented task-based authorization control to manage the execution of tasks and to provide an authorization mechanism for task execution through the control of run time execution status of tasks. They considered neither SoD nor authorizations among tasks, roles and users. Schier [23] also presented a role and task-based security model. Although authorization rules for SoD have been designed based on mutual exclusive (duty-conflict) tasks, the proposed work is merely an extension of RBAC model. Their definition of mutual exclusive tasks is simply derived from the definition of mutual exclusive roles in RBAC. In addition, the proposed authorization rules for SoD are merely derived from SoD in RBAC. Schier's authorization rules only contain static and dynamic SoD. Extending from the rule in RBAC that mutual exclusive roles cannot be assigned to the same user, mutual exclusive tasks con not be assigned to the same role or user. Moreover, Lee et al. [19] implemented

role-based access control in computer supported collaborative writing (CSCWriting). They focused on integrating distributed version management and RBAC for CSCWriting environments. Accordingly, they did not consider the issues of role-base authorizations and SoD.

2.2. Workflow management

Several researchers [1][2][3][5][6][13][14][15][16] have addressed role-based access control and authorization management in workflow systems. Workflow management systems allow businesses to analyze, simulate, design, enact, control and monitor their overall business processes. The major issues concern the design of role-based authorization mechanisms in support of separation of duty. Bertino et al. [5] elucidated a flexible model to specify and enforce authorization constraints in workflow management systems. A logical authorization language, defined as clauses in a logic program, is proposed to express authorization constraints on role assignments and user assignments. Ahn et al. [1] developed a system architecture to enforce role-based access control in Web-based workflow management systems. The architecture mainly consists of a role server for maintaining user-role assignments and issuing certificates with client's role information. Atluri and Huang proposed a Workflow Authorization Model (WAM) [3]. The model associates each task with authorization templates that specify static parameters of authorization defined during the design period. Huang and Atluri [13] also presented a secure Web-based workflow management system (SecureFlow). Botha and Eloff [6] considered conflicting tasks, conflicting users (such as family members) 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 [2][16]. Atluri et al. [2] 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 address such issues. The model mainly prevents sensitive dependency information or sensitive output of a task leaking to another task agent (organization) with conflict-of-interest. Kang et al. [16] 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 workflow.

Most of the above literatures have addressed SoD constraints in role-based authorizations of workflows. Although SoD has been discussed, most of the works do not consider SoD constraints in relation to various duty-conflict relationships and execution-dependencies among tasks. A task specifies the rights of each member of a set of roles to perform operations on each member of a set of object categories. [8] Moreover, they do not consider authorization planning for assigning users/roles to workflow tasks. If we only use role-based access control to achieve the objective of separation of duty for organization, it is not enough to show the real activity in the organization, i.e. task-based enterprising environment. Although Bertino et al. [5] 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. Although examples have been presented to illustrate how to express static and dynamic SoD via the proposed authorization language, they also do not consider the variations of SoD that arise from various duty relationships among tasks. Furthermore, the proposed authorization planning algorithms do not consider the AND/XOR split structure of a workflow.

2.3. Process-View

Workflow management system allows various participants to collaborate in effectively managing a workflow-controlled business process. Shen and Liu [26] propose a process-view workflow management to provide appropriate process abstraction for various roles within an enterprise. Process-views are derived through the bottom-up aggregation of activities to provide various levels of abstraction of a base process as Figure 2.

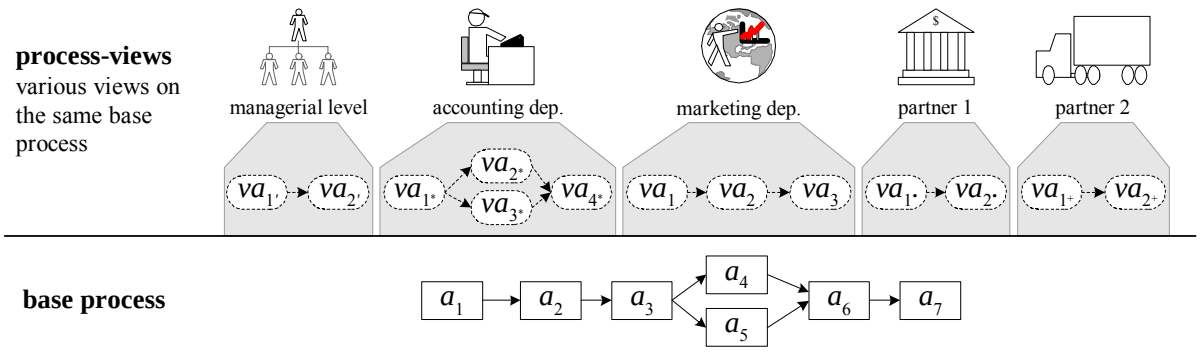

Figure 2. Examples of Process-views
Reference: [26]

For example, a process modeler can design an appropriate process-view for the marketing department as follows. The base activities, a_1, a_2 and a_3, in the base process are mapped into the virtual activity, va_1, in the process-views. The base activities a_4 and a_5 are mapped into the virtual activity va_2 and a_6 and a_7 are mapped into va_3.

A process-view is proposed for providing adaptable task granularity. The design of a process-view must first identify all base activities within it and then arrange them based on dependencies and ordering structures. A process-view is a two-tuple <VA, VD>, where VA is a virtual activities and VD is a set of virtual dependencies. Process-views allow a process modeler to flexibly provide different roles with appropriate views of an implemented process. For example, process-views provide high-level managers with aggregated information on a desired process which is different from the need for accountant. They focus the process-view derived from base process neither derive the aggregation of virtual activity nor analyze and define permissions for a role on a virtual activity in a process-view.

Chapter 3. Task-based Separation of Duty

A workflow consists of a set of tasks, and their order of execution, according to control flow dependencies [11][32]. 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.

Workflow management systems generally include process definition tools and workflow engines [32]. 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* [32]. 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.

Separation of duty (SoD) [9][12][21][22][27] 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 separation of duty. Static separation of duty prevents conflicting (mutually exclusive) roles or operations from being assigned to the same user. Dynamic separation of duty 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.

3.1. 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 separation of duty. 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

This section presents authorization rules based on various duty-relationships among tasks. The designed authorization rules can enforce the principle of separation of duty (SoD) in the assignment and activation of roles or tasks. 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 workflow management systems. For clarity, we only present some of the proposed authorization rules. Table 1 lists the defined functions used in the authorization rules. In this work, O denotes an object, T a task, R a role and S a subject. Table 2 shows the implicit meanings of functions used in authorization rules. For example, the function "T ∈ active_tasks(R)" implies the function "T ∈ authorized_tasks(R)". "Role R can activate task T", implies that role R is authorized to perform task T.

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
authorized_objects(T)	The set of objects that were assigned to task T

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
active_objects(S, R, T)	The set of objects that are accessed by user S in role R to execute task T
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
accessed_objects(S, R, T)	The set of objects that were accessed by user S in role R to execute task T

Table 2. The implied meaning of functions

Functions	Implied functions
T ∈ active_tasks(R)	T ∈ authorized_tasks(R)
R ∈ active_roles(S)	R ∈ authorized_roles(S)
T ∈ active_tasks(S, R)	R ∈ active_roles(S) ; T ∈ authorized_tasks(R)
O ∈ active_objects(S, R, T)	T ∈ active_tasks(S, R) ; O ∈ authorized_objects(T)
R ∈ activated_roles(S)	R ∈ authorized_roles(S)
T ∈ executed_tasks(S, R)	R ∈ authorized_roles(S) ; T ∈ authorized_tasks(R)
O ∈ accessed_objects(S, R, T)	R ∈ authorized_roles(S) ; T ∈ authorized_tasks(R) ; O ∈ authorized_objects(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 work-related

and the execution (process) of one task (B) depends on the execution (process) of the other task (A). For example, the tasks of a single workflow instance are execution-dependent. 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 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_x was authorized to perform task T_i; role R_y was authorized to perform task T_j, 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 $T_i \sim T_j$). Subject S executed task T_i in role R_x, and role R_y is authorized to perform task T_j. Subject S can activate R_y, but subject S cannot execute task T_i in role 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 subject S has activated role R_x, then S cannot activate 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】

$\forall T_i, T_j \in$ TaskSet, $R_x, R_y \in$ RoleSet, $S_A, S_B \in$ SubjectSet and $x \neq y$, $i \neq j$

$T_j \in$ active_tasks(S_B, R_y) and $T_i \in$ 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 subject 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】

$\forall T_i, T_j \in$ TaskSet, $R_x, R_y \in$ RoleSet, $S_A, S_B \in$ SubjectSet and $x \neq y$, $i \neq j$, $A \neq B$

$T_j \in$ executed_tasks(S_B, R_y) and $T_i \in$ active_tasks(S_A, R_x) and ($T_i \succ T_j$) and ($T_i \sim T_j$) $\Rightarrow$ $R_x > R_y$

Chapter 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 Chapter 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 Chapter 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

Figure 3 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 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 Chapter 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.

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)
   T1 = the first task of Tlist; AllRTplans = {}; RTplan = {};
   for each task Ti ∈ Tlist do assigned(Ti) = False;
   return RoleAssignment(T1);
end;

function RoleAssignment(Ti: task) : Boolean
begin
   valid_roles(Ti) = {r | r ∈ capable_roles(Ti)};
   for each task Tj where Tj ∈ Tlist , Tj ~ Ti and assigned(Tj) == True do
      if Tj ⊕ Ti then valid_roles(Ti) = valid_roles(Ti) − assigned_role(Tj)
      if Ti ≻ Tj then valid_roles(Ti) = {r | r ∈ valid_roles(Ti) and r > assigned_role(Tj)};
      if Tj ≻ Ti then valid_roles(Ti) = {r | r ∈ valid_roles(Ti) and r < assigned_role(Tj)};
  endfor;
   result = Fail;
   Tk = NextTaskFromTaskList(Tlist, Ti);
   R = ChooseNextRole(valid_roles(Ti));
   while R is not Null do
      assigned_role(Ti) = R;
      assigned(Ti) = True;
      if Tk == Null then
        { RTplan = CreateNewRTplan(Tlist);
        AddRoleTaskPlans(AllRTplans, RTplan);
        result = Success; }
      else if RoleAssignement(Tk) == Success then
            result = Success;
      R = ChooseNextRole(valid_roles(Ti));
   endwhile;
   assigned(Ti) = False;
   return result;
end
```

Figure 3. The role-task planning algorithm

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 role 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 <assigned_role(T_j), T_j> 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.

The complexity of the role-task planning algorithm is $\boldsymbol{O}(m^n)$ in worst case, which m is the maximum number of capable_roles of all tasks and n is the number of tasks in the workflow.

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 Chapter 3, 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

Figure 4 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, $<R_i, T_i>$, 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, $<U_i, R_i, T_i>$, 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 Figure 3, 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*. The complexity of the user-role-task planning algorithm is $\boldsymbol{O}(Q^n)$ in worst case, which Q is the maximum number of capable_users of all role-task assignments and n is the number of tasks in the workflow.

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, Ri: role, Ti: task) : Boolean
begin
  valid_users(Ri, Ti) = { u | u ∈ capable_users(Ri)};
  for each task Tj where Tj ∈ Tlist , Tj ~Ti and assigned(Rj, Tj) == True do
    if Tj ⊕ Ti then valid_users(Ri, Ti) = valid_users(Ri, Ti) − assigned_user(Rj, Tj)
 endfor;
  <Rk, Tk> = NextRoleTaskFromRTPlan(RTPlan, <Ri, Ti>);
  U = ChooseNextUser(valid_users(Ri, Ti));
  while U is not Null do
     assigned_user(Ri, Ti) = U;
     assigned(Ri, Ti) = True
     if <Rk, Tk> == Null then
       { URTplan = CreateNewURTplan(RTplan)
         return Success; }
      else if UserAssignement(RTplan, Rk , Tk) == Success then return Success;
      U = ChooseNextUser(valid_users(Ri, Ti));
 endwhile;
 assigned(Ri, Ti) = False;
 return Fail;
end
```

Figure 4. The user-role-task assignment algorithm

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. The complexity of the plan-adjust algorithm is $\boldsymbol{O}(Q^{n-k+1})$ which Q is the maximum number of capable_users of unassigned role-task assignments, n is the number of tasks in the workflow and k is from the k_{th} task need to be reassigned tasks.

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 activation 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

Figure 5. Plan adjust algorithm in run-time phase

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, ..., T_n, where T_1, T_2, ... , T_{k-1} have been activated and T_k is the current task to be activated. Let $<AU_i, AR_i, T_i>$ 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 $<PU_k, PR_k, T_k>$. 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 Figure 5, finds another valid *URTplan,* called *NUplan,* from *URTplanSet,* where *NUplan* follows the actual activation of task T_i, $<AU_i, AR_i, T_i>$, for i = 1 to *k-1*, such that the activation of current task T_k, $<U_k, R_k, T_k>$, satisfies the constraints of dynamic SoD variations. The *RTplan-adjust()* algorithm finds another valid *RTplan,* called *NRplan,* from *RTplanSet,* where *NRplan* follows the actual activation of task T_i, $<AR_i, T_i>$, 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, $<AU_i, AR_i, T_i>$, for i = 1 to *k-1*, by invoking the *UserAssignment*(*NRplan*, R_k, T_k) algorithm. The activation of T_k, $<U_k, R_k, T_k>$, in *NUplan*, must satisfy the constraints of dynamic SoD variations.

4.4. Illustrative examples

In this section, we use an example to illustrate how the algorithms perform in the assignment of tasks to roles and users. Figure 6 shows a workflow *W* which contains six tasks, from T_1 to T_6. After the task T_2, the execution is split into two paths. For the case of AND-split, two parallel paths will be executed concurrently; as a result, T_3, T_5 and T_4 will be executed in the same workflow instance, and thus may have execution-dependency. For the case of XOR-split, only one path will be executed; the execution of T_4 excludes the execution of T_3 and T_5, and thus there is no execution dependency between T_4 and T_3 (T_5). The algorithms proposed in Section 4.1.1 take into account the AND/XOR split structure in the assignment of tasks to roles/users.

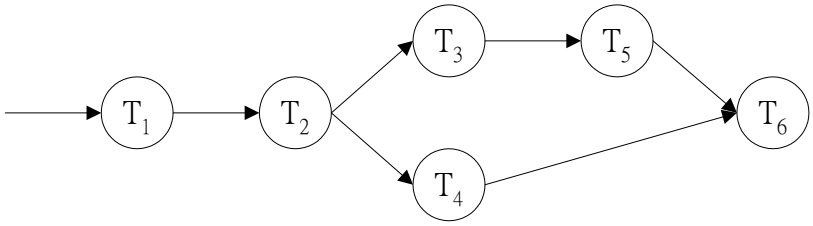

Figure 6. An example of workflow W

Table 3 shows the capable roles that can perform each task and the duty-conflict relationships among tasks of W. The role hierarchy is that R_p has a higher position than (supervises) R_x, R_y and R_z; R_x, R_y and R_z have higher positions than (supervise) R_a, R_b, R_c and R_d.

Table 3. Capable roles of each task and duty-conflict relationships in workflow W

Task	Capable roles	Duty-conflict relationship
T_1	R_a, R_b, R_x, R_y, R_z, R_p	$T_1 \oplus T_2$
T_2	R_a, R_x, R_c, R_d, R_y, R_z, R_p	$T_2 \oplus T_1$; $T_3 \succ T_2$; $T_4 \succ T_2$
T_3	R_x, R_y, R_z, R_p	$T_3 \succ T_2$; $T_3 \oplus T_5$
T_4	R_x, R_y, R_z, R_p	$T_4 \succ T_2$; $T_6 \succ T_4$
T_5	R_x, R_y, R_z, R_p	$T_3 \oplus T_5$; $T_6 \succ T_5$
T_6	R_p	$T_6 \succ T_4$; $T_6 \succ T_5$

The role-task planning algorithm first determines the execution dependency among tasks of W, as described in Section 4.1. Notably, the AND/XOR split structure is considered. The tasks of W are then assigned in topological order, T_1, T_2, T_3, T_4, T_5, T_6. When assigning the current task, the algorithm checks each previously assigned task with an execution dependency on the current task, to determine the roles that can be validly assigned to the current task. T_1 is assigned to role R_a and *assgned(T_1)* is set to true. T_2 has a duty-conflict relationship with T_1. Consequently, the *valid_roles(T_2)* do not include role R_a assigned to task T_1. The valid roles for T_2 are R_x, R_c, R_d, R_y, R_z and R_p. R_x is first assigned to T_2. The assignments of T_3 and T_4 are different in the cases of AND-split and XOR-split structures.

Suppose that the split structure is XOR-split. *XOR-SPLIT* structure splits into multiple mutually exclusive alternative paths (XOR-paths), only one of which is executed. Thus, no execution dependency exists among T_4 and T_3, as well as T_4 and T_5. There is no need to consider the duty-conflict relationships between tasks that have no execution dependency. Since task T_3 supervises task T_2, the role assigned to T_3 must have a higher position than the role assigned to T_2. Consequently, T_3 is assigned with role R_p. In the assignment of T_4, only the role assigned to T_2 is considered to determine the valid role of T_4, since T_4 has execution dependency with T_2, and T_4 does not have execution dependency with T_3. T_4 is assigned with role R_p, since T_4 supervises T_2. In the following steps, the valid roles of T_5 are R_x, R_y and R_z, which are derived by excluding the role R_p assigned to T_3. However, no valid role can be found for T_6, under the assignment of T_4 with R_p, since T_6 supervises T_4 and T_5. The algorithm backtracks to try another valid role of previous assigned task. By backtracking to the assignment of T_2, the algorithm chooses a valid role R_c of T_2. In the following recursive call of *RoleAssignment()* function, the valid roles of T_3 are R_x, R_y, R_z and R_p. R_x is chosen for T_3. The valid roles of T_4 are R_x, R_y, R_z and R_p. Notably, for AND-split structure, the valid roles of T_4 will be R_y, R_z and R_p, excluding the role R_x assigned to T_3. R_x is chosen for T_4. Next, the valid roles of T_5 are R_y, R_z and R_p, which exclude the role R_x assigned to T_3. R_y is chosen for T_5. Next, the valid role of T_6 is R_p, which satisfy the SoD constraints. One possible valid *RTplan* has been found, i.e., *RTplan* = {< R_a, T_1>, < R_c, T_2>, < R_x, T_3>, < R_x, T_4>, < R_y, T_5>, < R_p, T_6>}. The algorithm can continue to find another valid *RTplan* = {< R_a, T_1>, < R_c, T_2>, < R_x, T_3>, < R_x, T_4>, < R_z, T_5>, < R_p, T_6>}, by assigning T_5 with the valid role R_z. The algorithm can find all valid RTplans by exploring all valid roles of each task.

The *RTplans* generated by the role-task planning algorithm are used as the input to the user-role-task planning algorithm. The user-role-task planning algorithm finds valid users for each pair of (role, task) assignment in a *RTplan* by employing the similar approach of the role-task planning algorithm. Table 4 shows the capable users who have the capability to play the role. The assignment is similar to the assignment of the role-task planning algorithm, and thus the detailed illustrations of the assignment are omitted. The user-role-task planning algorithm finds one possible *URTplan* shown as the following: *URTplan* = {< Annie, R_a, T_1>, < Bob, R_c, T_2>, < Frank, R_x, T_3>, < Gary, R_x, T_4>, < Gary, R_y, T_5>, < Sam, R_p, T_6>}. Similarly, the algorithm can find all valid *URTplans* by exploring all valid users of each pair of (role, task) assignment.

Table 4. Capable users of roles.

Role	Capable users of a role
R_a	Annie, Bob, Calla, Gary, John, Sam
R_b	Annie, Bob, David, Gary, John, Sam
R_c	Bob, Calla, David, Kevin, Mary, Nancy, Tom
R_d	Calla, David, Ella, Kevin, Mary, Nancy, Tom
R_x	Frank, Gary, John, Sam
R_y	Gary, John, Kevin, Mary, Sam
R_z	Gary, John, Kevin, Nancy, Tom
R_p	Sam, Tom

Chapter 5. Authorization Management for Process-View

Adopting workflow management systems (WfMS) to manage business processes is an important trend in modern enterprises. The new value of WfMS for enterprise is agility, flexibility and visibility. WfMS can help decision makers fully utilize business processes. However, workers (representing organizational roles) cannot easily obtain a global view of a complex and large workflow.

A process-view, an abstracted process derived from a base process, is proposed to provide adaptable task granularity in previous related work [26]. Process-view is a good solution that different workflow participants can acquire different needs and types of authority. For example, a general manager may require aggregated information on a specific process rather than detailed information. In additional, an accounting manager may not have the authority or need to know each specific step of the production flow. These requirements necessitate the need to design a novel model for assigning the authority of process-oriented views of business tasks to organizational roles.

In base processes, access rights are associated with roles, and users are assigned to appropriate roles. The access rights/authorized permissions of roles for process-views are not considered in existing researches. This work discusses the authorization management of organizational roles in a process-view. Several base activities are aggregated into a virtual activity in a process-view. An organizational role *r*'s permissions on a virtual activity va_i can be derived from *r*'s permissions on those base activities belong to va_i. Moreover, the derivation needs to consider the duty-conflict relationships among base activities.

5.1. Grouping and data aggregations

The derivation of a virtual process (process-view) involves grouping the base activities in a base process into virtual activities. A virtual activity contains a set of base activities in the base process. Base activities may manipulate some data objects. Data aggregations may be specified on a virtual activity to provide aggregate views of data derived from the data objects of base activities. In general, data aggregations may apply aggregate functions on collections of data objects manipulated by base activities of a virtual activity. These aggregate functions are used in simple statistical computations,

including SUM, AVERAGE, MAXIMUM and MINIMUM, that summarize information from data objects handled by the base activities. Different roles may have different needs of data aggregations that are defined by the process modeler.

Let va_i represent a virtual activity and a_j denotes a base activity. $\mathbf{D}^r(va_i)$ denotes the aggregate data object of va_i for role r; $\mathbf{D}(a_j)$ denotes the data object handled by a_j, i.e., data object of a_j, for brevity; $S^r_{va_i}$ is the set of base activities that are specified in the data aggregation of va_i for role r; $DS^r_{va_i}$ is the set of data objects of base activities in $S^r_{va_i}$ used to derive the aggregate data object of va_i for role r; and $\mathbf{F}$ denotes an aggregate function.

$S^r_{va_i} = \{ a_j \mid a_j \in va_i : a_j$ is specified in the data aggregation of va_i for role $r \}$

$DS^r_{va_i} = \{ \mathbf{D}(a_j) \mid a_j \in S^r_{va_i} \}$

The formal expression of aggregating the data objects of base activities to derive the data object of va_i for role r is illustrated in the following.

$\mathbf{D}^r(va_i) = \mathbf{F}(\{\mathbf{D}(a_j) \mid a_j \in S^r_{va_i}\})$ i.e., $\mathbf{D}^r(va_i) = \mathbf{F}(DS^r_{va_i})$

Notably, $S^r_{va_i}$ may include all or partial base activities in va_i, as specified in the data aggregation for role r defined by the process modeler. The aggregate function may apply to some or all data objects of base activities in va_i.

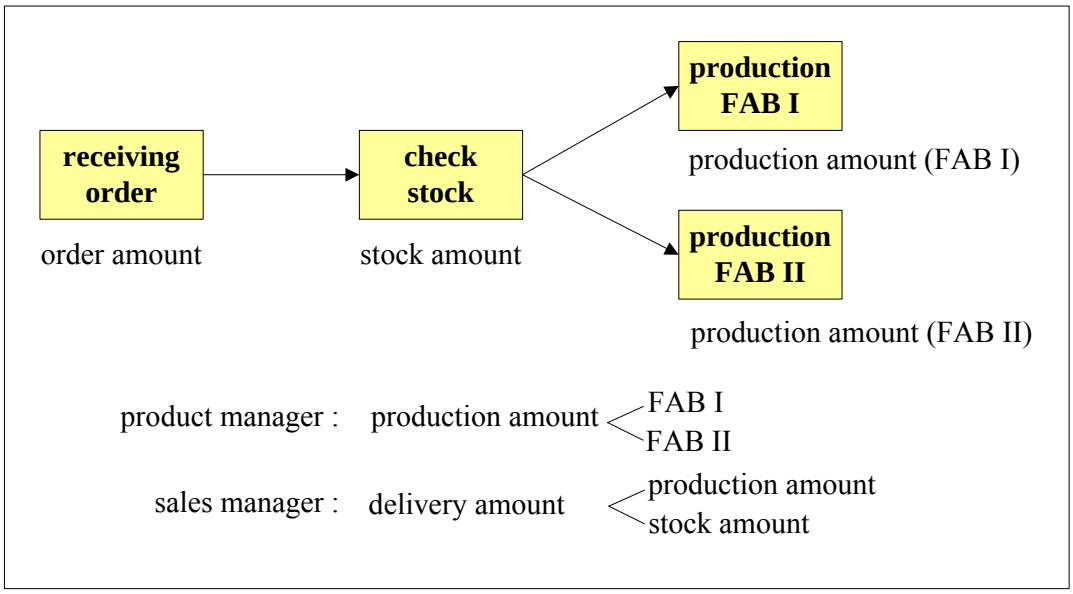

Figure 7. Example of a virtual activity "scheduling production"

Figure 7 illustrates a virtual activity "scheduling production" which represents an abstraction of four base activities "receiving order", "check stock", "production FAB I" and "production FAB II". Only production data are described for clarity. The data objects of "receiving order", "check stock", "production FAB I" and "production FAB II" are order amount, stock amount, production amount (FAB I), and production amount (FAB II), respectively. The sales manager only needs to know the delivery amount, including production distribution and stock amount but not the production details.

$$DS_{va_i}^{sales\ manager} = \{\ \mathbf{D}(\text{production FAB I}), \mathbf{D}(\text{production FAB II}), \mathbf{D}(\text{stock amount})\}$$

The production manager may want to know the production amount but not the stock amount.

$$DS_{va_i}^{product\ manager} = \{\ \mathbf{D}(\text{production FAB I}), \mathbf{D}(\text{production FAB II})\}$$

5.2. The permissions on an activity

A role is a job function defined as a named collection of responsibilities, which reflect organizational regulations and business procedures. Several permissions on an activity are defined and illustrated in Table 5. A role is assigned a collection of the permissions on base activities. The permissions of role *r* on a virtual activity va_i can be derived from role *r*'s permissions on base activities in va_i.

Table 5. Permissions on activity

Permissions on activity a_j	**Descriptions**	**Implied Permissions on activity a_j**
manage	Manage; Read the data object of a_j	*view, agg_view, awareness*
execute	Execute; Read/Write the data object of a_j	*view, agg_view, awareness*
view	Read the data object of a_j	*agg_view, awareness*
agg_view	Read aggregate data object	*awareness*
awareness	Be aware of a_j	*null*

For a role r, given a permission p on an activity a_j, the access privilege of role r is described as follows.

- *manage*: role r can monitor the progress of a_j and reassign a user to perform a_j; role r can read the data object of a_j.
- *execute*: role r can execute a_j and read/write the data object of a_j.
- *view*: role r can read the data object of a_j.
- *agg_view*: role r can read $\mathbf{D}^r(va_i)$, the aggregate data object of a virtual activity va_i, only if role r has the *agg_view* permission on all base activity $a_j \in S^r_{va_i}$. Notably, $S^r_{va_i}$ is the set of base activities specified in the data aggregation of va_i for role r.
- *awareness*: the basic and minimum permission for an activity; role r is aware of a_j.

Notably, if role r only has the *agg_view* permission without the *view* permission on a_j, then role r can not read the data object of a_j.

Moreover, the permissions have implied relationships as shown in Table 5. A role r has the *manage* or *execute* permission on an activity a_j implied that role r also has the *view*, *agg_view* and *awareness* permissions on a_j. A role r possesses the *view* permission on an activity a_j implied that role r also has the *agg_view* and *awareness* permissions on this activity. Finally, a role r has the *agg_view* permission on an activity a_j implied that role r also has the *awareness* permission on a_j.

5.3. Permissions on a virtual activity without considering duty-conflict relationships among base activities

This section presents the derivations of the permissions of a role r on a virtual activity without considering duty-conflict relationships among the base activities. Next section presents the derivations considering the duty-conflict relationships among the base activities.

Let $\boldsymbol{P}(r, a_j)$ be the set of permissions of role r on an base activity a_j and $\boldsymbol{P}(r, va_i)$ be the set of permissions of role r on a virtual activity va_i. For strict privilege principle, $\boldsymbol{P}(r,va_i)$ can be derived by the intersection of permissions on base activities belong to va_i, as illustrated in the following.

$$P(r, va_i) = \bigcap_{\forall a_j \in va_i} P(r, a_j)$$

We use the example illustrated in Figure 2, Section 2.3, to illustrate the derivation. The base activities, a_1, a_2 and a_3 in the base process, are aggregated into the virtual activity, va_1, in the process-views. If $\boldsymbol{P}(r, a_1)$, $\boldsymbol{P}(r, a_2)$, $\boldsymbol{P}(r, a_3)$ is {*manage*}, {*execute*} and {*view*}, respectively. According to the strict privilege principle, $\boldsymbol{P}(r, va_1)$ are the intersection of permissions on base activities belong to va_1, which results in {*view*}. Notably, the implied permissions shown in Table 5 should be considered in deriving the permissions on a virtual activity.

Above derivation may be too strict for data aggregation, since the derivation shows that if a role r does not have *agg_view* permission on all base activities in va_i, then the permissions of role r on virtual activity va_i will not contain *agg_view* permission. However, data aggregation may be specified on part of base activities in va_i. A role r should be able to read $\mathbf{D}^r(va_i)$, the aggregate data object of a virtual activity va_i, if role r has the *agg_view* permission on all base activity $a_j \in S^r_{va_i}$, as described in the following equation.

agg_view $\in \boldsymbol{P}(r, va_i)$, if $\boldsymbol{P}(r, a_j)$ contains *agg_view* permission for all $a_j \in S^r_{va_i}$

If a role r has the *agg_view* permission but not the *view* permission on a base activity a_j in a virtual activity va_i; the *agg_view* permission on va_i is derived for role r. Role r may deduce the data object of a_j that r does not have the permission to view. For example, a virtual activity, va_1, is aggregated from two base activities, a_1 and a_2. A personal computer manufacturer contains two factories. Each factory reports the amount of product to head office, where a_1 reports the amount of product in factory one, two thousands PCs, and a_2 reports the amount of product in factory two, three thousands PCs. The va_1 reports the total amount of product in both factories, i.e. five thousands PCs. However, the permission of a role r can only know the amount of product in factory one, and not factory two. Role r should not have the permission to know the total amount of product in both factories, since the amount of product in factory two may be deduced. Accordingly, the derivation of the *agg_view* permission on a virtual activity is modified as the following, by considering the data deduction rule.

Equation (1):

$$P(r, va_i) = \bigcap_{\forall a_j \in va_i} P(r, a_j)$$

$agg\_view \in \boldsymbol{P}(r, va_i)$, if $\boldsymbol{P}(r, a_j)$ contains *agg_view* permission for all $a_j \in S^r_{va_i}$;

and the data deduction rule is satisfied.

***Data deduction rule*:**

There is no data deduction on the data object of a_j for $a_j \in S^r_{va_i}$ and $view \notin \boldsymbol{P}(r, a_j)$.

Some enterprises adopt lenient privilege principle to increase the convenient and flexibility in process management. Least privilege principle is to make sure that only those permissions required for the activities conducted by members of the role are assigned to the role. For least privilege principle, the permissions of role *r* on a virtual activity are the necessary access rights defined by the process modeler.

5.4. Permissions on a virtual activity considering duty-conflict relationships among base activities

Section 3.1 has defined several duty-conflict relationships among tasks (activities). A base process contains a set of tasks, and some duty-conflict relationships may exist among tasks. The derivation of a role *r*'s permissions on a virtual activity needs to consider duty-conflict relationships among base activities.

For strict privilege principle, if the base activities a_x and a_y are duty-conflict tasks, $a_x \oplus a_y$; virtual activity va_i only contains two base activities a_x and a_y, the permission of a role *r* on virtual activity va_i is illustrated in Table 6.

Two base activities a_x and a_y are duty-conflict activities that are aggregated as a virtual activity va_i. According to the authorization rules for SoD defined in section 3.2, two duty-conflict tasks cannot be assigned to the same role. If role *r* has the *execute* permission on both base activities a_x and a_y, then only the minimum and basic permission "*awareness*" on va_i can be authorized to role *r*. Moreover, under the strict privilege principle, if role *r* has the *view* permission on both base activities a_x and a_y, then only the "*awareness*" permission on va_i can be authorized to role *r*. Notably, the implied permissions should be considered in the authorization.

Table 6. Permissions of role *r* under strict privilege principle

$a_x \oplus a_y$		**Permission on va_i** $va_i = \{a_x, a_y\}$
Permission on a_x	**Permission on a_y**	
execute	*execute* *manage* *view* *agg_view* *awareness*	*awareness* *awareness* *awareness* *agg_view* (if data deduction rule is satisfied) *awareness* (if data deduction rule is violated) *awareness*
manage	*manage* *view* *agg_view* *awareness*	*awareness* *awareness* *agg_view* (if data deduction rule is satisfied) *awareness* (if data deduction rule is violated) *awareness*
view	*view* *agg_view* *awareness*	*awareness* *agg_view* (if data deduction rule is satisfied) *awareness* (if data deduction rule is violated) *awareness*
agg_view	*agg_view* *awareness*	*agg_view* (if data deduction rule is satisfied) *awareness* (if data deduction rule is violated) *awareness*
awareness	*awareness*	*awareness*

For the case that role *r* has the *view* permission on a_x and the *agg_view* permission on a_y, the *agg_view* permission on va_i can be authorized to role *r*, if the data deduction rule is satisfied; otherwise, only the "*awareness*" permission on va_i can be authorized to role *r*. The data deduction rule states that there is no data deduction on the data object of a_j for $a_j \in S^r_{va_i}$ and *view* $\notin \boldsymbol{P}(r, a_j)$. A role should not deduce some unauthorized permissions on the data objects. For example, three base activities are aggregated into a virtual activity va_i. If role *r* has the *view* permission on one base activity a_x and the *agg_view* permission on the other two base activities a_y and a_z, the *agg_view* (e.g. SUM aggregation) permission on va_i can be authorized to role *r*, since role *r* can not deduce the data information of a_y and a_z. Nevertheless, if a role has the *view* permission on a_x and a_y; and the *agg_view* permission (no *view* permission) on a_z, the *agg_view* (e.g. SUM aggregation) permission on va_i can not be authorized to role *r*, since role *r* can deduce the

data information of a_z. The other derivations are similar, and thus are omitted for clarity.

In order to increase the flexibility, an organization may adopt lenient privilege principle. For lenient privilege principle, if the base activities a_x and a_y are duty-conflict tasks, $a_x \oplus a_y$; virtual activity va_i only contains two base activities a_x and a_y, the permission of a role r on virtual activity va_i is illustrated in Table 7.

Table 7. Permissions of role r under lenient privilege principle

$a_x \oplus a_y$		**Permission on va_i** $va_i = \{a_x, a_y\}$
Permission on a_x	**Permission on a_y**	
execute	*execute*	*awareness*
	manage	*awareness*
	view	*awareness*
	agg_view	*agg_view* (if data deduction rule is satisfied) *awareness* (if data deduction rule is violated)
	awareness	*awareness*
manage	*manage*	*awareness*
	view	*view*
	agg_view	*agg_view* (if data deduction rule is satisfied) *awareness* (if data deduction rule is violated)
	awareness	*awareness*
view	*view*	*view*
	agg_view	*agg_view* (if data deduction rule is satisfied) *awareness* (if data deduction rule is violated)
	awareness	*awareness*
agg_view	*agg_view*	*agg_view* (if data deduction rule is satisfied) *awareness* (if data deduction rule is violated)
	awareness	*awareness*
awareness	*awareness*	*awareness*

The main difference between the strict and lenient privilege principles is to loosen the view-view violation. Under the strict privilege principle, if role r has the *view* permission on both base activities a_x and a_y, then only the "*awareness*" permission on va_i can be authorized to role r. For the lenient privilege principle, if role r has the *view* permission on both base activities a_x and a_y, then the "*view*" permission on va_i can be authorized to role r. If role r has the *manage* permission on a_x and the *view* permission on

a_y, then the "*view*" permission on va_i can be authorized to role *r*. However, if role *r* has the *execute* permission on both base activities a_x and a_y, then only the "*awareness*" permission on va_i can be authorized to role *r*, in order to achieve the constraint of SoD.

As described in section 5.2, if the "*execute*" permission on a_x is authorized to role *r*, then the implied permissions are also assigned to role *r*, i.e., $\boldsymbol{P}(r, a_x)$ = {*execute*, *view*, *agg_view*, *awareness*}. If role *r* also has the "*execute*" permission on a_y, then the "*view*" permission on va_i is derived for role *r*, according to Table 7, which violates the SoD principle. To ensure no violation of SoD, the maximum privilege of role *r*'s permissions should be used to derive role *r*'s permissions on va_i. The privileges of the permissions are ranked as follows: *execute* > *manage* > *view* > *agg_view* > *awareness*. Let $\max P^r_{a_j}$ denote the maximum privilege of role *r*'s permission on activity a_j.

$\max P^r_{a_j}$ = the higest ranked permission in $\boldsymbol{P}(r, a_j)$

For duty conflict tasks, the maximum privileges of role *r*'s permissions on base activities are used to derive role *r*'s permissions on virtual activity, according to Table 7.

Table 7 shows the derivation of role *r*'s permission on a virtual activity that contains two duty-conflict activities. The derivation for general cases that a virtual activity may contain a set of base activities with duty-conflict relationships is described as Figure 8. The algorithm creates a temporary virtual activity, $tmpva_j$ for each pair of duty-conflict tasks a_x and a_y in va_i, where $tmpva_j$ = { a_x, a_y }. Then role *r*'s permissions on $tmpva_j$ are derived according to Table 7 by using the maximum privilege of $\boldsymbol{P}(r, a_x)$ and $\boldsymbol{P}(r, a_y)$ (i.e. $max\, P^r_{a_x}$ and $max\, P^r_{a_y}$), respectively. Once role *r*'s permissions on duty-conflict tasks are derived, the algorithm then uses equation (1) described in Section 5.3 to derive role *r*'s permissions on the virtual activity va_i.

Algorithm The algorithm of derivation for general cases

$tmpVA = \{\}$

for each pair of duty-conflict tasks $a_x \oplus a_y$ and $a_x, a_y \in va_i$

Create a temporary virtual activity $tmpva_j = \{ a_x, a_y \}$

$P(r, tmpva_j)$ = the set of permissions derived according to Table 7,

by using the $max\, P^r_{a_x}$ and $max\, P^r_{a_y}$

$tmpVA = tmpVA \cup \{ tmpva_j \}$

endfor;

for each base activity $a_x \in va_i$

if there is no $a_y \in va_i$ such that $a_x \oplus a_y$ ***then***

$tmpVA = tmpVA \cup \{ a_x \}$

endfor;

$\boldsymbol{P}(r, va_i) = \bigcap_{\forall a_j \in tmpVA} P(r, a_j)$

if $\boldsymbol{P}(r, a_j)$ contains *agg_view* permission for all $a_j \in S^r_{tmpVA}$; and the data deduction rule is satisfied.

$\boldsymbol{P}(r, va_i) = \boldsymbol{P}(r, va_i) \cup \{ agg\_view \}$

Figure 8. The algorithm of derivation for general cases

Chapter 6. System Implementation and Demonstration

This chapter will elucidate the implementation, demonstration and discussion of the system.

6.1. System implementation

Various authorization rules are incorporated into the system to achieve separation of duty 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. Figure 9 depicts the system architecture which integrates a workflow management system.

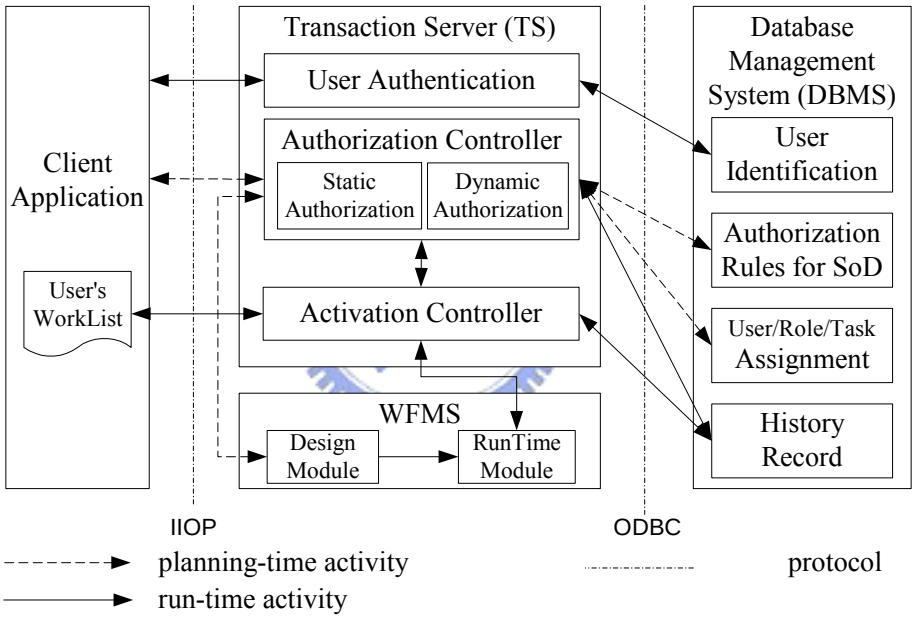

Figure 9. The system architecture

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 workflow management system (W*f*MS) 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 Chapter 4, to generate initial workflow activation plans. The run-time module is responsible for the enactment of 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 W*f*MS. The run-time module of W*f*MS 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 W*f*MS. Finally, the W*f*MS allows the user to execute the authorized task.

Sybase's EAS (Enterprise Application Server) 3.0 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, and a database management system (DBMS). The prototype system is a three-tier architecture with the transaction server (TS) as the middle tier between the client application and the DBMS server. The client application program provides the interface between the user and the system. The DBMS stores authorization data and process/task data. The system is implemented with Windows NT 4.0 as the server side and Windows 2000 as the client side. Rational Rose 4.0 is used to analyze and design the system, while PowerDesign 7.0 is used to develop databases managed by Sybase SQLAnywhere 6.0 DBMS. PowerBuilder 7.02 is used to develop client applications and the modules in the transaction server. Notably, the application program on the client side communicates with the Transaction Server 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.

6.2. 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 Figure 10.

The authorization management mainly supports the specification of authorization data necessary for authorization control. The specification is conducted in the planning phase. Figure 10 reveals that “Issuing item-request” and “Approving item-request” are tasks in the “Submitting purchase request” sub-process. “Approving item-request” supervises “Issuing item-request”. An initial workflow activation plan of the sub-process, “Submitting purchase request”, generated in the planning phase according to the user/role-task planning algorithms, 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 system provides the login user with a GUI to conduct role/task activations. The activation controller provides the user with the authorized role list and the assigned task list. Additionally, 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.

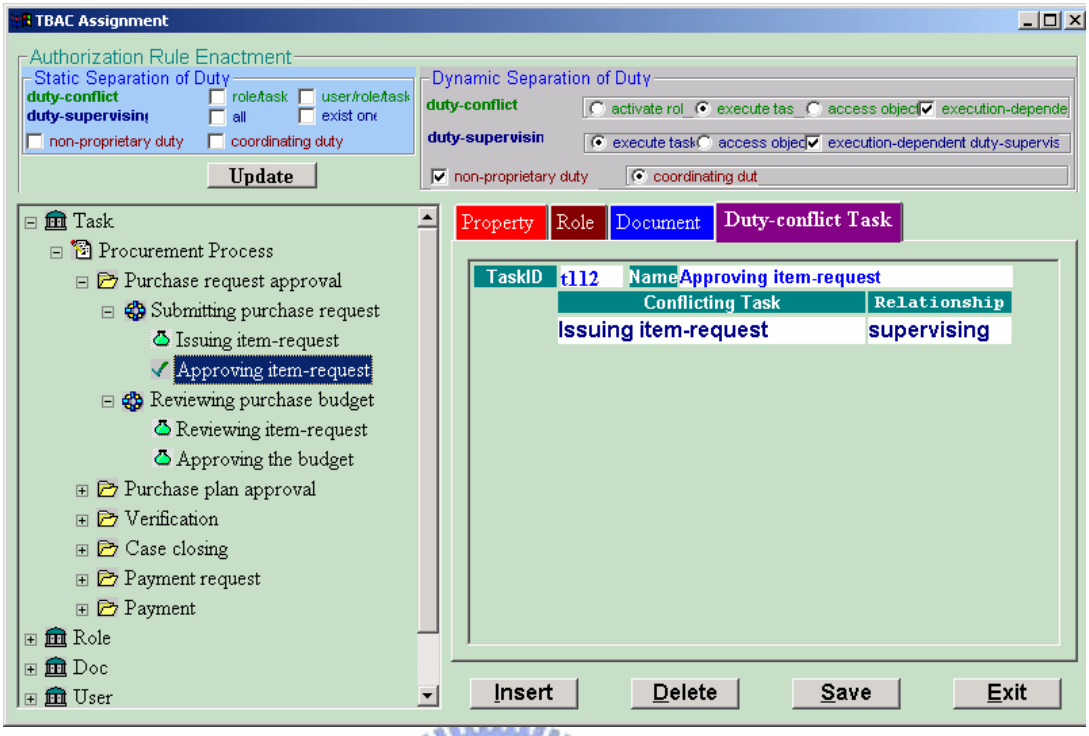

Figure 10. Enactment of data for authorization control

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). Figure 11 indicates that John is authorized to play the roles of the assistant manager and the clerk of the human resource department. John has successfully performed the task “Issuing item-request’ on issuing a request to purchase computers (CPU P4 1.4GHz, wf- no. 135). Figure 12 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 1.4GHz, 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.

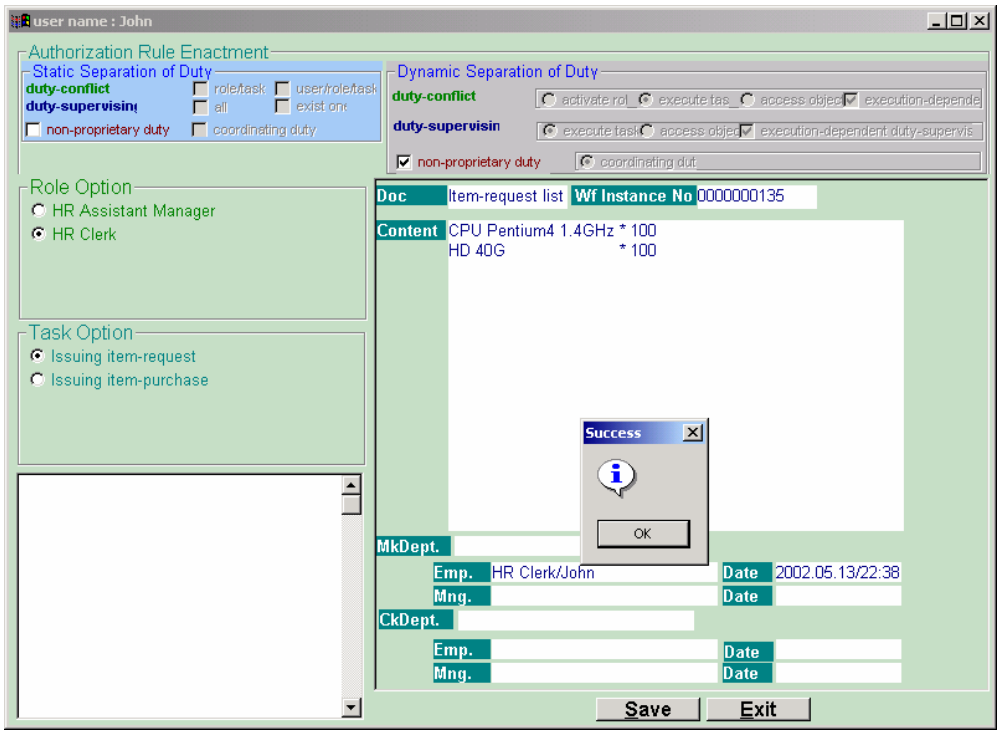

Figure 11. Activation of "Issuing item-request" by John as Clerk

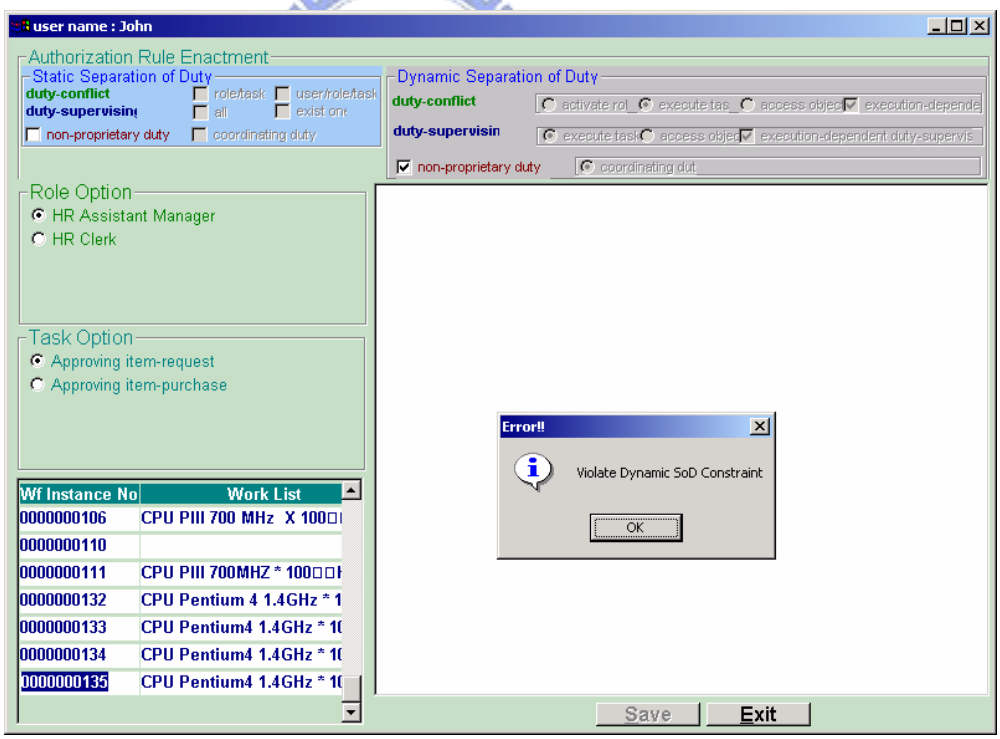

Figure 12. Verifying SoD in the activation of "Approving item-request" by John as assistant manager

6.3. Discussion

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 organization's understanding of the duties associated with roles/users. There is no system means and verification to ensure that the principle of separation of duty 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 separation of duty.

(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.

Chapter 7. Comparison with Related Work

Existing researches include role/task-based security model for SoD, task-based authorization model, and role-based authorizations for workflows. Thomas et al. [29] proposed task-based authorization control to manage the execution states of tasks by controlling the run-time execution status of tasks. They considered neither SoD nor authorizations among tasks, roles and users. Schier [23] also proposed a role and task based security model. Although authorization rules for SoD have been designed, they are merely derived from SoD in RBAC. The definition of mutual exclusive tasks is simply derived from the definition of mutual exclusive roles. Duty-conflict relationships between tasks have not been explored. In addition, the proposed authorization rules for SoD are merely extended from SoD in RBAC. They considered neither execution dependency nor role-based authorizations for workflow tasks. In contrast, this work provides a novel analysis and defines various duty-conflict relationships among tasks. Various authorization rules for execution-dependent SoD have been proposed.

Although several researchers have addressed role-based access control and authorization management in workflow systems, few have considered authorization planning in assigning workflow tasks to roles/users. Bertino et al. [5] proposed a flexible model for the specification and enforcement of authorization constraints in workflow management systems. A logical authorization language, defined as clauses in a logic program, is proposed to express authorization constraints on role assignments and user assignments. Deductive approach is then used to conduct consistency checking on the logical constraints. Moreover, algorithms have been proposed for authorization planning in assigning users and roles to workflow tasks such that no authorization constraints are violated. The comparison of their work with ours can be elucidated as follows. First, although examples have been presented to illustrate how to express static and dynamic SoD via the proposed authorization language, 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. Deductive inference needs to check all constraints to find if there is inconsistency. 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. Only tasks that are duty-conflict and execution dependent need to be verified. 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. [1] developed a system architecture for enforcing role-based access control in Web-based workflow management systems. The architecture mainly consists of a role server for maintaining user-role assignments and issuing certificates with client's role information. Role-based authorization is conducted as follows. The client needs to request a client certificate with role information, implemented as an X.509v3 certificate with role attributes, and presents it to the Web server of the workflow system. The Web server then retrieves role information from the certificate to verify if the client has the privileges to execute the task by the role. Detailed implementation has been presented to show the feasibility of the proposed system. However, the proposed role-based authorization is still based on the simple RBAC96 model.

Atluri and Huang proposed a Workflow Authorization Model (WAM) for workflows [3]. The model associates each task with authorization templates that specify static parameters of authorization defined during the design time. When a task of a workflow instance starts to execute in run time, the actual authorization of granting a subject to execute the task is derived from the authorization templates. The WAM model has also been enhanced to incorporate separation of duty constraints. Huang and Atluri [13] also presented a secure Web-based workflow management system (SecureFlow). The SecureFlow system is developed based on the WAM. A workflow authorization server, which is separated from the W*f*MS, is employed to support the specification and enforcement of security policies based on role-based access control and separation of duty. In addition, a simple 4GL language is used to specify authorization constraints.

Botha and Eloff [6] presented access control requirements in document-centric workflow systems. A Context-sensitive Access Control model, which is based on

role-based access control, is proposed to protect unauthorized access to documents (sensitive information) used in workflow systems. The model considers conflicting tasks, conflicting users (e.g. family members) and access history of document in supporting dynamic SoD requirements. Moreover, an agent-based approach is used to implement the proposed model.

As the demand of business globalization increases, inter-organizational workflows are gaining importance in collaborative business environments. From this aspect, access control mechanisms and security models have been proposed for inter-organizational workflows [2][16]. Kang et al. [16] 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 workflow. To participate in the inter-organizational workflow, an organization needs to map its role structure to the role domain for the workflow. The role domain approach decouples the workflow-specific security structure from an organization's security structure. X.509 certificate is used to provide user identity and role/organization information. The mechanism also supports context-based access control, in which data access is enforced according to the capability (context) of each task, i.e., read/write permissions on fields of data objects. Moreover, Atluri et al. [2] considered the issues of conflict-of-interest among competing organizations of inter-organizational workflows in decentralized workflow environments. The model mainly prevents sensitive dependency information or sensitive output of a task leaking to another task agent (organization) with conflict-of-interest.

The comparisons of our work with above literatures [1][2][3][6][13][16] 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. The user/role/task planning algorithms generate initial workflow activation plans in advance, which assign tasks to a set of valid roles/users, to satisfy the constraints of SoD. The plan-adjust algorithm identifies an available user authorized to activate the current task, and generates a new activation plan. 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. Inter-organizational workflows are gaining importance in B-to-B commerce. 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.

Chapter 8. Conclusions and Future Works

8.1. Summary

Authorization management and access control are essential in supporting secure workflow management systems. This work presents a novel analysis and defines various task-based SoD constraints, such as duty-supervising and execution-dependent SoD. The user/role/task planning algorithms in planning phase have been developed to generate initial workflow activation plans in advance, while the plan-adjust algorithm in run-time phase has been developed to determine an available user authorized to activate the current task. The proposed approach facilitates the effective authorization management of workflows to assign tasks to roles or users, while enforcing task-based SoD. Secure task-based access to workflow related data is enforced via effective authorization management.

Process-view is a good solution that different workflow participants acquire different needs and types of authority. This work analyzes the grouping and aggregate function of a virtual activity; and further, explains the permissions of a virtual activity in a process-view. Moreover, this work discusses the permissions for a role on a virtual activity aggregated from duty-conflict base activities.

Moreover, a prototype system has been developed to manage the authorization to perform tasks in workflow environments. 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 separation of duty 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 separation of duty and thus prevent fraud, by sacrificing flexibility and convenience.

8.2. Future Works

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 [18][24][25]. Future research will be to investigate the authorizations and access control in inter-organizational workflows.

References

[1] Ahn, G-J, Sandhu, R., Kang, M., Park, J., "Injecting RBAC to Secure a Web-based Workflow System", In Proceedings of 5th ACM Workshop on Role-Based Access Control, 2002.

[2] Atluri, V., Chun, S.A., Mazzoleni, P., "A Chinese Wall Security Model for Decentralized Workflow Systems", Proceedings of the 8th ACM conference on Computer and Communications Security, 2001.

[3] Atluri, V., Huang W-K, "An Authorization Model for Workflows", Proceedings of the fifth European Symposium on Research in Computer Security, Rome, Italy, pp. 44 – 64, 1996.

[4] Barkley, J., "Implementing Role Based Access Control Using Object Technology", First ACM Workshop on Role Based Access Control, November, 1995.

[5] Bertino, E., Ferrari, E., Atluri, V., "Specification and Enforcement of Authorization Constraints in Workflow Management Systems", ACM Transactions on Information and System Security, Vol. 2, No. 1, pp 65 – 104, 1999.

[6] Botha, R.A., Eloff, Jan H.P., "Access Control in Document-centric Workflow Systems — An Agent-based Approach", Computers & Security, Vol.20, No.6, pp.525-532, 2001.

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

[8] Coulouris, G., Dollimore, J., Roberts, M., "Role and Task-based Access Control in the PerDis Groupware Platform", Third ACM Workshop on Role-Based Access Control, George Mason University, VA, October, 1998.

[9] Ferraiolo, D.F., Cugini, J., Kuhn, R., "Role-Based Access Control (RBAC): Features and Motivations", Proceedings of 11th Annual Computer Security Application Conference, IEEE Computer Society Press, pages 241-248, December, 1995.

[10] Ferraiolo, D.F., Kuhn, R., "Role-Based Access Control", In Proceedings of 15th NIST-NCSC National Computer Security Conference, pages 554-563, October, 1992.

[11] Georgakopoulos, D., Hornick, M., Sheth, A., "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure", Distributed and Parallel Databases, pages 119-153, 1995.

[12] Gligor, V.D., Gavrila, S.I., Ferraiolo, D., “On the Formal Definition of Separation-of-Duty Policies and Their Composition”, Proceedings of IEEE Symposium on Security and Privacy, IEEE Computer Society, May, 1998.

[13] Huang, W-K, Atluri, V., “SecureFlow: A secure web-based workflow management system”, In Proceedings of 4th ACM Workshop on Role-Based Access Control, pages 83-94, Fairfax, VA, October, 1999.

[14] Joshi, J., Bertino, E., Shafiq, B., Ghafoor, A., “Dependencies and Separation of Duty Constraints in GTRBAC”, Proceedings of the eighth ACM Symposium on Access Control Models and Technologies (SACMAT’03), p51-63, June 2-3, 2003.

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

[16] Kang, M.H., Park, J.S., Froscher, J.N., “Access Control Mechanisms for Inter-organizational Workflow”, Sixth ACM Symposium on Access Control Models and Technologies (SACMAT 2001), May 3-4, 2001.

[17] Kappel, G., Lang, P., Rausch-Schott, S., Retschitzegger, W., “Workflow Management Based on Objects, Rules, and Roles”, IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 18/1, pp. 11-17, March, 1995.

[18] Koetsier, M., Grefen, P., Vonk, J., “Contracts for Cross-Organizational Workflow Management”, Proceedings of the 1st International Conference on Electronic Commerce and Web Technologies, pp. 110-121, London, UK, 2000.

[19] Lee, B.G., Narayanan, N.H., Chang K.H., “An Integrated Approach to Distributed Version Management and Role-based Access Control in Computer Supported Collaborative Writing”, The Journal of Systems and Software, 59 (2001), pp. 119-134, 2001.

[20] Li, N., Bizri, Z., Tripunitara, M.V., “On Mutually-Exclusive Roles and Separation of Duty”, Proceedings of the 11th ACM conference on Computer and Communications Security (CCS’04), p42-51, October 25-29, 2004.

[21] Nash, M.J., Poland, K.R., “Some Conundrums Concerning Separation of Duty”, Proceedings of IEEE Computer Society Symposium on Security and Privacy, IEEE Computer Society Press, May, 1990.

[22] Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman C.E., “Role-Based Access Control Models”, IEEE Computer, 29(2), pp.38-47, February, 1996.

[23] Schier, K., “Multifunctional Smartcards for Electronic Commerce — Application of the Role and Task Based Security Model”, 14th Annual Computer Security Applications Conference, December, 1998.

[24] Schulz, K.A., Orlowska, M.E., “Facilitating cross-organizational workflows with a workflow view approach”, Data & Knowledge Engineering, 51, p109-147, 2004.

[25] Shen, M., Liu, D.R., “Coordinating Interorganizational Workflows based on Process-Views”, Proceedings of the DEXA 2001 12th International Conference on Database and Expert Systems Applications, pp. 274-283, Munich, Germany, Sept., 2001, LNCS 2113, Springer-Verlag Berlin Heidelberg.

[26] Shen, M., Liu, D.R., “Discovering role-relevant process-views for disseminating process knowledge”, Expert Systems with Applications, 26, p301–310, 2004.

[27] Simon, R.T., Zurko, M.E., “Separation of Duty in Role-Based Environments”, 10th Computer Security Foundations Workshop, June 10-12, 1997.

[28] Strembeck, M., Neumann G., “An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments”, ACM Transactions on Information and System Security, Vol. 7, No. 3, p392-427, August 2004.

[29] Thomas, R.K., Sandhu, R.S., “Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management”, Proceedings of the IFIP WG11.3 Workshop on Database Security, August 11-13, 1997.

[30] Weitz, W., “Workflow modeling for Internet-Based Commerce: An Approach Based on High-Level Petri Nets”, Proceedings of International IFIP/GI Working Conference TREC’98, Hamburg, Germany, June 3-5, 1998.

[31] Wolf, R., Keinz, T., Schneider, M., “A Model for Context-dependent Access Control for Web-based Services with Role-based Approach”, Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA’03), 2003.

[32] Workflow Management Coalition, “Workflow Management Coalition: Workflow Reference Model”, Technical report WfMC TC-1003, Jan. 19, 1995.