JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 27, 1011-1028 (2011)

A Delegation Framework for Task-Role Based Access Control in WFMS*

HWAI-JUNG HSU AND FENG-JIAN WANG
Institute of Computer Science and Engineering
National Chiao Tung University
Hsinchu, 300 Taiwan
E-mail: {hjhsu@csie; fjwang@cs}.nctu.edu.tw

Access control is important for protecting information integrity in workflow management system (WfMS). Compared to conventional access control technology such as discretionary, mandatory, and role-based access control models, task-role-based access control (TRBAC) model, an access control model based on both tasks and roles, meets more requirements for modern enterprise environments. However, few discussions on delegation mechanisms for TRBAC are made. In this paper, a framework considering temporal constraints to improve delegation and help automatic delegation in TRBAC is presented. In the framework, the methodology for delegations requested from both users and WfMS is discussed. The constraints for delegatee selection such as delegation loop and separation of duty (SOD) are addressed. With the framework, a sequence of algorithms for delegation and revocation of tasks are constructed gradually. Finally, a comparison is made between our approach and the representative related works.

Keywords: delegation, task-role-based access control (TRBAC), workflow management system (WfMS), separation of duty (SOD), time constraints

1. INTRODUCTION

Workflow management systems (WfMS) systematically coordinate resources and business processes for modern enterprises [1], and regulates activities of employees through varieties of access control methods. Role-based access control (RBAC) model [2, 3] which groups users with similar permissions into roles is a popular solution for access control among enterprises. However, business processes are operated based on not only roles but also tasks. With both as core concepts, task-role-based access control (TRBAC) model provides more modeling power. Besides, in TRBAC, roles group tasks and users together, and permissions to business objects are bound with tasks. TRBAC prevents users from access business objects when not executing the corresponding tasks, and thus satisfies the confidentiality requirements of modern enterprises [4].

Delegation is to authorize subjects like access rights or works between users or roles, and is often built based on access control models. For example, RBDM0 [5], RBDM1 [6], and the methods in [7-10] describe various delegation models based on RBAC [2, 3]. On the other hand, tasks, the basic logic units in business processes, should also be considered for delegation in workflow systems. Some studies like [11, 12] extend the RBAC model with tasks, and develop respective delegation mechanisms for the relative access control models.

Received September 3, 2009; revised September 10, 2010; accepted December 6, 2010.
Communicated by Chih-Ping Chu.

* The preliminary work [13] of this study was presented in FTDC'08 (Kunming, China, October 21-23, 2008) by Dr. Feng-Jian Wang.

Delegation for TRBAC is also studied in some researches like [13, 14]. Jian *et al*. construct a framework and define the components for delegation in TRBAC [13]. Nevertheless, Jian's work [13] ignores the temporal issues, and is thus incomplete. In [14], Zhang *et al*. develop a delegation model with time constraints for TRBACM, a reduced TRBAC model. However, TRBACM violates the primary spirit of TRBAC because of binding roles with permissions and tasks separately.

In this paper, we refine the delegation framework from [13] by adopting a set of systematic operations for delegation in TRBAC. In our work, the temporal factors are considered as [15] does. The constraints like delegation loop and separation of duty (SOD) in delegatee selection are stated. Based on the discussions, a series of algorithms for delegation of a task instance and delivery of candidate delegatees from the role hierarchy are developed. With the algorithms, a user is able to delegate his work to the designated delegatee through an approval process, and WfMS can delegate an emergent work item to an appropriate delegatee automatically. The algorithm for users and WfMS to revoke a delegation is also established. Finally, we present a case study throughout our approach, and make a comparison between the existing methodologies and ours.

The rest parts of this paper are organized as follows. Related works in the past are presented in section 2. The workflow model adopted in this study is defined in section 3. In section 4, we construct the basic framework for delegation in TRBAC. The methods for delegation and revocation in TRBAC are described in section 5. In section 6, a case study is made to demonstrate our framework. Finally, we compare our approach with other related works, and make a conclusion in section 7.

2. BACKGROUNDS

2.1 Task-Role-based Access Control Model

Based on RBAC96 [2, 3], TRBAC model [4] illu strated in Fig. 1 works for modern enterprise environments in which tasks are the fundamental units of business processes. TRBAC model binds permissions on tasks and groups users operating the same tasks into roles. Rather than accessing business objects directly [2, 3], users accomplish their works

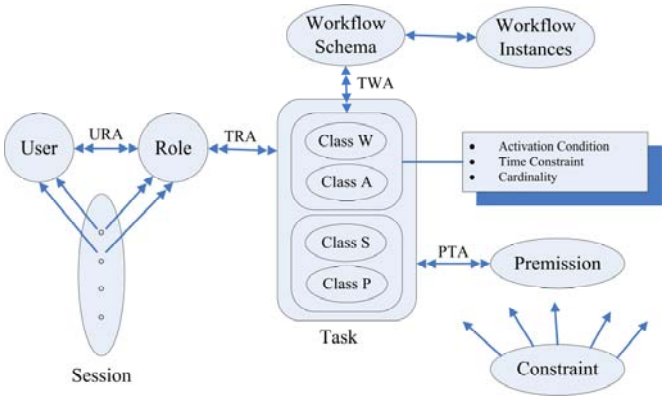


Fig. 1. The TRBAC model [4].

through tasks in which permissions are properly defined and protected. Restricting the access rights of business objects on tasks facilitates permission management and reduces the risks about inappropriate permission authority made by users. In TRBAC [4], tasks are classified into four classes according to whether the task participates in a business process and whether the task is inherited by the ancestor job. The classes of tasks in TRBAC model are illustrated in Table 1.

Table 1. Classes of tasks in TRBAC model [4].

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

2.2 Delegation Approaches in RBAC and TRBAC

In RBAC [2, 3], permissions to business objects are bound with roles. RBDM0 [5] provides a flexible way for granting and revoking permissions between roles. RBDM1 [6], an extension of RBDM0 [5], is more realistic since it organizes roles with hierarchy. Both techniques are focused on delegation of roles among human users through identifying can-delegation relationships between roles.

In [7], the essence of this delegation model is that a user delegates a particular right to another user, and delegation of partial permissions is allowed. Osborn separates users in organization, role hierarchies, and relationships among privileges into different graph models in [8, 16], and shows a simple way to delegate privileges to users by creating a delegatee role. In [10], Crampton gives a further discussion about both granting and transferring access rights between roles. When access rights are granted from the delegator to the delegatee, the delegated access rights are available for both the delegator and the delegatee [10]. On the other hand, if the access rights are transferred, only the delegatee holds the access rights after the delegation [10]. Besides, Crampton considers both can-delegate and can-receive relationships, and introduces the concept of administrative scope to improve the efficiencies in delegation controlling [10].

However, modern enterprises adopt WfMS to coordinate business processes. Therefore, tasks, the basic logic units of business processes, should also be considered in access control. In [10], Crampton addresses the issues like upward delegation and authorization of appropriate permissions for delegation to adopt RBAC-based delegation mechanism in workflow system. Bammigatti associates tasks into permission management and develops a new model for using RBAC in workflow system [11]. In TAC model [12], the permissions possessed by roles and required by tasks are described separately, and the assignment of tasks to roles is thus constrained. With such constraints, a protocol enabling delegation of task instances from users to roles is established [12].

TRBAC [4] binds the permissions with tasks, and the tasks with roles. With the roles assigned to users, users access business objects and accomplish their duties through tasks. Therefore, authorization of permissions is not necessary for delegation of tasks and task instances in TRBAC. In our previous work [13], a delegation framework for TRBAC has been initially established without considering the temporal issues. Zhang *et al*. develops a delegation model for time constraints-based TRBAC [14]. However, Zhang reduces

TRBAC model as TRBACM model in which permissions and tasks are separately bound with roles, and the reduction violates the primary sprits of TRBAC [4]. In [14], users delegate permissions together with tasks to accomplish their works.

2.3 Separation of Duty

SOD is a security principle which requires multiple users to be responsible for the completion of a work [17]. Ferraiolo defines SOD in [3] as "*For a particular set of transactions, no single individual is allowed to execute all the transactions within the set*." Botha discusses SOD in workflow environments both statically and dynamically [18]. In Botha's study, four possible conflicts, conflicting roles, conflicting permissions, conflicting users, and conflicting tasks, are described, and the corresponding methods for the conflicts are developed.

TRBAC [4] offers SOD policy at both task and instance level, and defines that some tasks are *mutually-exclusive* to each other. In task level SOD, for the roles played by a user, none of the tasks assigned to the roles are mutually-exclusive. In instance level SOD, the policy is effective for the tasks belonging to the same workflow instance. The task instances instantiated from the mutually-exclusive tasks in a workflow instance can not be executed by the same user [4].

2.4 Temporal Description

In [19], Allen gives several reasoning relationships between time intervals such as before, after, and overlapping, *etc*. In [15], the estimated active interval (EAI) for each task is calculated. Corresponding to the start time of a workflow, EAI of a task indicates when the task can be initiated and must be completed. The temporal relationships between tasks are considered through EAIs. In [9, 20], Bertino uses the periodic expressions to describe the temporal constraints in roles for temporal RBAC model. The periodic expressions are viewed as a combination of multiple time intervals, and are grouped as a *time description* in this paper. The definitions are as following.

Definition 1 A time interval $ti = [S(ti), E(ti)]$ indicates a duration from the time point $S(ti)$ to $E(ti)$ where $E(ti) \geq S(ti)$. A time point tp can be represented as a time interval $[tp, tp]$. *ctime* is the time point indicating the current time. For any two time intervals ti_1 and ti_2, ti_2 *contains* ti_1, notated as $ti_2 \supseteq_{TI} ti_1$, if and only if $S(ti_2) \leq S(ti_1)$ and $E(ti_2) \geq E(ti_1)$.

Definition 2 A time description $td = \{ti \mid ti \text{ is a time interval}\}$, and $\forall ti_1, ti_2 \in td$, ti_1 and ti_2 are exclusive. For any two non-empty time description td_a and td_b, $td_a \supseteq_{TD} td_b$ indicating td_a *contains* td_b if and only if $\forall ti_b \in td_b, \exists ti_a \in td_a$ such that $ti_a \supseteq_{TI} ti_b$.

3. THE WORKFLOW MODEL ADOPTED IN OUR APPROACH

A workflow specification consists of tasks, networks between tasks, the criteria indicating the start and the termination of the workflow, and the information about the individual tasks [1]. However, networks between tasks are not necessary in our discussion.

We model a workflow specification in a reduced way that a workflow specification is composed of a start task, an end task and a set of activity tasks. During run-time, workflow instances are instantiated from a workflow specification, and the tasks in the specification are also instantiated along with the workflow instance.

Tasks are the basic components describing pieces of works in logical steps within a workflow [1]. *Permissions* are the rules describing the admission in accessing business objects such as documents or computation resources. Based on TRBAC, individual permissions are bound with tasks. The active duration of a task is estimated with EAI which can be calculated with the methodology described in [15]. Besides, the task classes related to enactment of business processes, *i.e.* "Workflow" and "Approval", are considered as the types of tasks.

Users are participants of business processes. A user may play multiple roles for business, and a role can be played by multiple users. During run-time, users execute task instances in their work list to accomplish their daily duties. The status of a user is normally *available* and is transited to *unavailable* when he is not available for work.

Task instances are the basic units for users' daily duties. When a workflow instance is instantiated, related task instances are initiated as well. When a task instance is going to be executed, the system offers the instance to a role in accordance with the workflow specification. Then, the instance is allocated to the work list of one of the users playing the offered role. The user executes the instances in his work list, and submits the instance whenever it is complete. A task instance is suspended once the responsible user becomes unavailable for a certain time, is resumed from suspension when the responsible user is again available. An instance is failed if it is not completed in its active interval, and is discarded if it is not executed until the end of the workflow instance. The active interval of an instance indicates when the instance can be started and the corresponding deadline.

Roles represent collections of users with common responsibilities. In this paper, a role is modeled as a collection of the users responsible for the same tasks with certain timing constraints. The definition of workflow specification, workflow instances, tasks, users, task instances, users, and roles are formally described as follows.

Definition 3 (Workflow Specification & Instance) A workflow specification $rws = (T_{rws}, s, e)$. T_{rws} is the set containing the tasks operated in *rws*, and $s, e \in T_{rws}$ are the start and the end tasks of *rws*. A workflow instance $wi = (I_{wi}, rws, st)$. I_{wi} is the set of the task instances instantiated from each task in T_{rws}. *rws* is the workflow specification instantiating *wi*, and *st* is the time point *wi* being initialized.

Definition 4 (Task) Let T be the set of all tasks. $\forall t \in T$, $t = (rws, P_t, R_t, eai, type)$. *rws* is a workflow specification, and $t \in T_{rws}$. P_t is the set of permissions to business objects bound on *t*. R_t is the set of roles assigned to *t*. *eai* is a time interval indicating the EAI of *t*. *type* $\in$ {Workflow, Approval} is the class of *t*.

Definition 5 (Task Instance) Let I be the set of all task instances. $\forall i \in I$, $i = (wi, tk, ar, s, eu, ai)$. *wi* is the workflow instance where $i \in I_{wi}$. $tk \in T_{wi.rws}$ is the task instantiating *i*. $ar \in R_{tk}$ is the role *i* offered. $s \in$ {Initiated, Discarded, Offered, Allocated, Completed, Suspended, Failed} is the status of *i*. *eu* is the user executing the task instance. $ai = [wi.st + S(tk.eai), wi.st + E(tk.eai)]$ is the active interval of *i*.

Definition 6 (User) Let U be the set of all users. $\forall u \in U$, $u = (R_u, WL, cs)$. R_u is the set of roles played by u. $WL = \{i \mid i \in I, i.eu = u, i.s \in \{\text{Allocated, Completed, Suspended, Failed}\}\}$ is the work list of u. $cs \in \{\text{Available, Unavailable}\}$ is the current status of u.

Definition 7 (Role) Let R be the set of all roles. $\forall r \in R$, $r = (U_r, T_r, etd)$. U_r is the set of users playing r. T_r is the set of tasks assigned to r. etd is a time description indicating when r is active.

The role hierarchy indicates inheritance relationships and partial orders between roles to reflect organization lines of authority or responsibility [2]. The role hierarchy is modeled with directed acyclic graph (DAG) like in [2, 21, 22]. Among the role hierarchy, the roles in higher positions possess larger authority, and the connected roles are more coherent than disconnected ones [2, 21, 22]. The number of edges between two connected roles in the role hierarchy is defined as their *distance*. The roles closer in distance are related more tightly than roles farther. The role hierarchy and the function calculating the distance between two roles in a hierarchy are defined in Definition 8.

Definition 8 (Role Hierarchy) The role hierarchy $RH \subseteq R \times R$. $\forall(r_1, r_2) \in RH$, (r_1, r_2) shows a partial order that all inheritable tasks assigned to r_1 can also be assigned to r_2. $\forall r, r' \in R$, $r' \succ_{rh} r$ holds if there exists $(r, r_1), \ldots, (r_k, r') \in RH^*$. RH is acyclic, and if $r' \succ_{rh} r$ holds, $r \succ_{rh} r'$ does not. DisRH() shows the distance between two roles in the role hierarchy: (1) $DisRH(r, r) = 0$, (2) if $r' \succ_{rh} r$, $DisRH(r, r') = -(k + 1)$ and $DisRH(r', r) = k + 1$, (3) $DisRH(r, r')$ is undefined while neither $r' \succ_{rh} r$ nor $r \succ_{rh} r'$ holds.

4. THE DELEGATION FRAMEWORK FOR TRBAC

In this section, we construct a framework for delegation in TRBAC. The properties of delegation in TRBAC are described in section 4.1. The methods collecting candidate delegatees and filtering out inappropriate candidates are discussed in section 4.2.

4.1 The Properties of Delegation

A delegation is primarily composed of a delegator, a delegatee and a delegating subject. The delegator delegates the subject, and the delegatee receives it. In TRBAC, since the permissions are bound with tasks, the task instances are delegated between users during run-time.

For each delegation, the delegator, the delegatee, the delegated task instance, and the delegation duration are recorded in a delegation record. In this paper, the duration is constrained not exceeding the active interval of the delegated task instance. Our framework allows multi-level delegation [5, 7], and a task instance can be delegated several times. For each delegated task instance, a delegation record keeps tracking its status no matter how many times it is delegated. All the delegators who once delegated the task instance are put into the historical delegator list in the corresponding delegation record. We assume that the maximal number of times a task instance can be delegated is constrained by an enterprise policy named the *Maximal Levels of Delegation* (MLD). If MLD is equal to 1, multi-level-delegation is forbidden. With above features, the format of a delegation

record is defined in Definition 9. When a delegation occurs, the corresponding record is attached to the task instance for reference as Definition 10 shows. Algorithm 1 describes how a task instance is delegated in our framework.

Algorithm 1 Delegation Algorithm
Input: the delegating task instance *dti*, the delegatee *u*, and the designated delegating duration *ddur*
Pre-Condition: $dti.ai \supseteq_{TI} ddur$

```
DA {
1:   if (dti.dr ≠ Ø) {
2:      if (|dti.dr.HDRL| + 1 > MLD)
3:         EXCEPTION(MAX_DELEGATION_LEVEL_REACHED)
4:      else {
5:         add dti.eu to dti.dr.HDRL
6:         dti.dr.du := ddur
7:         dti.dr.de := u
8:      }
9:   } else {
10:      dti.dr := (dti, dti.eu, u, ddur, {dti.eu})
11:      add dti.dr to D
12:  }
13:  remove dti from dti.eu.WL
14:  add dti to u.WL
15:  dti.eu := u
}
```

Definition 9 (Delegation Record) Let D be the set of all delegation records. $\forall d \in D$, $d = (di, dr, de, dur, HDRL)$. $di \in I$, is the delegated task instance, and $\forall d' \in D$, if $d' \neq d$, $d'.di \neq d.di$. $dr \in U$ is the original delegator. $de \in U$, is the current delegatee. *dur* is a time interval indicating the duration *d* is effective, and $di.ai \supseteq_{TI} dur$. $HDRL = \{u_1, u_2, \ldots, u_k\}$ is the historical delegator list. $u_1 = dr$, and $\forall u_m \in HDRL$, $m < k$, u_m delegated *di* to u_{m+1}, and u_k delegated *di* to *de*. $|HDRL| \leq MLD$.

Definition 10 (Delegation Record in Task Instance) $\forall i \in I$, if *i* is delegated, $i.dr \in D$, $i.dr.di = i$; otherwise, $i.dr = Ø$.

The system invokes Algorithm 1 when delegating a task instance to the designated delegatee. At line 1, the algorithm checks whether the input task instance has been delegated. If so, the algorithms checks the size of historical delegator list of the task instance at line 2 to assure that further delegation will not exceed MLD. According to the input parameter, the delegation record is updated from lines 5 to 7. Otherwise, the task instance is delegated for the first time. A new delegation record is created and attached to *dti* at lines 11 and 12. After the delegation record is well updated or created, the task instance is transferred from the delegator's work list to the delegatee's from lines 14 to 16.

4.2 Delegatee Decision

4.2.1 Removing inappropriate users from candidate delegatees

Algorithm 1 does not concern whether a delegatee is appropriate for delegation or

not. In multi-level-delegation, if a task instance is delegated to one of the delegators who once delegated the task instance, a delegation loop occurs. Delegation loop causes redundancy in business and should be avoided [23]. Algorithm 2 is constructed to remove users causing delegation loop from the candidate users.

Algorithm 2 Removing Users causing Delegation Loop
Input: the candidate user set CUS, and target task instance *ti*
Pre-Condition: CUS $\subseteq$ U
User Set RUDL {
1: if ($ti.dr \neq \emptyset$) {
2: CUS := CUS\($ti.dr$.HDRL)
3: return CUS
}

Taking a set of users and a task instance as the input parameters, Algorithm 2 eliminates users causing delegation loop from the input user set. Each delegator user who once delegated the instance is recorded in the historical delegator list of the delegation record. After removing the historical delegators from the input user set at line 2, CUS is returned at line 3.

SOD is another issue in delegatee decision. Since delegation happens during runtime, we focus on maintaining instance level SOD policy for the task instances in a workflow instance. For each workflow specification, the mutually-exclusive tasks are grouped in records, and a task might belong to multiple records. For example, task "auditing" is mutually-exclusive to both task "ordering" and "purchasing", but "ordering" and "purchasing" are not mutually-exclusive. Therefore, two records are established, and "auditing" is contained in both records with "ordering" and "purchasing" separately. The record for mutually-exclusive tasks and the SOD constraints adopted in this paper is defined as follows.

Definition 11 (Mutually-Exclusive Tasks) MET is the set of all the records of mutually-exclusive tasks. $\forall met \in$ MET, $met = (w, T_{met})$. w is a workflow specification. $T_{met} \subseteq T_{w.rws}$ is a set of mutually-exclusive tasks and $\forall t_i, t_j \in T_{met}$, t_i and t_j are mutually-exclusive.

Definition 12 (Instance Level SOD constraints) $\forall$ workflow instance *wi* that $wi.rws = w$, $ti_1, ti_2 \in I_{wi}$. If $\exists(w, T_{met}) \in$ MET, that $ti_1.tk, ti_2.tk \in T_{met} \Rightarrow ti_1.eu \neq ti_2.eu$.

In a delegation, SOD also holds. For a task instance *ti* which is being delegated, a user executing a task instance mutually-exclusive to *ti* can not be the delegatee of *ti*. Taking a set of candidate delegatees and a task instance as the input parameters, Algorithm 3 eliminates the users violating instance SOD from the candidate delegatees.

***Proof*:** To show Algorithm 3 following SOD constraints in Definition 12, we assume that a user $u \in$ RUCT(CUS, *ti*) is now executing *ti′* which is mutually-exclusive to *ti*. Let a workflow instance wi_1 contains *ti′* and *ti*. With the assumption, $ti'.tk$ and $ti.tk$ are mutually-exclusive to each other, and $ti'.eu = u$. Since $ti.wi = wi_1$ and $ti' \in I_{ti.wi}$, *ti′* is selected at line 1. On the other hand, by Definition 11, there exists $met \in$ MET that $met.w = ti.wi.rws = ti'.wi.rws = wi_1.rws$ and $ti'.tk, ti.tk \in T_{met}$. Therefore, the expression at line 2 is true for *ti′*

and *ti′.eu* is removed from the result set at line 3. Thus, *u* is not included in the result set and the assumption is contradicted. Algorithm 3 follows SOD constraints in Definition 12.❑

Algorithm 3 Remove Users involving Mutually-Exclusive Tasks
Input: the candidate user set CUS, and target task instance *ti*
Pre-Condition: CUS $\subseteq$ U
User Set RUCT {
1: $\forall i \in I_{ti.wi} \backslash \{ti\}$ {
2: if ($\exists met \in$ MET, *met.w* = *ti.wi.rws*, *i.tk*, *ti.tk* $\in T_{met}$)
3: remove *i.eu* from CUS
4: }
5: return CUS
}

Intuitively, an unavailable user can not be the delegatee of any delegation, and a task instance should not be delegated to the user currently executing it. Concluding these two issues and the algorithms described in this section, the algorithm removing inappropriate users from a set of candidate delegatees is constructed as follows.

Algorithm 4 Removing Inappropriate Users
Input: candidate delegatee set CDS, delegating task instance *dti*
Pre-Condition: CDS $\subseteq$ U
Candidate Delegatee Set RCU {
01: $\forall u \in$ CDS
02: if (*u.cs* = Unavailable) remove *u* from CDS
03: CDS := RUCT(RUDL(CDS, *tdi*), *tdi*)\{*dti.eu*}
04: return CDS
}

Algorithm 4 first removes the unavailable users from the input set at line 2. At line 3, the algorithm invokes Algorithms 2 and 3 to remove the users causing delegation loop or violating SOD. After removing the current executing user of *dti*, Algorithm 4 returns the result set at line 4.

4.2.2 Deriving candidate delegatees

For system requested delegations, WfMS derives the delegatees automatically. The role hierarchy indicates the organization lines of authority and responsibility [2], and can be used for exploration of possible candidate delegatees. Along with the role hierarchy, a task instance can be delegated upward or downward. When a task instance of daily works is being delegated, the system gathers the users playing lower roles related to the offered role of the instance as candidate delegatees. On the other hand, for the instances of the tasks related to decision making, the system commits an upward discovery from the offered role in the role hierarchy for the candidate delegatees. Besides, the users playing roles closer to the offered role in the role hierarchies are considered as better candidates in delegatee decision. Based on Definition 8, the algorithm discovering the role hierarchy for the candidate delegatees is constructed as follows.

Algorithm 5 Discover Role Hierarchy
Input: the delegating task instance *dti*
Candidate Delegatee Set DRH {
1: if (*dti.tk.type* = Approval) $p := 1$
2: else if (*dti.tk.type* = Workflow) $p := -1$
3: $m := 0$
4: US := Ø
5: loop {
6: GR := Ø
7: $\forall r \in R$, DisRH(*dti.ar*, *r*) = $p*m$
8: add *r* to GR
9: if (GR = Ø) return Ø
10: $\forall r \in GR$, $r.etd \supseteq_{TD} \{dti.ai\}$
11: US := US $\cup$ U_r
12: US := RCU (US, *dti*)
13: if (US = Ø) $m := m + 1$
14: else break
15: }
16: return US
}

At lines 1 and 2, according to the class of the *dti*'s task, the algorithm decides the direction to explore the role hierarchy. Algorithm 6 commits an upward discovery for the tasks typed "Approval" or a downward discovery for the tasks typed "Workflow". From lines 5 to 14, the algorithm does a breadth first search in the role hierarchy. At line 9, empty GR set represents that all roles connected to the offered role along with the designated direction in the role hierarchy are explored, and no proper delegatee is found. Therefore, Algorithm 6 returns Ø as the result. If GR is not empty, the users playing roles in GR is gathered into user set US and filtered with Algorithm 4. If US set is not empty after the removal of conflict users, the algorithm returns US as the result set. Otherwise, the discovery continues with further distances.

5. DELEGATION PROCESS FOR TRBAC

In this section, we discuss how our framework works in workflow systems. In section 5.1, the methodology automating delegation requested by the system is discussed. In section 5.2, the process for users to delegate their current and forthcoming works are described. The method to revoke delegation is described in section 5.3.

5.1 Delegation from System Request

When a suspended task instance is nearly timed out, the system might spontaneously request a delegation for the task instance. We assume that a suspended task instance is *emergent*, and need to be delegated automatically if the proportion of its remaining active interval is less than an enterprise policy named the *Emergent Execution Ratio* (EER). With this assumption, the algorithm for delegation requested by the system is described as follows.

Algorithm 6 Delegation from System Request
Input: the delegating task instance *dti*
Pre-Condition: *dti*.*s* = Suspended, E(*dti*.*ai*) > *ctime*

(E(*dti*.*ai*) – *ctime*)/(E(*dti*.*ai*) – S(*dti*.*ai*)) < EER

```
DFSR {
1: CDS := DRH(dti)
2: if (CDS = Ø) EXCEPTION(NO_PROPER_DELEGATEE)
3: else {
4:     randomly choose a user u from CDS
5:     DA(dti, u, [ctime, E(dti.ai)])
6: }
}
```

The system tracks the status of the executing task instances, and invokes Algorithm 7 whenever an emergent task instance is found. Algorithm 7 acquires the candidate delegatees by exploring the role hierarchy with Algorithm 6 at line 2. Exception is raised if Algorithm 6 returns no candidates. Otherwise, the algorithm randomly chooses a delegatee from the candidate delegatees and invokes Algorithm 1 to delegate the emergent task instance.

5.2 Delegation from User Request

Many modern enterprises adopt user-authorized delegation as the primary delegation methodology. The RBAC-based researches like [6, 8-10], also describes how roles and permissions are delegated under user authorization.

With our framework, a user can authorize two types of delegation. First, a user may delegate task instances currently allocated to him. Second, a user may delegate the task instances going to be allocated to him during a specific period.

To request a delegation, the delegator fills in an authorization form which designates the delegating subject, the delegatee user and the activation duration of the delegation. For the first type of delegation, the delegator designates a task instance residing in his work list as the delegating subject. The duration to authorize the delegation must be contained by the active interval of the delegating task instance. For the second type of delegation, the delegator designates an executable task by any of the roles he playing as the delegating subject. The duration to authorize the delegation must be contained by the effective duration of the role.

After accomplishing the authorization form, the delegator user submits the form in requesting approval from his supervisor and the designated delegatee. If the delegation is approved, for the first type of delegation, the designated task instance is delegated to the delegatee user with Algorithm 1 immediately. For the second type of delegation, the approved form is put into *Forthcoming Delegation Table* (FDT). According to the form, the task instances of the designated task which is allocated to the delegator in the specified duration are delegated to the designated delegatee. Fig. 2 represents the process of delegation from user request, the authorization form is defined in Definition 13, FDT is defined in Definition 14, and finally, Algorithm 7 shows how WfMS handles the second type of delegation.

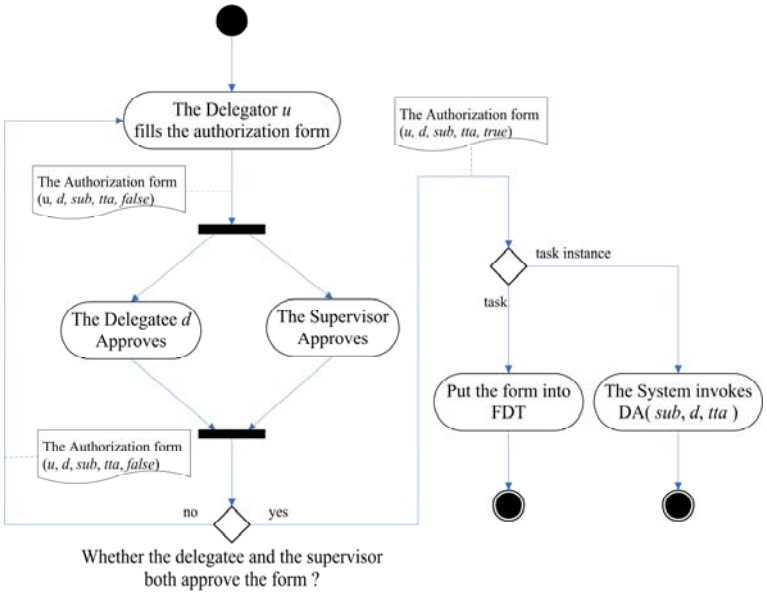


Fig. 2. The process of delegation from user request.

Algorithm 7 Handle Forthcoming Delegation
Input: Task Instance *i*, User *u*
Pre-Condition: *i* is allocating to *u*
HFD {
1: if ($\exists ap \in$ FDT, $ap.dr = u$, and $ap.sub = i.tk$) {
2: if (RCU({$ap.de$}, i) $\neq \emptyset$ {
3: DA(i, $ap.de$, [$ctime$, min(E($ap.tta$), E($i.ai$)))
4: } else EXCEPTION(INAPPROPRIATE_DELEGATEE)
}

Definition 13 (Authorization Form) Let AP be the set of all authorization forms. $\forall ap \in$ AP, ap = (dr, de, sub, tta, $is\_approved$). dr is the delegator user. de is the designated delegatee user, $dr \neq de$. sub is the subject of delegation, where if $sub \in$ T, there exists $r \in u.R_U \cap sub.R_T$ and $r.etd \supseteq_{TD} \{tta\}$, and if $sub \in$ I, $sub \in dr$.WL, $sub.s \neq$ Completed, and $sub.ai \supseteq_{TD} tta$. tta, time to authorize, is the time interval that dr delegates the subject to de.$is\_approved$ is a boolean variable showing whether ap is approved.

Definition 14 (Approved Form) $\forall ap \in$ AP, if $ap \in$ FDT, $ap.sub \in$ T, $ap.is\_approved$ = true, and E($ap.tta$) $\geq ctime$.

The system invokes Algorithm 8 whenever a task instance is being allocated to its execution user. At line 1, the algorithm first checks if the user and the task of the task instance are recorded on an authorization form in FDT. If the task instance is authorized to be delegated, Algorithm 8 then invokes Algorithm 4 to check whether the designated delegatee user on the form violates any delegation constraints. If the check is not passed, an exception is raised and further handling is necessary. According to different policies,

the hanging task instance might be handled manually or be automatically delegated by the system. Otherwise, Algorithm 1 is invoked to perform the delegation.

5.3 Revocation

A successful delegation can be revoked before it ends [10]. For revoking a delegated task, the authorization form is removed from the FDT. For revoking a delegated task instance, according to the system settings and the enterprise policies, the delegatee's contribution on the task instance might be preserved or discarded. The task instance is transferred back to the work list of the user requesting the revocation, the revoker, and the revoker continues executing the task instance after the revocation.

Revoking a multi-level delegation is complex. For a multi-level delegation, all the users recorded in the historical delegator list might revoke the delegation. If the revoker is the original delegator, after the delegated task instance is transferred back, the delegation record is totally eliminated. Otherwise, if the revoker is the other delegator in the historical delegator list, the revoker becomes the delegatee of the delegation after the revocation. The revoker and the other delegators behind the revoker are removed from the historical delegator list.

When a delegation runs out of its effective duration, the system revokes it automatically. The delegated task instance is transferred back like the revocation is requested by the original delegator. Algorithm 8 is constructed as follows for revocation.

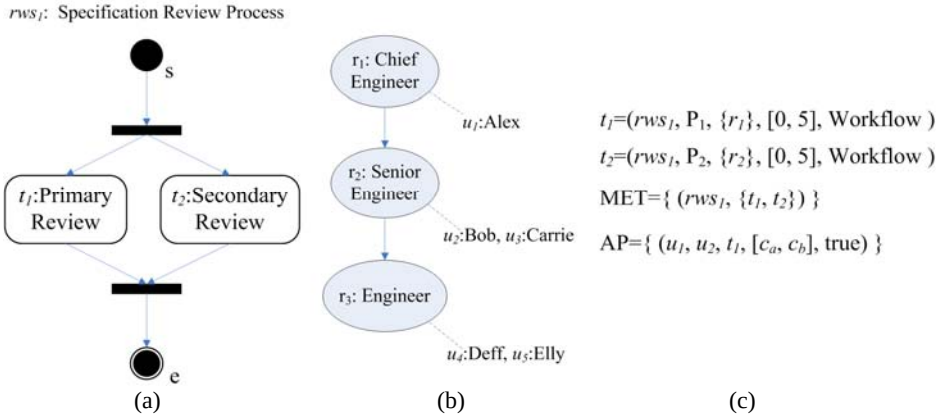
Algorithm 8 Revocation Algorithm
Input: the subject to be revoked *rsub*, the revoker *u*
Pre-Condition: $rsub \in T \cup I$, $u \in U$
RA {
1: if ($rsub \in T$ && $\exists ap \in$ FDT that $ap.dr = u$, and $ap.sub = rsub$) {
2: remove *ap* from FDT
3: } else if ($rsub \in I$ && $rsub.dr = d$ that $u \in d$.HDRL, and $rsub.s \neq$ Completed) {
4: alert *dti.eu* that *d* is going to be revoked
5: remove *rsub* from *rsub.eu*.WL
6: add *rsub* to *u*.WL
7: *rsub.eu* := *u*
8: if($u = d.dr$) {
9: remove *d* from D
10: *rsub.dr* := Ø
11: } else {
12: *d.de* := *u*
13: *u* and all the users behind *u* in the *d*.HDRL are removed from *d*.HDRL
14: }
15: alert *dti.eu dti* is transferred back to his work list
16: } else EXCEPTION(INVALID_REVOCATION)
}

Algorithm 8 takes the subject being revoked and the revoker as the input parameters. If the subject is a task, Algorithm 8 checks whether there is corresponding authorization form, and removes the form from FDT at lines 1 and 2. Otherwise, if the subject is a task instance, Algorithm 8 checks the corresponding delegation record to assure the revoca-

tion is valid at line 3. If valid, the current delegatee of the delegated instance is first alerted at line 4. The delegated instance is removed from the delegatee's work list, and transferred to the revoker from lines 5 to 7. If the revoker is the original delegator of the delegation, the delegation record is eliminated from lines 8 to 10. Otherwise, the record is updated. The delegatee is assigned to the revoker at line 9; the revoker and the delegators behind him are removed from the historical delegator list at line 10. The revoker is alerted at line 15. At line 16, an exception is raised if the revocation is invalid.

6. CASE STUDY

In this section, we adopt a specification review process illustrated as an example to show the feasibility of our approach. The workflow specification of the review process, the partial role hierarchy, and the other related information are illustrated in Fig. 3. In this case, the review process is composed of two tasks, primary review and secondary review. Chief Engineer is in charge of the primary review, and Senior Engineer is responsible for the secondary one. These two review tasks are mutually-exclusive, and concurrent with the same EAI, [0, 5]. Let Alex is busy in his duty, and gets an approved delegation of the reviews allocated to him during the time interval $[c_a, c_b]$. The approval is done by his supervisor and Bob, the designated delegatee.



(a) (b) (c)

Fig. 3. (a) The sample workflow specification; (b) The sample role hierarchy and the user assignment; (c) The information about tasks, mutually-exclusive tasks, and authorization applications.

At time c_1, $c_a < c_1$ and $c_1 + 5 < c_b$, a workflow instance of rws_1, $wi_1 = (\{i\_t_1, i\_t_2\}, rws_1, c_1)$, is instantiated so that the task instances i_t_1 and i_t_2 are instantiated based on t_1 and t_2. i_t_1 and i_t_2 are offered to Chief Engineer and Senior Engineer, and allocated to Alex and Carrie correspondingly. Now, $i\_t_1 = (wi_1, t_1, r_1, \text{Allocated}, u_1, [c_1, c_1 + 5], \emptyset)$, and $i\_t_1 = (wi_1, t_2, r_2, \text{Allocated}, u_3, [c_1, c_1 + 5], \emptyset)$. Because Alex has been approved to delegate all the reviews during $[c_a, c_b]$ to Bob, Algorithm 7 invokes Algorithm 1 to delegate i_t_1 to Bob. The delegation record $d = (i_1, u_1, u_2, [c_1, c_1 + 5], \{u_1\})$ is created, and i_t_1 becomes $(wi_1, t_1, r_1, \text{Allocated}, u_2, [c_1, c_1 + 5], d)$ after the delegation.

At time c_2 which is in the middle of the active interval of i_t_1, $c_1 < c_2 < c_1 + 5$, Bob gets an emergent call and becomes unavailable right away. The task instances in his work list are all suspended. If EER equals to 1, the workflow system would immediately invokes Algorithm 6 to delegate i_t_1 to another appropriate delegatee. In Algorithm 6, Algorithm 5 is first invoked to explore the role hierarchy for a proper delegatee. Because t_1 is typed "Workflow", the role hierarchy is explored downward from Chief Engineer, the role i_t_1 offered. Alex is the only user now playing Chief Engineer, and is eliminated from the candidate delegatee set by Algorithm 2 to avoid delegation loop. When considering Senior Engineer, Carrie is eliminated from the candidate set by Algorithm 3 because of the SOD policy, and Bob is eliminated from the candidate set by Algorithm 4 because he is unavailable. No users playing Senior Engineer are appropriate to take the task. Therefore, Engineer is then considered. After all, Deff and Elly are included in the candidate set, and Deff is randomly decided as the new delegatee of i_t_1. Algorithm 1 is invoked to delegate i_t_1 to Deff. d is updated as (i_1, u_1, u_4, [c_2, $c_1 + 5$], {u_1, u_2}), and i_t_1 is updated as (wi_1, t_1, r_1, Allocated, u_4, [c_1, $c_1 + 5$], d).

At c_3, $c_2 < c_3 < c_1 + 5$, Alex finishes his jobs ahead of time, and decides to finish i_t_1 himself. Alex invokes Algorithm 8 to revoke i_t_1. Deff is first alerted and i_t_1 is then revoked. The delegation record d is removed, and i_t_1 is updated as (wi_1, t_1, r_1, Allocated, u_1, [c_1, $c_1 + 5$], Ø). In summary, this case demonstrates the delegations requested from a user and the system, and indicates how the constraints like delegation loop and SOD work in automatic delegatee decision.

7. COMPARISON AND CONCLUSION

7.1 Comparison

In this section, we compare our framework with the latest popular approaches: [6, 10, 12, 14], and Table 2 illustrates the characteristics of above approaches and ours correspondingly. RBDM1 [6] is a classic delegation model for RBAC, and can be adopted in managing delegation of permissions between users. Crampton develops another RBAC-based delegation model for workflow systems [10]. Crampton's approach allows both grant and transfer operations for delegation of permissions while RBDM1 adopts only grant

Table 2. Delegation characteristics in various delegation models.

Characteristics	RBDM1 [6]	Crampton [10]	Gaaloul [12]	VTTRDM [14]	Our Approach
Access Control	RBAC [3]	RBAC [3]	TAC [12]	TRBACM [14]	TRBAC [4]
Delegation of Permissions	Grant	Grant & Transfer	No	Grant	No
Delegation of Tasks	No	No	Transfer	Yes	Transfer
Delegation of Task Instances	No	No	No	No	Transfer
Time Constraints	No	No	No	Yes	Yes
Automatic Delegation	No	No	No	No	Yes

operation [10]. Crampton also raises the issues like upward delegation and permission authorization for delegation of tasks in work-list based workflow systems [10]. However, both RBDM1 and Crampton's approach describe no methods about delegation of tasks.

With various access control models based on tasks and roles, Gaaloul's methodology [12], VTTRDM [14], and our approach can be adopted in managing delegation of tasks for workflow systems. Gaaloul's methodology describes constraints for delegation of tasks based on Task-oriented Access Control (TAC) model [12]. TAC model describes the permissions which a role owns and a task needs. Gaaloul's methodology allows a user to delegate his tasks to a role which has sufficient permissions to execute the tasks. Since Gaaloul's methodology allows no delegation of permissions, it is limited and inflexible when selecting the delegatee for a delegation.

In VTTRDM [14], both permissions and tasks can be delegated between users. RBDM1 [6] is adopted in VTTRDM to manage the delegation of permissions. When delegating a task, if the delegatee does not have sufficient permissions to execute the task, permission delegation from the delegator to the delegatee is necessary to enable the execution [14]. Since in VTTRDM, the delegated permission is not limited being used for the delegated tasks only, security risk exists.

In our approach, tasks are delegated through user's authorization. Our approach is based on TRBAC, and a task is executed with a set of associate permissions. Therefore, the delegatee can execute the delegated task without delegation of permissions, and the security risks brought by delegation of tasks are eliminated. Besides, in [24], delegation is defined as "*A user allocates a task instance previously allocated to him to another user*." While delegation of task instances is ignored in [12, 14], our approach clearly states how to delegate task instances between users. For delegation of task instances, our methods could gather candidate delegatees and remove inappropriate users from the candidates. With our approach, a workflow system can automatically delegate an emergent task instance to an appropriate user to prevent the task instance from failure.

Regarding temporal issues, in VTTRDM, delegation is effective during a single time interval, and the delegated tasks are revoked after the interval [14]. Our approach is based on the time constraints between the delegated task instances and the related roles. Because a role might be activated in multiple time intervals, multiple or periodical time intervals are considered in our approach to provide a more realistic temporal constraints.

7.2 Summaries and Future Works

In this paper, a delegation framework for TRBAC in WfMS is presented. The components of delegation are discussed. In the framework, the constraints in delegatee selection *e.g.* delegation loop and SOD are stated, and the methods to derive candidate delegatees from the role hierarchy are established. The temporal constraints are considered with the active interval of a task instance, the active durations of a role, and the effective duration of a delegation. The algorithms for delegation and revocation are constructed. With the algorithms, a user can delegate his tasks manually, and the workflow system can delegate an emergent task instance automatically. Finally, we make a comparison between the related works and ours, including the advantages and the disadvantages.

In the future, the feasibility and security issues in sharing a task instances among users can be further studied to adopt grant operation in delegation of task instances. More

constraints about temporal factors and SOD might be discussed to construct methods for exception detection and handling. Besides, the possibility and user's factors in delegation automation will be further studied by applying experiences in knowledge management.

REFERENCES

1. WfMC, *Workflow Management Coalition Terminology and Glossary*, the WfMC Specification, Winchester, 1999.
2. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, Vol. 29, 1996, pp. 38-47.
3. D. Ferraiolo and R. Kuhn, "Role-based access controls," in *Proceedings of the 15th National Computer Security Conference*, Vol. 2, 1992, pp. 554-563.
4. S. Oh and S. Park, "Task-role-based access control model," *Information Systems*, Vol. 28, 2003, pp. 533-562.
5. E. Barka and R. Sandhu, "A role-based delegation model and some extensions," in *Proceedings of the 23rd National Information Systems Security Conference*, 2000, pp. 125-134.
6. E. Barka and R. Sandhu, "Role-based delegation model/hierarchical roles (RBDM1)," in *Proceedings of the 20th Computer Security Applications Conference*, 2004, pp. 396-404.
7. J. Wainer and A. Kumar, "A fine-grained, controllable, user-to-user delegation method in RBAC," in *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, 2005, pp. 59-65.
8. H. Wang and S. Osborn, "Delegation in the role graph model," in *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies*, 2006, pp. 91-100.
9. J. B. D. Joshi and E. Bertino, "Fine-grained role-based delegation in presence of the hybrid role hierarchy," in *Proceedings of the 11th ACM Symposium on Access Control Model and Technologies*, 2006, pp. 81-90.
10. J. Crampton and H. Khambhammettu, "Delegation in role-based access control," *International Journal of Information Security*, Vol. 7, 2008, pp. 123-136.
11. P. H. Bammigatti and P. R. Rao, "Delegation in role based access control model for workflow systems," *International Journal of Computer Science and Security*, Vol. 2, 2008, pp. 1-10.
12. K. Gaaloul and F. Charoy, "Task delegation based access control models for workflow systems," *Software Services for e-Business and e-Society*, *IFIP Advances in Information and Communication Technology*, Vol. 305, 2009, pp. 400-414.
13. P. Jian, H. J. Hsu, and F. J. Wang, "A delegation framework for access control in WfMS based on tasks and roles" in *Proceedings of the 12th IEEE International Workshop on Future Trends of Distributed Computing Systems*, 2008, pp. 165-171.
14. D. W. Zhang, X. Pei, J. Q. Qiu, Y. Zhang, and J. Peng, "A delegation model for time constraints-based TRBAC," in *Proceedings of the 8th International Conference on Machine Learning and Cybernetics*, 2009, pp. 2027-2032.
15. H. J. Hsu and F. J. Wang, "An incremental analysis for resource conflicts to workflow specifications," *Journal of Systems and Software*, Vol. 81, 2008, pp. 1770-1783.
16. M. Nyanchama and S. Osborn, "The role graph model and conflict of interest," *ACM*

Transactions on Information and System Security, Vol. 2, 1999, pp. 3-33.
17. R. T. Simon and M. E. Zurko, "Separation of duty in role-based environments," in *Proceedings of the 10th Computer Security Foundations Workshop*, 1997, pp. 183-195.
18. R. A. Botha and J. H. P. Eloff, "Separation of duties for access control enforcement in workflow environments," *IBM System Journal*, Vol. 40, 2001, pp. 666-683.
19. J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, Vol. 26, 1983, pp. 832-843.
20. J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "Generalized temporal role-based access control model," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, 2005, pp. 4-23.
21. M. J. Moyer and M. Ahamad, "Generalized role-based access control," in *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems*, 2001, pp. 391-398.
22. G. Ding, J. Chen, R. F. Lax, and P. P. Chen, "Graph-theoretic method for merging security system specifications," *Information Sciences*, Vol. 177, 2007, pp. 2152-2166.
23. C. H. Chang, and F. J. Wang, "An analysis of delegation mechanism in workflow management system," Master's Thesis, Institute of Computer Science and Engineering, National Chiao Tung University, 2003.
24. N. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst, "Workflow resource patterns," BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004.

Hwai-Jung Hsu (許懷中) received the B.E. and M.E. degrees in Computer Science and Information Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2001 and 2003. He is now a Ph.D. candidate of the Institute of Computer Science and Engineering, National Chiao Tung University, Hsinchu, Taiwan. His research interests are software engineering, workflow technology, service-oriented computation, and pervasive computing, *etc*.

Feng-Jian Wang (王豐堅) received the B.S. degree in Physic from National Taiwan University, Taipei, Taiwan, in 1980, and the M.S. and Ph.D. degrees in Electrical Engineering and Computer Science from Northwestern University, IL, U.S.A., in 1983 and 1988. He is a Professor of the Department of Computer Science of the National Chiao Tung University, Hsinchu, Taiwan. His area of research includes software engineering, pervasive computing system, workflow technology, and grid computation.