Capability-Role-Based Delegation in Workflow Systems

Koji Hasebe and Mitsuhiro Mabuchi
Graduate School of Systems and Information Engineering
University of Tsukuba
1-1-1 Tennodai, Tsukuba 305-8573, Japan
hasebe@iit.tsukuba.ac.jp, mmabu@osss.cs.tsukuba.ac.jp

***Abstract*—Various security models for supporting delegation in workflow systems have been proposed to achieve flexible access control in collaborative business processes. Since workflow systems come into their own when controlling large-scale business processes in a well-structured organization, these models are often based on role-based access control (RBAC). However, to realize a higher level of collaboration enabling users in different organizations to complete a common workflow, it is necessary to support cross-domain delegation of tasks. For this purpose, we propose a delegation model for workflow systems that extends the capability-role-based access control (CRBAC) model introduced in our previous work. The central idea behind our proposed model is that authority to perform tasks, as well as roles, are mapped to capabilities, thereby realizing delegation by capability transfer. By adopting the approach of a capability-based access control mechanism, our model provides both flexibility and reduced administration costs, thus allowing it to cope with unexpected changes in task assignments. We demonstrate these advantages by considering an example.**

***Keywords*-RBAC, capability-based access control, delegation, workflow systems**

I. Introduction

Workflow systems are used to specify and control the execution of business processes. A workflow system stores a specification of workflow, which is defined as a collection of tasks with a specific execution order designed to achieve a common business objective. In addition, a workflow system stores information that identifies the owner of the execution authority for each task. The authorization information is specified in access control models. Since workflow systems come into their own when controlling large-scale business processes in a well-structured organization where users are assigned roles, these models are often based on role-based access control (RBAC) [10]. (An example of such a model is given in [7].)

To achieve flexible and dynamic access control in workflows, various attempts have been made at modeling the delegation of authority for executing tasks [1], [13], [3], [4]. Although most studies in the literature restrict their scope to delegation within a single domain, to achieve a higher level of collaboration enabling users in different organizations to complete a common workflow, it is necessary to realize cross-domain delegation. For example, considering a development process with collaboration between different departments of a company, delegation of tasks is required across multiple departments to cope with emergent calls for reinforcement or unexpected changes in team members.

To address the issue of cross-domain delegation in workflow systems, in this paper we propose a delegation model that extends the capability-role-based access control (CRBAC) model introduced in our previous work [6]. CRBAC comprises a capability-based access control mechanism integrated with the RBAC96 model [10]. A capability-based access control mechanism (cf. [8]) is considered a means for delegation of authority. In general, a capability, which is an unforgeable token, consists of an object identifier and a list of permitted operations for that object. Therefore, a capability represents a self-authenticating permission to access a specified object through a permitted operation, thus allowing owners of the capability to access the object without any authentication. Moreover, it is used for cross-domain delegation by means of middleware such as HomeViews [5] or CapaFS [9]. In terms of the capability, the central idea of CRBAC is the mapping of permissions as well as roles to capabilities in each domain, thereby realizing the delegation of permissions and roles by capability transfer. Thus, in the CRBAC model, by considering a set of permissions to perform tasks in a workflow, we obtain the intended delegation model. More precisely, our model, called W-CRBAC, is obtained by incorporating the following extensions. First, we introduce the notion of tasks to the CRBAC model, and then provide a definition of the workflow specification and its instantiation (i.e., assignment of a user to each task). Thus, as the models in [13], [3], we also consider two kinds of delegations, namely abstract task delegation and concrete task delegation. The former authorizes the delegatee to perform the delegated task in any instance of the workflow, while the latter authorizes the delegatee to perform only the delegated instance. Next, we define the delegation of tasks and roles as state transitions in a similar manner to the definition in CRBAC.

By adopting the approach of capability-based access control, the W-CRBAC provides delegation of tasks as well as roles in workflow systems with the following advantages. As in the CRBAC, cross-domain delegation can be achieved without specific requests to administrators. This makes flexible and smooth user-to-user delegation possible even in

emergent situations. Moreover, by avoiding authentication, the administration cost is reduced especially in large-scale workflow systems. On the other hand, as a trade-off for these advantages, capability based-access control has a potential security issue, that is, unintended propagation of capabilities. To avoid this issue, our model imposes certain restrictions on capabilities, such as lifetime and the number of activation, focusing on workflow management. Finally, in this paper, we demonstrate the effectiveness of the W-CRBAC model for workflow systems by considering an example situation.

This paper is organized as follows. Section II presents related work. Section III introduces the W-CRBAC model, while Section IV demonstrates the application of our model to workflows using an example. Section V discusses how our model can be implemented. Finally, Section VI concludes the paper and presents future work.

II. Related Work

There have been a number of studies on delegation in access control, with most of these classified into two classes according to the underlying model, either RBAC or workflow. As an example of the first class, the RBDM0 [2] was the first attempt at modeling user-to-user delegation of roles based on the RBAC96 model. RDM2000 [11] extended the RBDM0 to support hierarchical roles and multi-step delegation, while PBDM [12] considered delegation based on both roles and permissions. On the other hand, as an example of the second class, Atluri and Warner [1] addressed the issue of delegation in the context of workflow and presented a conditional delegation model.

Compared with the number of studies in each class, studies focusing on both classes are far fewer. As an example of this approach, DW-RBAC [13] proposed a role-based access control model supporting delegation and revocation in workflow systems. This model provided a mechanism to delegate both abstract and concrete tasks, where the delegation request was accepted so long as certain authorization conditions were satisfied. In [3], Crampton and Khambhammettu considered various kinds of delegation operations based on the differences of task type (introduced in [13]), workflow execution model, and delegation type. Moreover, in their subsequent work [4], satisfiability problem (i.e., problem of whether a set of authorized users can complete a workflow) was investigated for each of these kinds of delegation operations. The main difference between these three papers and ours is that our motivation is to support flexible delegation by adopting the approach of a capability-based access control mechanism, instead of supporting various kinds of delegations investigated in [3], [4].

III. Delegation in Workflow with CRBAC

In this section, we present formal definitions of our W-CRBAC model and the delegation in workflow systems.

A. Overview

CRBAC [6] is an extension of the RBAC96 model obtained by integrating a capability-based access control mechanism into the RBAC96 model. Our proposed model, called W-CRBAC, is an extension of CRBAC with the focus on its application to workflow systems. In a similar manner to the RBAC96 family, W-CRBAC (as well as CRBAC) has been developed from the base model (called W-CRBAC0) by adding either role hierarchy (W-CRBAC1) or various constraints (W-CRBAC2). (Although we also considered a consolidated model, called W-CRBAC3, we omit the definition thereof in this paper.) Moreover, to model cross-domain delegation, we consider a collection of sub-models, each of which represents a single domain. More precisely, our model has been developed according to the following steps.

First, we introduce certain components to the original RBAC0, namely, a set of capabilities, a set of tasks, a mapping to determine the owner of the capabilities, and the assignment, to the capabilities, of roles and permissions to perform tasks. On the other hand, in terms of the set of tasks, we define a workflow specification as well as its instantiations. Thus, as in [13], [3], in our model we distinguish between abstract and concrete tasks and treat roles and tasks of these kinds as delegation units. Next, we define the delegation operations in terms of state transition. In our model, new capabilities are created either from the delegator's role or from capabilities that were previously obtained. Finally, we add the role hierarchy and constraints. These extensions are essentially the same as in the RBAC96 model, but in this study we focus mainly on constraints on capabilities that are useful for preventing unintended propagation of capabilities and maintaining workflow security.

In the rest of this section, we introduce our model in a stepwise fashion: in Section III-B, we first present the definition of CRBAC0 then extend it to W-CRBAC0, including a model of the workflow; in Section III-C, we define the delegation; finally, in Sections III-D and III-E, we extend the base model to W-CRBAC1 and 2, respectively.

B. W-CRBAC0 and Workflow Model

As a preliminary step, we first introduce CRBAC0 presented in [6] then define W-CRBAC0. (For a graphical illustration of the family of W-CRBAC models, see Fig. 1, where the components drawn with bold lines and underlines are extended parts of the RBAC96 and CRBAC models, respectively.)

Definition 1 (CRBAC0): The CRBAC0 model includes the following basic components:

- Sub, Dom (sets of *subjects* and *domains*, respectively).
- $Role_i$, Per_i, Ses_i, Cap_i (sets of *roles*, *permissions*, *sessions*, and *capabilities*, respectively, in the i-th domain for each $i \in Dom$).

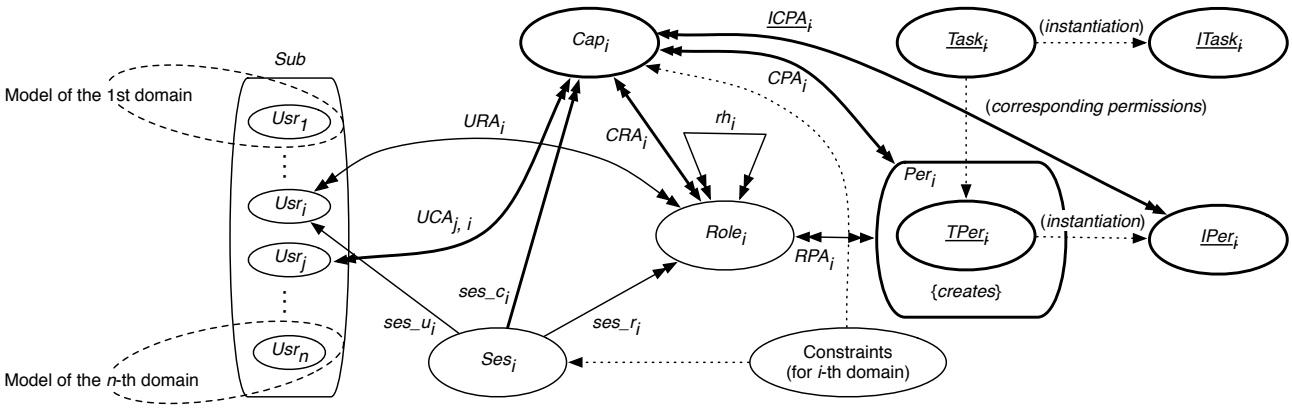

Figure 1. The family of W-CRBAC models (for i-th domain)

In addition to the above components, the following functions and relations are defined for the i-th domain for each $i \in Dom$:

- $usr : Dom \rightarrow 2^{Sub}$, a function that determines the set of users in the i-th domain for each $i \in Dom$. We also use the notation Usr_i to denote $usr(i)$ and assume that $Sub = \bigcup_{i \in Dom} Usr_i$.
- $ses\_u_i : Ses_i \rightarrow Usr_i$, a function mapping each session in the i-th domain to the user who has established it.
- $ses\_r_i : Ses_i \times 2^{Role_i}$, a function mapping each session in the i-th domain to the set of roles that is activated by this session.
- $URA_i \subseteq Usr_i \times Role_i$, a many-to-many user-to-role assignment relation.
- $RPA_i \subseteq Role_i \times Per_i$, a many-to-many role-to-permission assignment relation.
- $ses\_c_i : Ses_i \rightarrow 2^{Cap_i}$, a function mapping each session in the i-th domain to a set of capabilities.
- $UCA_{j,i} : Usr_j \rightarrow 2^{Cap_i}$, a function mapping each user in the j-th domain to a set of capabilities.
- $CRA_i \subseteq Cap_i \times Role_i$, a many-to-many capability-to-role assignment relation.
- $CPA_i \subseteq Cap_i \times Per_i$, a many-to-many capability-to-permission assignment relation.

These components satisfy the following conditions:

(C0-1) $ses\_r_i(s) \subseteq \{r \mid \langle ses\_u_i(s), r\rangle \in URA_i\}$, which means that any role activated by a session is one that is assigned to the user who establishes the session.

(C0-2) Session s has the permissions $\bigcup_{r \in ses\_r_i(s)} \{p \mid \langle r, p\rangle \in RPA_i\}$.

(C0-3) $ses\_c_i(s) \subseteq \{c \mid \langle ses\_u_i(s), c\rangle \in UCA_i\}$, which means that any capability activated by a session is owned by the user who establishes the session.

(C0-4) For any $r \in Role_i$ and $c \in Cap_i$, if $\langle c, r\rangle \in CRA_i$ session s has the permissions that are determined by (C0-2) above, otherwise (i.e., $\langle c, r\rangle \notin CRA_i$) s has the permissions $\bigcup_{c \in ses\_c_i(s)} \{p \mid \langle c, p\rangle \in CPA_i\}$. □

As explained before, CRBAC0 is a pure extension of the RBAC96 model, where the components of Cap_i, ses_c_i, $UCA_{j,i}$, CRA_i, and CPA_i are the extended parts. The first five components play the following roles: Cap_i denotes the set of capabilities issued in the i-th domain; function ses_c_i determines the set of capabilities activated by each session; function $UCA_{j,i}$ represents the owner of the capabilities; and relations CRA_i and CPA_i determine, respectively, which roles and permissions are assigned to each capability.

Before presenting the definition of W-CRBAC0, we here introduce the definitions of workflow specification and its instantiation as follows.

Definition 2 (Workflow specification): A workflow specification W is a partially ordered set of abstract tasks $(Task_i, \leq)$, where $Task_i = \{t_1, \ldots, t_m\}$ is a set of tasks in domain $i \in Dom$. □

Intuitively, the instantiation is an assignment of a subject to each task in a workflow specification. The formal definition is as follows.

Definition 3 (Instance of workflow): An *instance* w *of workflow* $W = (Task_i, \leq)$ is a sequence $[(t_1, u_1), \ldots, (t_m, u_m)]$ of pairs of tasks and subjects, such that $Task_i = \{t_1, \ldots, t_m\}$, $\{u_1, \ldots, u_m\} \subseteq Sub$, and $t_j \leq t_k$ for any j, k with $j \leq k$. We also use $ITask_i^w$ to denote the set $\{(t_1, u_1), \ldots, (t_m, u_m)\}$ of concrete (i.e., instantiated) tasks in instance w. □

In addition, for a given workflow and its instance, we consider the corresponding permissions as follows.

Definition 4 (Permissions in workflow): For an abstract task t in a workflow specification $W = (Task_i, \leq)$, the permission to perform t in any instance of W is denoted by $per(t)$. The set $\{per(t) \mid t \in Task_i\}$ is denoted by $TPer_i$.

On the other hand, for a concrete task t in an instance w of $W = (Task_i, \leq)$, the permission to perform t in instance w is defined as tuple $\langle u, per(t)\rangle$. □

In terms of these notions, W-CRBAC0 is defined as follows.

Definition 5 (W-CRBAC0): The W-CRBAC0 model (for a workflow specification $W = (Task_i, \leq)$ and for a set of instances $\{w_1, \ldots, w_k\}$ of W) is obtained from CRBAC0 by adding the following components:

- $Task_i$, the set of abstract tasks in W.
- $ITask_i^{w_1}, \ldots, ITask_i^{w_k}$, the sets of concrete tasks.
- $TPer_i$, the set of corresponding permissions to the abstract tasks in $Task_i$.
- $IPer_i$, a set of permissions to perform concrete tasks, initially defined as $\{\langle u, per(t)\rangle \mid \langle u, r\rangle \in URA_i,\ \langle r, per(t)\rangle \in RPA_i,\ t \in Task_i\}$ (which represents the set of all concrete permissions determined by role assignments).
- $ICPA_i \subseteq Cap_i \times IPer_i$, a many-to-many assignment relation between capabilities and permissions to perform concrete tasks. □

To make our discussion simpler, in the rest of this paper, we consider only a fixed single workflow instance. We thus omit the superscript w.

C. Delegation of Tasks

Based on W-CRBAC0 and the workflow model, we define the delegation of authority of abstract tasks and concrete tasks using capability transfer. (Although our model also supports delegation of roles, due to space limitation, we omit the delegation of this type. Cf. also [6].) As mentioned in Section III-A, the delegation process can be regarded as the sequential composition of the following three basic operations:

(Step 1) *Creation*: A user creates a new capability.
(Step 2) *Assignment*: A user assigns authority to the capability.
(Step 3) *Transfer*: A user sends it to another user.

In each of the first two steps, we consider the following cases. In Step 1, permission to create new capabilities is assigned to either (1) a role, or (2) a capability owned by the creator. In Step 2, the authority consists of either (3) a set of authorities of abstract tasks or (4) concrete tasks. According to these cases, we classify delegations into four types as shown in the following table. However, more precisely, DC-I is further divided into the following two cases: the permission assigned to a capability owned by the creater is to perform an abstract task or a concrete task. In the table, these two cases are named "DC-AI" and "DC-II", respectively.

Step 2 \ Step 1	(1) Role	(2) Capability
(3) Abst. tasks	DR-A	DC-A
(4) Inst. tasks	DR-I	DC-AI/-II

Furthermore, in Step 3, we classify each of these five types into two categories depending on whether or not the delegation is carried out across multiple domains.

To provide a formal definition of the delegation of authority in our model, we first represent Steps 1 to 3 as a set of rules for state transitions of the model. (In the definition, we use the symbols $S, S', \ldots$ to denote states.) Then, by considering all possible sequential compositions of the transitions, we can obtain the delegation of these types.

Definition 6 (Basic operations for delegation): The basic operations for delegation (i.e., *creation*, *assignment*, and *transfer*) are defined by the following rules for state transitions of a model.

Creation rules: Let S be a state, in which $u \in Usr_i$, $c \in Cap_i$. If user u has (1) role r or (2) capability c to which permission "*create*" is assigned, then he can create a new capability c'.

Formally, these rules are defined as follows.

(1) If $r \in Role_i$ with $\langle u, r\rangle \in URA_i$ and $\langle r, create\rangle \in RPA_i$, then this rule can be applied to S and defines new state S' by replacing Cap_i in S with Cap_i', such that $Cap_i' := Cap_i \cup \{c'\}$ (where $c' \notin Cap_i$).
(2) If $\langle u, c\rangle \in UCA_i$ and $\langle c, create\rangle \in CPA_i$, then this rule can be applied to S and defines new state S' as (1).

Assignment rules: Let S be a state, in which $p \in TPer_i$, $u \in Usr_i$, $t \in Task_i$, and c' ($\in Cap_i$) is created by u from his role r or capability c. If (AA) permission p to perform an abstract task is assigned to r or c, then (3) he can assign p to new capability c'. Also, if (AI) permission p to perform an abstract task is assigned to r or c, or if (II) permission $\langle u, per(t)\rangle$ to perform concrete task $\langle u, t\rangle$ is assigned to c, then (4) he can assign this permission to c'.

Formally, these rules are defined as follows.

(3) If $\langle u, r\rangle \in URA_i$ and $\langle r, p\rangle \in RPA_i$, or if $\langle c, p\rangle \in CRA_i$, then this rule can be applied to S and defines new state S' by replacing CPA_i in S with $CPA_i' := CPA_i \cup \{\langle c', p\rangle\}$.
(4) If one of the following conditions holds
- (For AI) $\langle u, r\rangle \in URA_i$ and $\langle r, p\rangle \in RPA_i$, or if $\langle c, p\rangle \in CPA_i$,
- (For II) $\langle u, per(t)\rangle \in ICAP_i$ and $\langle c, \langle u, per(t)\rangle\rangle \in ICPA_i$,

then this rule can be applied to S and defines new state S' by replacing $ICPA_i$ in S with $ICPA_i' := ICPA_i \cup \{\langle c', \langle u', per(t)\rangle\rangle\}$ for any $u' \in Sub$.

Transfer rule: Let S be a state, in which $u \in Usr_i$, $u' \in Usr_j$, $c' \in Cap_i$, and c' is created by u (where i and j may be the same domain number). Then new capability c' can be transferred from user u to another user u'. In particular, such a transfer is referred to as a *cross-domain transfer* if $i \neq j$.

Formally, this rule can be applied to S and defines new state S' by replacing $UCA_{i,j}$ in S with $UCA'_{i,j} := UCA_{i,j} \cup \{\langle u', c' \rangle\}$. □

By considering all possible sequential compositions of these basic operations, we can define our aimed delegations as shown in the table. (See also [6] for the detailed definition.)

Here, it should be noted that the delegation operations defined in Definition 6 are known as "grant delegation", i.e. delegator has the delegated permissions after the delegation. To define the other type of delegation operations, called "transfer delegation" can be also defined by modifying the assignment rules.

In closing of the definitions of our base model and workflow, to help the readers understand the formal definitions we provide a simple example.

Example 1: Consider a development process of some electronic device in Hardware R&D Division of a company. One day, to support development of control software in the device, Alice is invited from Information System R&D Division and assigned to an abstract task, say t, in a workflow. Since then, however, Alice should be absent from the office, thus she delegate another staff, say Bob, in the same division.

To deal flexibly with such a situation, our model provides different types of delegations for different purposes. For example, delegating task t to Alice can be realized by the following procedure: the project leader creates a new capability (say, c) by activating his role and assigns permission to perform task t, and then sends it to Alice. In our model, this delegation process can be described by the following state transtions. First, by creation rule for case (1), we obtain a new state by replacing Cap_H with $Cap'_H := Cap_H \cup \{c\}$. Then by assignment rule for case (3), we obtain a new state by replacing CPA_H with $CPA'_H := CPA_H \cup \{\langle c, per(t) \rangle\}$. Finally, by the transfer rule, we obtain a new state by replacing UCA_H with $UCA'_H := UCA_H \cup \{\langle Alice, \{c\} \rangle\}$.

On the other hand, when Alice delegates her task, she may choose a suitable type among DC-A and DC-AI in accordance with the aim. For example, DC-A would be used if she should leave the office for a long time due to an extended business trip or hospitalization, while DC-AI would be used to delegate a concrete task. Furthermore, during the absence of Alice, if Bob needs to delegate the task to another staff and if his delegated capability has the ability to create a new capability, it can be realized by Bob without asking the administrator in Hardware Division. However, such flexible delegation may cause a security issue, namely unintended propagation of capabilities. To avoid this issue, we also provide some kinds of constraints on capabilities, which shall be presented in Section III-E.

D. W-CRBAC1

Based on the W-CRBAC0 model, we develop an extended model, called W-CRBAC1, by introducing role hierarchy. This extension is essentially the same as that in RBAC1. W-CRBAC1 is defined as follows:

Definition 7 (W-CRBAC1): The W-CRBAC1 model is obtained by adding rh_i, a partial order over $Role_i$, called a *role hierarchy*. (The infix notation $r' \geq_{rh_i} r$ is also used to denote the role hierarchy and we can say that "r' is a senior role of r" or "r is a junior role of r'" in the same sense as in the RBAC96 model.)

In addition to the above, the following conditions are satisfied for each i.

(C1-1) $ses\_r_i(s) \subseteq \{r \mid \exists r' \geq_{rh_i} r(\langle ses\_u_i(s), r' \rangle \in URA_i)\}$.

(C1-2) Each session $s \in Ses_i$ has the set of permissions $\bigcup_{r \in ses\_r_i(s)} \{p \mid \exists r'' \leq_{rh_i} r(\langle r'', p \rangle \in RPA_i)\}$.

(C1-3) Each session $s \in Ses_i$ has the set of permissions $\bigcup_{r' \in \{r \mid \langle c_i, r \rangle \in CRA_i \wedge c_i \in ses\_c_i(s)\}} \{p \mid \exists r'' \leq r' (\langle r'', p \rangle \in RPA_i)\}$

The former two conditions are related to role hierarchy, and are the same as in the RBAC96. Intuitively, condition (C1-1) imposes the requirement that every role activated by session s is equally or less powerful (junior) than any role of the user establishing session s. (C1-2) imposes the requirement that the permissions in session s are those directly assigned to the session's roles and all of their junior roles. On the other hand, the third condition guarantees the inheritance of permissions in terms of the role hierarchy. In other words, (C1-3) means that if a role is assigned to a capability and it is activated, then all the permissions are assigned to this role and to all the junior ones.

E. W-CRBAC2

Similar to the development process of the RBAC96 family, here we develop another extension, called W-CRBAC2. In this paper, we focus on the constraints on a capability, and in the next section we explain how these constraints are used to prevent unintended propagation of authority to perform tasks using an example workflow.

As examples, we introduce the following two constraints.

(R1) Lifetime

(R2) Number of activations

There are also other constraints, such as the number of creations of new capabilities, the number of hops in a capability transfer, or the depth of inheritance of permissions to create new capabilities. (See also [6] for more constraints.)

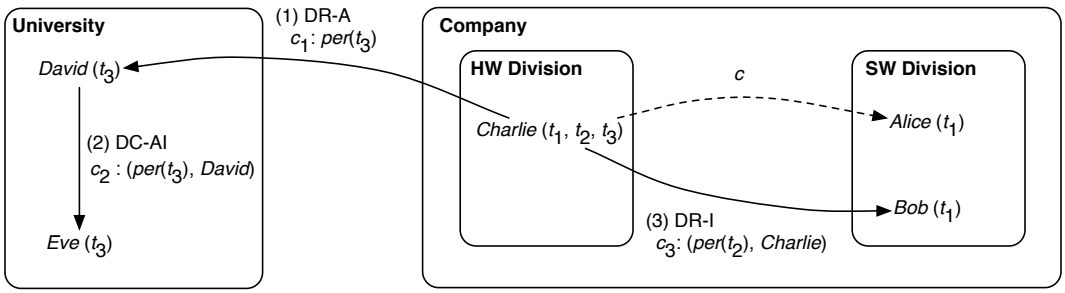

Figure 2. Example 2

We present the formal definitions of constraints R1 and R2 below.

R1 (*Lifetime*): To formalize the notion of lifetime, we first introduce the following constant and function:

- $time$ ($\in \mathbb{Z}^+$), the current time of the global clock represented by a non-negative integer;
- $life_i : Cap_i \to \mathbb{Z}^+ \times \mathbb{Z}^+$, a function mapping each capability in the i-th domain to the time after which it can be activated and that after which it expires.

Using this constant and function, the lifetime of a capability can be represented by the following conditions for any $c \in Cap_i$ and $i \in Dom$:

- $fst(life(c)) < time \Rightarrow ses\_c_i(c) \notin Ses_i$;
- $snd(life(c)) > time \Rightarrow ses\_c_i(c) \notin Ses_i$.

Here fst and snd are the first and second projections of the input.

R2 (*Number of activations*): To define this constraint we first introduce the following functions:

- $lim\_act_i : Cap_i \to \mathbb{N}$;
- $cnt\_act_i : Cap_i \to \mathbb{N}$.

The former determines the possible number of activations, while the latter counts the number of activations thus far. By using these functions, this constraint can be represented as the following condition for any $c \in Cap_i$ and $i \in Dom$:

$$lim\_act_i(c) \geq cnt\_act_i(c).$$

IV. Example of Delegations

In this section, we demonstrate how to apply our model to workflow systems by considering the following example scenario, which is an extended version of Example 1. Here we consider three tasks, t_1, t_2, and t_3. Initially, Charlie is assigned to these tasks, and then t_1 and t_3 are delegated by capability transfers. (See also Fig 2 for the graphical presentation of this scenario. In this figure, assigned tasks are denoted after the corresponding user name, while delegations are depicted as arrows.)

Example 2: After Charlie, the project leader, delegated the abstract task t_1 to Alice via the capability c (described in Example 1), he also invites David, a professor in a university, to join the team for promoting the project by business-academia collaboration. David accepts the invitation, so Charlie creates a new capability (called c_1) by activating his role and assigns the permission to perform abstract task t_3, and then passes it to David. At this time, Charlie permits David to delegate his task to Eve, one of his students, if necessary. So, during his absence, David creates a new capability (called c_2) from this capability c_1 then delegates a concrete task $\langle per(t_3), David\rangle$ for each time when Eve works in his place. On the other hand, one day Alice is absent without notice, so Charlie creates a new capability (called c_3) and delegates a concrete task $\langle per(t_1), Charlie\rangle$ to Bob via c_3.

The delegations appearing in this scenario (denoted by (1) to (3) in Figure 2) are respectively classified into DR-A, DC-AI, and DR-I. In our model, these delegation processes can be described as state transitions. For example, the second delegation (i.e., from David to Eve) is described by the following state transitions (where Cap_H, $ICPA_H$, and $UCA_{H,Univ}$ are in the state before this delegation): $Cap'_H := Cap_H \cup \{c_2\}$; $ICPA'_H := ICPA_H \cup \{\langle c_2, \langle David, per(t_2)\rangle\rangle\}$; $UCA'_{H,Univ} := UCA_{H,Univ} \cup \{\langle Eve, c_2\rangle\}$.

Finally, we would like to note that the following points. First, as the scenario indicates, our model supports a flexible user-to-user delegation of both abstract tasks and concrete tasks without any authentication. This would be helpful to cope flexibly with unexpected situation over different domains. On the other hand, our model provides delegators to set some constraints on the propagation of capability by combining basic constraints. This would be useful to compensate for the intrinsic weakness in the capability-based access control. For example, considering the scenario in Example 2, it would be better to restrict the lifetime (R1) or the number of activations (R2) of capability c_3. Also, it would be better to restrict the number of hops capability transfer (cf. [6]) of capability c_1 so that David cannot propagate unnecessary permissions.

V. Discussion

In this section we briefly discuss how this model can be implemented. A possible way is to implement RBAC permissions using access control list (ACL) approach and the capability-based permissions are implemented by extension of the ACL approach. That is, the i-th domain (for each i) has a reference monitor that manages RBAC permissions by ACL-based access control mechanism with two matrices specifying URA_i and RPA_i. Also, a reference monitor stores workflow specifications and the execution schedules (which are not explicitly modeled in this paper). When a user in the j-th domain tries to perform a concrete task in the i-th domain by activating his capability (say, c), the reference monitor checks the validity of this capability then dynamically adds a tentative user, whose name is just specified as "guest", to the matrix of URA_i. If some permissions are assigned to c, then the specific role, called r^*, is assigned to the tentative user in the matrix of URA_i, while a list which specifies that all the permissions of c are assigned to r^* is also added to the matrix of RPA_i, thereby the guest user has all the permissions. (Note that r^* plays a role in associating the guest user with the permissions.) When performing a concrete task, a reference monitor checks the authority by referring to the extended matrices. Then, after closing the session, the server removes both the tentative user and the corresponding permissions from the matrices.

VI. Conclusions and Future Work

We presented an access control model called W-CRBAC, by extending the CRBAC model, which was introduced in our previous work [6]. To realize flexible cross-domain delegation of tasks and roles, we introduced the notion of authority to perform both abstract and concrete tasks, and mapped these to capabilities to be transferred. Owing to the capability-based access control mechanism, our model provides both flexibility and reduced administration costs, thus making it possible to cope with emergent calls for reinforcement or unexpected changes in task assignments in large-scale workflows. On the other hand, we proposed various constraints on capabilities to prevent unintended propagation of authority of tasks.

In future work, we intend developing formal methods to check the satisfiability problem considered in [4] and to verify security in the W-CRBAC model. In particular, we are interested in a method for checking satisfiability of delegation conditions, as well as unintended propagation of capabilities, by searching all possible traces of workflow executions using a model-checking approach.

Acknowledgments

This study was supported in part by the Grant-in-Aid for the Global COE Program on "Cybernics: fusion of human, machine, and information systems" at the University of Tsukuba.

References

[1] V. Atluri and J. Warner. Supporting conditional delegation in secure workflow management systems, *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT'05)*, pp.49-58, 2005.

[2] E. Barka and R. Sandhu. A Role-based Delegation Model and Some Extensions. *Proceedings of the 23rd National Information Systems Security Conference*, pp.101-114, 2000.

[3] J. Crampton and H. Khambhammettu. On Delegation and Workflow Execution Models, *Proceedings of the 23rd ACM Symposium on Applied Computing (SAC'08)*, pp.2337-2144, 2008.

[4] J. Crampton and H. Khambhammettu. Delegation and Satisfiability in Workflow Systems, *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (SACMAT'08)*, pp.31-40, 2008.

[5] R. Geambasu, M. Balazinska, S. D. Gribble, and M. Levy. HomeViews: Peer-to-Peer Middleware for Personal Data Sharing Applications, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.235-246, 2007.

[6] K. Hasebe, M. Mabuchi, and A. Matsushita. Capability-Based Delegation Model in RBAC, *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies (SACMAT'10)*, pp.109-118, 2010.

[7] S. Kandala and R. Sandhu. Secure Role-Based Workflow Models, *Database Security XV: Status and Prospects*, pp.45-58, 2002.

[8] H. M. Levy. *Capability-Based Computer Systems*, Digital Equipment Corporation, 1984.

[9] J. T. Regan and C. D. Jensen. Capability File Names: Separating Authorization from User Management in an Internet File System, *Proceedings of the USENIX Security Symposium*, pp.211-233, 2001.

[10] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models, *IEEE Computer*, vol.29, no.2, pp.38-47, 1996.

[11] L. Zhang, G. Ahn, B. T. Chu. A rule-based framework for role based delegation, *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT'01)*, pp.153-162, 2001.

[12] X. Zhang, S. Oh, and R. Sandhu. PBDM: A Flexible Delegation Model in RBAC, *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT'03)*, pp.149-157, 2003.

[13] J. Wainer, A. Kumar, and P. Barthelmess. DW-RBAC: A formal security model of delegation and revocation in workflow systems, *Information Systems*, vol.32, pp.365-384, 2007.