Team-and-Role-Based Organizational Context and Access Control for Cooperative Hypermedia Environments

Weigang Wang
GMD - German National Research Center for Information Technology
IPSI - Integrated Publication and Information Systems Institute
Dolivostr. 15, D-64293 Darmstadt, Germany
wwang@darmstadt.gmd.de

ABSTRACT
Access control needs to be more flexible and fine-grained to support cooperative tasks and processes performed by dynamic teams. This can be done by applying state-of-the-art role-based access control (RBAC) technology. This paper examines how to integrate RBAC in a team-based organization context and how to apply such access control to hypermedia structures. Based on the analysis of these issues, a team-and-role-based access control model is proposed, which describes various aspects of role-based access control in cooperative hypermedia environments. The model has been implemented in CHIPS, a cooperative hypermedia-based process support system. Application examples demonstrate that its organizational context management and access permission authorization retain the simplicity of RBAC. Our extensions provide effective and flexible access control for managing various kinds of shared workspaces, especially shared process spaces, where access control is not only used for managing security, but also for supporting coordination.

KEYWORDS
Cooperative hypermedia, groupware, coordination, workflow, role-based access control, process support

INTRODUCTION
Due to the emerging trend towards distributed and virtual organizations there is a growing demand for better support of processes executed by distributed, virtual teams. Groupware is one technology that aims at providing such support. Within the groupware area cooperative process support environments specifically address the issues of flexible coordination and cooperation support for emerging processes tackled by virtual organizations [8, 16]. A cooperative process support environment provides support for distributed teams to cooperatively define and modify emerging processes as well as to cooperatively execute these processes. Access control can address two central issues in such an environment: Firstly, it can be used to restrict access to information and functionality in the shared environment to those trusted. Secondly, it can be used to help coordination by providing only those functions to team members that are currently needed to fulfil their role. The latter is especially useful when dealing with emerging or frequently changing processes. Due to the dynamic nature of virtual organizations, these environments have to provide a great deal of flexibility to cope with changing teams, processes, and cooperation styles. These requirements raise many challenging issues, such as how to manage access permissions assigned to members of dynamic teams and how to provide different access permission overtime upon emergent process structures.

Hypertext 99 Darmstadt Germany

The requirements on access control derived from this situation lead to complex access models developed in groupware and workflow areas [15, 5]. However, none of these models has been fully implemented and enjoyed large-scale usage as part of a widely used CSCW systems [17]. The complexity of such models makes it far from trivial to design a user interface that offers the user an adequate set of access control operations and which is easy to understand [6, 14, 16, 17].

Role-based access control (RBAC) is an alternative to traditional discretionary and mandatory access control policies [11]. It can simplify authorization management and provide flexible access control policies. In RBAC, roles are created for the various job functions in an organization and users are assigned roles based on their responsibilities and qualifications [11]. The essence of RBAC is that permissions are assigned to roles rather than to individual users. Users acquire these permissions by virtue of being assigned membership in appropriate roles. In this way, the task of specifying user authorization is divided into two logically independent parts: one which assigns users to roles and one which assigns access rights for objects to roles. This greatly simplifies authorization management, because authorization can be administrated as a whole for all users belonging to a role, rather than at the level of individual users, objects, and permissions [18]. Users can be easily reassigned from one role to another without modifying the underlying access structure. Roles can be granted new permissions as new applications or actions are incorporated, and permissions can be revoked from roles as

needed [11]. RBAC is a policy-rich framework [11]. It facilitates the definition of flexible, customized policies adaptable to an organizational structure and to the means of conducting business. Also, policies implemented under RBAC can evolve over time as organizational structure and security needs change.

The Problems

To support the emerging virtual organization forms and their work styles, it seems a natural idea to maintain an organizational context for both the team-based virtual organization structures and the role-based access control mechanism in a cooperative hypermedia environment. However, there are still many open issues on how this can be done. Problems of applying RBAC to cooperative hypermedia include:

- How to integrate RBAC in a team-based virtual organizational context? I.e. how to deal with changing teams, shifting responsibilities and emerging processes represented in hypermedia? Whether to define roles within each team or across teams in an organization?
- How to apply such access control to hypermedia structures? I.e. how to effectively specify permissions for different object types and specific instances? How to deal with composites and navigation between composites?
- How to balance between the complexity of fine-grained access control (in terms of many different operations explicitly controlled) and the need for simple-to-use solutions (requiring few, easy-to-understand categories of operations)? How to make access control for cooperative hypermedia understandable by end users?
- How to support context-dependent access control (i.e. when the access permissions vary with the state of cooperation or the state of the hypermedia workspace)?

Our Approach

Our approach to these problems can be outlined as follows:

- An organizational context characterized by teams and roles is integrated in a cooperative hypermedia environment for supporting access control and other groupware features. Teams and roles are used in conjunction to deal with access control in a dynamic organizational context. Roles are defined across teams in an (extended) organization;
- Access permissions can be assigned based on the types or the instances of hypermedia objects. The node nesting structures of composite nodes are used as folders or wrappers representing workspaces. The access permissions defined by roles upon object types are independent of teams and individuals;
- To ease understanding, operations upon hypermedia objects are classified into four categories: Query, Update, Execute, and Assign. They correspond to the well-known "read", "write", "execute", and "own" modes in the UNIX operating system; and
- To support context dependent access control to process structures, teams and states of tasks or processes are represented in a "Protection State" and used in the access review procedure. Individuals and their permissions for specific objects are brought together by teams that are assigned to composite nodes.

The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 analyses the problems raised in the introduction sections. In Section 4, a scenario is given to provide an overview on the kind of access control we want to support. Section 5 then introduces our proposed access control model. Section 6 presents an implementation of the model in the CHIPS system. Section 7 presents an application example. Finally section 8 compares our approach to related work, and discusses future work.

RELATED WORK

In the last few years, many RBAC models have been developed and efforts have been made towards a general reference model for role-based access control [12]. However, there are only a few RBAC models developed specifically for cooperative environments. Shen and Dewan developed such an access control model [15]. They identified a set of collaborative rights and developed a set of inheritance rules on the subject, object, and access rights dimensions. The model is quite general and complex. It is not clear how its user interfaces is designed and how its access rights are specified by end-users. Edwards proposed a model with dynamic roles (such as 'people who are in my lab right now') for users to define the behavior of a system in reaction to the state of the world [5]. These two models present novel access control concepts, but they also add quite an amount of complexity. Sikkel developed a group-based access control model for the BSCW system that supports asynchronous sharing of documents [17]. Access permissions on documents and workspaces are granted to user groups for specific operations. User groups can be used to model roles by, for instance, assigning a job function name to a group and defining many subgroups for various tasks. However, such modeling has some limitations, because their groups are collections of people without the attributes and operations for various types of roles. In addition, groups are not typed, so that they can not be used in different ways in the access review procedure.

The RBAC approach has also been applied to the workflow systems area. Thomas proposed some ideas on a team-based access control model as a primitive for applying role-based access controls in collaborative environments [17]. However, as the model has not yet been fully developed, it is not clear how to incorporate the team concept into a general RBAC framework. Atluri and Huang developed a workflow authorization model, in which an authorization is a primitive concept representing the fact that a subject has a privilege on an object for a certain time interval [2]. Bertino et al. proposed a model on specification and enforcement of role-based authorization constraints for workflow systems

[4]. Each of them addressed an important area of access control in workflow systems while they have not placed a focus on dealing with the needs for process support systems that are built on shared information spaces.

Hypermedia systems can be used to provide shared information spaces for networked chunks of information. With the advent of the WWW, hypermedia has become an everyday tool that can also be used to support collaboration. Usually, hypermedia systems used standard user-based or group-based access protection based on file systems or databases [14]. A good example is KMS [1], which provides a user-based access model for individual nodes and guarantees workspace consistency using a versioning approach. A more prominent example is BSCW, which has already been discussed (see above). To our knowledge, there is no published work dedicated to a role-based access control model for cooperative hypermedia environments.

Historically, RBAC has been developed for supporting access security in non-collaborative application domains. Although it has been successfully used in many commercial applications, such as the latest versions of Sybase, Informix, and Oracle databases [9], its focus has not yet been placed on supporting collaboration and coordination. A RBAC model for cooperative hypermedia environments in general and for cooperative process support environments in particular is still missing.

PROBLEM ANALYSIS

In addition to role-based access control, a team-and-role-based organizational context can be also used to support many other groupware features, such as group awareness, notification, and job balancing. However, how can we integrate RBAC in such an organizational context and how can we apply such an integrated RBAC to hypermedia structures? More specifically, should we define roles for each team or across teams in an organization? How to apply such access control on hypermedia components and composites? How to deal with access control for cooperative process definition and execution? Next, an analysis on these problems is given and some answers to these questions are suggested. They provide a basis for our team-and-role-based access control model.

Roles, Teams, and Individuals

The concept of role stems from organizational theory that has a much longer history then role-based access control models. However, the scope of the meaning of roles used in current role-based access control models is much narrower than their original meaning in an organizational context [10]. A "role" in an organizational context is a job function defined as a named collection of responsibilities, which reflect organizational regulations and business procedures. When used for access control purpose, a role is defined as a named collection of access permissions, or a named collection of users and access permissions [10]. When talking about a role for access control purpose, what comes to people's mind is the purpose of the role as indicated by its name. This purpose may have a meaningful match to the permissions, if the permitted actions or their corresponding tasks are defined in a higher-level, application-specific way reflecting the responsibilities of the role. The association of a role to a collection of users taking the role is implicit.

The typical roles for access control purpose are "Organizational roles", which include professional roles (such as software engineer), domain expert roles (such as Smalltalk expert), and administrative roles (such as manager). These roles provide a meaningful classification of people; independent of the teams the people work in. A process role in workflow area is defined as a mechanism to associated participants to a collection of workflow activities. Compared with widely understood organizational roles, process roles are more task-specific and temporary. Nevertheless, in the trend towards dynamic virtual teams, such roles may be used increasingly often and become a kind of organizational role. Therefore, we consider them as organizational roles.

"Group" is a very general term referring to a collection of people. By definition, it should have at least two persons, but this restriction is not always important in practice. For access control purpose, a group (or user group) serves as a shorthand notion of a collection of users who share a set of whatsoever access permissions that may not be inferred from the arbitrary name of the group.

For access control purpose, both group and role serve as shorthand notions that bring a collection of users and a collection of access permissions together. From this point of view, it is possible to use groups to model roles or use roles to model groups. However, from the organizational point of views, such mixed use of concepts may be counterintuitive. It might be better to make a clear classification and use different terms for different categories of roles or groups.

"Team" is another organizational concept. A team is a group of people working together for a common task. In virtual organizations, teams also serve as work units for their projects. Cooperative tasks are often carried out by multi-disciplinary teams, in which multiple roles are taken by their members.

To bring people together in a team, based on the needed roles, for a timely challenging business opportunity is one of the characteristics of virtual organizations. People working in such a team may also work in other teams and they may live in different cities or countries. Therefore, it makes sense to maintain roles across teams in an (extended) organization (that includes real plus virtual organizations), rather than within each team in such an organization. On the other hand, teams can be set up on a per project basis. A team (or teams) can be assigned to a workspace (or a process) as a whole to allow team members to access and perform the tasks of the process. In addition, roles can be assigned to parts (i.e., contents or sub-workspaces) of the workspace for fine-grained access control. The system can use the roles in conjunction with the team(s) to identify

relevant team members who can access and perform the corresponding tasks in the workspaces (or processes).

In some occasions, user-based access control is still needed. For example, when a cooperative task is very informal and involves only a few collaborators, to assign permissions indirectly though roles may do more trouble than help. Therefore, it is desirable to accommodate this option in a role-based access control model.

Hypermedia Components and Composites

Hypermedia objects, as defined in the Dexter model [7], are components (node and link objects) and composites (structures). Analog to the classification of users according to their roles, hypermedia components and composites can be classified according to their semantics types. Access authorizations of roles can be assigned based on object types or specific objects. When they are assigned based on types, access authorization for each object is automatically determined according to the type of the object. There is no need of specifying authorization upon each object. If a specific object needs to have a set of permissions that are different from those inherited from its type, the inherited permissions can be overwritten.

Typical hypermedia composites (composite nodes) are constructed by means of non-linking containment relationships of nested nodes. Composite nodes can be used as nested folders or wrappers for creating various shared information spaces. Teams for a workspace can be assigned to a composite node. The components contained in a node usually have all the permissions of their containing node unless more fine-grained permissions are assigned to them. The composite mechanism may also control whether or not to allow links from outside of a composite connect to its components. When such 'jump in from side doors' is allowed, the access permissions defined upon the component and its containing node will still put the person entering from the 'side doors' under access control. It is very likely that he or she can only read the part and can not go to any other parts of the composite, if he or she has no such permissions.

Access and manipulation operations upon hypermedia objects can be classified within their object types or across all the object types. The classification on a per type basis can be more fine-grained, while classification across all the types can be easier to handle by end-users. In addition, if task-specific operations are defined upon hypermedia structures, application-specific classifications can be made, which could provide a better mapping from access permissions (roles for access control) to the responsibilities of roles (roles in organizational context).

Cooperative Process Definition and Execution

Wang and Haake in [19] presented a general cooperative hypermedia based process model and a general method to incorporate process-related semantics into hypermedia structures. In such a process model, tasks are represented by task nodes; Control and data flow connectors between tasks can be represented by process links. Process definition corresponds to hypermedia schema definition and template creation. Process execution corresponds to an extended guided tour [19].

When hypermedia structures are used to represent processes, the states of a task (or process) should be reflected in access permission representations. To maintain the integrity of a shared process structure and to keep a cooperative process execution under proper control, there should be dedicated access rights for manipulating process definitions and for executing processes. The rights for process execution may also include rights for switching among cooperation modes (i.e., loosely coupled and tightly coupled modes) and for handling cooperative transactions (i.e., check-in/check-out a shared workspace). In each cooperative session, one person may take multiple roles at a time and multiple roles may work in the same workspace. The permissions assigned to roles and teams would not change within a transaction.

To address the requirement of automating the change of access permissions of objects used by several cooperative tasks (e.g. workflow steps); the concept of a 'wrapper' is developed. A wrapper can "wrap" a collection of objects of the same or of different types. The wrapped objects combine their access permissions with those of their wrappers. To access the wrapped objects, the wrapper has to be opened. A wrapper may correspond to a task in a process. The membership of objects in wrappers may change dynamically as they *flow* from one wrapper into another (e.g., along workflow steps). There could be many different *flow* semantics, such as moving or coping by value. This solves the problem of how to automatically change access permissions of objects (passed to consecutive steps of a process) overtime.

A SCENARIO

The following scenario, from user points of view, provides an overview on the kind of access control we want to support. This, together with the above problem analysis, may help the understanding of the concepts defined and the technical choices made in our team-and-role-based access control model to be presented in next section.

Due to a successful project proposal, a *team* is set up to start the project as outlined in the project proposal. The selection of team members is based on the job functions (*roles*) needed for the project. Then, work plans in the proposal are developed into more detailed work processes. For a process that is new to the team, the team members may work together in a shared hypermedia workspace to create a definition of the process (i.e., a process schema). At this stage, to avoid being to restrictive to people working on an emerging process structure, access control to the workspace can be very loose (i.e., all team members may have the same set of access permissions to the workspace). When more detailed process structure emerges, more fine-grained access permissions to the process structure are assigned to various roles. The need for access control for some shared artifacts may change overtime. This change

can be supported by moving the artifacts from one sub-workspace into anther that has different access permissions. A sub-workspace may be assigned to multiple roles that have different access permissions to different parts of information structure or functions in the workspace. Through such access control, the consensus on the working procedure and job allocation may be maintained.

The process definition together with its role-based access permissions can be reused by all its process instances. The process instances created from the definition can also be performed by other teams without the need to repeat access permission assignment. Just assigning a team to the composite representing a process instance would be sufficient if the team members can take all the roles needed for the process. If one person taking a needed role is absent, other people available for the same role in the team can take over the work. If no people in the team can take a needed role, then a new member can be brought in to take the role and perform the job.

A TEAM-and-ROLE BASED ACCESS CONTROL MODEL

Conceptually, an access control model describes what a protection state is and how protection state transitions occur 514]. The general subject-action-object based protection state can be represented as a subset of the Cartesian product of *subjects*, *actions*, and *objects*: $P \subseteq S \times A \times O$. Subject s is allowed to perform action a on object o, if there exists $\langle s, a, o \rangle$ in P. Particular to RBAC, S is defined as a set of roles rather than individual users. User u can perform action a on object o, if there exists $\langle r, a, o \rangle$ in P, such that $u \in r$. This protection state representation corresponds to the authorization relation in RDBMS [11].

The principle of dealing with collections of users in the subject dimension by using the *role* concept, that earns the advantage for RBAC, can also be applied to its object and action dimensions. In the following subsections, first our conceptual protection state representation and general access review procedure are introduced, and then the categories developed for each dimension of the protection state representation are described. Finally, the authorization specification and the issue of authorization policies for administrating the role-based access model itself are discussed.

Protection State and Access Review

Conceptually, the *protection states* (i.e., permissions) in our access model can be represented as: $P \subseteq S \times A \times O \times T \times E$, where S is a set of roles, A is a collection of function categories, and O is a collection of object semantic types or instances of these types. The additional dimensions of T and E are context-dependent ones. They denote a collection of teams and a collection of process states respectively. User u can perform function f on object instance i, if there exists $\langle r, a, i, t, e \rangle$ in P, such that $u \in r$, $f \in a$, $u \in t$, $team(i) = t$, $state(i) = e$. Otherwise the access can also be granted if there exists $\langle r, a, o, t, e \rangle$ in P, such that $u \in r$, $f \in a$, $i \in o$, $u \in t$, $team(i) = t$, and $state(i) = e$. Here $team(i)$ denotes a team that is assigned to access object instance i; while $state(i)$ denotes the process state of object instance i. Corresponding to the above conceptual representation, in our model an access control list (i.e., a list of authorized roles and their permitted function categories) is used for representing access permissions for hypermedia objects. An *access-review procedure* is performed to grant or deny an access request. This procedure includes two parts: first the context dependent dimensions of T and E are checked, and then, the normal access control list is checked.

Subject Categories: Roles, Teams, and Individuals

To provide flexible access control, the advantages of user-based, group-based, and role-based access control should be combined. This can be achieved by using three types of roles in a RBAC framework:

- *Organizational roles:* a collection of participants (users) exhibiting a specific set of attributes, qualifications and/or skills. They are the very roles that are supported by most RBAC models. Typically, any of the participants *taking* a particular organizational role has a common set of permissions for performing the job function that is indicated by the name of the role. Organizational roles are usually assigned to actors in a process schema to achieve a global effect on all process instances of the schema. Examples of such roles are "Software engineer", "Hypertext researcher" and "Division manager".
- *Teams:* a collection of participants (users) working in the same work unit or for the same project. They correspond to teams or groups in an organizational structure. Each team has a responsible person (head or manager). Any of the participants *joining* a team shares a common goal and may share a default set of permissions for their cooperative work. The notion of a team role is used to restrict access permissions to those individuals who not only have the right organizational roles but also are associated to the task via team membership. In this way, it associates relevant individuals to a specific group of object instances (e.g., to all task instances in a process instance). Examples of such roles are "CONCERT DIVISION", "TELE-LEARNING GROUP", and "CHIPS PROJECT TEAM".
- *Personal roles* (individuals and their delegates): a collection of individuals acting for a person. Each personal role has a responsible person that the role is primarily representing. Personal roles represent individuals, but they differ from individuals in that a personal role is a collection (although in most time with only one member). It may also include other members that are *named* by the responsible person to act in their name for some of the person's job. Personal roles can be used to create private workspaces. Personal roles are also used in situations where user-based access control is required. Examples of such roles are "haake" and "wwang".

Action Categories: Query, Update, Execute, and Assign

To reduce the conceptual overhead for introducing access control into a cooperative hypermedia environment, we divide functions of each object class that are available at the user interface into four categories: Query, Update, Execute, and Assign. These correspond to the widely used Read, Write, Execute, and Own access modes in operating systems. Query functions allow read-only access, such as open a node (or follow a navigational link) and inspect object attributes. The Update category includes object creation and manipulation functions for non-process objects. Of particular importance to cooperative process support are the Execute and Assign categories. The Execute category covers *rights for process definition and execution*, such as triggering of state transitions and modification of process structures. The Assign category includes functions for access permission assignment and functions for *coupling shared aspects* (such as shared scrollbar and navigation controls). In this way, different access concerns of information objects people manipulate in a process and the process structure themselves can be managed using the same RBAC framework.

Object Categories: Types, Instances, and Composites

In this dimension, objects of each class are divided into many semantic types. These semantic types can be implemented as object prototypes. Prototypes are initialized instances of a class; they are used for creating other instances. An instance created from a prototype inherits all attribute values that have not been set in the instance. Access permissions on prototypes affect all their instances. These permissions can be overwritten for object instances and they can be changed back to the inherited rights (by setting the attribute for access rights to nil). In this way, the advantage of type-based control and fine-grained instance-based access control can be combined.

Wrappers as described in the problem analysis section can be implemented as composite node types. They may correspond to steps (tasks) in a process. The access permissions for wrappers can be assigned when the corresponding process steps are defined. Wrappers may have different semantic types. They can also be nested and component objects within wrappers can set their own access rights if more fine-grained control is needed.

Context Categories: Teams and Process States

Context-dependent access control could be very complicated. To make it simple, currently in our model only responsible teams and process states of hypermedia objects are included. Teams are assigned to composite nodes and are inherited along the node nesting structure of the composite nodes. Therefore, given any hypermedia object, a team (or teams) working on it can be identified.

The process states of hypermedia objects are application-specific, such as process states (ready, running, suspended, completed) for task nodes, process transition states (true, false) for process links, version states (frozen), and document states (draft, stable, release). The state information can be used for constructing various processes from very formal workflow processes to very informal process with only state-based notification support. For the access review procedure to check upon these dimensions, a list of states that prohibit functions of the certain Action categories should be provided.

For other multimedia objects, such text and images, the state dimension of Protection State is irrelevant and therefore is ignored in their access review procedures.

Authorization Specification

The authorization specification in our approach follows the typical two-step process of a RBAC model:

- *User-Role assignment*: Roles and users are created and their attribute values are modified, and users are assigned to or removed from various types of roles. Both user and role in the model are represented by objects with their own attributes. The attributes of users include 'name', 'login name' and 'qualification'. The attributes of roles include 'name', 'coordinator', and 'responsibility'.
- *Permission-Role assignment*: permissions are assigned to roles either for an object type or for an object instance. Roles with permissions on an object are called *authorized roles* of the object.

For objects involved in a process (or any cooperative task), there are two additional steps:

- *Role-Actor assignment*: Roles authorized in the above step are assigned to actors of each task in a process (usually for task node prototypes, and sometimes for task node instances). *Actors* refer to those authorized roles that may be notified to perform the tasks.
- *Team-Task assignment*: Teams are assigned *to* a process *instance (or any cooperative task)*. For example, they might be assigned to the root node of a process instance, and/or to its sub-task node instances. The latter is important if the teams assigned at a higher level of the task hierarchy differ from those at a lower level. Given authorized roles, actors and teams as defined in the above steps, two specific collections can be defined (computed): *authorized team members for a task* and *authorized actors for a task*. The *authorized team members* are defined as ($\cup$ authorized roles) $\cap$ ($\cup$ teams) and the *authorized actors* are equal to ($\cup$ actors) $\cap$ ($\cup$ teams). They relate specific individuals to specific object instances. This knowledge is used for access review and notifying relevant individuals to take over a task.

Administration Policies

Also important for authorization specification and access review are administrative authorization policies. They determine who is authorized to modify the allowed accesses. In our model, by default an ownership policy and a decentralized administration policy are adopted. Other policies are possible. By default, a user is considered the

owner of the objects he or she creates. The owner can grant and revoke access rights for other roles to the object. The owner can also grant other roles the privilege of administrating authorization for the object. If this done, then the owner relinquishes his or her own assignment right for the object. However, if all the roles revoke themselves such rights, then the owner's assignment right would resume. In addition, a "super user" role is supported to by-pass access control in exceptional situations. Private workspaces can be created by owners or by using personal roles. One person can play multiple roles and multiple people taking different or the same roles can work together in the same workspace in a cooperative session.

Three types of roles are managed differently. Personal roles are managed by the responsible person (who takes the 'Coordinator' role of the personal roles when they are created). Only the responsible person can name his or her delegates. Team roles are managed by their managers (by default, the creator of the team is the manager of a team (i.e., Coordinator of a team role)). Organizational roles are managed flexibly. Any individual taking any role in the system can create new organizational roles, add members to them, or remove members from them. The 'Coordinator' of the role (as recorded in the 'coordinator' attribute value of the role by default the creator of the role) can modify the attribute values of the role object.

In addition, an 'any user' role represents any users of a system. Like 'super user', the permission assigned to 'any user' is checked in the access review procedure before those to roles and teams are checked. Audit is supported as a deterrence of the misuse of the permission assignment privileges [11].

IMPLEMENTATION IN CHIPS

CHIPS stands for Cooperative Hypermedia Integrated with Process Support [8, 19]. It is a cooperative hypermedia-based process support system. It uses cooperative hypermedia to model both the content of teamwork and the working processes of the teamwork. Access control is an important mechanism of the system to support various shared hypermedia workspaces, including process spaces in different formalities: i.e., from manually coordinated to automated processes.

Strict access control often faces the tradeoffs between security and task accomplishment, while a general tailorable model often meets tradeoffs between flexibility and ease of use. Although we tried to make our model simpler, to meet various requirements in cooperative hypermedia environments led to a still quite complex model for many end users. Nevertheless, it is possible to layout simpler models upon it, and there is still of quite amount of flexibility. Therefore, we target two levels of users. One is for application programmers or power users who can understand the relatively low-level customization interface for tailoring the model to meet their needs. The other is for ordinary users who would rely on a predefined incarnation of the model as a starting point. Such a simplified model has been implemented in our CHIPS system. In the implementation, default settings include a matrix defining the mapping of functions into the Action categories and a matrix defining in which states functions of the Action categories for each relevant class of objects are permitted

The CHIPS system and its RBAC model are implemented using our COAST toolkit [13]. COAST supports a Cooperative Model-View-Controller (CMVC) framework. All application data objects modeled using the framework automatically get transaction-based concurrency control, coupling support, and prototype mechanisms. The cooperative MVC paradigm supports the manipulation of shared objects, e.g., in a whiteboard-style. All co-located or distributed users can see the changes made by others immediately.

In our implementation, an access control list (ACL) is implemented as an attribute of each class of hypermedia objects (such as node, link, and other multimedia objects), which are sub-classed from the COAST Model class. A method for access review is defined in the controllers of corresponding objects under the COAST Controller class. When an action is activated at the user interface, this method is called. Among the four action categories, Update implies Query. Execute implies Update and Query. Implied rights are automatically set in the Permission-Role assignment phase. This reduces some burden on the access review phase.

Next, an example is given to present the user interface of the tools for access authorization. The example is derived from the previously described scenario.

EXAMPLES OF USE

The CHIPS system consists of several tools, i.e., the hypermedia activity space browser and the hypermedia schema editor. The user interface of each editor/browser has seven major areas (see the lower window in Figure 1). At the top is a title bar, which presents the name of the tool, the name of the current node and the type of the current page. Under the title bar is a system logo and a list of users (represented by little pictures) who are currently working together on the same page. On the right-hand side is the current node label whose color indicates the state of the current task node. The largest area in the middle displays the content of the current page. To the left is a palette of tools for navigation, editing, and task-related triggers. On the right is a palette of hypermedia object types that are allowed in the current page. Each instance of a node is represented by a box, which carries its type as a little red label and shows its name in the box. Links are represented as arrows carrying their type name as a label, too. Color-coding is used to signal the state of a task node, e.g. green task nodes are active. The node type label is also used to display planned task duration and end date as well as the logic of preconditions for task node activation. Here, '&' means AND-joint. Similarly, the letter 't' (for true) and 'f' (for false) at the end of link label denote the current value of the trigger condition of the link.

Figure 1 shows in the lower window a schema for an equipment purchase process. This process has three steps: purchase request, purchase approval, and the actual purchase task. The internal structure of the document to be processed and transferred in the process is defined in the first step of the process. As shown in the upper window of Figure 1, the document has a form-like structure. Some of its column items are hypertext nodes that may have their own internal structures and may contain multimedia contents. The process schema is created by multiple people (here, users wwang and haake).

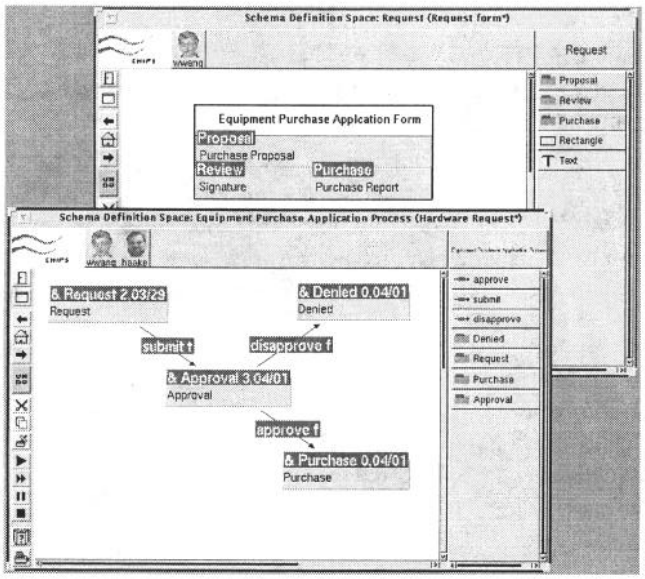

Figure 1. Process schema: composite type definition

At the initial stage, they change the process structure very frequently. The access control at this stage has to be very loose to be non-intrusive to people working on an emergent process structure. Gradually, the structure is changed from ordinary hypertext structure to a process structure with specific task nodes and process links (see also [19]). Finally, the created structure consists of one process task node functioning as the root of the structure, and four atomic task nodes representing each step. When a process schema emerges and is ready to be instantiated, more tight access control on the process schema becomes necessary. This is important, because any change to it would affect all of its instances. This kind of flexible handling of access permissions to an emergent process structure can avoid the problem of formalizing a premature process and the problem of trying to formalize an informal process structure. The simplest way to loosen or tighten up access control is to change the access permission of the root node of the process schema. The root node serves as a wrapper for all of its content, which may be a nested structure.

After a process structure is defined, other attributes (whose values will affect all its instances) can be assigned. Among these attributes are access control related and process execution related attributes, such as 'access rights' and 'actor'. These are to be set for each of the object prototypes (that appear in the type palette). If required roles and users for the process do not exist in the organizational context of the system, new roles and/or users have to be created. Figure 2 shows that a newly added user 'schummer' is being assigned to the 'Software engineer' role. As illustrated in Figure 2, different types of roles can be recognized by their naming conventions. The ones in all upper case are teams; the initial capital ones are organizational roles; and ones in all lowercase are personal roles. It is sometimes a problem to decide whether a team needs to bring in new members to fill a needed role or just to let a person take multiple roles (if he or she is capable and has the time to take over the corresponding tasks). The User-Role assignment tool can help here. By selecting a team role, all roles available in the team and all members of the team can be seen. By selecting a role, all the people taking the roles can be seen, no matter which teams these people belong to. By inspecting a user, all the possible roles he or she can take is given. Thus, a team can act appropriately.

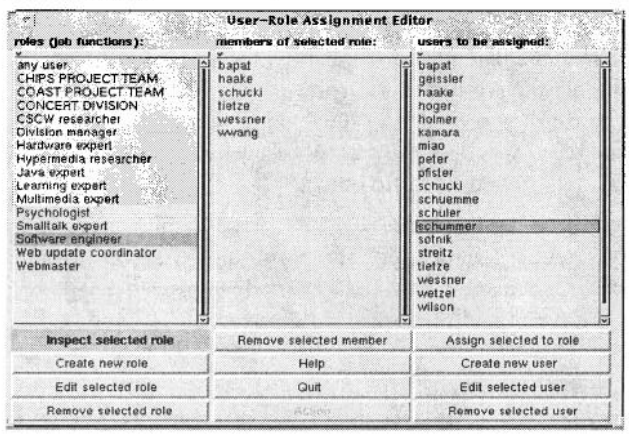

Figure 2. User-Role assignment: organizational context

Figure 3 shows user wwang, who is the owner (creator) of the first step (the Request task) of the process, is using the Permission-Role assignment tool to assign full permissions of the task to 'Hardware expert' role and Execute permission to 'Software engineer' role. The Query and Update permissions that are implied by the Execute permission are automatically set by the tool. He also sets Query permission to the whole 'CONCERT DIVISION'. The internal structure of the task is a form as illustrated in the upper window in Figure 1. The task node serves as a wrapper for the form contained in it. Therefore, only users who can open the node (with Query permission) can see the form.

More fine-grained permission can be assigned to the columns in the form. For instances, an Update permission to the "Review" column in Figure 1 is assigned to the 'Division manager' role. Therefore, although many members having the above mentioned three authorized roles may read this page or comment on the purchase proposal column (which has a multimedia page as its content), only a 'Division manager' can write in the "Review" column.

After the authorized roles are assigned for the first task, user wwang assigns some of the authorized roles to be 'actor' of this type of task. In this case, 'Hardware expert' and 'Software engineer' roles are assigned to 'actors' of the task. The actual performers of the task will be decided later when a process instance is created and one or more team(s) are assigned to the process instance.

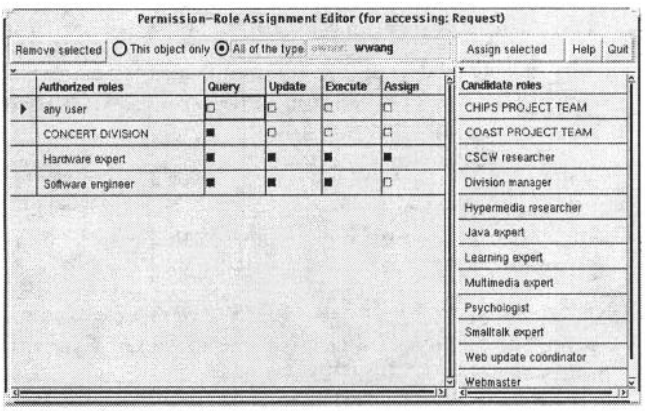

Figure 3. Permission-Role assignment for a node type

For the second step (the Approval task) of the process, the Update permission is assigned to the 'Division manager' role. For this task the 'Division manager' role is its only actor. Therefore, when the form is passed from the Request task (in the first step) into the Approval task node (the second step) the access permission to the form changes automatically (i.e., only 'Division manager' can access it).

The authorized role and actor of the third step "Purchase" or "Denied" is assigned to the 'Hardware expert' role. If the 'Division Manager' role approves the proposal, hardware experts would be notified to carry out the purchase and complete a brief report on the task.

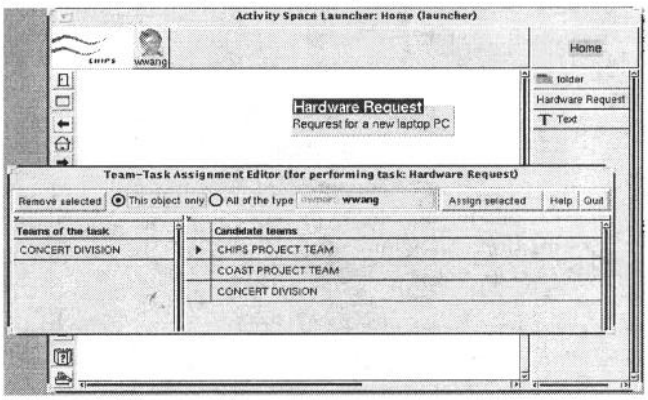

Figure 4. Team-Task assignment for a composite node

After relevant attributes that have global effects have been assigned; the process schema is ready for use. Users can create an instance from it with a copy-as-template function. The activity space launcher tool as illustrated by Figure 4 shows the root node of an instance of the process. User wwang is assigning a team to the root node of the process instance. This permits the access review procedure to identify the *authorized actors* and *authorized team members,* who will perform the tasks of the process.

After task-related attribute values that are local to this instance have been assigned (by any roles with Execution permissions), the process can be started by an actor. For process consistency and record-keeping purposes, when a task is completed, the task node is protected by our RBAC mechanism (i.e., by its task state-related contextual condition) to prevent any further changes.

CONCLUSION

In this paper, we presented an extended RBAC model for cooperative hypermedia environments. This access model extends the RBAC approach by

- adding the concepts of team role and personal role,
- introducing four categories of actions including process control and sharing aspects,
- introducing two context dimensions to the protection state representation for teams and states of tasks (or processes),
- applying access control dynamically not only to object types and instances but also to composites and nested wrappers, and
- providing a cooperative environment for specifying and applying access permissions.

Using our approach, the problems of applying RBAC to cooperative hypermedia can be addressed:

- By integrating a team-and-role-based organizational context in the cooperative hypermedia system, it is possible to apply the RBAC approach. This aids changing teams, shifting responsibilities, and emerging processes and hypermedia workspaces.
- Using the wrapper approach and by defining access permissions on the object type and instance levels permissions can be flexibly assigned. By using inheritance mechanisms, access permissions to dynamic composites and for navigation between composites can be computed.
- Categorizing functions into few categories require limited learning by end users. Furthermore, the cooperative definition of access permissions should facilitate understanding of access policies in the teams.
- Context-dependent access control (i.e. when the access permissions vary with the state of cooperation or the state of the hypermedia workspace) is supported via our definition of Protection State and the access review procedures.

An example demonstrated how the extended RBAC model is used in CHIPS. It enables teams to work cooperatively and in a coordinated fashion on a shared information space. Different styles of cooperation can be supported by using emergent process definitions. The extended RBAC model provides the necessary support for dynamically adapting access permissions.

Compared to related work our approach differs in several aspects. In [18], each team has a set of roles that are defined within the team. In our model, the concept 'team' is used to identify relevant team members among authorized actors and roles that are defined across teams. In the object dimension, many access control models have used object types and inheritance relationship between types and their instances. In those models, types refer to object classes [11, 15] while in our model types refer to semantic object types (or prototypes). This allows more fine-grained categorization. In general, we tried to incorporate RBAC into a cooperative hypermedia environment, that exhibits the simplicity virtue of a RBAC model, rather than making major changes that may make it too much complicated for users to understand.

Our approach can be applied to any cooperative hypermedia environment that is built on shared information spaces. Currently, our model has been implemented in the CHIPS cooperative hypermedia system using the COAST framework. Using CHIPS in our group has led to some early observations: (1) managing changing team membership and new types of information has become easier; (2) in the case of emerging process structures, the wrapper concept has eased the dynamic adaptation of access permissions to changing document spaces; (3) the set-based representation of user-role relationship is simple and flexible for dynamic organization structures. However, how to specify constraints between roles needs to be addressed.

Our next plans include testing the applicability of our model in a more realistic setting (i.e. outside of our group). We are also working on including the extended RBAC models and tools into the COAST kernel system. This would make the access control features available to all applications built on COAST. Finally, we want to investigate alternative policies and flexible ways to select and apply them in different situations.

ACKNOWLEDGMENTS

The author wants to especially thank Jörg Haake for his concrete suggestions and constructive comments on this paper. Thanks are also due to Christian Schuckmann and the anonymous referees for their very helpful comments.

REFERENCES

1. Akscyn, R., McCracken, D., Yoder, E. KMS: A distributed hypermedia system for managing knowledge in organizations, Communications of the ACM, 31, 7, 820-835, 1988.
2. Atluri, V., and Huang, W. An authorization model for workflows. In Proceedings of the IEEE Symposium on Research in Security and Privacy, 122-136, 1992.
3. Baker, D.B., Barnhart, R.M., and Buss, T.T. PCASSO: Applying and extending state-of-art security in healthcare domain, Proceedings of CSAC'95, 1997.
4. Bertino, E., Ferrari, E., and Atluri, V. A flexible model supporting the specification and enforcement of role-based authorization in workflow management systems. Proceedings of ACM RBAC'97, 1997.
5. Edwards, W.K. Policies and roles in collaborative applications, Proceedings of CSCW'96, 11-20, 1996.
6. Ellis, C.A., Gibbs, S.J., and Rein, G.L. Groupware: some issues and experiences, CACM, 34, 1, 38-58, 1991
7. Gronbaek. K, and Trigg. R. Design Issues for a Dexter-Based Hypermedia System, Proceedings of ACM Hypertext'92, pp. 191-200, 1992
8. Haake, J. M., and Wang, W. Flexible support for business Processes: Extending cooperative hypermedia with process support, Proceedings of Group'97, 341-350, Nov. 1997.
9. Ramaswamy, R., Sandhu, R. Role-Based Access Control Features in Commercial Database Management Systems. In Proceedings of NISSC'98, 1998
10. Sandhu, R. Roles Versus Groups in Workshop Summary, ACM RBAC Workshop'95, pp. I-25, 1995
11. Sandhu, R. and Samarati, P. Authentication, access control, and audit, ACM Comput. Surv. 28, 1, 241-243, March 1996.
12. Sandhu. R. Rationale for the RBAC96 family of access control models. In Proceedings of ACM RBAC'97, 1997
13. Schuckmann, C., Kirchner, L., Schümmer, J., and Haake, J.M. Designing object-oriented synchronous groupware with COAST, In Proceedings of the ACM CSCW '96, 30-38, Boston, 1996.
14. Shackelford, D.E., Smith, J.B., and Smith, F.D. The architecture and implementation of a distributed hypermedia storage system, Proceedings of ACM Hypertext'93, 1-13, 1993.
15. Shen, H. and Dewan, P. Access control for collaborative environments, in *Proceedings of ACM CSCW'92*, 1992.
16. Sheth, A. From contemporary workflow process automation to adaptive and dynamic work activity coordination and collaboration, in SIGGRUOP Bulletin, 18, 3, 17-20, 1997.
17. Sikkel, K.: A Group-based authorization model for Cooperative Systems, Proceedings of ECSCW '97, 345-360, Sept., 1997.
18. Thomas, R.K. Team-based access control (TMAC): A primitive for applying role-based access controls in collaborative environments, ACM *RBAC'97*, 1997.
19. Wang, W. and Haake M.J. Flexible Coordination with Cooperative Hypermedia, *Proceedings of ACM Hypertext'98*, pp. 245-255, June, 1998