Specification of Secure Distributed Collaboration Systems

Anand R. Tripathi and Tanvir Ahmed
{tripathi, tahmed}@cs.umn.edu
Department of Computer Science
University of Minnesota, Minneapolis MN 55455

ABSTRACT
The focus of this paper is on specifying security policies for distributed collaboration and workflow systems. We identify unique requirements for secure collaboration, specifically role admission and activation constraints, separation of duties, and a model for multi-user participation in a role. We present a role-based model for specifying dynamic security requirements in collaboration systems which is unified with coordination specification. It also supports hierarchical structuring of a large collaboration environment using nested activities, which can be created dynamically. An activity defines a naming scope and a protection domain to specify security and coordination policies. For decentralized management of activities, access control and event subscription-notification policy modules are derived from the high level collaboration specification and are distributed.

Keywords: Distributed collaboration, Role based access control, Security policy specification.

1. INTRODUCTION
The objective of our research is to realize a distributed collaboration environment from its high level specification. In this paper, our primary focus is on the specification model for secure distributed collaboration systems. The specification model is developed to address security and coordination requirements of these systems. The approach to realize the runtime environment for a collaboration system by means of a generic policy-driven middleware is presented in [20]. There are three steps involved in this approach as shown in Figure 1. Initially, the coordination and security policy for a collaboration is specified based on a schema. From the specification, various policy modules are derived for different kinds of requirements, such as role based security, object level access control, and secure event notification for coordination. Finally, through these modules, the collaboration environment is realized.

The general requirements of security in CSCW systems are related to confidentiality and integrity of the shared data, privacy of a user's actions, auditability and non-repudiation of users' actions, and the integrity of operations to ensure that only authorized participants perform certain sensitive tasks. Additionally, CSCW systems pose several unique security requirements not present in traditional operating systems and databases [6]. An important characteristic of collaboration environments is the need to have dynamic security policies that require context sensitive access control mechanisms that depend on participants' past actions and execution stages. Privileges may also need to be changed when certain events occur in the system. In this paper, we show that the specification of such security requirements necessitates a unified model for expressing coordination and security policies.

The existing research on security in collaboration targets either synchronous groupware applications or asynchronous workflow. However, the security requirements and solutions proposed for these two areas are different. Shen and Dewan [4] presented an access control model with a set of rights that are unique to shared-view based GUI interface control in groupware environments. In workflow, database oriented security is prevalent [8, 1]. However, in practice, synchronous collaboration comes into existence in many stages of asynchronous workflow; moreover, different organizational workflows have to interact [11]. In [1], how tasks can be dynamically assigned to roles following authorization constraints is addressed. Integration of object types and time interval for tasks based active security policy specification is addressed in [8]. Though these works motivated us, they do not address security policy requirements on task creation, meta-policies like users assignment to roles, and most importantly lack a specification model for ease of security policy specification for distributed collaboration systems. We propose a security model from organization workflow con-
We identify here several important security requirements that a role-based model for collaboration systems should support. The traditional concerns of confidentiality and integrity of shared data are naturally present in collaboration systems. However, access rights, privileges, and ownership of objects may change in collaborative environments as activities progress.

In a role-based security model, a role represents a set of privileges [14]. One can view a role as the characterization of a protection domain. A user assigned to a role acquires those privileges. The use of role-based security policies in collaboration and workflow systems has been found to be quite natural as participants perform a set of well-defined tasks pertaining to their expertise and responsibilities in the organization [3].

In an organization, many activities exist and new activities are constantly created. Existing roles and new participants in these activities. A large collaborative environment may sometimes need to be structured hierarchically. For example, a system supporting a team-based design project running over several weeks or months may need to have several dynamically created synchronous shared "whiteboard-like" activities of relatively shorter life-spans.

2. REQUIREMENTS FOR SECURE COLLABORATION

2.1 Role Admission and Activation Constraints

A role-based model needs to address role admission and role activation constraints. Role admission constraints specify the conditions that need to be satisfied before a user can join a role. The admission constraints can be based on several different kinds of criteria. It may specify a list of users that should be allowed to join a role, or it may also include a list of those who should never be admitted. A role admission constraint can be based on previous qualifications, which could be that the requesting user is currently admitted in some given role. Additional role admission constraints can be based on cardinality, i.e., a count specifying the maximum number of participants allowed to be admitted in the role. Another kind of constraint could specify the events that must happen before a user could be admitted in a role.

The admission into a role itself does not guarantee that the member can use that role. Role activation constraints specify the conditions which need to be satisfied for a member to perform the role-related tasks. For example, a cardinality constraint may specify the maximum and minimum number of participants that must be present for a role to be operational. When a user is admitted to several roles, for simultaneously activating multiple roles, there are various models for the set of privileges to be given to that user [13]. The obvious option is the user acquires union of all the privileges. However, in some cases the roles may conflict and user's privileges may be reduced.

2.2 Coordination Model in a Role

Coordination between participants in different roles within an activity is referred to as inter-role coordination, which is widely used in role-based coordination systems [9, 2]. However, when multiple users are allowed to simultaneously join and activate a role, users within the role may require to coordinate among themselves to achieve collaboration objectives, which can be termed as intra-role coordination. When mul-
iple users are present in a role, they can participate either independently or cooperatively. In independent participation, all of the role specific task responsibilities are assumed individually by a participant, irrespective of the presence of the other participants. A participant in a cooperative or shared role may need to create and use some private objects, not visible to the others in that role until explicitly granted access permissions. On the other hand, when the participants in a role are assuming task responsibilities cooperatively, they coordinate among themselves on deciding who is performing which role specific task. Another type of cooperation may require a task to be performed by all the participants of a role, like jointly opening a bank vault. Moreover, in some collaboration environments there may be no coordination among participant activities, e.g. in an unrestricted whiteboard sharing.

2.3 Dynamic Access Control Policies
The privileges assigned to a user in a role may change with time due to the actions executed by the other participants. Sometimes permissions may change due to the user’s own actions, such as making a final agreement on a document. After an object is finalized, its owner may be different from the one who created the object. The need of dynamic access control mechanisms also arises during the lifetime of an object that may move from one activity to another. In some situations, a role should be allowed to execute an operation only after another role has performed some other action. Though this policy may be used as a coordination constraint, it has to be enforced securely using an access control mechanism.

In the context of role based access control, dynamic access control policies have to address issues like role invalidation, role activation, several types of “separation of duty” constraints, or other history-based conditions which may occur during the lifetime of an activity. Traditional RBAC and most of the existing MAC (Mandatory Access Control), DAC (Discretionary Access Control) style security policies are passive, i.e. they do not differentiate permission assignment and permission activations [17]. Recent research in task-based workflow systems [8, 17] addresses dynamic access control policies in the context of tasks.

Several researchers have discussed “separation of duty” requirements in role-based access control models [16, 18, 15], namely static separation of duty, dynamic separation of duty, user-user conflict, user-role conflict, object based separation of duty, and operational separation of duty. A specification model for collaboration has to be able to express these constraints.

3. OVERVIEW OF THE COLLABORATION MODEL
Here we discuss the basic elements of our collaboration model: shared object, role, activity, and event.

3.1 Shared Object Model
In our model, shared objects are managed by a set of trusted servers, which can be specified in the collaboration specification. Based on the security and coordination requirements specified in a collaboration, our system derives appropriate policy modules, which are used by the trusted nodes to control access to their objects. Shared objects are represented in our specification model only in terms of their types and method signatures, keeping the semantics and implementation details transparent.

We can specify access control at the granularity of the methods invoked on these objects. Our system supports both role based access control as well as traditional DAC. Access control lists are maintained based on user-ids as well as role names. Access control policies can be specified for each method of an object or the object itself. Negative access rights (i.e., constraints) enable easier coarse-grain access control specification.

3.2 Definition of Roles
In defining a role, we consider the constraints addressed in Section 2. These constraints are of three types: the role admission condition puts constraints on admitting a participant to a role, the role activation condition needs to be satisfied when any role specific task is executed, and the precondition of an operation is required to be satisfied for a member in the role to execute it. Coordination policies are specified as preconditions for operations using an event based model.

The owner of a role can admit users to it, subject to the admission constraints; it can also remove an existing participant. The constraint specifications can include event names and event counters.

An operation in our model represents a task of a role. An operation may consist of an execution of a method of a shared object, a synchronization action, or a role request. A role request is an operation either to leave the current role or to join another role.

3.3 Activity Definition
An activity defines a scope for shared objects, roles, and operations as shown in Figure 2. A nested activity is defined when a collection of tasks within an activity can be performed concurrently using different objects of the same type with different or same participants. In Figure 2, in Course activity Examination activity is defined as a nested activity as the tasks associated with each Examination activity instance uses different ExamPaper objects.

The specification for a role in an activity can refer to the objects and other roles in that activity. A nested activity may need to have access to the objects in the scope of its parent activity, or a role in the parent activity may need to be bound to a role in a nested activity. For this purpose, an activity definition needs mechanisms for role binding and passing object references as parameters to activity instance. A role in the parent activity can be bound to a role in the child activity in the static definition. We refer to it as role reflection, which means that all the members of the parent role implicitly become members of the role in the child activity. Removal of a participant from the reflected role, also implies removal from the role in the child activity. A role reflection can also be specified at the time of activity instantiation. A participant in the reflected roles (i.e. a role in the parent activity) gains expanded privileges comprising of
the operations of the child activity role. Moreover, the child activity role can have operations that access the objects in the scope of the reflected role. However, a participant of the reflected role has to comply with the child role's admission constraints. In our model, roles are not defined based on permission inheritance [14, 10], rather defined in the scope of hierarchically nested activities.

In Figure 2, a Course activity is defined consisting of an Instructor role, an Assistant role with possibly multiple teaching assistants as its members, and a Student role having all registered students as its participants. In the nested Examination activity, the participant in the Examiner role creates an ExamPaper object. In this example, both the Instructor and the Assistant roles in the Course activities are reflected in the Grader role of the Examination activity. When an instance of this activity is created, the members of these two roles who comply with the Grader role's admission constraints are admitted to the Grader role. Therefore, an admitted participant in the Grader role can perform operations on the GradeSheet object in the Course activity as well as the objects in the Examination activity.

Each student in an Examination activity can instantiate an ExamSession activity. Only the student creating this activity is assigned to the Candidate role, and one of the participants in the Grader role is assigned to the Checker role. References to two objects are passed as parameters to an exam session activity. A single ExamPaper object is shared by all the sessions. On the other hand, a new AnswerBook object is created for each student's exam session.

An entity (such as activity, object, object method, role, role operation, or event) encapsulated in the scope of an activity can be referenced by a fully qualified name. In the specification model, within an activity, one can refer to its current instance using the pseudo variable thisActivity, and its creator using parentActivity. The pseudo variable this refers to its immediate nesting entity, which can be an activity, role, object, or operation.

![Diagram of Hierarchical Structuring of Collaborative Activities](image)

**Figure 2: Hierarchical Structuring of Collaborative Activities**

### 3.4 Event Model

Events and event counters are used in our model for specifying coordination and dynamic security policies. Event types are related to different kinds of entities such as activities, roles, operations, and objects.

The multiple occurrences of a given event type — such as multiple executions of an operation — are represented by a list. The expression `eventName` returns the list including all the instances of this type of event. This list can be indexed using notation `eventName[i]` to refer to the i-th event in the list. We provide a count operator `#` on lists. Hence, `#(eventName)` returns the number of times the event has occurred. The event name itself refers to its most recent occurrence.

We use event counters for synchronization specification based on the model presented in [12]. Related to each role operation are two types of events: `start` and `finish`. For example, `role-name.op.start` or `role-name.op.finish`. These event types are also defined for each object method. For a role, we have event types defined for `join`, `leave`, `admit`, and `remove` operations. For each activity definition, we have `start` and `finish` events. In our model, one can also specify a derived event type by filtering an event list based on its attributes. For example, for a role operation execution, we can define a filter based on an event id, such as `opName.start(invoker=John).

### 4. ROLE SPECIFICATION

A role is defined in the scope of an activity and can manipulate the objects in that activity. A role specification includes role name, role owners, reflected roles if any, role admission constraints, role activation constraints, and role operations with their preconditions. The basic terms of a role specification is shown below, where `[ ]` encloses optional terms. If not specified, the creator of a role is the owner.

```
ROLE name ([OWNER name], [REFLECT role]) {
[ADMISSION_CONSTRAINTS condition]
[ACTIVATION_CONSTRAINTS condition]
[OPERATION name
[PRECONDITION condition]
[ACTION method_invocation]]
```

In the specification model several functions are defined for a role. A boolean function `member(role, user)` checks if a participant is present in a role. The function `members(role)` gives the list of participants in a role. Hence, a count of the participants admitted in a role is given by `#(members(role))`.

#### 4.1 Role Admission Constraints

The Assistant role in Figure 2 has several admission constraints. The maximum member count for this role is two; only a staff can be an assistant; a student cannot be an assistant; only after a user is assigned to the Instructor role, users can be admitted to the Assistant role; there is a list of users who can be and cannot be assigned to the Assistant role; and lastly, both users B and C cannot be assigned to this role. The specification of admission constraints of the Assistant role is shown below.

```
#members(thisRole) ≤ 2
& member(thisUser, parentActivity.Staff)
```
4.2 Role Activation Constraints
A dynamic separation of duty constraint, like a user should not activate two roles at the same time can be easily specified as part of role activation constraints. In the following example, we present several activation constraints for an admission committee member role of a computer engineering department. Such a role may specify a minimum and a maximum number of participants to be present for the committee to be active. Moreover, a constraint can be that at least a member from both the computer science and the electrical engineering departments must be present during role activities.

\[
#\text{members(thisRole)} \geq 2 \land #\text{members(thisRole)} < 5 \\
#(\text{members(thisRole)} \cap \text{members(EE.Professor)}) \geq 0 \\
#\text{members(thisRole)} \cap \text{members(CS.Professor)}) > 0
\]

4.3 Operation Specification
An operation specification includes a name, and may include a precondition and an action which invokes predefined methods of some objects in the shared workspace. A keyword new is reserved for specifying an object or object creation. Other methods are specified as part of object specification and are referenced here. Follows an example of an operation specification of the Examiner role in the Examination activity as shown in Figure 2. The operation SetPaper can be performed only once as specified by the precondition. This operation results in creation of an object exam of type ExamPaper and an invocation of the setPaper method of this object.

\[
\text{OPERATION SetPaper} \\
\text{PRECONDITION} #\text{(SetPaper.start)} = 0 \\
\text{ACTION} \{ \text{exam=}\text{new OBJECT(ExamPaper)}; \text{exam.setPaper(data)} \}
\]

The precondition must be true when the operation is invoked. Thus precondition enables us to specify fine grain “separation of duty” policies, like the object based separation of duty and the operational separation of duty. For example, in an office system, a manager may prepare an invoice and approve an invoice, but should not be able to approve his/her own invoice. This specification is shown below which can be also considered as an example of an operational separation of duty policy.

\[
\text{OPERATION ApproveInvoice} \\
\text{PRECONDITION} \{ \text{PrepareInvoice.finish(invoker=thisUser)}=0 \}
\]

5. ACTIVITY SPECIFICATION
An activity specification contains the specification of shared objects, roles, static assignment of users to these roles, references to objects of other activities, owner of the activity, and conditions which will terminate the activity. In Figure 3, we describe a specification of the Examination activity as shown in Figure 2.

In the Examination activity template definition, the owner of an Examination activity is the the Instrucor role of the parent activity. Participants for the Examiner role have to be assigned during activity creation as specified in the activity declaration. The specification of ExamPaper and AnswerBook object types are shown in Figure 3.

A student can initiate an ExamSession activity by invoking the StartExam operation with a newly created AnswerBook object. However, he/she cannot initiate two sessions of this
examination as enforced by the precondition. Also, the exam object is passed to the session, and the user initiating this activity is assigned to the Candidate role. When any activity is instantiated, the creator of the activity is the role, which instantiates it, here the Student; whereas the owner is the role which manages the activity, here the Grader as specified. Though the activity is created by a student, the student cannot be trusted to maintain the exam paper and answer book objects or to enforce the constraints.

The ExamSession activity terminates when the checker grades the answer book. The admission constraints in the Candidate role ensure that the creator of the ExamSession is the only member of the Candidate role. The activation constraints ensure that the examination can be taken only during the specified three hour periods, and during the session, the participant is a valid student. The date function returns the current date.

A candidate can read the exam paper and write the answer book. The Submit operation does not invoke any method, rather creates an event subscribed by the Checker role for coordination. The checker can only grade after the exam book is submitted.

6. POLICY MODULES DERIVATION

In our model, every object - activity template, activity instance, object type, object instance, and role - has an access control policy, which is derived from the collaboration specification. Moreover, the enforcement of proper access control depends on proper distribution of events among all the entities as the access control policies for the entities are inter-linked with event count based conditions. To ensure the integrity of the events, only the trusted nodes can be event subscribers or event notifiers.

In our distributed execution model [20], nested activities can be instantiated in various roles' protection domain. Moreover, all the nodes of the distributed users cannot be trusted, like a client node cannot maintain a shared object unless the client node is trusted. We derive a distributed trust relationship among the activity owners specified as part of an activity template. If not specified, the default owner of any activity is the owner of its parent activity. An activity owner's trusted site maintains all the roles, activity templates, and objects created within its scope including the activity instance, we call these primary nodes; whereas other users interacting with these role and objects maintain RMI stub references to these objects. A primary node is responsible for managing all security sensitive functions and event subscriptions for the entity it represents.

In our model, policy modules are specified as policy templates in the context of activity instances, i.e., a policy is specified based on activity types, roles, and object types. The access control policy (ASP) module, derived from the Course activity specification, for an Examiner object instance is shown below. The participants of this role can be assigned by the Instructor role as specified for instantiating an Examination activity, not shown in Figure 3.

Each entry in the ASP module is shown as an ACL entry extended with a condition tag. The condition tag specifies the event condition and the node where this event is subscribed from. In the distributed execution model, each trusted server maintains a subscriber list for the objects it maintains. The $ indicates a variable for binding with an object of the corresponding type. When an Examination activity midterm is instantiated in the activity chemistry, an instance of Course, the following entry for the Examiner role is added in the access control module in the role's primary node.

7. RELATED WORK

In contrast to SecureFlow's [8] Authorization Template(AT), an activity template in our model has multiple roles and object types capturing the interaction among these roles and objects through operations. If one thinks of an AT as analogous to a method specification, then an activity template is like a module specification with multiple methods. An activity template specification may contain multiple tasks or operations and is able to capture workflow stages.

In [17], team based access control (TMAC) is presented as an approach of applying RBAC in workflow like collaborative environments. In conjunction with RBAC, TMAC proposes an active security model similar to trigger oriented active databases. A team has participants from different roles. Our concept of activity, where roles are created or reflected inside an instance of an activity, subsumes the team concept.

A framework for role based access control (RBAC) models with constraints and role hierarchies is presented in [14]. In [16], role based management (RBM) is presented, which views role definitions similar to classes, which can be instantiated or reused through inheritance. Similar to observation made in [5], we haven't face a strong need for role inheritance. We use and build upon many of the fundamental concepts presented in these papers. In our model, a role represents a protection domain, and a set of tasks (responsibilities) is associated with a role. In our system, roles are defined as instantiable classes in the context of an activity definition. Instances of the same role in different instances of an activity represent different protection domains. Our
role admission criteria are conceptually similar to those presented in the context of RDL (Role Definition Language) in [7]. The focus of that work is on distributed and decentralized management of roles in a distributed service model. We are concerned with an integrated centralized specification of a collaboration environment and deriving from such specification distributed realizations of the collaboration. We introduce here, the concept role reflection which provides similar functionality of role reusability as addressed in [10]; however, not through inheritance rather using automated user assignment.

In [19], we investigated the use of mobile agents in a middleware for supporting distributed collaborations. The focus of this paper is on the development of a role based model for specifying collaboration environment.

8. CONCLUSION
The objective of our research is to realize secure distributed collaboration systems from high level specifications. We presented in this paper a role-based specification model for collaboration systems based on the requirements including dynamic security policies, role admission and activation constraints, and separation of duties constraints. The specification model unifies coordination specification with active access control policy specification providing a task based view of role specifications. Roles are defined and instan- tiated in the context of activities. We presented the concept of usable activity template, which enables dynamic creation of activities. The concept of role reflection is present to automate user assignment to sub-activities. Access control policy modules for collaboration objects and events are derived from the specification and are distributed for decentralized management of the collaboration. A trust relationship is derived among the collaborating nodes based on the ownership specification of the activities.

9. REFERENCES


Design of a Policy-Driven Middleware for Secure
Distributed Collaboration. Technical report,
Department of Computer Science, University of
Minnesota, November 2001. Available at