Specification of Secure Distributed Collaboration Systems

Anand R. Tripathi, Tanvir Ahmed, and Richa Kumar

{tripathi, tahmed, richa}@cs.umn.edu

Department of Computer Science

University of Minnesota, Minneapolis MN 55455

Abstract

The focus of this paper is on a specification model for defining security and coordination policies for distributed collaboration and workflow systems. This work is motivated by the objective to build distributed collaboration systems from their high level specifications. We identify here unique requirements for secure collaboration, specifically role admission and activation constraints, separation of duties, dynamic access control, and a model for multi-user participation in a role. We present a role-based model for specifying coordination and dynamic security requirements in collaboration systems. It also supports hierarchical structuring of a large collaboration environment using the concept of activities, which define a naming scope and a protection domain to specify security and coordination policies. We have implemented this specification model in XML and used it to construct the runtime environments for distributed collaboration systems using a policy based middleware.

Keywords: Distributed collaboration, Role based access control, Security policy specification, Decentralized management of collaboration systems.

1 Introduction

The objective of our research is to realize distributed CSCW (Computer Supported Cooperative Work) systems from their high level specifications. 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 from this specification model by means of a generic policy-driven middleware is presented in [23]. A policy-based approach decouples the coordination and security aspects of a collaboration system from the implementation of the collaboration objects. This makes it easy for different policies to be plugged in, allowing flexibility in designing as well as modifying a collaboration environment. Several factors motivate this approach. Collaboration environments often span multiple administrative domains. The requirements of a collaboration system may evolve with changes in administrative policies and user experience. With advances in technology, new devices, tools or artifacts may need to be integrated into a collaboration environment. This can warrant a reconfiguration of the system with changes in the governing policies.

Coordination requirements in collaboration systems include synchronization and precedence constraints. The general requirements of security in such 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 [8]. An important characteristic of collaboration environments is the need to have dynamic security policies such as dynamic "separation of duties" constraints as well as context sensitive access control mechanisms that depend on participants' past actions and the execution state of the collaboration. 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, when, in practice, synchronous collaboration comes into existence in many stages of asynchronous workflow. 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 [1, 10].

In this paper, we present a specification model that can capture the various coordination and security requirements outlined above. Past research in this area, such as Task Based Access Control (TBAC) [19, 1], shows that the specification of many dynamic security requirements necessitates a unified model for expressing coordination and security policies. Our objective is similar to COCA [12] and DCWPL [2] in their approach of constructing a distributed collaboration environment from a high level specification. However, these specifications are limited to coordination policies. They do not provide adequate support for security policies like user-assignment to roles, dynamic access control or

history based security constraints. Moreover, they do not provide any mechanisms for hierarchical organization of collaboration systems and context sensitive privileges.

The role based specification model for collaboration systems presented here supports the expression of requirements such as "separation of duties", intra-role and inter-role coordination, admission control policies, role activation constraints, and dynamic access control policies. An important aspect of this work is reflected by the fact that the specification model has been integrated with a policy-driven middleware to realize the runtime environments for collaboration systems [23]. In the specification model, we introduce the notion of reusable *activity templates*, which provide a facility to dynamically create activity instances, which can be hierarchically nested. Based on this specification model, we have devised an XML schema to express collaboration specifications using XML. In this paper, for the sake of simplicity and brevity, rather than using XML, we use a notation that is simple and conceptually easy to follow.

In the next section, we present our role based collaboration model. Section 3 discusses the requirements for secure collaboration systems. Our specification model is described in Section 4. Section 5 shows how the specification model is used by a policy based middleware to build runtime collaboration environments. Sections 6 and 7 present the related work and the conclusions.

2 A Role Based Collaboration Model

In a role based security model, a role represents a set of privileges [15]. 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]. However, in traditional RBAC (Role Based Access Control) model [15], all the privileges in a role are assigned statically thus lacking support for dynamic access control policies. As roles are globally defined, context based access control is not supported, unless new roles are created based on the context. Additionally, the issue of users' assignment to roles is not addressed.

In an organization, many activities exist and new activities are constantly created. Existing roles and new roles participate 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.

In workflow systems, such as [1, 10], dynamic task assignments to roles following authorization constraints are addressed. Integration of roles, object types, and time intervals in a task-based active security policy specification is addressed in [10]. Though these works motivated us, they do not address policy requirements on task creation, meta-policies such as users assignment to roles, and administrative meta-policies such as who can enforce collaboration policies. Also, they lack a framework for realizing a system from its specifications.

In our collaboration model, an activity is an abstraction of a shared task. An activity defines how a group of users cooperate towards some common objectives by performing their individual tasks on a set of shared objects. It represents a protection domain and a scope for the roles, objects, and privileges in a collaboration. In an activity, users are represented by their roles, and roles within an activity are assigned privileges to perform certain tasks. We term these role specific tasks as *operations*. An operation can be method invocations on shared objects, synchronization actions, or activity management actions. Besides the application-defined roles in an activity, there are several administrative roles or meta-roles, such as *Creator* and *Owner* of an activity. An activity can be structured hierarchically, consisting of multiple nested concurrent activities.

An *activity template* specifies a generic collaboration pattern among a set of roles using some shared objects. To support hierarchical structuring of a large collaboration system into smaller activities, an activity template specification can define nested activity templates. Activities are instantiated from templates. Any number of instances of a template can be dynamically and concurrently created.

In a collaboration environment, users may be distributed over a network, and they may be associated with different administrative domains. In a typical situation, a user participates in a collaboration by joining a role in an activity. A role operation can result in a sequence of interactions between the invoker and a set of shared objects. This represents a *session* in the context of that role operation.

3 Requirements for Secure Collaboration

In this section, we identify several important security requirements that a role-based model for collaboration systems should support.

3.1 Role Admission Constraints

A role based model needs to address role admission constraints. Role admission constraints specify the conditions that need to be satisfied when a user requests to 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 other given roles. 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.

3.2 Coordination Requirements

Coordination between participants in different roles within an activity is referred to as *inter-role* coordination. The primary motivation of this requirement is to enforce precedence constraints among different roles' operations. For example, an *inter-role* coordination requirement in a conference workflow can be that the *Reviewer* role can review a paper after the *Author* role of the paper has made the final submission.

However, when multiple users are allowed to be present simultaneously in a role, users within the role may require to coordinate among themselves, which is be termed as *intra-role* coordination. When multiple 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. For example, every participant of a conference *Reviewer* role has to independently write a review.

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. For example, in a hospital patient ward, several nurses may be present in the role of *nurse-on-duty*. However, some medical procedure on a patient may need

to be performed only once by any of the members. 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 actions, e.g. in an unrestricted whiteboard sharing.

3.3 Separation of Duties

Several researchers have discussed "separation of duties" requirements in role-based access control models [17, 16, 20], 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. The notion of *static separation of duty* requires that two given roles should never be assigned to the same person. The concept of *dynamic separation of duty* requires that two given roles cannot be concurrently assigned to or activated by the same person. The notion of *user-user conflict* requires that two particular users should not be assigned to the same role. A *user-role conflict* specifies that a specified user should never be assigned to a given role. Another kind of requirement is *object based separation of duty.* A user cannot perform multiple operations of the same object by participating in two different roles. The *operational separation of duty* requires that no single participant of a role can perform all the operations related to a business transaction.

3.4 Dynamic Access Control Policies

The privileges assigned to a user in a role may change with time due to the actions executed by other participants. Sometimes permissions may change due to the user's own actions, such as making a final agreement on a document. In some situations, a role should be allowed to execute an operation only after another role has performed some other action. For example, in a course examination activity, a security requirement can be that students can only view the question after the examiner has released it and only during the specified time period of the exam-session activity.

Several types of "separation of duty" constraints and history-based access control conditions also fall into the category of dynamic access control policies. In the context of role based access control, dynamic access control policies have to address issues, such as constraints requiring a maximum and minimum number of participants that must be present for a role to perform any operation.

Traditional RBAC, and most of the existing MAC (Mandatory Access Control) and DAC (Discretionary Access

Control) style security policies are static, i.e., they do not depend on time or other events. Though permissions are assigned to roles, they may be specific based on various contexts and can be activated in those contexts. For example, in the course examination activity, a candidate can access the answer book only during an exam session. Additional context based access control may be related to physical environment's events. For example, in the course examination activity, context based access control may specify that the candidates can access answer books only when they are physically present in the class room and that too only during a predefined period.

3.5 Privacy

Privacy becomes an issue when one may need to hide the identity of one participant from another. In such cases, the presence of a participant may be only visible through his/her role or a pseudonym in a role but not by name. It may be required to hide identities of the participants of a role from other roles. Consider a course examination activity, which has two roles: the *Candidate* takes the exam and the *Grader* grades the answer book. A requirement can be that the graders do not know the identities of the *Candidate* role's participants. A similar collaboration requirement can specify that only the owner of a role knows the identities of the role participants. For example, in a conference submission workflow, the *Program Chair* role owns the *Reviewer* roles, who review submitted papers. However, none other than the users in the *Program Chair* role are permitted to view the identities of the participants of the *Reviewer* roles.

3.6 Meta-level Security Policies

An activity requires many administrative security policies: who can define new activities or instantiate an activity; during the lifetime of an activity, who can change various policies and enforce additional constraints on shared objects. These meta level administrative policies need to be specified based on meta roles, such as *Creator* and *Owner* for various entities like activities, roles, and objects. Policies need to be specified on who can join or leave these meta roles. Users in these roles are trusted with management responsibilities of the assigned entities.

4 Overview of the Collaboration Specification Model

In this section, we present the seminal elements of the collaboration specification model that we developed in our project for supporting policy-driven construction of collaboration activities [23, 21]. In this overview, we focus on the following features of this model:

- Definition of an activity in terms of roles and shared objects.
- Nesting of activities, passing of objects to a nested activity, and assignment of users to roles in a nested activity.
- Event model for coordination of role operations.
- Definition of shared objects.
- Definition of a role and its operations.

4.1 Activity Definition

An activity defines a scope for shared objects, roles, and nested activities. We illustrate the basic elements of activity definition using an example of a course management related activities. We first present the structure of an activity and its related concepts using a a schematic representation shown in Figure 1. Here a *Course* activity is defined with a nested activity named *Examination*. In an environment, there can be many instances of the *Course* activity, e.g. a *chemistry* course. The *Course* activity consists 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. Participants in the *Student* role of the course are not allowed to join the *Assistant* role. 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 made members of the *Grader* role of the *Examination* activity.

Each student in an *Examination* activity can instantiate a nested *ExamSession* activity. The *ExamSession* activity contains the roles *Candidate* and *Checker*. 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. When an *ExamSession* activity instance is created, references to the *ExamPaper* object and a new *AnswerBook* object are passed to it.

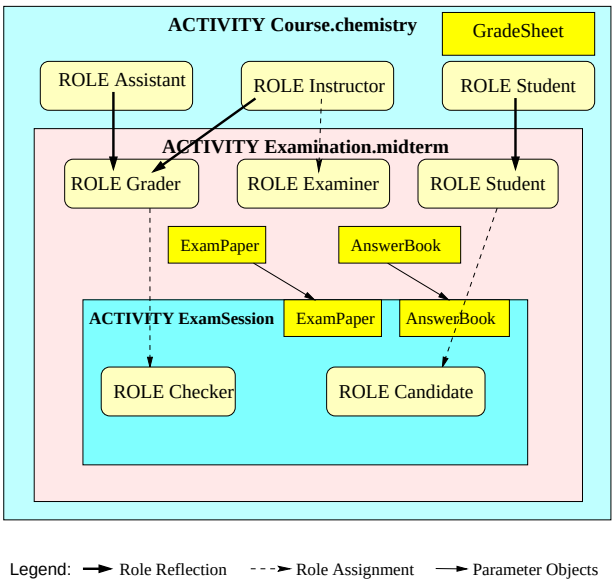

Figure 1: Hierarchical Structuring of Collaborative Activities

In the specification model, an entity (such as activity, object, or role) encapsulated in the scope of an activity can be referenced by a fully qualified name. Within an activity, one can refer to its current instance using the pseudo variable *thisActivity*, and its parent activity using *parentActivity*. The user executing an operation is identified by *thisUser*. Within a role's operation, one can refer to the role by *thisRole*. The pseudo variable *this* refers to its immediate nesting entity, which can be an activity, role, object, or operation.

The basic schema for an activity specification is shown below, where [ ] encloses optional terms.

```
ACTIVITY_TEMPLATE name ([OWNER name], [ASSIGNED_ROLES roles]) {
    [TERMINATION_CONDITION condition]
    [OBJECT_TYPE type (CODEBASE=codebase) {
        [METHOD name [RETURN type] [PARAM type]]
    }]
    [ROLE {..}]
    [ ACTIVITY_TEMPLATE {..}]
}
```

Figure 2, shows the complete specification of the nested *Examination* activity of our course example. The various elements of this activity specification are discussed in the following parts of this section.

```
ACTIVITY_TEMPLATE Examination (OWNER Instructor,
                      ASSIGNED_ROLES Examiner) {
   OBJECT_TYPE ExamPaper (CODEBASE=http://codeserver) {
      METHOD setPaper {PARAM PaperType}
      METHOD readPaper {RETURN PaperType}
   }
   OBJECT_TYPE AnswerBook (CODEBASE=http://codeserver) {
      METHOD writeAnswer {PARAM AnswerType}
      METHOD setGrade {PARAM GradeType}
   }
   ROLE Examiner {
      OPERATION SetPaper {
         PRECONDITION #(SetPaper.start)=0
         ACTION { exam=new OBJECT(ExamPaper);
                  exam.setPaper(data)}}
   }
   ROLE Examinee (REFLECT parentActivity.Student) {
      OPERATION StartExam {
         PRECONDITION #(Examiner.SetPaper.finish)=1
                  & #(StartExam.start(invoker=thisUser))=0
         ACTION {obj=new OBJECT(AnswerBook);
                 act=new ACTIVITY ExamSession(
                        (exam,obj), Canadidate=thisUser)}}
   }
   ROLE Grader (REFLECT parentActivity.Assistant,
                        parentActivity.Instructor) {
   }
   ACTIVITY_TEMPLATE  ExamSession( OWNER Grader,
                           OBJECTS (Course.ExamPaper exam,
                       Course.Examination.AnswerBook ans),
                           ASSIGNED_ROLES Candidate ) {
     TERMINATION_CONDITION: #(Checker.Grade.finish)>0
     ROLE Candidate {
        ADMISSION_CONSTRAINTS
            member(thisUser, parentActivity.Examinee)
            & #members(thisActivity.Creator) =1
            & member(thisActivity.Creator, thisUser)
            & #members(thisRole)<1
        ACTIVATION_CONSTRAINTS
            date > DATE(Jan, 12, 2001, 12:00)
            & date < DATE(Jan, 12, 2001, 15:00)
        OPERATION Read {
            ACTION exam.readPaper()}
        OPERATION  Write {
            ACTION ans.writeAnswer(data) }
        OPERATION Submit {
            PRECONDITION #(Write.finish)>0 }
     }
     ROLE Checker {
        ADMISSION_CONSTRAINTS
           #members(thisRole)<1
           & member(thisUser, parentActivity.Grader)
        OPERATION Grade {
           PRECONDITION #(Candidate.Submit.finish)=1
           ACTION ans.setGrade(data)}
} } }
```

Figure 2: Specifi cation of Collaborative Examination Activity Template

4.2 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. For example, instantiation of an activity, execution of a role operation, admission of user in a given role etc. represent different types of events. The multiple occurrences of a given event type — such as multiple executions of an operation — are represented by a list. We provide a count operator # on lists. Hence, *#(eventName)* returns the number of times the event has occurred.

We use event counters for synchronization specification based on the model presented in [14]. Related to each role operation are two types of events: *start*, and *finish*. For example, *role-name.op.start* or *role-name.op.finish*. For a role, we also have event types defined for *join*, *leave*, *admit*, and *remove* operations. For each activity, 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 invoker id, such as `opName.start(invoker=John)`. We can count how many times a user has invoked an operation using `#(opName.start(invoker=John))`.

Events related to the physical environment can be imported from the underlying runtime system, if the specification requires such events for context sensitive policies. For example, the collaboration system may need to detect the presence of a participant in a specific physical location to enforce context based access control.

4.3 Shared Object Specification

Shared objects are represented in our specification model only in terms of their types and method signatures, keeping the semantics and implementation details transparent. A specification for an object includes a type name to facilitate parameter bindings of operations in roles, a codebase to load the class of the object, and method signatures. The specification of an *ExamPaper* object type is shown in Figure 2. In addition of a codebase location, the object type has two method signatures defined: *setPaper* and *readPaper*. The specification of the types of the parameters in these methods are not shown, which are mapped to some Java object types or primitive types using XML DTD.

Our system supports both RBAC as well as traditional DAC. We can specify access control at the granularity of the methods invoked on these objects. For an object, access control is derived from the various roles' operations involving that object. Based on the security and coordination requirements specified in a collaboration, our system derives

appropriate policy modules, which are used by the object servers to control access to their objects [23]. Moreover, after an object is created the owner can specify additional access control based on the traditional DAC models.

4.4 Definition of Roles

In our model, a role can be viewed as a protection domain with a set of privileges to perform certain tasks in the shared workspace defined by its activity's scope. A role definition involves specification of two aspects:

- Meta-level policies in regard to admission of users to the role.
- Role related operations and associated preconditions for coordination and dynamic security requirements.

A role is defined in the scope of an activity and it can refer to the objects and other roles in that activity. It can invoke methods on the objects in that activity. As detailed below, a role specification includes role owners, reflected roles, admission constraints, activation constraints, and role operations with their preconditions. The basic terms of a role specification are shown below, where [ ] encloses optional terms.

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

The owner of a role must be an existing role from its outer scope. If the owner is not specified in a role definition, the owner of its activity is the owner. If the activity owner is not specified, the default owner of any activity is the owner of its parent activity. For the top level activity, a meta-role *Convener* is defined as the owner. The owner of a role can admit users to it, subject to the specified admission constraints; it can also remove an existing participant. Importantly, a role is managed by participants in its owner role. For example, in Figure 2, the *Instructor* role from the outer scope is specified as the owner of the *Examination* activity. As no owner is specified for the individual roles in this activity, by default, the *Instructor* becomes the owner of the *Examiner*, *Examinee* and *Grader* roles.

In the specification model several functions are defined for a role. A boolean function *member(user, role)* 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))*. A role definition may specify privacy constraints as to which other roles are permitted to query its participants' identities.

4.5 Role Admission Constraints

These constraints control a user's admission to the role to meet the security requirements stated in Section 3. For example, the *Assistant* role in Figure 1 can have several admission constraints, as shown below.

#members(thisRole) $\leq$ 2
& member(thisUser, parentActivity.Staff)
& !member(thisUser, Student)
& #(Instructor.admit) $>$ 0
& valid_user(thisUser)
& !(member(B,thisRole) & member(C,thisRole))

This example illustrates the following aspects of security requirements:

- *Static separation of duties* constraint specifying that a student in the *Course* activity cannot join the teaching assistant role.
- Role cardinality constraint requires that the member count for this role cannot exceed two.
- *Previous role membership* constraint, which requires that the person joining this role must already be a member of the *Staff* role defined in the parent activity.
- A user is admitted to this role only after some user has been assigned to the *Instructor* role.
- A "user-user separation of duties" requirement specifying that users *B* and *C* both cannot be assigned concurrently to this role.

4.6 Role Reflection and User Assignment

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 an 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 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 if any. In our model, roles are not defined based on permission inheritance [15, 13], rather defined in the scope of hierarchically nested activities. In Figure 2, the *Student* role is reflected in the *Examinee* role of any of the nested *Examination* activities.

The specification also indicate assignment of users to certain roles at the time of activity creation. For example, in Figure 2, users must be assigned to the *Examiner* role in the *Examination* activity, and the *Candidate* role in the *ExamSession* activity. The user assignments maybe specified as part of the activity creation specification. In Figure 2, the examinee invoking the *StartExam* operation is assigned to the *Candidate* role of the new *ExamSession* activity instance. The users specified to be assigned to a role also need to comply with the role's admission constraints. As opposed to role reflection and role assignment, when a user attempts to join a role directly, absence of a *previous qualification* constraint in the admission constraints prevents the user from joining.

4.7 Operation Specification

An operation specification includes a name, and may include a precondition and an action. The precondition must be true when the operation is invoked. The preconditions associated with operations allows one to specify coordination constraints as well as various dynamic security requirements, such as condition-based access control, dynamic "separation of duties", context-based access control.

The action part of an operation may invoke an object method, or create a new object or a nested activity. If the action part is empty, then the operation is used primarily for coordination purposes. A keyword *new* is reserved for specifying creation of an object or activity. Follows an example of an operation specification of the *Examiner* role in the *Examination* activity as shown in Figure 1. 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.

Use of preconditions enables us to specify fine grain "separation of duties" policies, like the "object based sepa-

```
OPERATION SetPaper {
    PRECONDITION #(SetPaper.start) = 0
    ACTION { exam=new OBJECT(ExamPaper)
             exam.setPaper(data)}
```

ration of duties" and the "operational separation of duties". 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 duties" policy.

```
OPERATION ApproveInvoice
    PRECONDITION
        #(PrepareInvoice.finish(invoker=thisUser))=0
    ACTION * approve the invoice *
```

Preconditions also enable us to specify coordination constraints, for both *inter-role* and *intra-role* coordination. In Figure 2, a student in the *Examinee* role can not execute the *StartExam* operation until the *Examiner* has set the exam paper. The precondition for this operation also illustrates an intra-role coordination policy, which allows each participant in the *Examinee* role to independently start an exam session. This illustrates the *independent participation model* for the members in the *Examinee* role. The precondition of the *SetPaper* operation in the *Examiner* role illustrates the *cooperative participation model* for the members in this role – only one member in this role can execute the *SetPaper* operation.

4.8 Role Activation Constraints

Role activation constraints must be true when a user executes a role operation. In contrast, role admission constraints are checked only when a user is admitted to a role. Such constraints may not be valid when an operation is executed. Role activation constraints apply to all role operations and can be viewed as preconditions that are common to all operations of a role. Certain security requirements can be enforced only by activation constraints. A "dynamic separation of duties" constraint, such as a user should not activate two roles at the same time needs to be specified as part of role activation constraints. Similarly, *previous qualifications* that must be ensured during role operation invocation need to be specified as activation constraints. For example, the following activation constraint of the

Assistant role in Figure 1 ensures that the role can perform any operation only as long as the participant is a member of the *Staff* role. Moreover, minimum cardinality constraints, which specify a minimum number of participants that

```
ACTIVATION_CONSTRAINTS
    member(thisUser, parentActivity.Staff)
```

must be present before any role operation can be performed, are specified as 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 (e.g. 2) 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.

```
#members(thisRole)>2
& #(members(thisRole) ∩ members(EE.Professor))>0
& #(members(thisRole) ∩ members(CS.Professor))>0
```

5 Policy Based Construction of Collaboration Environment

A collaboration environment in our model is realized in several steps through a policy driven middleware that we have developed [23]. Initially, the coordination and security policy for a collaboration is specified based on the schema. From the specification, various policy modules are derived for different kinds of requirements: role based security, object level access control, and secure event notification for coordination. A generic middleware facility discussed in [23] provides a set of generic components which are coupled with these application specific policy modules to realize the desired runtime collaboration environment.

In our distributed execution model [22], entities – roles, objects, and activities – are managed in a decentralized manner. In our collaboration model, all users are not equally trusted. To ensure the proper enforcement of various collaboration policies, the collaboration entities need to be maintained at trusted sites. We derive a distributed trust relationship among the owners of various entities based on owner assignment as specified in the activity definition. As roles from the outer scope of an entity are specified (or assigned through default rules as discussed in Section 4) as the owner of that entity, the owners represent a hierarchical trust relationship. An entity is managed at its owner's site.

In our specification model, the coordination policies are specified based on serialized evaluation of dependent operation preconditions. In a decentralized environment, where users' actions are communicated by means of events, communication of coordination states to other users may be delayed. In the absence of a central coordinator, such delays may cause coordination inconsistencies. In [22], we have presented a technique for for ensuring coordination consistency among distributed entities.

Coordination and dynamic access control are based on events communicated among distributed entities. To ensure integrity of our collaboration system, the middleware needs to ensure reliable delivery of events, and that events are not falsified or omitted. In [22], we present a protocol which ensures integrity of events by subscribing corroborating events.

6 Related Work

Our goal is similar to those of COCA [12] and DCWPL [2] in their approach of constructing a distributed collaboration environment from a high level specification. COCA [12] is a logic-based coordination policy specification language for interactive CSCW applications which views security policy as an integral part of coordination policy. At the implementation level, COCA is tied with IP multicast models and Prolog. Recent work on COCA [7] applied formal verification techniques for ensuring privacy in user presence awareness systems. DCWPL [2] addresses user level mechanisms to deal with group interaction issues and is limited to its predefined policies and functions. The specification of confidentiality using Z notation in [6] is limited to its theoretical foundation and lacks an implementation model.

The concept of role has been used in many CSCW systems, e.g. XCP [18], MPCAL [8], Quilt [11]. These applications use roles to represent groups of users with different tasks within a collaboration. They do not satisfy the access control requirements specified in this paper. Decentralized management of role memberships based on role certificates in a distributed service model is presented in [9]. Suite [4] presents an access-control model for multiuser GUI interfaces, mainly for coordination of shared editing-based synchronous collaboration. For that, it deals with fine grain access rights on shared data. In contrast, our work focuses on a broad range of application wide security requirements, addressing the needs of data confidentiality, integrity, and dynamic security policies. Intermezzo [5] is

one of the first systems to introduce the basic concepts of role based policies, with primary focus on user-presence awareness environments.

A task-based constraint specification language for workflow management systems is discussed in [1]. There, constraints are specified with a mapping between roles and tasks. In contrast, our work specifies both the security and the coordination policies, with realization of such policies in an implemented system. SecureFlow [10] imposes workflow authorization constraints on tasks using Authorization Template (AT), which is a tuple specifying privileges to be granted to a subject of a given role on an object of a given type during a specified time interval. There, the permissions are activated based on tasks. In contrast, an activity in our model is a higher level abstraction than AT, representing a collaboration pattern, which involves multiple roles, objects, and their coordination. An activity specification may contain multiple tasks or operations and is able to capture workflow stages. Moreover, in [10] roles are bound to tasks at runtime based on AT. This requires certain rules to resolve conflicts when multiple roles for a user can be bound to a task. This approach is mainly to impose role based access control on existing workflow tasks. In our specification, role and activity modeling incorporates collaboration tasks, and tasks are encapsulated as operations within the role definition. This provides a deterministic view of roles privileges.

7 Conclusion

We have presented in this paper a role-based specification model for collaboration systems based on requirements of dynamic security policies, role admission and activation constraints, and separation of duties constraints. The specification model unifies coordination specifications with dynamic access control policy specifications. Roles are defined and instantiated in the context of activities. We have presented the concept of *activity template*, which enables dynamic creation of activities and hierarchical structuring of sub-activities. An activity defines a protection domain for roles, objects, and operations. We have implemented this specification model using an XML schema and shown how the model supports different coordination and security requirements characteristic of collaboration systems. Policy modules are derived from the XML specifications and are integrated with a generic middleware for the automated realization of collaboration environments. This approach represents a novel technique for constructing the runtime environments for collaboration systems from their high level specifications.

References

[1] E. Bertino, E. Ferrari, and V. Atluri. A Flexible Model Supporting the Specification and Enforcement of Role-based Authorizations in Workflow Management Systems. In *ACM Workshop on Role-based Access Control*, pages 1–12, 1997.

[2] M. Corts and P. Mishra. DCWPL: a programming language for describing collaborative work. In *Proc. of CSCW'96*, pages 21 – 29, November 1996.

[3] S. Demurjian, T. Ting, and B. Thuraisingham. User-role based security for collaborative computing environments. *Multimedia Review*, 4(2):40–47, Summer 1993.

[4] P. Dewan and H. Shen. Controlling access in multiuser interfaces. *ACM Transaction Computer-Human Interaction*, 5(1):34 – 62, March 1998.

[5] W. K. Edwards. Policies and Roles in Collaborative Applications. In *Proc. of CSCW'96*, pages 11–20, 1996.

[6] S. Foley and J. Jacob. Specifying Security for Computer Supported Collaborative Computing. *Journal of Computer Security*, 3(4):233–253, 1995.

[7] P. Godefroid, J. D. Herbsleb, L. J. Jagadeesany, and D. Li. Ensuring Privacy in Presence Awareness Systems: An Automated Verification Approach. In *Proc. of CSCW'2000*, pages 59–68, December 2000.

[8] I. Greif and S. Sarin. Data sharing in group work. *ACM Transactions on Information Systems*, 5(2):187–211, 1987.

[9] R. Hayton, J. Bacon, and K. Moody. Access control in an open distributed environment. In *IEEE Symposium on Security and Privacy*, pages 3 –14, 1998.

[10] W.-K. Huang and V. Atluri. SecureFlow: a secure Web-enabled workflow management system. In *ACM Workshop on Role-based Access Control*, pages 83 – 94, 1999.

[11] M. Leland, R. Fish, and R. Kraut. Collaborative Document Production using Quilt. In *Proc. of CSCW'88*, pages 206–215, 1988.

[12] D. Li and R. Muntz. COCA: Collaborative Objects Coordination Architecture. In *Proc. of CSCW'98*, pages 179–188, 1998.

[13] E. C. Lupu and M. Sloman. Reconciling Role-Based Management and Role-Based Access Control. In *ACM workshop on Role-based Access Control*, pages 135–141, 1997.

[14] P. Roberts and J.-P. Verjus. Towards Autonomous Descriptions of Synchronization Modules. In *Proc. of IFIP Congress*, pages 981–986, 1977.

[15] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.

[16] R. S. Sandhu. Transaction control expressions for separation of duties. In *Fourth Annual Computer Security Application Conference*, pages 282–286, December 1988.

[17] R. Simon and M. Zurko. Separation of duty in role-based environments. In *10th Computer Security Foundations Workshop*, pages 183 –194, 1997.

[18] S. Sluizer and P. M. Cashman. XCP: an experimental tool for managing cooperative activity. In *Proceedings of the 1985 ACM Thirteenth Annual Conference on Computer Science*, pages 251 – 258, 1985.

[19] R. K. Thomas and R. S. Sandhu. Conceptual Foundations for a Model of Task-based Authorizations. In *In Proceedings of IEEE Computer Security Foundations Workshop* , pages 66–79, 1994.

[20] J. E. Tidswell and T. Jaeger. Integrated constraints and inheritance in DTAC. In *ACM Workshop on Role-based Access Control*, pages 93 – 102, July 2000.

[21] A. Tripathi. Adaptive middleware: Challenges designing next-generation middleware systems . *Communications of the ACM*, 45(6):39–42, June 2002.

[22] A. Tripathi, T. Ahmed, and R. Kumar. Secure Management of Distributed Collaboration Systems. Technical report, Dept. of Computer Science, Univ. of Minnesota, Aug. 2002. Available at http://www.cs.umn.edu/Ajanta.

[23] A. Tripathi, T. Ahmed, R. Kumar, and S. Jaman. Design of a Policy-Driven Middleware for Secure Distributed Collaboration. In *Proc. of ICDCS'2002*, pages 393 – 400, July 2002.