

# USING PARAMETERIZED UML TO SPECIFY AND COMPOSE ACCESS CONTROL MODELS

Indrakshi Ray, Na Li, Dae-Kyoo Kim, Robert France  
*Department of Computer Science, Colorado State University*

**Abstract:** Situations can arise in which organizations have to merge policies that are based on different access control frameworks, such as Role Based Access Control (RBAC) and Mandatory Access Control (MAC). Integrating policies requires addressing the following question: How will the integration impact access to protected resources? In particular, one needs to determine that the integration does not result in security breaches or unavailability of resources. A way of addressing these concerns is to model the access control frameworks, compose the models, and analyze the resulting model to identify problems. In this paper we outline a technique for modeling and composing access control policy frameworks. Specifically, we model RBAC and MAC using a variant of the Unified Modeling Language (UML) and show how to compose the models. The composed model can be used as a base for defining access control policies in a military domain.

**Key words:** Mandatory Access Control, Role-Based Access Control, Unified Modeling Language

## 1. INTRODUCTION

An access control framework provides a set of rules, concepts, and guidelines for defining access control policies. Two popular frameworks are Role Based Access Control (RBAC) and Mandatory Access Control (MAC). Situations can arise in which organizations have to merge policies that are based on different access control frameworks. In such cases one needs to ensure that unauthorized persons are not inadvertently given access to protected resources, and that authorized persons are not denied access to resources as a result of emergent properties of the merged policies.

In this paper we propose a modeling approach that can be used to describe the effect of composing policies developed under different frameworks. In the approach the frameworks involved are modeled and the resulting models are composed to produce a hybrid model. The hybrid model can then be analyzed to uncover undesirable emergent properties that can potentially prevent authorized access or allow unauthorized access.

The variant of the Unified Modeling Language (UML) [10] is used to specify access control frameworks. A number of reasons motivated the use of the UML. First, the UML is a de-facto industry standard for modeling software artifacts. Second, the graphical nature of the UML and its use of widely understood concepts make it relatively easy to use and understand. Third, the graphical notation allows one to more easily detect inconsistencies and other problems in model.

In our modeling approach, access control frameworks are specified by generic models that describe the family of access control behaviors and structures that conform to the rules defined in the framework. A framework is described by a set of UML template models. Instantiation of a template model results in a UML model that describes an application-specific policy that conforms to the framework.

Composition of access control models results in a *hybrid access control* (HAC) model that is also expressed in the template form. To compose two access control models we first determine (1) the elements in the two models that will be merged, (2) the elements that will appear “as is” in the composed model and (3) the elements that will be modified or omitted in the composed model. We also determine the application domain specific constraints that affect the hybrid model. This information is then used to compose the models to produce the HAC for a given domain. In this paper we illustrate our approach by integrating the hierarchical RBAC framework and the MAC framework for a military domain.

The rest of the paper is organized as follows. Section 2 gives an overview of RBAC and MAC and shows how one can model them using template forms of UML diagrams. Section 3 describes how the models can be composed to produce a hybrid access control model. Section 4 describes some of the related work in this area, and Section 5 concludes the paper.

## 2. MODELING RBAC AND MAC

In this section we give an overview of the UML, RBAC and MAC, and illustrate how RBAC and MAC can be modeled using template forms of UML diagrams. The UML is a standard modeling language maintained by the Object Management Group (OMG) (See <http://www.omg.org/uml> for

details). The UML defines notations for building many diagrams that each presents a particular view of the artifact being modeled. In this paper we utilize the following diagram types:

**Class Diagram:** A class diagram depicts the static structural aspects of an artifact being modeled. It describes the concepts and the relationships between them. Relationships are expressed using associations and generalizations/specializations. Concepts are described in terms of their properties, where a property can be represented by an attribute, a behavioral unit (an operation), or a relationship with another concept.

**Interaction Diagram:** An interaction diagram describes a behavioral aspect of an artifact being modeled. Specifically, it describes how instances of the concepts described in a class diagram collaborate in order to accomplish behavioral goals. The UML has two interchangeable forms of interaction diagrams: collaboration diagrams show both the interactions and the supporting instance structure; sequence diagrams show only the interactions. In this paper we use collaboration diagrams because they capture more information.

Examples of these diagrams will be described when used in this paper.

## 2.1 Role-Based Access Control

RBAC [1] is used to protect information objects (henceforth referred to as objects) from unauthorized users. To achieve this goal, RBAC specifies and enforces different kinds of constraints. Depending on the nature of the constraints present in a specific RBAC, the RBAC will use one or more of the following components: *Core RBAC*, *Hierarchical RBAC*, *Static Separation of Duty Relations*, and *Dynamic Separation of Duty Relations*.

Core RBAC embodies the essential aspects of RBAC. The constraints specified by Core RBAC are present in any RBAC application. The Core RBAC requires that users (human) be assigned to roles (job function), roles be associated with permissions (approval to perform an operation on an object), and users acquire permissions by being members of roles. The Core RBAC does not place any constraint on the cardinalities of the user-role assignment relation or the permission-role association. Core RBAC also includes the notion of user sessions. A user establishes a session during which he activates a subset of the roles assigned to him. Each user can activate multiple sessions; however, each session is associated with only one user. The operations that a user can perform in a session depend on the roles activated in that session and the permissions associated with those roles.

Hierarchical RBAC adds constraints to Core RBAC for supporting role hierarchies. Hierarchies help in structuring the roles of an organization. Role hierarchies define an inheritance relation among the roles in terms of

permissions and user assignments. In other words, role  $r1$  inherits role  $r2$  only if all permissions of  $r2$  are also permissions of  $r1$  and all users of  $r1$  are also users of  $r2$ . There are no cardinality constraints on the inheritance relationship. The inheritance relationship is reflexive, transitive and anti-symmetric.

Static Separation of Duty Relations (SSD) are necessary to prevent conflict of interests that arise when a user gains permissions associated with conflicting roles (roles that cannot be assigned to the same user). SSD relations are specified for any pair of roles that conflict. The SSD relation places a constraint on the assignment of users to roles, that is, membership in one role that takes part in an SSD relation prevents the user from being a member of the other conflicting role. The SSD relationship is symmetric, but it is neither reflexive nor transitive. SSD may exist in the absence of role hierarchies (referred to as SSD RBAC), or in the presence of role hierarchies (referred to as hierarchical SSD RBAC). The presence of role hierarchies complicates the enforcement of the SSD relations: before assigning users to roles not only should one check the direct user assignments but also the indirect user assignments that occur due to the presence of the role hierarchies.

Dynamic Separation of Duty Relations (DSD RBAC) aims to prevent conflict of interests as well. The DSD relations place constraints on the roles that can be activated within a user's session. If one role that takes part in a DSD relation is activated, the user cannot activate the other conflicting role in the same session.

In this paper we do not consider the constraints specified by DSD RBAC relations. We use hierarchical SSD to explain our ideas. In other words, our model comprises the following components: Core RBAC, hierarchical RBAC and SSD RBAC. A model of hierarchical SSD RBAC is shown in Fig. 1.

The hierarchical SSD RBAC (Fig. 1) consists of: 1) a set of users (*USERS*) where a user is an intelligent autonomous agent, 2) a set of roles (*ROLES*) where a role is a job function, 3) a set of objects (*OBS*) where an object is an entity that contains or receives information, 4) a set of operations (*OPS*) where an operation is an executable image of a program, and 5) a set of permissions (*PRMS*) where a permission is an approval to perform an operation on objects. The cardinalities of the relationships are indicated by the absence (denoting one) or presence of arrows (denoting many) on the corresponding associations. For example, the association of user to session is one-to-many. All other associations shown in the figure are many-to-many. The association labeled *Role Hierarchy* defines the inheritance relationship among roles. The association labeled *SSD* specifies the roles that conflict with each other.

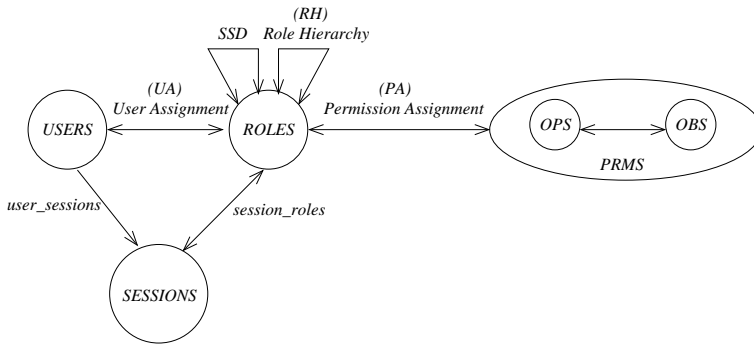


Figure 1. Hierarchical RBAC with SSD

The constraint in hierarchical SSD is given below (see [1] for formal definition): users cannot be assigned to roles that are involved in an SSD relation. In the following sections we give a graphical approach by which such constraint violations can be easily detected.

## 2.2 Modeling RBAC using Parameterized UML

In this section we specify the hierarchical SSD RBAC in terms of UML class diagram templates. A class diagram template consists of parameterized class elements, for example, parameterized class attributes, class operations, and relationships. A parameterized class is a class descriptor with parameters. It defines a family of classes where each class is obtained by binding the parameters to actual values.

Fig. 2 shows a class diagram template describing the hierarchical SSD RBAC. In the diagram, we use a symbol “|” to indicate parameters to be bound. This is not standard UML: we use this notation instead of the parameter list notation defined in the UML standard because it is more concise when there are many parameters involved. Each class template consists of two parts: one part consists of attribute templates that produce class attributes when instantiated, and the other part consists of operation templates that produce class operations when instantiated.

The RBAC template consists of the following parameterized classes: *User*, *Role*, *Session*, *Permission*, *Object*, *Operation*. The class template *User* has one attribute template named *UserID*. The behavior of the class template *User* is described by the operation templates *CreateSession* (creates a new session), *DeleteSession* (deletes an existing session), *AssignRole* (assigns a new role to the user) and *DeassignRole* (removes an existing role from the user). Typically there are many other *User* RBAC behaviors that can be described by operation templates. To keep the diagram simple, we do not

show all of them. One operation template not shown in the figure is *GetRoles* (returns the roles assigned to the user). The class templates *Role*, *Session*, and *Permission* are similarly specified.

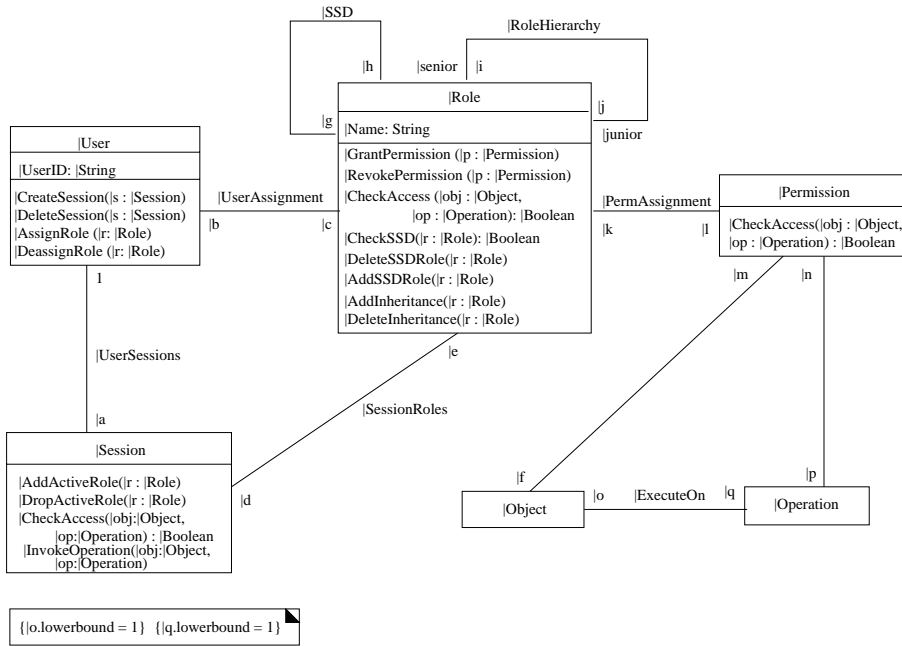


Figure 2. UML Class Diagram Templates Modeling Hierarchical SSD RBAC

Association templates, such as *UserAssignment* and *SessionRoles* produce associations between instantiations of the class templates they link. An association template consists of multiplicity parameters (one at each end) that yield association multiplicities (integer ranges) when instantiated. A *UserSessions* link is created by the *CreateSession* operation in an instantiation of the RBAC model; this link gets deleted by the corresponding *DeleteSession* operation. The operation *AssignRole* creates a *UserAssignment* link; the *DeassignRole* removes this link. The operations *GrantPermission* and *RevokePermission* are responsible for creating and deleting, respectively, the link *PermAssignment*. An *SSD* link is created by the *AddSSDRole* operation; this link is deleted by the *DeleteSSDRole* relation. The operation *AddInheritance* adds a link *RoleHierarchy*; *DeleteInheritance* deletes this link. For lack of space, we do not show the full specification of each operation. Pre- and postcondition templates in the operation template *CreateSession* is given below:

**context** |User :: |CreateSession(|s : |Session)

**post:** result = |s and |s.ocIsNew() = true and self.|Session includes(|s)

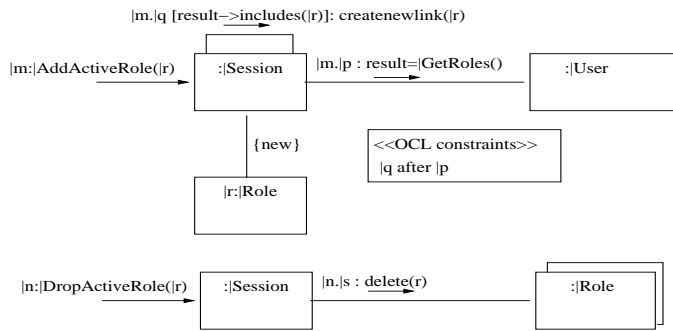


Figure 3. UML Collaboration Diagram Templates Modeling AddActiveRole and DropActiveRole

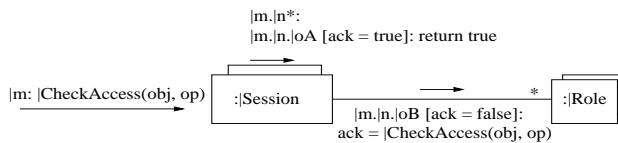


Figure 4. UML Collaboration Diagram Templates Modeling CheckAccess

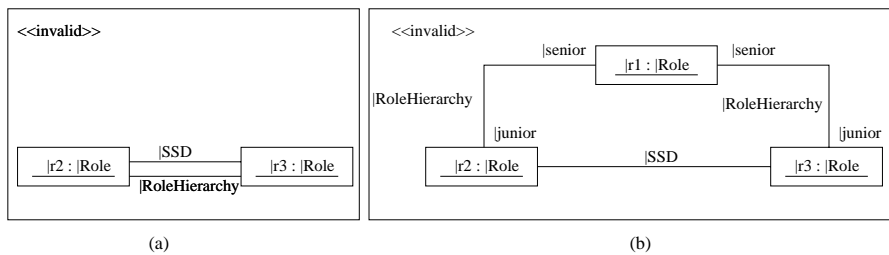


Figure 5. UML Object Diagram Templates Modeling Conflicts

The constraints in the bottom left corner of Fig. 2 impose restrictions on the cardinalities of the associations. For example,  $\{o.lowerbound = 1\}$  restricts an association that is an instance of *ExecuteOn* to having a multiplicity of at least one at the association end *Object*. The multiplicity “1” of *UserSessions* association end on *User* is strict: a session can only be associated with one user.

We can specify constraints using the Object Constraint Language (OCL) in class diagram templates (as is done in Fig. 2). Alternatively, one can express valid and invalid structures using object diagram templates. For

example, Fig. 5 shows an object diagram template specification that illustrates the violation of the constraint of the hierarchical SSD RBAC. Fig. 5(a) shows that there cannot be a policy in which two roles  $r2$  and  $r3$  connected by a hierarchy are also involved in an SSD relation. Fig. 5(b) specifies that there cannot be a policy in which two roles  $r2, r3$  that are in an SSD relation have the same senior role  $r1$ . Presence of such patterns indicates a problem with the specification.

The class diagram templates do not specify the interactions between operations. Collaboration diagram templates are used for this purpose. Fig. 3 shows interactions involving the operations *AddActiveRole* and *DropActiveRole* in the class template *Session*. For example, the *AddActiveRole* causes the invocation of the operations *GetRoles* in *User*. If the role to be activated  $r$  is included in the result, then a new link is established between the session and the new role  $r$ . *DropActiveRole* causes this link to be deleted. The details of *CheckAccess* operation in *Session* appears in Fig. 4.

### 2.3 Mandatory Access Control

The Mandatory Access Control framework that we use is adapted from the Bell-Lapadula model [8]. The Bell-Lapadula model is defined in terms of a security structure  $(L, \geq)$ .  $L$  is the set of security levels, and  $\geq$  is the dominance relation between these levels. The main components of this model are objects, users, and subjects. Objects contain or receive information. Each object in the Bell-Lapadula model is associated with a security level which is called the classification of the object. User, in this model, refers to human beings. Each user is also associated with a security level that is referred to as the clearance of the user. Each user is associated with one or more subjects. Subjects are processes that are executed on behalf of some user logged in at a specific security level. The security level associated with a subject is the same as the level at which the user has logged in.

The mandatory access control policies in the Bell-Lapadula model are specified in terms of subjects and objects. The policies for reading and writing objects are given by the Simple Security and Restricted-★ Properties.

- *Simple Security Property*: A subject  $S$  may have read access to an object  $O$  only if the security level of the subject  $L(S)$  dominates the security level of the object  $L(O)$ , that is,  $L(S) \geq L(O)$ .
- *Restricted-★ Property*: A subject  $S$  may have write access to an object  $O$  only if the security level of the subject  $L(S)$  equals the security level of the object  $L(O)$ , that is,  $L(O) = L(S)$ .

The static structural aspects of the MAC are described in the class diagram template shown in Fig. 6. The write and read operations prohibited by MAC are shown using object diagram templates in Fig. 7 and Fig. 8. The behavior of the *CheckAccess* operation is described by the collaboration diagram template shown in Fig. 9.

### 3. HYBRID ACCESS CONTROL: HAC

In this section we show how RBAC and MAC can be composed. The following steps identify how the composition of access control models takes place. Later in Section 3.1 we show how the application domain causes modification to the hybrid model.

**Step 1** Identify the entities in each of the access control frameworks.

**Step 2** Compare the definition of an entity in one model to that of another in the second model, and determine which represent similar concepts and which represent dissimilar concepts.

**Step 3** Matched entities (those representing similar concepts as determined in the previous step) are merged in the composed model.

**Step 4** Dissimilar entities that must be present in the composed model are added “as is” to the composed model, or are modified (as determined in step 2). Access model entities that have been identified for elimination are not added to the composed model.

We make the following observations about the elements in MAC and RBAC.

- *User*, *Object*, *Operation* are used in both the models and they each refer to the same concepts (see Section 2).
- *Subject* (used in MAC) and *Session* (used in RBAC) refer to the same concept [9].
- *SecurityLevel* appears in one model (MAC) but not in the other (RBAC).

Based on the above observations, we apply our algorithm to generate the hybrid access control model.

1. The *User* elements are merged in the two models. The structural feature templates are identical in both RBAC and MAC. The MAC specifies only one operation *CreateSubject* for *User*. The merged element includes the behavioral feature template from the element in RBAC and also those of MAC. The merged element appears in the hybrid model and we refer to it as *User*.
2. The elements *Object* and *Operation* in both the models are identical. Each of these elements is added to the hybrid model.

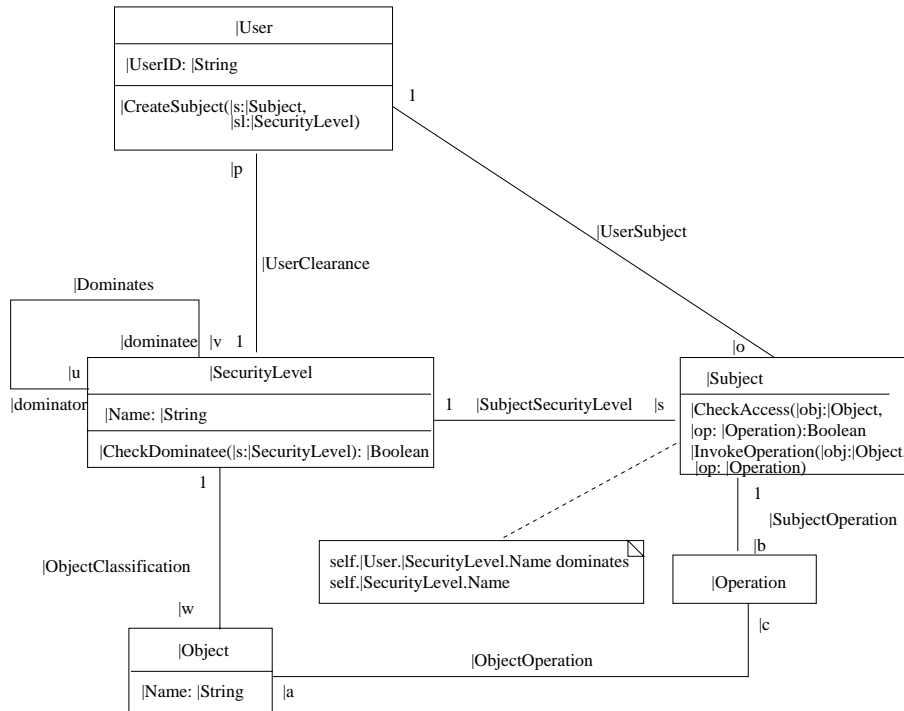


Figure 6. Mandatory Access Control (MAC)

3. The element *SecurityLevel* (present in MAC but not RBAC) is added to the hybrid model.
4. The elements *Session* and *Subject* refer to the same concept. These elements are merged. Since *Subject* is associated with *SecurityLevel*, the merged element is now associated with a security level. The merged element is referred to as *Session* in the hybrid model. Comparing these two elements in the two models, we see that RBAC *Session* has behavioral feature templates, such as *AddActiveRole* and *DropActiveRole* that are not present in *Subject*. These behavioral feature templates are added in the *Session* element of HAC. Both *Subject* and *Session* have *CheckAccess* and *InvokeOperation*. These operations if different must be merged. The *CheckAccess* operation in HAC is changed to reflect this. The collaboration diagram template of *CheckAccess* appears in Fig. 12. The merging of *Subject* and *Session* also affects other model elements. For instance, consider the class template *User* in HAC. From Step 1, the operations of *User* are *CreateSession*, *DeleteSession*, *AssignRole*, *DeassignRole*, and *CreateSubject*. Since *Subject* and *Session* are merged, we need to have only one operation called *CreateSession* (because the merged entity in HAC is called *Session*). Moreover, the *CreateSession* of

RBAC must be changed because now the security level also has to be passed as a parameter. The collaboration diagram templates of *CreateSession* in RBAC, *CreateSubject* in MAC, and *CreateSession* in HAC are given in Fig. 10.

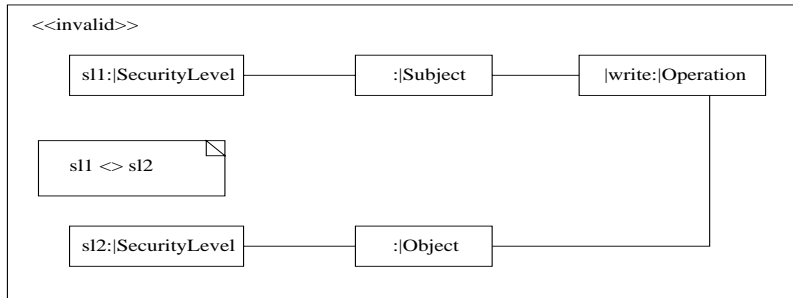


Figure 7. Write Operation Prohibited by MAC

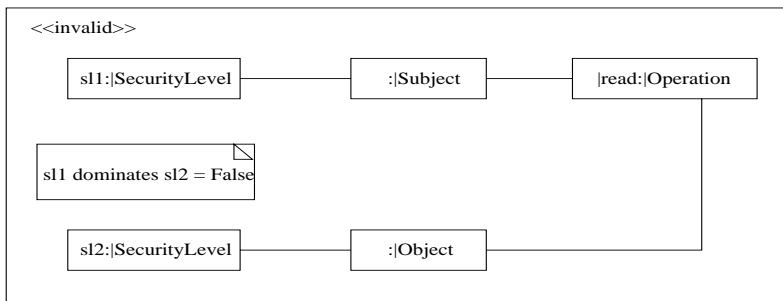


Figure 8. Read Operation Prohibited by MAC

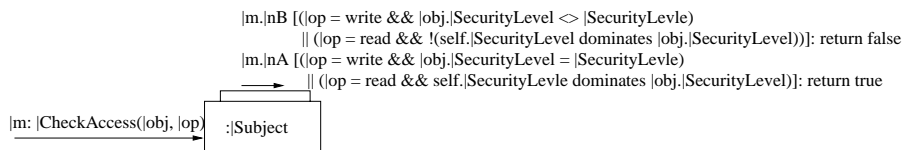


Figure 9. CheckAccess Operation in MAC

The class diagram template of this HAC is shown in Fig. 11.

### 3.1 Military Domain Requirement

The HAC shown in Fig. 11 may be suitable for applications, such as, an academic environment, that do not require any additional domain-specific constraints. For others, we may need to modify the hybrid model to incorporate the application domain specific constraints. Consider, for example, the military environment. The concept of *Role* in the military environment is associated with security levels [7]. For example, the role *CentralCommander* is associated with a security level of *TopSecret*. The roles *JointPlanner*, *ArmyLogisticsOfficer* are associated with a security level of *Secret* etc. To capture this association, we add an additional link to Fig. 11 in order to get the HAC for the military domain in Fig. 15 (the additional link is shown as a dark line in the figure).

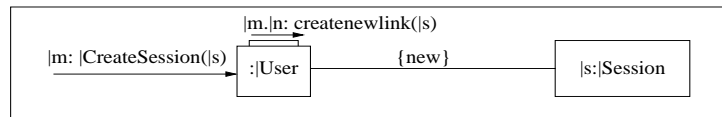
Roles being associated with security levels have other implications. For example, whenever a user is assigned to a role, the security level of the *User* must dominate the security level of the *Role*. In other words the *AssignRole* operation in the class *User* needs to be changed. Similarly, the roles that can be activated in a session must be the roles whose security levels are the same as the security level of the session. The *AddActiveRole* must be modified to check this additional precondition.

These additional constraints modify the HAC in the following way.

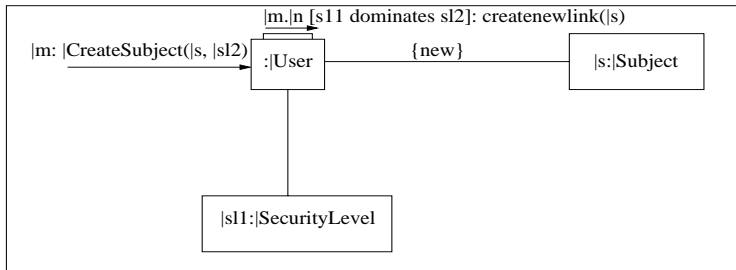
- An association is established between *Role* and *SessionLevel*.
- The operation *AssignRole* now must check that the security level of the *User* must dominate the security level of the *Role*. The operation *AddActiveRole* must be changed to check that the security level of the role activated must be equal to the level of the *Session*. To observe the change in the operation *AddActiveRole*, compare the *AddActiveRole* in RBAC (Fig. 3) with that of the *AddActiveRole* in HAC (Fig. 13).

The collaboration diagram templates in Fig. 14 show the *AssignRole* operations in RBAC and HAC.

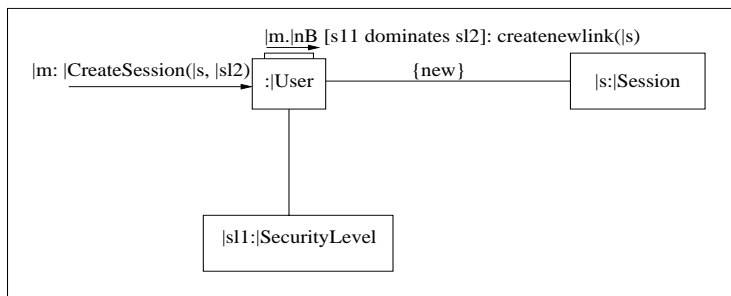
Merging access control frameworks may result in conflicts. For example, RBAC may allow a specific role  $r_1$  at security level  $s_2$  to write an object at the same security level. Because of role hierarchy the senior role  $r_2$  at security Level  $s_1$  (where  $s_1$  dominates  $s_2$ ) is also allowed to write on that object. However, this behavior is prohibited by MAC. Fig. 16 describes some undesirable patterns that indicate a presence of conflict in the hybrid model.



(a) CreateSession in RBAC



(b) CreateSubject in MAC



(c) CreateSession in HAC

Figure 10. Collaboration Diagram Templates for CreateSession

## 4. RELATED WORK

Several researchers have looked into integrating the mandatory access control and role-based access control models. Osborn [6] examines the interaction between RBAC and MAC. The author discusses the possible structures of role graphs that do not violate the constraints imposed by MAC.

In their approach when a subject is assigned to a role, the subject can perform all the privileges in the role, the possible structures that ensure that no violations of secrecy can occur are discussed. Their research shows that the combination of the structure imposed by the role graphs and the MAC rules means that the possible structure of a role graph in which roles are assignable to subjects without violating MAC rules is greatly restricted.

Nyanchama and Osborn [5] discuss the realization of MAC in role-based protection systems.

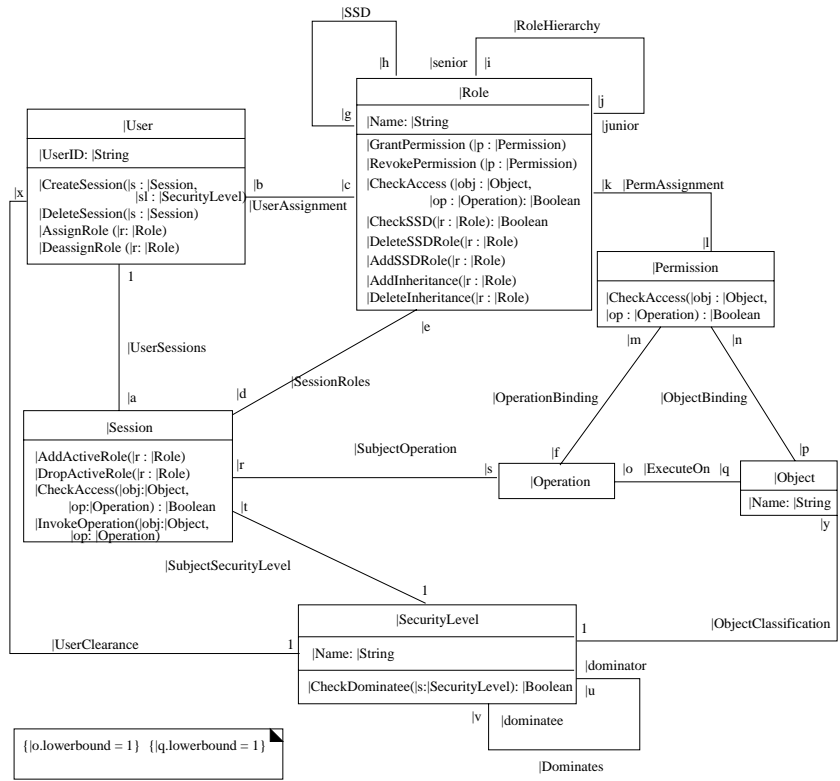


Figure 11. Hybrid Access Control (HAC)

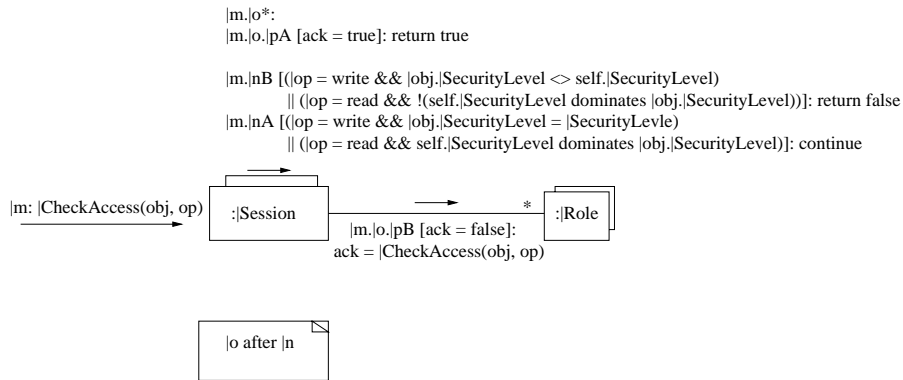


Figure 12. CheckAccess Operation in HAC

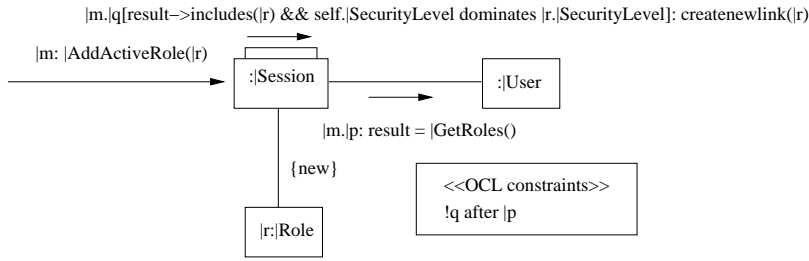


Figure 13. Collaboration Diagram Template for AddActiveRole Operation in HAC

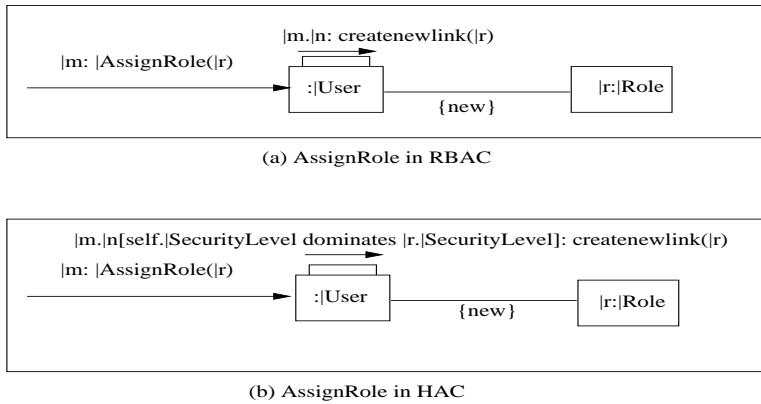


Figure 14. Collaboration Diagram Templates for AssignRole Operation in HAC

Phillips et al. [7] examine the unification of MAC and RBAC into a security model and enforcement framework for distributed applications. In their work, RBAC is extended to include MAC to ensure that the clearance of users playing roles meets or exceeds classification of resources, services, and methods being utilized. A role is assigned a classification, the authorized user must possess a classification greater than or equal to the role classification.

Researchers [4,2] have also investigated extending UML for representing access control. Lodderstedt et al. [4] propose SecureUML and define a vocabulary for annotating UML-based models with information relevant to access control. Jurgens [2] model security mechanisms based on the multi-level classification of data in a system using an extended form of the UML called UMLsec. The UML tag extension mechanism is used to denote sensitive data. Statechart diagrams model the dynamic behavior of objects, and sequence diagrams are used to model protocols.

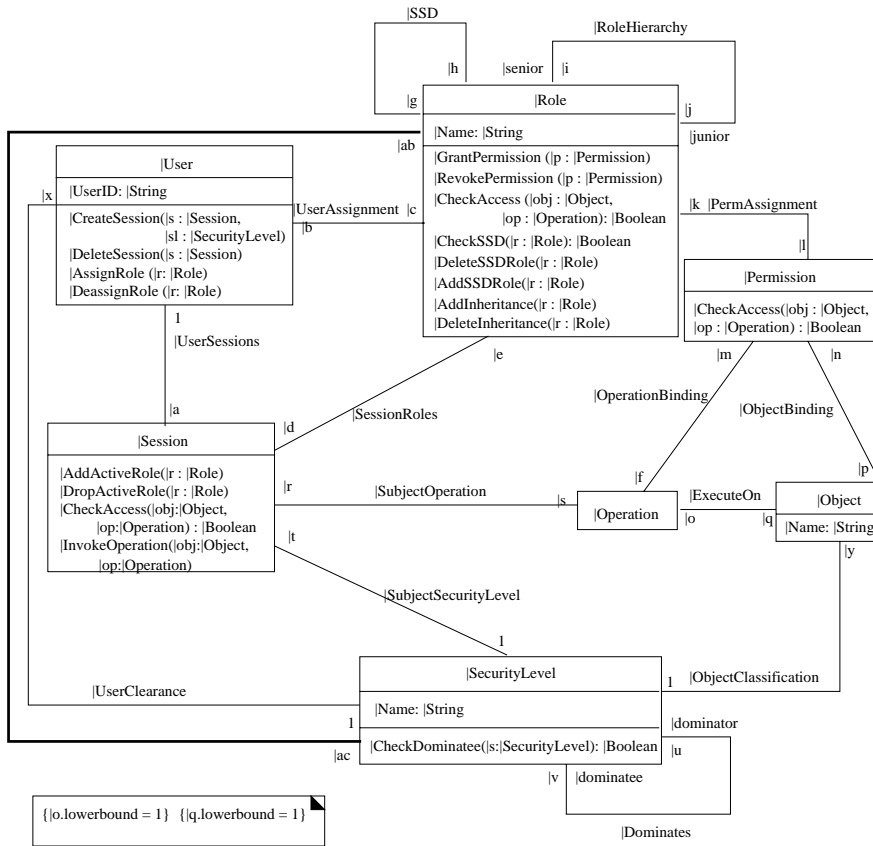
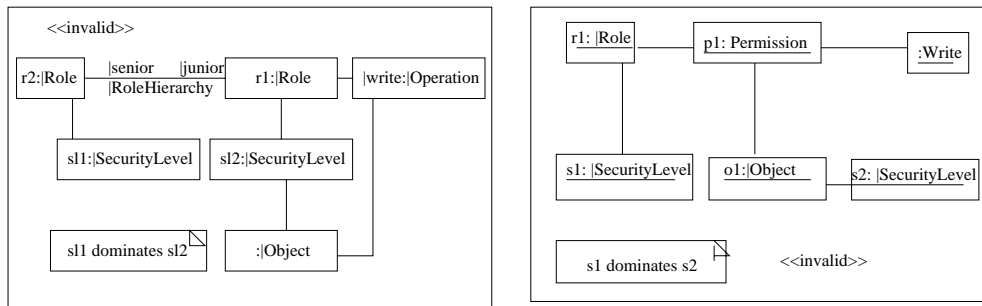


Figure 15. Hybrid Access Control (HAC) for Military Domain



(a) Example 1

(b) Example 2

Figure 16. Examples of Invalid Patterns in HAC

## 5. CONCLUSION

The need to integrate different kinds of access control frameworks arise when organizations using these frameworks need to work together in a collaborative environment. In this paper we show how to model these frameworks in a form that allows for easy composition, how to compose these models, and identify conflicts arising because of the composition.

A lot of work still remains to be done. The integration process, presented in this paper, is done manually. We plan to develop tools that will automate, in part, this composition process. The tool can compare the elements in the different models based on their structural and behavioral specifications and aid the user in resolving conflicts. Other types of conflict, such as mismatch in multiplicity parameters, mismatch in the specification of operations, that occur during the integration process can also be automatically detected by tools.

## References

- [1] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and Systems Security*, 4(3), August 2001.
- [2] Jan J-urjens. UMLsec: Extending UML for Secure Systems Development. In *UML2002*, 2002.
- [3] C. E. Landwehr, C. L. Heitmeyer, and J. McLean. A security model for military message systems. In *ACM Transactions on Computer Systems* 2(3): 198-222, August, 1984.
- [4] Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *5th International Conference on the Unified Modeling Language*, 2002.
- [5] Matunda Nyanchama and Sylvia Osborn. Modeling Mandatory Access Control in Role-Based Security Systems. In *IFIP Workshop on Database Security*, 1995.
- [6] Sylvia Osborn. Mandatory access control and role-based access control revisited. In *Proceedings of the Second Workshop on Role-Based Access Control*, 1997.
- [7] C. E. Phillips, S. A. Demurjian, and T. C. Ting. Toward information assurance in dynamic coalitions. In *Proceedings of the IEEE Workshop on Information Assurance*, United States Military Academy, West Point, NY, June 2002.
- [8] R. Sandhu and P. Samarati. Access Control: Principles and Practice. In *IEEE Communications*, *Volume 32, Number 9*, September, 1994.
- [9] Ravi Sandhu. Role-Based Access Control. In *Advances in Computers*, volume 46. Academic Press, 1998.
- [10] The Object Management Group (OMG). Unified Modeling Language. Version 1.4, OMG, <http://www.omg.org>, September 2001.