Towards Realizing a Formal RBAC Model in Real Systems

Gail-Joon Ahn and Hongxin Hu
The University of North Carolina at Charlotte
{gahn,hxhu}@uncc.edu

ABSTRACT

There still exists an open question on how formal models can be fully realized in the system development phase. The Model Driven Development (MDD) approach has been recently introduced to deal with such a critical issue for building high assurance software systems.

The MDD approach focuses on the transformation of high-level design models to system implementation modules. However, this emerging development approach lacks an adequate procedure to address security issues derived from formal security models. In this paper, we propose an empirical framework to integrate security model representation, security policy specification, and systematic validation of security model and policy, which would be eventually used for accommodating security concerns during the system development. We also describe how our framework can minimize the gap between security models and the development of secure systems. In addition, we overview a proof-of-concept prototype of our tool that facilitates existing software engineering mechanisms to achieve the above-mentioned features of our framework.

Categories and Subject Descriptors

D.2.1 [**Requirements/Specifications**]: Language, Methodologies, Tools; D.2.4 [**Software/Program Verification**]: Model Checking, Validation; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Design, Security, Languages, Verification

Keywords

Access Control, Policy Specification, Model Validation, Code Generation

SACMAT'07, June 20–22, 2007, Sophia Antipolis, France.

1. INTRODUCTION

Security is a crucial aspect in the development and management of modern software systems. Even though additional security countermeasures are integrated into existing systems after significant security problems are identified in the administration or usage phase, there exists a gap between security models and building secure systems with such models. Several issues must be taken into account to minimize this gap. More importantly, security models are generally described in some form of formalism. However, software developers are unlikely to adopt a formal security model for their development tasks. Therefore, it is very important to have a mechanism and corresponding tool to help software developers or system administrators understand and articulate the security model and associated polices in the software analysis and design stage. To address this issue, we present an empirical methodology called Assurance Management Framework (AMF) that ensures security models are fully realized and employed in real systems. Our main contribution in this paper is to show how the MDD approach and our systematic tool can be applied to build secure systems through security model representation, policy specification and validation, and automatic generation of security enforcement codes.

The rest of this paper is organized as follows. In Section 2, we briefly describe related technologies. Section 3 discusses our AMF approach including major components and tasks followed by a proof-of-concept prototype of our tool called RBAC Authorization Environment (RAE) in Section 4. Several related works are discussed in Section 5. Section 6 concludes the paper.

2. RELATED TECHNOLOGIES

2.1 Role-based Access Control Standard

RBAC standard was proposed by National Institute of Standards and Technologies (NIST) in 2001 [16] and formally adopted as an ANSI standard in 2004 [4]. The NIST/ANSI RBAC standard is composed of two parts: RBAC *Reference Model* and RBAC *System and Administrative Functional Specification*. The reference model defines sets of basic RBAC elements and relations, such as a set of roles, a set of users, a set of permissions, relationships between users, roles, and permissions. The system and administrative functional specification identifies all necessary functionalities required by role-based systems. These functionalities are divided into three categories: administrative operations, administrative reviews, and supporting system functions. In

addition, the NIST/ANSI RBAC standard has four components: Core RBAC, Hierarchical RBAC, Static Separation of Duty (SSD) Relations, and Dynamic Separation of Duty (DSD) Relations. In this paper, we adopt this standard model as a basis for security model and model representation.

2.2 Unified Modeling Language and Object Constraint Language

Unified Modeling Language (UML) [21] is a general-purpose visual modeling language in which we can specify, visualize, and document the artifacts of software systems. It captures decisions and understanding about systems that must be constructed. UML has become a standard modeling language in the field of software engineering. UML defines notions for building many diagrams, such as use case diagram, class diagram, collaboration diagram and so on, to depict a particular view of a system. In this paper, we focus on the class diagram and object diagram of UML. A class diagram depicts a structural view of information in a system. Classes are defined in terms of their attributes and relationships. The relationships include association, generalization/specialization, and aggregation of classes. An object diagram is an instance of a class diagram. It shows the system states as a collection of objects at a particular point in time. In this paper, we concentrate on class and object diagrams for representing RBAC model and system configuration, respectively.

The semi-formal semantics of UML has ambiguity and inconsistency issues [17]. Object Constraint Language (OCL) is a constraint expression language that enables us to describe constraints and expressions for UML-based models. OCL constraints typically specify restrictions that state conditions for all instances of the classes. In this paper, we adopt OCL to specify RBAC policies. OCL offers a number of advantages over the use of UML diagrams for modeling a software system. First, OCL expressions make the definition of UML graphical models more consistent and precise. Second, OCL has a formal semantic based on mathematical set theory and predicate logic. Thus, it is possible to check OCL expressions to ensure whether the expressions are correct and consistent with other elements of the model. Third, UML diagrams and OCL expressions can be integrated to support system development. As pointed out in [32], the OCL can be regarded as a key ingredient of UML-based model representation.

2.3 Role-based Constraints Language 2000

Role-based Constraints Language 2000 (RCL2000) [5] is a formal specification language for RBAC policies and helps identify useful role-based authorization constraints such as prohibition, obligation and cardinality constraints. The users of RCL2000 are security policy designers who understand organizational objectives and articulate security policies to support these objectives. RCL2000 also provides *n-ary* expressions and more flexibility in expressing access control constraints [19].

RCL2000 has six entity sets called users (`U`), roles (`R`), permissions (`P`), sessions (`S`), objects (`OBJ`) and operations (`OP`). Additional elements used in RCL2000 are three conflicting sets `CR`, `CP` and `CU`. `CR` is defined as a collection of conflicting role sets; `CP` is denoted as a collection of conflicting permission sets; and `CU` is a collection of conflicting user sets. RCL2000 supports six RBAC system functions `user`, `roles`, `sessions`, `permissions`, `operations` and `object`. Also, RCL2000 defines two nondeterministic functions, `OE` (one element) and `AO` (all other). The `OE(X)` function allows us to get one element form set `X`, and `AO(X)` is used to get a set by taking out one element. In addition, RCL2000 successfully demonstrated its capability to articulate most separation of duty properties [5]. In this paper, RCL2000 expressions are also used to specify formal authorization policies derived from the NIST/ANSI RBAC standard.

3. ASSURANCE MANAGEMENT FRAMEWORK (AMF)

In this section, we propose an empirical assurance management framework (AMF) that is designed for the representation of security model, the specification of security policy, and the validation of both the security model and security policy. The AMF framework consists of four tasks as follows:

1. ***Model representation.*** The representation of security model should enable software engineers to integrate security aspects into the application without knowing all details of a security model. In this regard, a well-designed visual representation can be considered as a powerful means to represent the security model in an intuitive fashion. Since UML is a general-purpose visual modeling language and widely used in modeling software systems as a de facto standard, our AMF facilitates UML features to represent the security model. Furthermore, changes and evolutions in modeling security models should be considered as well.

2. ***Policy specification.*** Security policies are an important means for laying out high level security rules for organizations to avoid unauthorized access. A considerable amount of work has been carried out in the area of specifying security policies. The policy specification languages can be classified into three categories: formal logic-based language, high-level language and visual specification language. Formal logic-based languages are precise and analyzable and can be used by policy designer to create complicated and fine-grained security polices. However, it is relatively hard for ordinary system developers to understand and to apply them in developing secure systems. High-level languages are readable and understandable to both policy designers and system developers but it is not suitable for analyzing security policies. Visual specification approaches are intuitive and easy to use while it is difficult to specify complex polices. The formal logic-based policy specification and visual policy specification should be translated into high-level policy specification in practical system development process so that security policies can be integrated into system design by system developers. Obviously, we need a supporting tool to achieve such functionalities.

3. ***Model and policy validation.*** It is necessary to check the consistency and validity of security models and policies before actual implementation commences. In the AMF framework, we validate a security model by producing a set of system states as snapshots and checking these states against the security policies. These

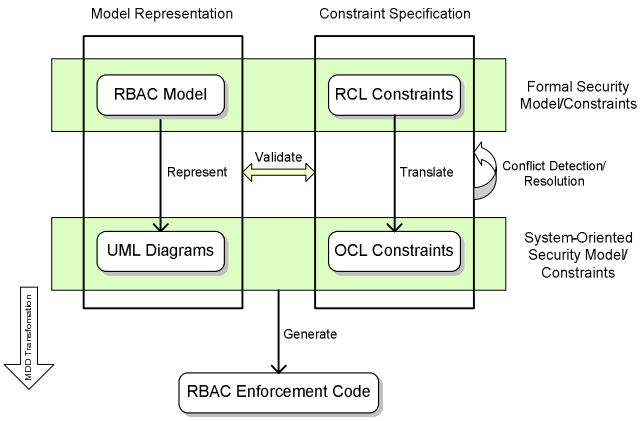


Figure 1: Realizing RBAC model and authorization constraints in AMF.

system states can be obtained by instantiating the security model and policies. For example, in the context of UML, an object model is an instance of the class model including objects and links and is also considered as a system state.

4. ***Conflict detection and resolution.*** The crucial aspect of a policy specification is to ensure that a policy is not conflicted with other existing policies. The validation approach should be used to detect the conflicts between security policies. Since it is not possible to generate all system states, there is no guarantee to detect all conflicts of policies. Therefore, conflict resolution mechanism is required for resolving the identified conflicts systematically [1].

For building a secure system based on a particular access control model, it is very important to have a system-oriented representation of the access control model for software engineers. Since UML is the standard language in modeling community, the usage of UML for the representation of security models is an attractive aim [29]. Therefore, our framework supports the representation of a formal RBAC model using UML. The constraints in RBAC have been considered as one of the important components that enforce the principal motivation of RBAC models [18]. Our framework specifies RCL2000-based authorization constraints. RCL2000 as a formal specification language is powerful to specify complicated constraints. In order to make RCL2000 expressions more meaningful to ordinary system developers, our framework translates RCL2000-based constraints into OCL specifications. Then UML-based modeling and OCL specifications are facilitated to *automatically* generate system modules, called RBAC enforcement codes which can be deployed in a real system implementation to support RBAC features.

As we discussed above, formal access control model and constraints are mapped to system-oriented system design, which involves representing the RBAC standard model and RCL2000-based authorization constraints with UML diagrams and OCL expressions. Then we validate the RBAC model and constraints to detect and resolve conflicts in system design as illustrated in Figure 1. We now demonstrate how an existing RBAC model can be realized in system development through the realization steps in Figure 1, identifying specific mechanisms for each AMF task.

3.1 Model Representation

We adopt the NIST/ANSI RBAC standard as a basis model since it includes most of well-recognized RBAC features [28, 15] and has been widely cited in information assurance community. The standard defines three models: Core RBAC, Hierarchical RBAC and Constrained RBAC. The Constrained RBAC in the standard adds separation of duty relations. However, there are two limitations in the Constrained RBAC model. First, the SD constraints in the standard are applied only to the activation of roles without considering other components in RBAC model. Second, the standard defines SSD relations with respect to user-role assignments over pairs of roles and DSD relations with the aspect of role activation in a user's session. These two constraints in the standard mainly reflect the simplest separation of duty properties. More fine-grained constraints cannot be defined adequately. Thus special constraint specification languages should be used to provide much richer expression for RBAC constraints. To reduce these limitations, we extend the Constrained RBAC to consider all aspects of role-based constraints and specify separation of duty constraints using RCL2000, which can express a variety of separation of duty properties.

Figure 2 shows a UML class diagram which depicts a complete representation of the NIST/ANSI RBAC model including Core RBAC, Hierarchical RBAC and Constrained RBAC. The representation can be decomposed to partially support Core RBAC or different compositions of three reference models. It contains classes, relationships between classes, and cardinalities in relationships. The basic entities are user, role, permission, and session classes. The permission class is represented as a composition of operation and object classes. The role hierarchy relationship is reflected in role class as a recursive relationship. The standard RBAC model only supports two separation of duty relations: SSD and DSD relations. As discussed above, constraints should be applied to all RBAC entities. Thus, in our model representation, we introduce two components such as SCR (Static Conflicting Roles) and DCR (Dynamic Conflicting Roles) that support SSD and DSD relations in the standard. Four new components SCP, DCP, SCU and DCU are created to support constraints in other RBAC entities such as permissions and users. These six components have dependency relationships with corresponding RBAC components in UML class diagram and are utilized by constraint expressions to identify more fine-grained SD constraints. The notations of those six components are derived from RCL2000 [2].

The functional specification in the standard defines various functions that role-based systems should provide. Since

[1]Our validation approach only checks the current system state as a snapshot (object model) against security constraints. In other words, the constraints are evaluated and enforced whenever the system configuration is changed. At the moment, we are not considering state transitions.

[2]RCL2000 defines three collections, CR, CP and CU, for the sets of conflicting roles, permissions, users respectively. The static separation of duty constraints and the dynamic separation of duty constraints employ separate conflicting element sets, thus we extended CR to SCR and DCR, CP to SCP and DCP, and CU to SCU and DCU, to support SSD constraints and DSD constraints separately.

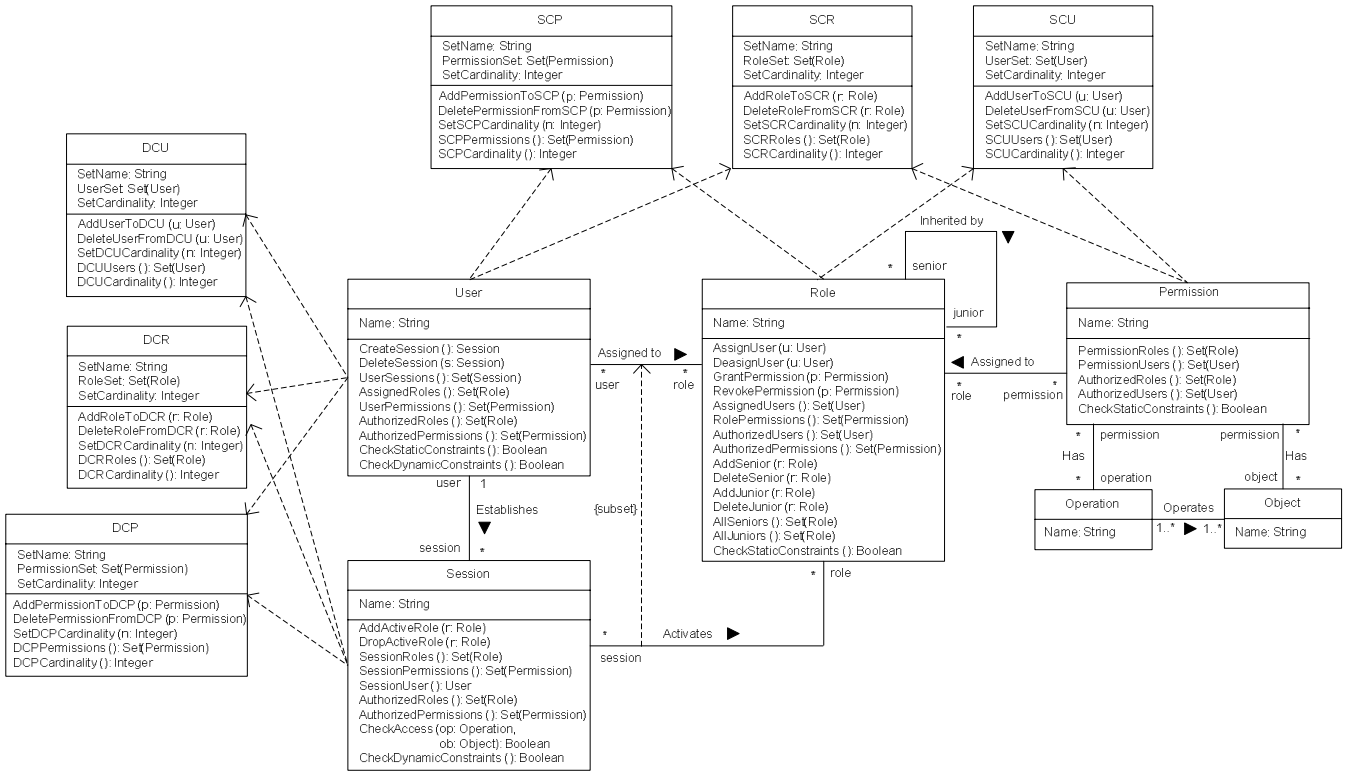


Figure 2: RBAC model representation in UML class diagram.

we use an object-oriented approach to express these functionalities of the standard, some subtle changes must be conducted in the definition of each function. For example, the RBAC standard defines two functions, `AddInheritance` and `AddAscendant`, to support building a new role inheritance relationship in role hierarchy. The standard explains `AddInheritance` is used to establish a new immediate inheritance relationship between two existing roles and `AddAscendant` is used to create a new role and to add this new roles as an immediate ascendant of an existing role. In object-oriented system design, every function is attached to a class. Therefore, the function `AddInheritance` cannot be used as a single class (`role` class in our model representation). On the other hand, since `CreateRole` function can be implicitly derived from `role` class in the UML class diagram at the implementation stage, `AddAscendant` may not include the operation for creating a new role. In our model representation, we define a function named `AddSenior` which adds an immediate senior role object to current role object instead of adopting two functions proposed in the standard. For role hierarchy, we also add two review functions `AllSeniors` and `AllJuniors` to query all seniors and juniors of a role object, because these two functions can frequently be called by many other functions, as well as by constraints in the presence of role hierarchy. Similarly, several review functions related to role hierarchy are added into our model representation. For example, two review functions, `AuthorizedRoles` and `AuthorizedUsers`, for a permission to find a set of roles that authorize the given permission and to get a set of users that can authorize the given permission through their roles, respectively. For brevity, we elaborate a few typical functional definitions of three components in the standard and corresponding OCL-based definitions.

1. Functional definition of Core RBAC

Administrative commands: These commands are for the creation and maintenance of RBAC element sets and relations by administrators. The functions for adding and deleting an element such as `AddRole` and `DeleteRole` can be created from UML class diagrams in the implementation step. A command specification for `AssignUser` is defined as follow:

context `Role::AssignUser(u:User)`
pre : `self.user->excludes(u)`
post: `self.user->includes(u)`

Review functions: These functions are for administrators to query RBAC element sets and relations. Query operations do not change the system states and they return a value or a set of values. In OCL, they are defined as a `body` expression. The following OCL definition supports a review function `UserPermissions`:

context `User::UserPermissions():Set(Permission)`
body: `self.role.permission->asSet()`

Supporting system functions: The functions are applied to create and maintain RBAC dynamic properties with regard to users' sessions and access control decisions.
`CheckAccess` checks whether an operation on an object is allowed to be performed in a particular session. OCL representation for this function is defined as follows:

```
context Session::CheckAccess(op:Operation,
        ob:Object):Boolean
pre :   true
post:   self.SessionRoles()->exists(r|r.permission
        ->exits(p|p.operation->includes(op)
         and p.object->includes(ob)))
```

2. Functional definition of Hierarchical RBAC

As illustrated in Figure 2, we have four major functions such as `AddSenior`, `DeleteSenior`, `AddJunior` and `DeleteJunior`. These functions are used for administrators to maintain inheritance relationships among roles. We define two new review functions `AllSeniors` and `AllJuniors` for role hierarchy. The following definition is for `AllSeniors`:

```
context Role::AllSeniors():Set(Role)
body:   self.senior->union(self.senior
        ->collect(r|r.AllSeniors()))->asSet()
```

3. Functional definition of Constrained RBAC

In our approach, the definitions related to constraint expressions are incorporated with corresponding components in UML-based model representation. We introduce two new system functions `CheckStaticConstraints` and `CheckDynamicConstraints` for RBAC model to enforce constraint expressions and to check conflicts. The functions for constraint checking can be used by other related functions in RBAC model as well[3]. The following definition is for `AssignUser` function.

```
context Role::AssignUser(u:User)
pre :   self.user->excludes(u)
post:   self.user->includes(u) and
        if (not self.CheckStaticConstraints()) or
           (not u.CheckStaticConstraints())
        then
          self.user->excludes(u)
        endif
```

In `AssignUser`, the assignment operation can affect the status of two objects, user and role objects. Hence, the static constraints for user and role classes should be enforced at the same time to prevent possible conflicts resulted from the assignment operation.

3.2 RBAC Constraint Specification

In NIST/ANSI RBAC standard, SSD constraints are defined with two arguments: a role set `rs` that includes two or more roles, and a natural number n, called the cardinality, with the property that $2 \leq n \leq |rs|$ which means a user can be assigned to fewer than n roles from role set `rs`. The similar definition is used in DSD constraints with respect to the activation of roles in sessions. The definition of constraints in the standard has limitations. To overcome such obstacles for considering other components in RBAC, we use RCL2000 that defines three sets, CR, CP and CU, as the collections for conflicting role sets, conflicting permission sets and conflicting user sets, respectively. Each conflicting set can include two or more elements. However, there is no

[3]For example, `AssignUser` and `GrantPermission` utilize `CheckStaticConstraints` to check the static assignment relations, and `CreateSession` and `AddActiveRole` employ `CheckDynamicConstraints` to check dynamic attributes related to sessions.

Input: RCL2000 expression;
Output: OCL expression;

Let **Simple-OE** term be either `OE(set)`, or `OE(function(element))`, where `set` is an element of {`U, R, OP, OBJ, P, S, SCR, DCR, SCP, DCP, SCU, DCU, scr, dcr, scp, dcp, scu, dcu`} and `function` is an element of {`user, roles, roles*, sessions, permissions, permissions*, operations, object`}

1. `AO` elimination
   Replace all occurrences of `AO`($expr$) with ($expr$-`OE`($expr$));

2. `OE` elimination
   While There exists **Simple-OE** term in RCL2000 expression
     case (1) **Simple-OE** term is `OE(set)`
          choose `set` term;
          call set_reduction procedure;
     case (2) **Simple-OE** term is `OE(function(element))`
          choose `function(element)` term;
          call reg_function_reduction procedure;
   End
   Procedure set_reduction
     case (1) `set` is `U`
          put `u:User=User.allInstances->any(true)` to top of existing quantifier(s);
          replace all occurrences of `OE(U)` with `u`;
     case (2) `set` is `R`
          put `r:Role=Role.allInstances->any(true)` to top of existing quantifier(s);
          replace all occurrences of `OE(R)` with `r`;
   .....[a]
   End
   Procedure reg_function_reduction
     case (1) `function(element)` term is `user(r)`
          put `u1:User=r.user->any(true)` to top of existing quantifier(s);
          replace all occurrences of `OE(user(r))` with `u1`;
     case (2) `function(element)` term is `roles(u)`
          put `r1:Role=u.role->any(true)` to top of existing quantifier(s);
          replace all occurrences of `OE(roles(u))` with `r1`;
   .....
   End

3. **System Functions** elimination
   While There exists `function(element)` term in RCL 2000 expression
     choose `function(element)` term;
     call sys_function_reduction procedure;
   End
   Procedure sys_function_reduction
     case (1) `function(element)` term is `user(element)`
          replace all occurrences of `user(element)` with `element.user`;
     case (2) `function(element)` term is `roles(element)`
          replace all occurrences of `roles(element)` with `element.role`;
   .....
   End

4. **Operators** elimination
   replace all occurrences of `set1` $\in$ `set2` with `set2->include(set1)`;
   replace all occurrences of `set1` $\cap$ `set2` with `set1->intersection(set2)`;
   replace all occurrences of `set1` $\cup$ `set2` with `set1->union(set2)`;
   replace all occurrences of |`set`| with `set->size()`;
   replace all occurrences of $\Rightarrow$ with `implies`;
   replace all occurrences of $\wedge$ with `and`;
   replace all occurrences of $\neq$ with `<>`;
   replace all occurrences of $\leq$ with `<=`;
   replace all occurrences of $\geq$ with `>=`;
   replace all occurrences of $\emptyset$ with {};

[a]Due to the page limit, we omitted the detailed procedures for `OE` elimination and system function elimination.

Figure 3: Translation algorithm

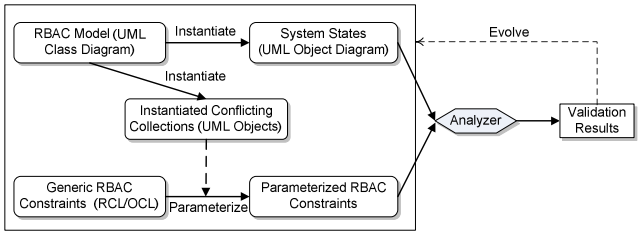


Figure 4: Validation of RBAC model and constraints

notation for the cardinality attribute in conflicting sets itself. Normally, we can regard the cardinality number n in RCL2000 always equals to *two* for each conflicting set.

Policy designers can employ RCL2000 to specify complex authorization policies to meet high-level security requirements along with the NIST/ANSI RBAC standard. In our framework, the next important step is that RCL2000 policy specifications need to be realized in UML-based RBAC representation which, in turn, we need to translate RCL2000 expressions to OCL expressions. In the previous section, we have defined several review functions. Using such definitions, `roles`* and `permissions`* function can be translated. For example, `roles`*`(u)` and `permission`*`(r)` are converted to `AuthorizedRoles(u)` and `AuthorizedPermissions(r)`. Each term in RCL2000 is converted to corresponding OCL operator or function. For example, operators like $\neq$, $\leq$, $\geq$, $\Rightarrow$, and $\wedge$ are replaced by <>, <=, >=, `implies`, and `and` operation correspondingly. The detailed translation algorithm is described in Figure 3.

Given a complicated constraint expression in [5] such that conflicting roles in the conflicting role set cannot be assigned to multiple conflicting users who are members of mutually exclusive roles, the translation algorithm generates an equivalent OCL expression as follows. This specific SD propriety is useful in a practical application system because it is necessary to prevent conflicting users such as family members from committing fraud.

RCL2000 Expression:

$$| \texttt{roles}^*(\texttt{OE(U)}) \cap \texttt{GS(OE(SCR))} | \leq \texttt{GC(OE(SCR))}$$
$$\wedge | \texttt{user(GS(OE(SCR)))} \cap \texttt{GS(OE(SCU))} | \leq \texttt{GC(OE(SCU))}$$

Translated OCL Expression:

```
context User
inv: let
       scr:SCR = SCR.allInstances()->any(true)
     let
       scu:SCU = SCU.allInstances()->any(true)
     in
       self.AuthorizedRoles()->intersection(scr.
         roleset)->size()<=scr.cardinality
       and
       scr.roleset.allInstances()->select(r|r.
         user->intersection(scu.userset)->size()
         <=scu.cardinality)
```

In this paper, we mainly focus on separation of duty constraints. Other types of constraints can be defined in OCL as well [6, 30]. Obviously, such OCL expressions can be produced from RCL2000 expressions using this translation algorithm.

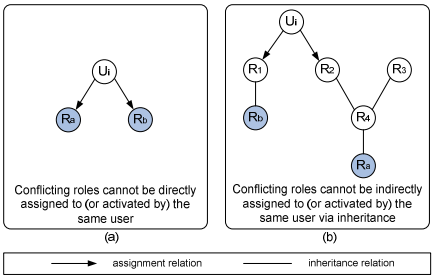


Figure 5: Conflict checking for SOD.

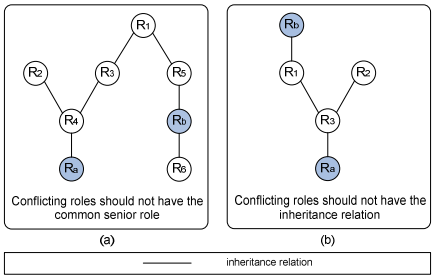


Figure 6: Conflict checking for role hierarchy.

3.3 RBAC Model and Constraint Validation

As we discussed earlier, our validation uses a set of system states and checks such states against authorization policies. Figure 4 presents our approach to validate RBAC model and constraints. In UML-based representation, a system state is a UML object diagram, which can be changed by creating and deleting objects as well as inserting and removing links between objects. Our validation approach supports both RCL and OCL expressions. In Figure 2, we have six components, SCR/DCR, SCP/DCP and SCU/DCU that are used to specify conflicting sets for roles, permissions and users. Constraint expressions employ these components to specify separation of duty constraints. These components are instantiated as objects which include conflicting object sets.

Figure 5 shows two typical examples. It illustrates conflicting roles cannot be directly or indirectly (via inheritance) assigned to (or activated by) the same user. On the contrary, two roles that have been assigned to (or activated by) the same user cannot be defined as statically (or dynamically) mutual exclusive. In our approach, a conflicting role set is extended to static conflicting role set and dynamic conflicting role set to support SSD and DSD constraints respectively. Figure 5 (a) shows that the user `Ui` owns (or activates) both roles `Ra` and `Rb` simultaneously. However, since `Ra` is statically (or dynamically) mutually exclusive to `Rb`, this assignment (activation) operation cannot be permitted. Figure 5 (b) depicts a more complex example taking role hierarchy into account. The user `Ui` acquires (or activates) two conflicting roles `Ra` and `Rb` through permission inheritance. Figure 6 shows two conflicts arising in role hierarchy. Figure 6 (a) depicts `Ra` and `Rb` are two mutually exclusive roles, however, they are owned by a common senior role `R1` via inheritance relation. This state should not be allowed in role hierarchy. Figure 6 (b) illustrates another conflict in role hierarchy. Two roles `Ra` and `Rb` are mutual exclusive, but now `Rb` is defined as ancestor of `Ra`. The inher-

```
1 public boolean CheckStaticConstraints() {
2   boolean result = true;
3   try {
4       Constraint_SSD_CR();  //Performing the SSD_CR constraint
5   } catch (ConstraintException e) {
6       throw new ConstraintError(e.getInstance(), e.getMessage());
7       result = false;
8   }
9   try {
10      Constraint_SSD_CP();  //Performing the SSD_CP constraint
11  } catch (ConstraintException e) {
12      throw new ConstraintError(e.getInstance(), e.getMessage());
13      result = false;
14  }
15  return result;
16 }
```

Figure 7: Generated Java code for `CheckStaticConstraints` function.

itance relation between conflicting roles should be avoided. On the other hand, when conflicting role sets are modified, or the SSD or DSD constraints related to the role hierarchy are established, the conflicts can also be identified through checking conflicting sets or constraints expressions.

The analyzer in Figure 4 is responsible for model and constraint validation. The main task of the analyzer component is to parameterize and interpret RBAC constraint expressions and evaluate these constraints against the current system state. When conflicting element sets are changed, or constraint expressions are established or modified, the analyzer checks if the constraints are violated by the current system states. Also, if the system state is changed, the analyzer also evaluates all authorization constraints against the changed system state. If any of the authorization constraint is violated during this process, it indicates that the authorization constraint is false, or the system state is undesirable. If the system state violates the RBAC constraints, the result component gives a conflict report to indicate root causes of the conflict. The conflict report assists users to find the reasons for the conflict and make decisions for resolving the conflict.

The result of the validation process may lead to two cases. Firstly, if there are reasonable system states that do not satisfy the authorization constraints, this may indicate that the constraints are too strong or the model is not adequate in general. Hence, we should modify the design, such as reducing the restriction of constraints. The other case is that the constraints may permit undesired system states, which points out the constraints are too week. As a result, the constraints must be further refined to exclude the undesired system states.

3.4 Code Generation

The code generation part of our approach enables users to build a real application by creating a platform independent model and then transforming it to platform dependent codes. Our objective for code generation is to generate security enforcement codes with some degree of assurance based on model specification represented by UML and OCL. As we addressed in the previous section, all model components and constraints are evaluated so the enforcement codes generated from our model representation should fully reflect features and functionalities of a formal security model, especially the NIST/ANSI RBAC standard in this paper. Although we select the Java language as the target language in this paper, we believe our mechanisms can be used for other

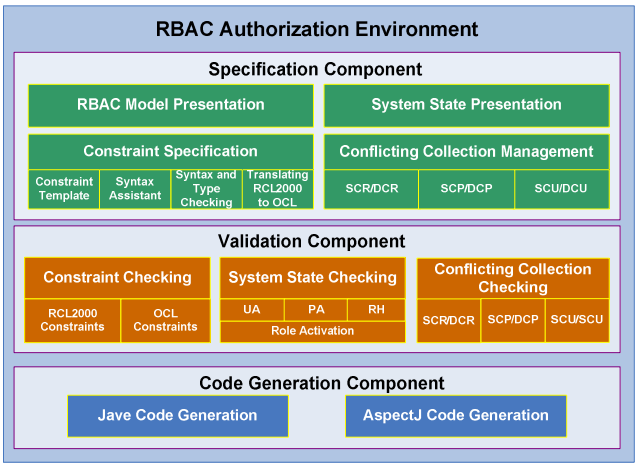


Figure 8: Structural overview of the RAE.

languages as well. The process of mapping model specification to enforcement codes may be performed by the adoption of the tools such as Octopus [3] and Dresden OCL toolkit [2]. In this paper, we omit the details of the translating process from UML and OCL to Java codes. Instead, we discuss some issues and solutions related to this translating process.

In our specification of RBAC model, RBAC model elements and relations are defined using the UML class diagram and the functionalities and constraints of RBAC model are specified with OCL expressions. To implement UML model elements, the classes, attributes, operations and associations need to be translated into corresponding Java classes or operations. Then, each class in the model is mapped to one Java class; an operation for the class is created by one operation in Java class; and an attribute and its association with the class in the model generate a private class member and a `get` and `set` operations in the Java class. Also, the basic types of OCL are mapped to corresponding Java types. For example, `Real` in OCL is mapped to `float` in Java. OCL collection type is implemented as a library using `Set` or `List` of Java language. It is a little complicated when implementing this library, because OCL collections have a large amount of predefined operations, such as `select` and `collect`. These operations should be defined as standard operations using Java. Based on the implemented standard OCL library, OCL expressions can directly generate Java codes.

In our implementation, two special system functions, `CheckStaticConstraints` and `CheckDynamicConstraints`, for Constrained RBAC are created automatically to collect and enforce static and dynamic constraint expressions respectively for corresponding components. Although we can use a universal function, such as a `CheckConstraints` function, to check all constraints for one component, for the purpose of making checking procedures more efficient, we provide two system functions for constraint checking. Session-related constraint expressions are performed by `CheckDynamicConstraints`, and other constraints are enforced by `CheckStaticConstraints`. Figure 7 generated Java codes for `CheckStaticConstraints` function of `user` class. Note that `CheckStaticConstraints` function includes the code for checking two static constraints with CR and CP as well.

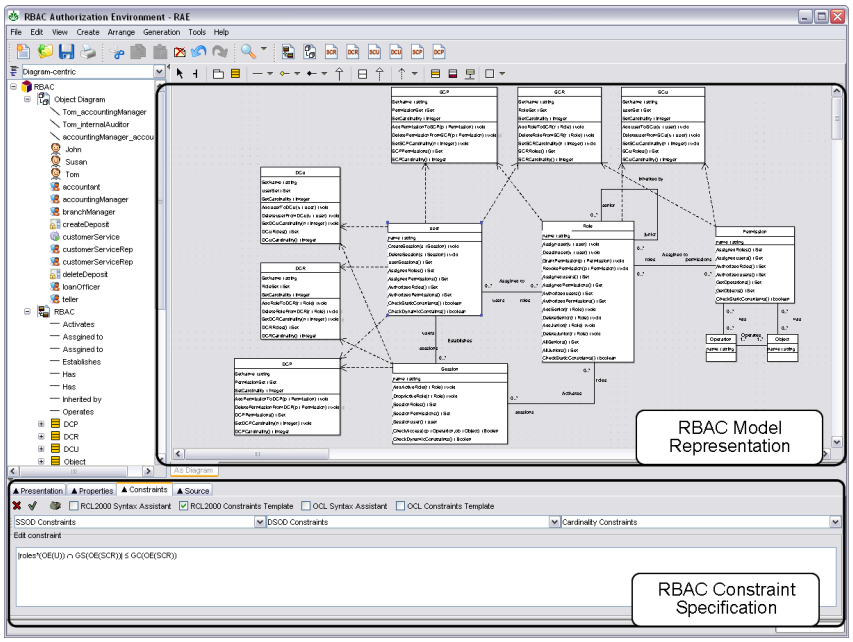


Figure 9: RBAC model and constraint specification.

4. RBAC AUTHORIZATION ENVIRONMENT (RAE)

To demonstrate the feasibility of our framework, we developed a tool called RAE which is based on ArgoUML, an open source UML-based modeling tool [1]. RAE tool is composed of three major components as shown in Figure 8: specification component, validation component and code generation component. Specification component is responsible for specifying RBAC model and constraints. Validation component is in charge of conflict checking so as to validate RBAC model and constraints. Code generation component is used to automatically generate enforcement codes.

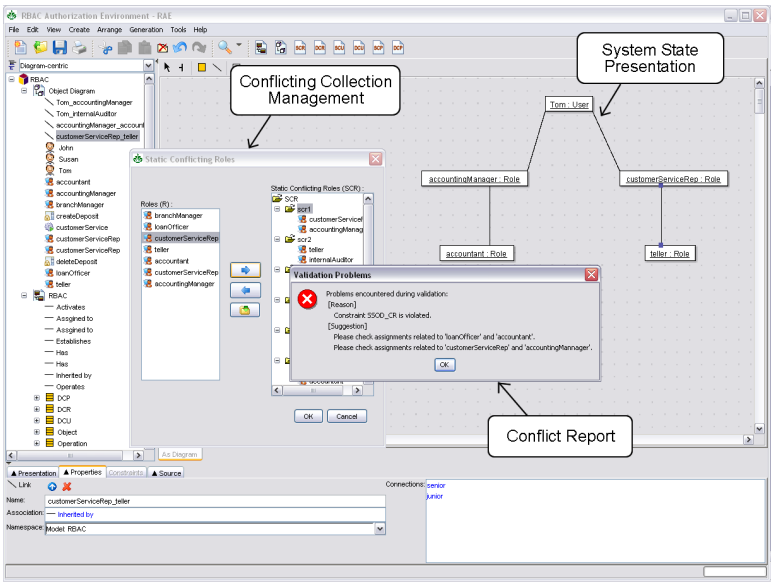
Specification component consists of four sub-components: RBAC model representation, system state presentation, constraint specification and conflicting collection management. Figure 9 shows the implementation of the RBAC model presentation and constraint specification components.

- *RBAC model representation* employs UML diagrams to represent RBAC model. This component is implemented based on ArgoUML.
- *System state presentation* is responsible to create snapshots of the system model at particular points using UML object diagrams, which is composed of a set of objects and a set of association links.
- *Constraint specification* provides an editing environment to easily specify authorization constraints using RCL2000 and OCL. It is further divided into four subcomponents: constraints template, syntax assistant, syntax and type checking, and translating RCL2000 to OCL. To simplify the constraint definition, some reusable constraint templates for typical RBAC constraints are provided in the constraints expression editor. Also it has a syntax assistant for user to construct complicated RCL2000 or OCL constraint expressions conveniently. The syntax checking verifies the constraints expression against the grammar of the specification language.
- *Conflicting collection management.* In the RAE tool, conflicting collections (SCR/DCR, SCP/DCP and SCU/DCU) are maintained separately from the constraint expressions.

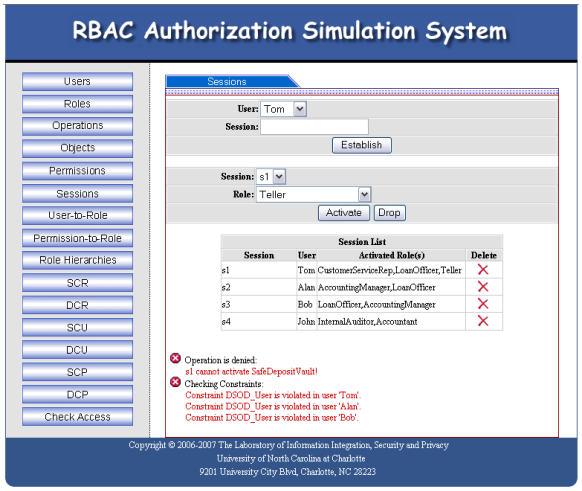
Validation component is composed of three sub-components: constraint checking, system state checking and conflicting collection checking.

- *Constraint checking* tests whether the constraint is violated by the current system state (snapshot), when a new constraint is established or an old constraint is modified. It supports both RCL2000 constraint checking and OCL constraint checking.
- *System state checking* is response for identifying the conflicts whenever a system state is changed. The RAE tool supports four kinds of assignment relation checking: UA (user-to-role assignment), PA (permission-to-role assignment), RH (role-to-role assignment) and role activation. When any assignment occurs, all related authorization constraints are evaluated against the changed system state.
- *Conflicting collection checking* is in charge of detecting the conflicts resulting from any changes of conflicting collections. Changing a conflicting collection affects the semantics of relevant SD constraint expressions. Figure 10(a) gives a snapshot of checking conflicts when static conflicting role sets are modified.

Code generation component is used to generate java code automatically for RBAC model and constraints. The enforcement codes are used by developers for role-based systems.



(a) Model and constraint validation.



(b) Simulation of creating session, activating role and checking dynamic constraints.

Figure 10: RAE tool and testbed environment

In addition, we developed a testbed environment to evaluate our RAE tool. We use a small banking system taken from [11] as a real application to test the functionality of RAE. Our experimental results clearly confirmed our expectations and functionalities mentioned in our framework. Figure 10(b) shows the simulation of creating session, activating role and checking dynamic constraints in our testbed environment.

5. RELATED WORK

There are several works concerning the specification of access control policies such as formal logic-based languages [12, 20, 8], high-level languages [14, 25, 26] and visualization of access control policies [31, 23]. In our framework, we combine three approaches to specify authorization policies. In [13], Crampton introduced a constraint monitor for enforcing authorization constraints. The constraint monitor checks whether granting the request would violate an authorization constraint and takes appropriate action. However, this approach omitted the policy enforcement issues during the modeling stage. In [30], we demonstrated how the USE tool, a validation tool for OCL constraints, can be utilized to validate authorization constraints against RBAC configurations. The policy designers can employ the USE-based approach to detect certain conflicts between authorization constraints and to identify missing constraints. However, the USE tool mainly focuses on the analysis of OCL constraints and has some limitations for specifying models and policies.

There are few works [22, 24, 27, 7] on UML-based modeling of security model. All of these approaches accommodated security requirements without considering the validation of security model and policy, and the translation to a concrete implementation. Our approach uses the standard UML to represent security model, supports the model and policy validation, and translates security model to enforcement codes.

6. CONCLUSION AND FUTURE WORK

In this paper, we have proposed an empirical framework that is designed to reduce the gap between security models and system development. We also discussed how the identified components and features in our framework can be utilized for representing the NIST/ANSI RBAC standard model using UML and OCL, and specifying authorization policies using RCL2000 and OCL. In addition, we implemented a systematic tool called RAE. We believe that this is the first attempt to implement such an intuitive tool including critical features such as the validation and code generation for role-based systems.

There is still a lot of work to be done in this research. Other constraints, such as history-based constraints and context-aware constraints, need to be considered. Also, different types of access control model including TRBAC [9] and GEO-RBAC [10] should be studied using our tool as well. Another possible direction for RAE tool is to investigate how the generated codes can be refined when the change of system requirements occurs, maintaining the same assurance level.

ACKNOWLEDGMENTS

This work was partially supported by the grants from National Science Foundation (NSF-IIS-0242393 and NSF-DUE-0416042), Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565) and Department of Defense (H98230-04-1-0210).

7. REFERENCES

[1] The ArgoUML Project. http://argouml.tigris.org.

[2] Dresden OCL toolkit. http://dresden-ocl.sourceforge.net.

[3] The Octopus Project. http://www.klasse.nl/octopus.

[4] *American National Standards Institute Inc.* Role Based Access Control, ANSI-INCITS 359–2004, 2004.

[5] G.-J. Ahn and R. S. Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 3(4):207–226, November 2000.

[6] G.-J. Ahn and M. E. Shin. Role-based authorization constraints specification using object constraint language. In *Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 157–162, 2001.

[7] K. Alghathbar and D. Wijesekera. authUML: a three-phased framework to analyze access control specifications in use cases. In *Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 77–86, New York, NY, USA, 2003. ACM Press.

[8] J. Bacon, K. Moody, and W. Yao. A model of OASIS role-based access control and its support for active security. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 5(4):492–540, 2002.

[9] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 4(3):191–233, 2001.

[10] E. Bertino, B. Catania, M. L. Damiani, and P. Perlasca. GEO-RBAC: a spatially aware RBAC. In *Proceedings of the tenth ACM symposium on Access control models and technologies (SACMAT)*, pages 29–37, New York, NY, USA, 2005. ACM Press.

[11] R. Chandramouli. Application of XML tools for enterprise-wide RBAC implementation tasks. In *Proceedings of the fifth ACM workshop on Role-based access control*, pages 11–18, Berlin, Germany, July 2000.

[12] F. Chen and R. S. Sandhu. Constraints for role-based access control. In *Proceedings of the first ACM Workshop on Role-based access control*, Gaithersburg, Maryland, United States, 1995.

[13] J. Crampton. Specifying and enforcing constraints in role-based access control. In *Proceedings of the eighth ACM symposium on Access control models and technologies(SACMAT)*, pages 43–50, June 2003.

[14] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 18–38, Bristol, UK, 2001.

[15] D. Ferraiolo and D. Kuhn. Role based access control. In *Proceedings of the fifth National Computer Security Conference*, pages 554–563, 1992.

[16] D. F. Ferraiolo, R. S. Sandhu, S. I. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 4(3):224–274, 2001.

[17] R. France. A problem-oriented analysis of basic UML static requirements modeling concepts. In *Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 57–69, New York, NY, USA, 1999.

[18] T. Jaeger. On the increasing importance of constraints. In *Proceedings of the fourth ACM workshop on Role-based access control*, pages 33–42, 1999.

[19] T. Jaeger and J. Tidswell. Practical safety in flexible access control models. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 4(3):158–190, 2002.

[20] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *IEEE Symposium on Security and Privacy*, pages 31–42, Oakland, CA, May 1997.

[21] J.Rumbaugh, I. Jacobson, and G.Booch. *The Unified Modeling Language Reference Manual, Second Edition.* Object Technology Series, Addison Wesley Longman, Reading, Mass, 2004.

[22] J. Jürjens. UMLsec: Extending UML for secure systems development. In *Proceedings of the 5th International Conference on The United Modeling Language*, pages 412–425. Springer Verlag, 2002.

[23] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A graph-based formalism for RBAC. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 5(3):332–365, 2002.

[24] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security, 2002.

[25] V. V. M. Hitchens. Tower: a language for role-based access control. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 88–106, Bristol, UK, 2001.

[26] OASIS. *XACML Language Proposal, Version 0.8.* Technical Report, Organization for the Advancement of Structured Information Standards, 2002, Available electronically from http://www.oasisopen.org/committees/xacml.

[27] I. Ray, N. Li, R. France, and D.-K. Kim. Using UML to visualize role-based access control constraints. In *Proceedings of the ninth ACM symposium on Access control models and technologies(SACMAT)*, pages 115–124, 2004.

[28] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[29] M. E. Shin and G.-J. Ahn. UML-based representation of role-based access control. In *Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 195–200, 2000.

[30] K. Sohr, G.-J. Ahn, and L. Migge. Articulating and enforcing authorisation policies with UML and OCL. In *Proceedings of the 2005 workshop on Software engineering for secure systems building trustworthy applications*, pages 1–7, 2005.

[31] J. Tidswell and T. Jaeger. An access control model for simplifying constraint expression. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 154–163, Athens, Greece, November 2000.

[32] J. Warmer and A. Kleppe. *The Object Constraint Language: Getting your models ready for MDA.* Addison-Wesley, Reading/MA, 2003.