Using UML To Visualize Role-Based Access Control Constraints *

Indrakshi Ray
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
iray@cs.colostate.edu

Na Li
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
na@cs.colostate.edu

Robert France
Computer Science Dept.
Colorado State University
Fort Collins, CO 80523
france@cs.colostate.edu

Dae-Kyoo Kim
Computer Science Dept.
Colorado State University
Fort Collins, C0 80523
dkkim@cs.colostate.edu

ABSTRACT

Organizations use Role-Based Access Control (RBAC) to protect information resources from unauthorized access. We propose an approach, based on the Unified Modeling Language (UML), that shows how RBAC policies can be systematically incorporated into an application design. We consider an RBAC model to be a pattern which we express using UML diagram templates; RBAC policies for an application conforming to this model can be generated by instantiating these templates with values obtained from the application. The constraints of the RBAC model are expressed using the Object Constraint Language (OCL). OCL constraints, based on first-order logic, are difficult to understand. To alleviate this problem, we show how violation of such constraints can be visually represented using object diagram templates. With adequate tool support, developers can use these to demonstrate constraint violations in their applications. Our approach is illustrated using a small banking application.

Categories and Subject Descriptors

D.2.1 [**Requirements/Specifications**]: [Languages, Methodologies]; K.6.5 [**Management of Computing and Information Systems**]: [Security and Protection]

General Terms

Design, Languages, Security

*Any opinions, findings, and conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of AFOSR.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profi t or commercial advantage and that copies bear this notice and the full citation on the fi rst page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specifi c permission and/or a fee.
SACMAT'04, June 2–4,2004, Yorktown Heights,New York, USA.
Copyright 2004 ACM 1-58113-872-5/04/0006 ...$5.00.

Keywords

Modeling, RBAC, UML

1. INTRODUCTION

Role-Based Access Control (RBAC) [10] is used by organizations to protect their information resources from unauthorized access. In RBAC, users are assigned to roles and roles are associated with permissions. A permission determines what operations a user assigned to a role can perform on information resources. In addition, various kinds of constraints can be specified in RBAC.

A lot of research appears in the area of specification of policies [3, 4, 5, 7, 8, 13, 14, 15, 16, 23, 22, 26]. Some researchers [3, 4, 5, 7, 13, 16, 23] use formal logic for specifying security policies; others [14, 15, 22, 26] use high level languages. Since formal-logic based approaches are difficult to use and understand, application developers are unlikely to use them. High-level languages are easy to use and understand, but are not amenable for analysis. A language is needed that is easy to understand and use, and also allows for the analysis of policy specification.

Researchers [12, 27, 28] have often advocated that security policies must be kept separate from the application. This allows security requirements to be clearly documented, policies to be changed independently of the application, policies to be independently analyzed, and policies to be centrally managed. Specifying policies independently creates an additional problem – how to integrate the policy concerns in an application. Thus, there is an additional requirement of policy specification language: the language should also allow the policy specification to be methodically integrated with the application.

In this work, we show how the Unified Modeling Language (UML) [29] can be used to specify RBAC policies. UML is the de facto modeling language used in the software industry. UML is easy to use and understand and is also amenable to analysis. Other researchers [2] have also advocated the use of UML for specifying RBAC policies. For instance, Ahn and Shin [2] show how RBAC constraints can be expressed in UML using the Object Constraint Language (OCL) [31]. However, they do not provide a systematic modeling approach that can be used by developers to create applications with RBAC features.

We present an approach for systematically incorporating RBAC

policies into an application design model that is specified using UML. We use UML diagram templates to specify patterns of reusable RBAC policies. The patterns describe reusable structures and constraints that developers can use to describe their application-specific RBAC policies. Class diagram templates are used to describe RBAC entities (e.g., user, permission, session) and their relationships. Applying an RBAC pattern in an application domain involves binding the template parameters to design elements in the domain.

To aid in the analysis of policies, RBAC constraints can be specified using the Object Constraint Language (OCL). OCL is based on first order logic which is not much comprehensible to the ordinary user. We provide an approach for visualizing RBAC constraints; this makes it easier for the end users to recognize problems with the constraints. Developers can specify application specific RBAC constraints as object diagrams. To assist in the task of identifying conflicts we provide object diagram templates that describe object structure patterns that describe violations to RBAC constraints. These patterns can be used by developers to check for the presence of policy violations.

Note that, other researchers [24, 30] have also focussed on visualization of access control policies. Osborn and Guo [24] use group and role graphs, and Tidswell and Jaeger [30] use a set-based notation. Unlike the technique described in this paper, these approaches use their own notations and could require some effort to integrate with the industry-based standard notations.

The rest of the paper is organized as follows. In Section 2, we give a brief overview of RBAC and UML. In Section 3 we present a generic RBAC model expressed as a class diagram template. Section 4 describes how to use object diagram templates to specify RBAC constraints. Section 5 illustrates how the violation patterns described by object diagram templates can be used to detect violations in application-specific RBAC policies. An overview of related work is provided in Section 6. Section 7 concludes the paper.

2. BACKGROUND

2.1 Role-based Access Control

RBAC 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. Fig. 1 describes the general model of RBAC. RBAC has three components: base model, role hierarchies, and constraints.

The base model embodies the essential aspects of RBAC. The base model 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 base model 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.

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 $r2$.

Constraints are an important aspect of RBAC and are sometimes argued to be the principal motivation for RBAC. The common examples of RBAC constraints include static separation of duty, dynamic separation of duty, prerequisite roles, and cardinality constraints. These constraints will be explained when we describe our approach for specifying RBAC constraints later in this paper.

2.2 Unified Modeling Language (UML)

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. 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.

Object Diagram: A static object diagram is an instance of a class diagram; it shows a snapshot of the detailed state of a system at a point in time. In this paper, we use object diagrams both to capture RBAC constraints and to model system specific security policies.

3. MODELING RBAC

In this section we specify RBAC in terms of UML template classes. A template class diagram is a class descriptor with parameters. A class diagram is obtained from a template diagram by binding the parameters to values. Fig. 2 shows a class diagram template describing hierarchical RBAC with SSD and DSD. The symbol "|" is used to indicate parameters to be bound. We use this notation when there is a large number of parameters that makes use of the standard UML parameter list cumbersome.

Class templates are associated with attribute templates (e.g., $|Name : String$ in *Role*) and operation templates (e.g., $|GrantPermission$ in *Role*). Association templates (e.g., $|UserAssignment$) consist of parameters for association names and association-end multiplicities. The OCL constraints in Fig. 2 restrict the values that can be bound to multiplicity parameters. For example, $\{|$o.lowerbound = $1\}$ restricts the multiplicities that can be bound to the parameter o to ranges that have a lower bound of 1. The multiplicity "1" on the $UserSessions$ association-end attached to $User$ is strict: a session can only be associated with one user.

The *User* class template represents users. A user can create a new session (*CreateSession*), delete a session (*DeleteSession*), associate self with a new role *AssignRole* and remove an associated role (*DeassignRole*). The operation *AssignedRoles* returns the set of roles directly assigned to the user while the operation *AuthorizedRoles* returns the set of roles directly assigned to the user as well as those roles that are inherited by the directly assigned roles. The class templates *Role*, *Session*, and *Permission* are similarly specified.

Association templates, such as *UserAssignment* and *SessionRoles* produce associations between instantiations of the class templates they link. A *UserSessions* link (i.e., an instance of an association obtained by binding the parameters of $UserSessions$ to values) is created by a *CreateSession* operation (i.e., an operation obtained by binding the operation template parameters to values) and deleted by a *DeleteSession* operation. The *AssignRole* operation creates a *UserAssignment* link; the *DeassignRole* operation removes a *UserAssignment* link.

Each operation template is associated with an OCL template expression that produces OCL pre- and post-conditions when the template parameters are bound to values. Pre- and post-condition templates associated with the $CreateSession$ and $GrantPermission$ operation templates are given below:

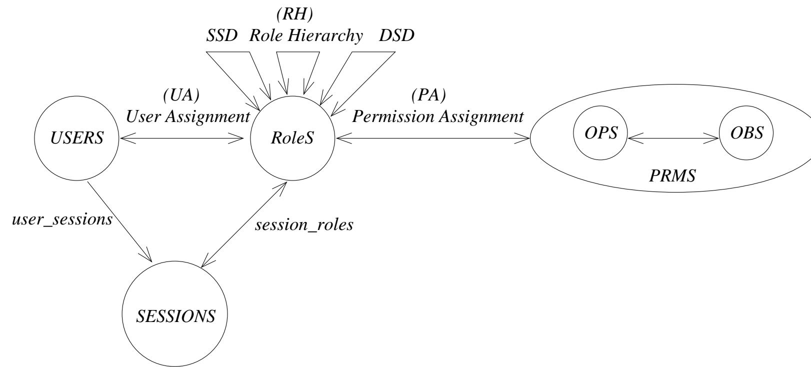


Figure 1: Describing the different components of RBAC

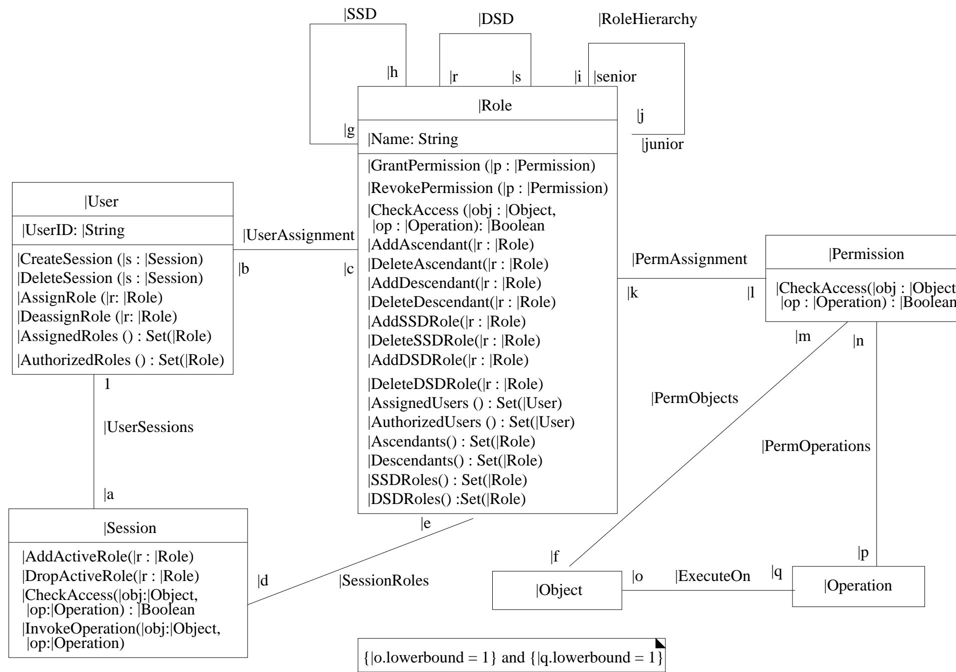


Figure 2: RBAC class diagram template

context |User::|CreateSession():(|s:|Session)
  post: result = |s and
  |s.oclIsNew() = true and
  self.|Session → includes(|s)

context |Role::|GrantPermission (|p:|Permission)
  pre: self.|Permission → excludes(|p)
  post: self.|Permission → includes(|p)

For a detailed description of how to use the templates to incorporate RBAC features into an application, please refer to our previous work (e.g., see [25, 18]).

4. MODELING RBAC CONSTRAINTS

Constraints are an important aspect of RBAC. Consequently, we focus on the specification of constraints. We can specify constraints using OCL in class diagram templates (as is done in Fig. 2). Alternately, one can visualize constraints using object diagram templates. We employ both techniques. The OCL approach is formal and precise; while visualization of constraints makes it easier to understand and recognize violation of those constraints.

In our visualization approach, we create object diagram templates that describe object structure patterns that are violations of RBAC constraints. These patterns can be used by developers to check for the presence of constraint violations. Developers can specify application specific security policies as object diagrams. Presence of invalid patterns in object diagrams that represent security policies indicate problems with the specification of security policies.

Our expression includes separation of duty constrains, prerequisite constraints, and cardinality constraints.

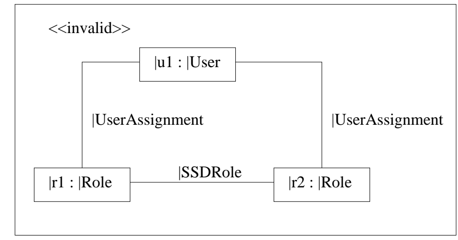
4.1 Separation of Duty Constraints

Separation of duty constraints are used to enforce conflict of interest policies. Static separation of duty (SSD) constraints aim 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 constraints are specified for any pair of roles that conflict. SSD constraints place constraint on the assignment of users to roles, that is, membership in one role that takes part in an SSD constraint prevents the user from being a member of the other conflicting role. We refer to this type of constraints as SSD-Role constraints. SSD-Role constraints may exist in the absence of role hierarchies or in the presence of role hierarchies. A role hierarchy defines inheritance relationships between roles. Through the inheritance relationship, a senior role inherits the permissions of its junior roles and any user assigned to the senior role is also assigned to the junior roles.The presence of role hierarchies complicates the enforcement of the SSD-Role constraints: before assigning users to roles not only should one check the direct user assignment but also the indirect user assignment that occurs due to the presence of the role hierarchies. An SSD-Role constraint is expressed as follows using the OCL template notation:

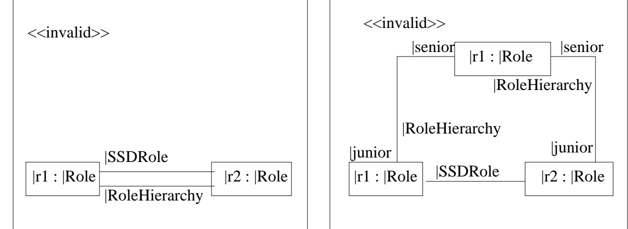
- SSD-Role constraint: Conflicting roles cannot be assigned to the same user.

  context |SSDRole **inv**:
    |r1. |User → excludesAll(|r2.|User)

Alternately, SSD-Role constraint violations can be visualized using object diagram templates as in Fig. 3. Fig. 3a describes structures in



(a) Violation of the SSD constraint in the absence of role hierarchies



(b) Violation of the SSD constraint in the presence of role hierarchies

Figure 3: Using UML object diagram templates to show SSD-Role constraint violations

which a user is assigned to roles in an SSD-Role constraint; Fig. 3b describes structures in which two roles connected by a hierarchy are also involved in an SSD-Role constraint, or two roles that are in an SSD constraint have the same senior role.

Dynamic separation of duty (DSD) constraints aim to prevent conflict of interests as well. DSD constraints place restrictions on the roles that can be activated within the same user session. If one role that takes part in a DSD constraint is activated, the user cannot activate the other conflicting role in the same session. We refer to these types of constraints as DSD constraints. A DSD constraint is expressed using the OCL template notation as follows:

- DSD constraint: Conflicting roles cannot be activated in the same session.

  context |DSD **inv**:
    |r1. |Session → excludesAll(|r2.|Session)

Alternately, DSD constraint violation can be visualized using object diagram template as in Fig. 4. Fig. 4 describes structures in which two roles in a DSD constraint are activated in the same session (violation of DSD constraints).

This conflicting notion can be applied to other elements such as user and permission in RBAC. The concept of conflicting permissions defines conflicts in terms of permissions rather than roles. Constraints of conflicting permissions are referred to SSD-Permission constraints. SSD-Permission constraints place constraint on the permissions that can be assigned to roles. Assignment of a permission that takes part in an SSD-Permission constraint to a role prevents the other conflicting permission being assigned to the role. Similarly, the concept of conflicting users defines conflicts in terms of users rather than roles. Constraints of conflicting users are referred to SSD-User constraints. SSD-User constraints place constraint on the users that can be assigned to roles. Assignment of a user that takes part in an SSD-User constraint to a role prevents

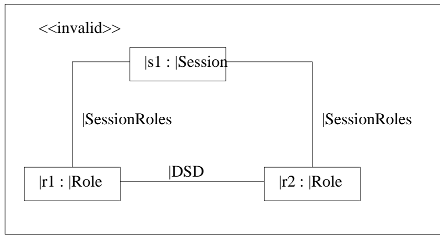


Figure 4: Using UML object diagram template to show DSD constraint violation

the other conflicting user being assigned to the role. The SSD-Permission and SSD-User constraints are expressed using OCL template expressions as follows:

- SSD-Permission constraint: Conflicting permissions cannot be assigned to the same role.

  context |SSDPerm **inv**:
  |p1. |Role $\rightarrow$ excludesAll(|p2.|Role)

- SSD-User constraint: Conflicting users cannot be assigned to the same role.

  context |SSDUser **inv**:
  |u1. |Role $\rightarrow$ excludesAll(|u2.|Role)

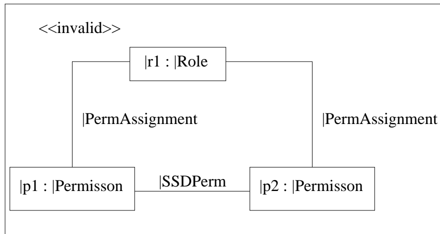


Figure 5: Using UML object diagram template to show SSD-Permission constraint violation

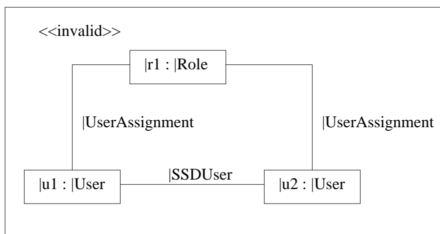


Figure 6: Using UML object diagram template to show SSD-User constraint violation

Alternately, SSD-Permission and SSD-User constraints can be visualized using object diagram template as in Fig. 5 and Fig. 6. Fig. 5 describes the structure in which a role is assigned to permissions in an SSD-Permission constraint (violation of SSD-Permission constraints). Fig. 6 describes the structure in which two users in an SSD-User constraint are assigned to the same role (violation of SSD-User constraints)

4.2 Prerequisite Constraints

The concept of prerequisite roles is based on competency and appropriateness. Prerequisite constraints require that a user can be assigned to a role only if the user is already assigned to the role's prerequisites. We refer to such constraints as Prerequisite-Role constraints.

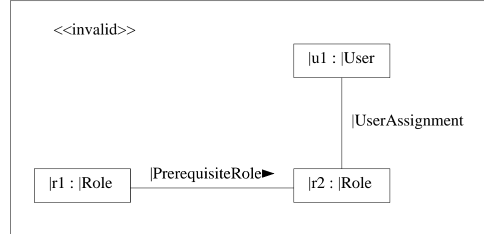


Figure 7: Using UML object diagram template to show Prerequisite-Role constraint violation

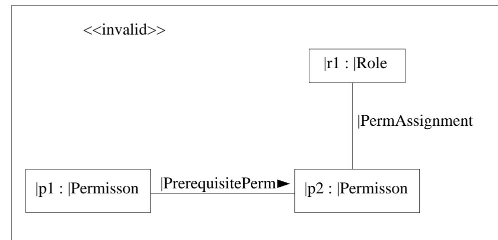


Figure 8: Using UML object diagram template to show Prerequisite-Permission constraint violation

This notion of prerequisite can also be applied to other elements such as permission in RBAC. The concept prerequisite permissions requires that a permission can be assigned to a role only if the role already possesses the permission's prerequisites. We refer to such constraints as Prerequisite-Permission constraints. Prerequisite-Role and Prerequisite-Permission constraints are expressed using OCL template expressions as follows:

- Prerequisite-Role constraint: A user can be assigned to a role only if the user is already assigned to the role's prerequisites.

  context |PrerequisiteRole **inv**:
  |r1. |User $\rightarrow$ includesAll(|r2.|User)

- Prerequisite-Permission constraint: A permission can be assigned to a role only if the role already possesses the permission's prerequisites.

  context |PrerequisitePerm **inv**:
  |p1. |Role $\rightarrow$ includesAll(|p2.|Role)

Alternately, Prerequisite-Role and Prerequisite-Permission constraints can be visualized using object diagram template as in Fig. 7 and Fig. 8. Fig. 7 describes the structure in which a user is assigned to a role without being assigned to the role's prerequisite (violation of Prerequisite-Role constraints). Fig. 8 describes the structure in which a role is assigned to a permission without being assigned to the permission's prerequisite (violation of Prerequisite-Permission constraints).

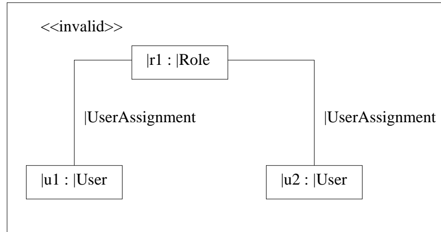


Figure 9: Using UML object diagram template to show cardinality constraint violation

4.3 Cardinality Constraints

Another constraint type is cardinality constraints. Cardinality constraints can be used to restrict, for example, the number of users that can be assigned to a role, the number of roles a user can play, the number of roles a permission can be assigned to, or the number of sessions a user is allowed to activate at the same time. Cardinality constraints place constraint on the relationship between elements. They restrict the number of elements that can be related to each other.

This numerical limitation may vary depending upon organizational policies. For example, we may have one type of organizational policies stating that there is at most one person in a role. Fig. 9 shows the specification of this type of constraints. Fig. 9 describes the structure in which two users are assigned to a role which, according to a cardinality constraint, should have at most one user assigned to it (violation of cardinality constraints).

5. IDENTIFYING CONFLICTS IN SYSTEM-SPECIFIC RBAC POLICIES

In this section we illustrate how the violation patterns shown in the previous section can be used to identify policy conflicts. If the violation pattern exists in an object diagram describing a system-specific policy, then a conflict exists. Checking for the presence of a pattern in an object diagram specifying a set of policies is essentially a search for a sub-graph in an object diagram.

To illustrate our approach we use a simple banking application taken from [6]. The application is used by various bank officers to perform transactions on customer deposit accounts, customer loan accounts, ledger posting rules, and general ledger reports. A set of system-specific RBAC policies for the banking system is given below:

Core policies: The roles of the banking system (instances of *BankRole*) are *teller*, *customerServiceRep*, *accountant*, *accountingManager* and *loanOfficer*. The permissions assigned to these roles are given below:

P1 A teller can modify customer deposit accounts.

P2 A customer service representative can create or delete customer deposit accounts.

P3 An accountant can create general ledger reports.

P4 An accounting manager can modify ledger-posting rules.

P5 A loan officer can create and modify loan accounts.

Hierarchical policies: The hierarchical policy in the banking application is stated below:

H1 Customer service representative role is senior to the teller role.

SSD-Role policies: For the banking system the following pairs of roles are conflicting:
{*(teller, accountant), (teller, loanOfficer), (loanOfficer, accountant), (loanOfficer, accountingManager), (customerServiceRep, accountingManager)*}

DSD policies: For the banking system the following pair of roles is in DSD relation:
{*(customerServiceRep, loanOfficer)*}

Prerequisite-Role policies: For the banking system the following pair of roles is in a prerequisite role constraint:

PR1 Accountant role is a prerequisite role for the accounting manager role.

Fig. 10 shows the object diagram that integrates the policies listed above. The reader can visually check that the patterns described by object diagram template in Fig. 3, Fig. 4, Fig. 5, Fig. 6, Fig. 7, Fig. 8, and Fig. 9 does not occur in Fig. 10.

Formally, an object diagram has the violation described by a violation pattern if there exists a binding that produces an object structure contained in the object diagram. To illustrate how conflicts can be identified, consider the case in which the following policy is added to the set of policies described in the previous section: "The branch manager role is senior to all the other roles in the bank." Fig. 11 shows this policy. A number of occurrences of the pattern described in Fig. 3b can be found in Fig. 11. For example, if we assign a user to the branch manager role, the user is also assigned to the roles *customerServiceRep* and *accountingManager* through inheritance. However, the roles *customerServiceRep* and *accountingManager* are in an SSD constraint.

The second example of conflict occurs when a new employee *Peter*, to whom both customer service representative role and loan officer role are assigned, activates both roles in a session. Fig. 12 shows that the new policy violates DSD constraints in Fig. 4.

The third example of conflict occurs when an employee *John* is assigned to an accounting manager role without being assigned to an accountant role. Fig. 13 shows that the new policy violates Prerequisite-Role constraints in Fig. 7.

Considering the case in which the following policy is added to the set of organizational policies: "There is at most one person in the branch manager role. An occurrence of the pattern described in Fig. 9 occurs in Fig. 14 when two employees *Peter* and *John* are both assigned to the branch manager role.

6. RELATED WORK

Tidswell and Jaeger [30] propose an approach to visualizing access control constraints. They point out the need for visualizing constraints and the limitations of previous work (e.g., [1, 21, 24]) on expressing constraints. A drawback of their work is that they

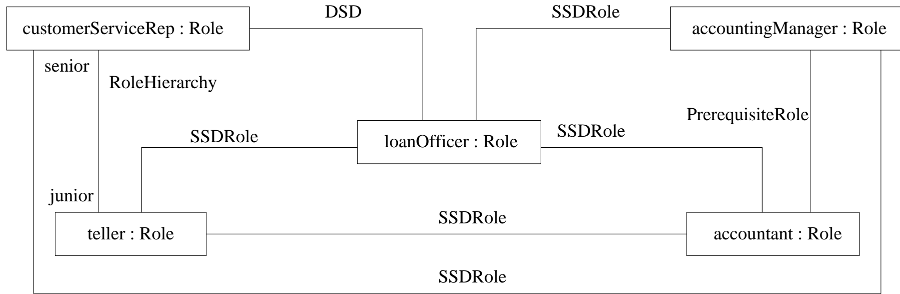
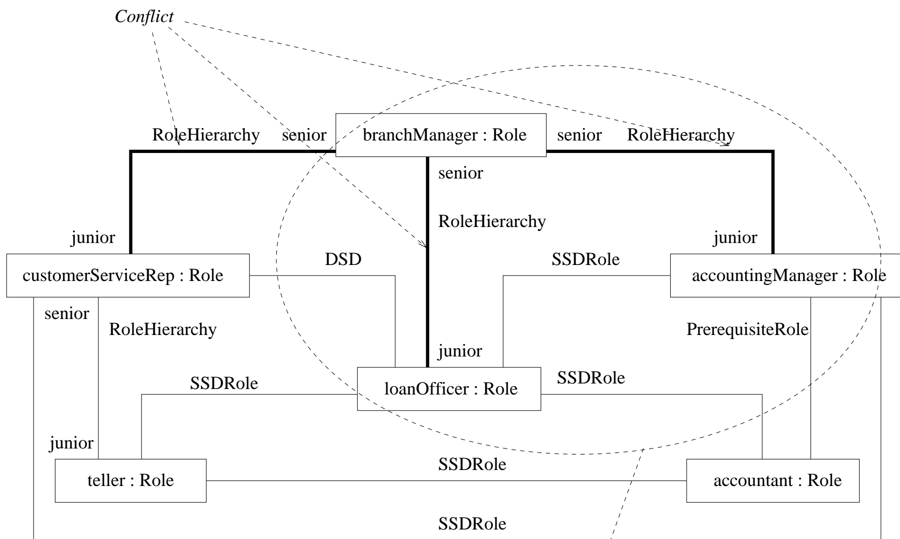
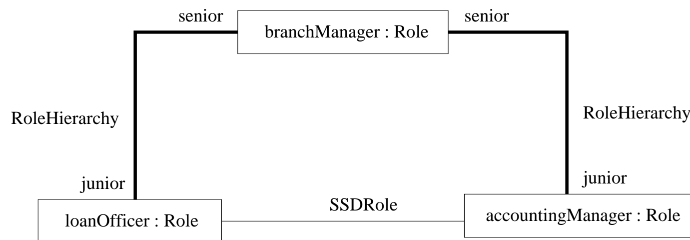


Figure 10: Object diagram for RBAC policies in a banking system



(a) Conflicting Policies



(b) Detecting Conflict

Figure 11: Violation pattern occurrence: SSD-Role

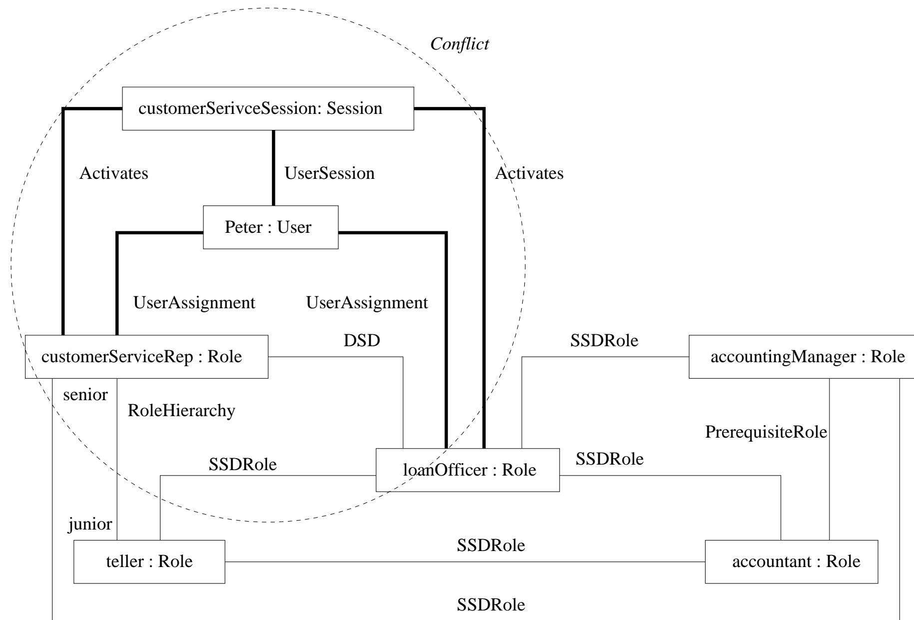


Figure 12: Violation pattern occurrence: DSD

created a new notation for specifying constraints and it is not clear how the new notation can be integrated with other widely-used design notations. The approach described in this paper utilizes notations from a standardized modeling language and also integrates the policy specification activity with design modeling activities.

Another effort to graphical specification of RBAC is proposed by [19]. In their approach, RBAC policies are represented by graph transformations. A graph consists of nodes and edges. Nodes represent notions such as users and roles. Edges represent relationships between notions. Transformation rules are defined for administration activities such as *add a user to a role* and *remove a user from a role*. Consistency properties such as DSD constraints are also specified graphically. Verification of RBAC policies is carried on by showing that graphical constraints do not occur in the graph specifying RBAC policies. The drawback of this approach is similar to what has been discussed in previous paragraph.

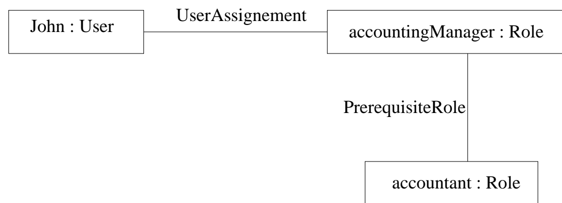


Figure 13: Violation pattern occurrence: Prerequisite-Role

A large volume of research [3, 4, 5, 7, 8, 13, 14, 15, 16, 23, 22, 26] exists in the area of specification of access control policies. Formal logic-based approaches [3, 4, 5, 7, 13, 16, 23] are often used to specify security policies. They assume a strong mathematical background which makes them difficult to use and understand. Other researchers have used high-level languages to specify policies [14, 15, 22, 26]. Although high-level languages are easier to understand than formal logic-based approaches, they are not analyzable. Our

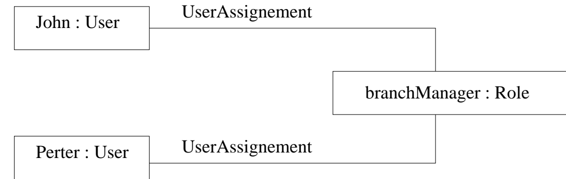


Figure 14: Violation pattern occurrence: Cardinality

work, on the other hand, provides a easy-to-use language supported by mechanisms for detecting problems with the specifications.

Some work [9, 17, 20] has been done on modeling system security using UML. Jurjens [17] proposes UMLsec, a UML profile for modeling and evaluating security aspects based on the multi-level security model. Lodderstedt *et al.* propose SecureUML [20], an extension of the UML that defines security concepts based on RBAC. These approaches mainly focus on extending the UML notation to better reflect security concerns. The approach described in this paper tackles the complementary task of capturing RBAC policies in patterns that can be reused by developers of secure systems.

7. CONCLUSION

In this work we have shown how RBAC policies can be modeled such that they can be easily integrated with the application, easy to understand and use, and are amenable to analysis. We specify reusable RBAC policies using UML diagram templates. The UML diagram templates can be instantiated to obtain application-specific RBAC policies. RBAC constraints can be specified using OCL. Since comprehending OCL may be difficult for the end user, we show how to represent RBAC constraint violations using object diagram templates. Application-specific policies are expressed using object diagrams. The object diagrams can be checked to detect any constraint violations.

A lot of work remains to be done. The work described in this

paper focuses on specifying the static structure of RBAC. A complete RBAC model should also include descriptions of patterns of behaviors supported by RBAC. In previous works [11, 18], we developed template forms of interaction diagrams that can be used to specify interaction patterns. The interaction patterns can be used to characterize families of allowed and prohibited behaviors.

An important question is validation of security policies. The RBAC policies for a given application can be tested against a set scenarios, some of which can be obtained by instantiating the interaction patterns of RBAC. For example, in order to evaluate the impact of an RBAC policy on a system, test scenarios that model prohibited behaviors can be obtained by instantiating RBAC interaction patterns that describe prohibited behaviors. Such tests can be used to determine if the manner in which the policies are addressed in the design are sufficient to prevent unauthorized access.

8. ACKNOWLEDGEMENT

This material is based upon work funded by AFOSR under Award No. FA9550-04-1-0102. The authors thank Dr. Robert Herklotz for supporting this work. The authors also thank the anonymous reviewers for their useful comments.

9. REFERENCES

[1] G. Ahn and R. Sandhu. The RSL99 Language for Role-Based Separation of Duty Constraints. In *Proceedings of ACM Workshop on Role-Based Access Control*, pages 43–54, 1999.

[2] G.J. Ahn and M. E. Shin. Role-based authorization constraints specification using object constraint language. In *Proceedings of the 10th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '01)*, pages 157–162, Cambridge, Massachusetts, June 2001.

[3] S. Barker. Security Policy Specification in Logic. In *Proceedings of the International Conference on Artificial Intelligence*, pages 143–148, Las Vegas, NV, 2000.

[4] S. Barker and A. Rosenthal. Flexible Security Policies in SQL. In *Proceedings of the 15th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Niagara-on-the-Lake, Canada, 2001.

[5] E. Bertino, P. Bonatti, and E. Ferrari. TRBAC: A Temporal Role-Based Access Control Model. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, pages 21–30, Berlin, Germany, 2000.

[6] R. Chandramouli. Application of XML Tools for Enterprise-Wide RBAC Implementation Tasks. In *Proceedings of the 5th ACM workshop on Role-based Access Control*, Berlin, Germany, July 2000.

[7] F. Chen and R. Sandhu. Constraints for Role-Based Access Control. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*, Gaithersburg, MD, 1995.

[8] N. Damianou and N. Dulay. The Ponder Policy Specification Language. In *Proceedings of the Policy Workshop*, Bristol, U.K., 2001.

[9] P. Epstein and R. S. Sandhu. Towards a UML Based Approach to Role Engineering. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, pages 145–152, 1999.

[10] 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.

[11] Geri Georg, Robert France, and Indrakshi Ray. An Aspect-Based Approach to Modeling Security Concerns. In *Proceedings of the Workshop on Critical Systems Development with UML*, Dresden, Germany, 2002.

[12] R. Grimm and B. Bershad. Providing Policy-Neutral and Transparent Access Control in Extensible Systems. Technical Report UW-CSE-98-02-02, University of Washington, 1998.

[13] R. J. Hayton, J. M. Bacon, and K. Moody. Access Control in Open Distributed Environment. In *IEEE Symposium on Security and Privacy*, pages 3–14, Oakland, CA, May 1998.

[14] M. Hitchens and V. Varadarajan. Tower: A Language for Role-Based Access Control. In *Proceedings of the Policy Workshop*, Bristol, U.K., 2001.

[15] J. A. Hoagland, R. Pandey, and K. N. Levitt. Security Policy Specification Using a Graphical Approach. Technical Report CSE-98-3, Computer Science Department, University of California Davis, July 1998.

[16] 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.

[17] J. Jurjens. UMLsec: Extending UML for Secure Systems Development. In *Proceedings of the 5th International Conference on the Unified Modeling Language*, pages 412–425, Dresden, Germany, October 2002.

[18] Dae-Kyoo Kim, Robert France, Sudipto Ghosh, and Eunjee Song. Using Role-Based Modeling Language (RBML) as Precise Characterizations of Model Families. In *Proceedings of the International Conference on Engineering Complex Computing Systems (ICECCS 2002)*, Greenbelt, MD, December 2002. ACM Press.

[19] M. Koch, L. V. Mancini, and F. Parisi Presicce. A Graph Based Formalism for RBAC. *ACM Transactions on Information and System Security*, 5(3):323–365, 2002.

[20] T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of the 5th International Conference on the Unified Modeling Language*, pages 426–441, Dresden, Germany, October 2002.

[21] M. Nyanchama and S. Osborn. The Role Graph Model and Conflict of Interest. *ACM Transactions on Information Systems Security*, 2:3–33, 1999.

[22] OASIS. XACML Language Proposal, Version 0.8. Technical report, Organization for the Advancement of Structured Information Standards, January 2002. Available electronically from http://www.oasis-open.org/committees/xacml.

[23] R. Ortalo. A Flexible Method for Information Systems Security Policy Specification. In *Proceedings of the 5th European Symposium on Research in Computer Security*, Louvain-la-Neuve, Belgium, 1998. Springer-Verlag.

[24] S. Osborn and Y. Guo. Modeling Users in Role-Based Access Control. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, pages 31–37, Berlin, Germany, July 2000.

[25] I. Ray, N. Li, D. Kim, and R. France. Using Parameterized UML to Specify and Compose Access Control Models. In *Proceedings of the 6th IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, Lausanne, Switzerland, November 13-14 2003.

[26] C. Ribeiro, A. Zuquete, and P. Ferreira. SPL: An Access Control Language for Security Policies with Complex Constraints. In *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, February 2001.

[27] J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[28] E. G. Sirer, R. Grimm, A. J. Gregory, N. R. Anderson, and B. N. Bershad. Improving the security, scalability, manageability and performance of system services for network computing. Technical Report UW-CSE-98-09-01, University of Washington, September 1998.

[29] The Object Management Group (OMG). Unified Modeling Language: Superstructure. Version 2.0, Final Adopted Specification, OMG, http://www.omg.org, August 2003.

[30] J. E. 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.

[31] J. Warmer and A. Kleppe. *The Object Constraint Language, Second Edition*. Addison-Wesley, 2003.