5

Modelling Access Control with Dynamic Role Binding

Al-Dahoud[1] Ali and Dr.K.Chitra[2]
[1]*Al-Zaytoonah University, Jordan,* [2]*Department of Computer Applications, Thiagarajar School of Management Madurai, Tamil Nadu, India*
aldahoud@alzaytoonah.edu.jo, chitra@tsm.ac.in

Abstract

To achieve the goal of realizing object adaptation to environments, we model the object with its role using parameterized UML models. An environment is defined as a field of collaboration between roles and an object adapts to the environment assuming one of the roles. Objects can freely enter or leave environments and belong to multiple environments at a time so that dynamic adaptation or evolution of objects is realized. Organizations use Role-Based Access Control to protect computer-based resources from unauthorized access. This paper describes a method for modeling access control for dynamic role binding using parameterized UML design models. Reusable parameterized UML models are specified as patterns and are expressed using UML template diagrams. Developers can use the models to identify their policy violations. The method is illustrated using a small banking application.

Key Words: Parameterized UML, Object adaptation, Evolution, Reusable.

1. Introduction

Objects represent things or concepts of the real world and it is this representation feature that gives the object-oriented technology the high modeling capability. Objects in the real world exist in various environments. If an object permanently resides in a fixed environment, the structure and behavior of the object can possibly stay unchanged over time. However, environments surrounding objects may not be stable due to various reasons. If objects are humans or manufacturing equipment, their environment changes periodically between the day and the night and between the weekdays and the weekend. When an object moves, the surrounding environment naturally changes. Even if an object stays at the same place for a certain period of time, the environment itself may dynamically change. Corresponding to such environmental change, objects adaptively change themselves. Conversely, objects may spontaneously evolve, causing change in their relation to the environment and that in turn may trigger change in the environment. Moreover, there generally exist multiple environments around an object and the object may selectively

belong to a subset of them at a time and the selection of environments may also change dynamically.

How is such adaptation or evolution of objects handled in the world of object-oriented modeling and programming? As many researchers have pointed out, current widely-used object-oriented modeling and programming languages do not conveniently support such flexibility. Motivation of our research is to build a parameterized UML model that is flexible enough to cope with future changes but simple enough to describe and reason about the design validity.

The model to be described in the succeeding sections has the following features.
1. Objects can freely enter or leave environments and belong to multiple environments at a time so that dynamic adaptation or evolution of objects is realized.
2. Environments and roles are the first class constructs at model description time so that separation of concerns is not only materialized as a static structure but also observed as behaviors.
3. Environments are independent reuse components to be deployed separately from objects that participate in them.

2. Access Control for Dynamic Role Binding (DRBAC)

Roles are considered for listing up functions or behaviors of an object to define a clear boundary of the object, thus their granularity is smaller than objects and conceptually comparable to the level of methods. However, the major motivation of this paper is to design access control for object adaptation to environments using parameterized UML. An environment in the context of role model is regarded as a collaboration field and in order to realize adaptation, objects should be allowed to enter collaboration environments by assuming roles and to leave from environments by discarding roles dynamically.

Dynamic Role Binding Access Control (DRBAC) constraints can be organized as follows: Core DRBAC, Hierarchical DRBAC, Static Separation of Duty Relations, and Dynamic Separation of Duty Relations.

The Core DRBAC requires that users (i.e. Objects) be assigned to environment, users be assigned to roles (job function) corresponding to that environment, roles be associated with permissions (approval to perform an operation on a database), and users acquire permissions by being assigned to roles. The Core DRBAC places a constraint on the cardinalities of the user-role assignment relation that when a user enters an environment he is assigned a role when he/she leaves that environment he/she discards that role. The Core DRBAC does not place any constraint on the cardinalities of the permission-role association. Core DRBAC also includes the notion of user environments. A user enters an environment during which he activates a subset of the roles assigned to him. Each user may belong to multiple environments; however, each environment is associated with only one user. The operations that a user can perform in an environment depend on the roles activated in that environment and the permissions associated with those roles.

Hierarchical DRBAC adds features supporting role hierarchies (RH). Hierarchies are used to describe a structure of roles in an organization. Role hierarchies define an inheritance relation among the roles. Role r1 inherits from role r2 only if all permissions of r2 are also permissions of r1 and all users of r1 are also users of r2. The inheritance relationship is reflexive, transitive and anti-symmetric.
Static Separation of Duty (SSD) relations 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, assignment to a role that takes part in an SSD relation prevents the user from being assigned to the related 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 DRBAC), or in the presence of role hierarchies (referred to as hierarchical SSD DRBAC). 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 (DSD) relations aim to prevent conflict of interests as well. The DSD relations place constraints on the roles that can be activated in a user's environment. If one role that takes part in a DSD relation is activated, the user cannot activate the related (conflicting) role in the same session. A model of DRBAC is shown in Fig. 1.

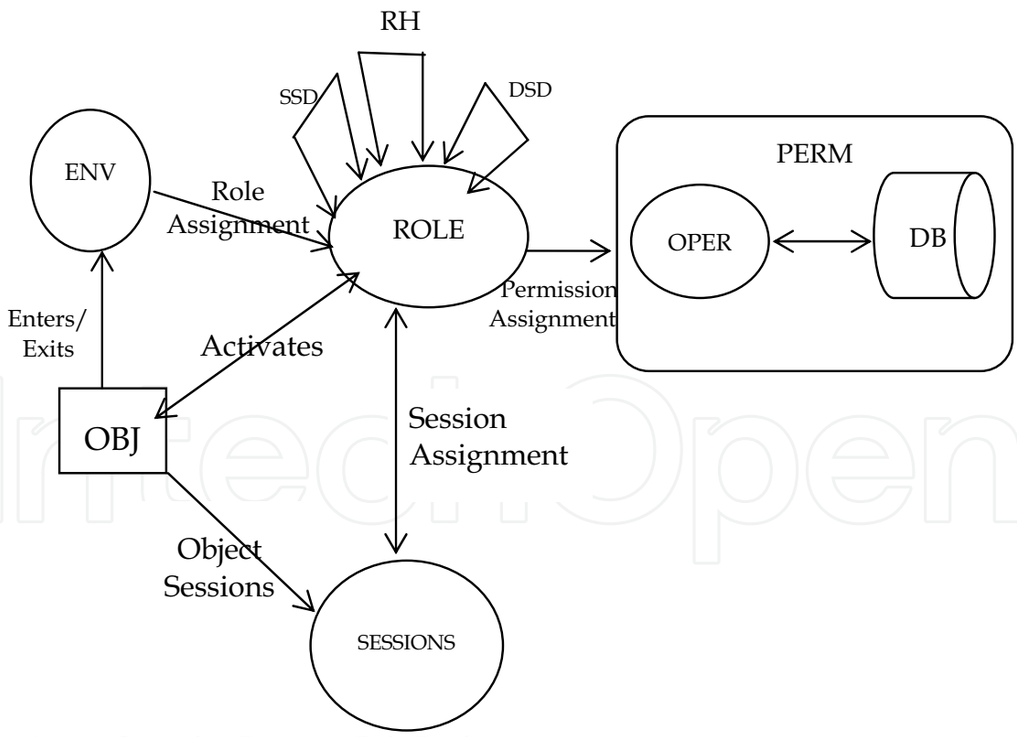


Fig. 1. Access Control in Dynamic Role Binding

The DRBAC in Fig. 1 consists of: 1) a set of objects (OBJ) where an object is an intelligent autonomous agent, 2) a set of environments (ENV) where an object enters or leaves 2) a set

of roles (ROLE) where a role is a job function which is assigned to the object when it enters an environment and is discarded when it exits the environment, 3) the database (DB) where DB is an entity that contains or receives information, 4) a set of operations (OPER) where an operation is an executable image of a program, and 5) a set of permissions (PERM) where a permission is an approval to perform an operation on database. 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 an object to an environment and the association of an object to role are one-to-many. All other associations shown in the figure are many-to-many. The association labeled Role Hierarchy (RH) defines the inheritance relationship among roles. The association labeled SSD specifies the roles that conflict with each other. The association labeled DSD specifies the roles that cannot be activated within an environment by the same user.

3. A Reusable DRBAC Model

In this section a DRBAC pattern is described as a UML template class diagram. A class diagram is obtained from a template diagram by binding the parameters to values. Fig. 2 shows a class diagram template describing hierarchical DRBAC with SSD and DSD. The symbol "|" is used to indicate parameters.

The class diagram template shown in Fig. 2 consists of class and association templates. A class template is a class descriptor with parameters. 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 association-end multiplicity parameters. The multiplicity "1" on the UserSessions association-end attached to Object is strict: a session can only be associated with one user.

The User Object class template defines classes that describe Objects. When a user enters an environment, he is assigned a role (assignRole). Then he creates a new session (createSession), delete a session (deleteSession). When he exits that environment, his role is discarded (deassignRole). 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 operation assignRole creates a RoleAssignment link; the deassignRole removes a RoleAssignment link.

The class template Role is used to produce classes representing roles with behavior that (1) associates a new permission with the role (grantPermission), (2) deletes an existing permission associated with the role (revokePermission), (3) adds an immediate inheriting role (addInheritance), (4) deletes an immediate inheriting role (deleteInheritance), (5) adds a role to the set of conflicting roles (addSSDRole), (6) deletes a role from the existing set of conflicting roles (deleteSSDRole), (7) checks whether the role is in an SSD relationship with a given role in the presence of hierarchies (checkSSD), (8) checks whether the role has a given permission (checkAccess), (9) checks whether the role is in a DSD relation with a given role (checkDSD), (10) deletes a DSD relation between the role and a given role (deleteDSDRole), and (11) adds a DSD relation with a given role (addDSDRole).

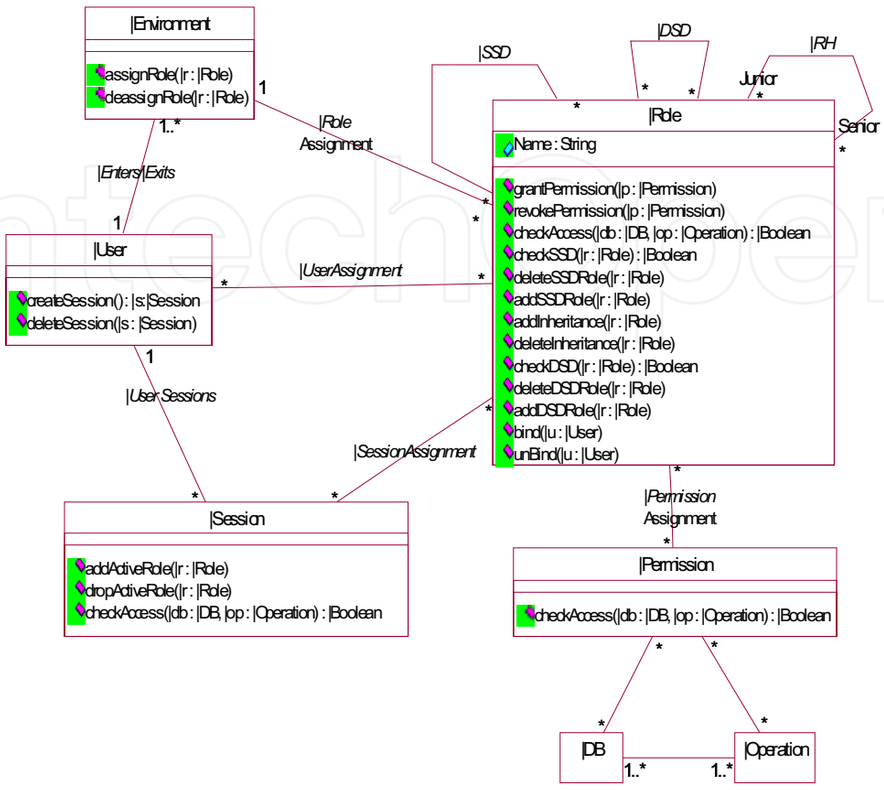


Fig. 2. A DRBAC Class Diagram Template

The class template Session is associated with the template operations: addActiveRole (activates a role in a session), dropActiveRole (deactivates a role in a session), and checkAccess (checks whether the role has the permission to perform an operation on the database).

The class template Permission is associated with an operation template, checkAccess, that checks whether the role has the permission to perform the operation on the database. 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:

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

context |Role:: |grantPermission (|p: |Permission)
post: self.| Permission includes(|p)

We express DRBAC constraints that restrict SSD and DSD relationships as OCL template expressions. Examples of these constraints are given below:

– SSD constraint. A user cannot be assigned to two roles that are involved in an SSD relation.

```
context |User inv:
self. |Role → forAll(r1, r2 | r1.| SSD → excludes(r2))
```

– Hierarchical SSD constraint. There cannot be roles in an SSD relation which have the same senior role.

```
context _ Role inv:
let allSenior(r1) = r1.senior -> union(r1.senior -> collect(r2 | allSenior(r2)))
in
self. |SSD -> forAll(r1 | allSenior(r1) -> excludesAll(allSenior(self))
```

– DSD constraint. A user cannot activate two roles in DSD relation within a session.

```
context | User inv:
|self. |Session. |Activates -> forAll(r1, r2 |r1. |DSD -> excludes(r2))
```

4. DRBAC Model on Banking Application

To illustrate this approach we use a simple banking application taken from [5]. 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. The transactions include 1) create, delete, or modify customer deposit accounts, 2) create, delete, or modify customer loan accounts, 3) modify the ledger posting rules, and 4) create general ledger report. When a user enters Bank environment, he is assigned a BANKROLE when he exits bank environment he is assigned another role according to the environment he enters and assigned permissions depending on the role he is assigned. Fig. 3 shows DRBAC class diagram for the bank environment.

5. DRBAC Policies applied using Object Diagrams

DRBAC policies when applied to a role in an environment constrain how system users access system resources. They determine 1) the assignment of roles to system users, 2) the permissions associated with roles in the environment, 3) the inheritance relationships between roles, and 4) the SSD and DSD relationships between roles. In this section we illustrate how DRBAC policies can be described by object diagrams when the user is assigned a ROLE.

The DRBAC model supports the specification of four types of policies: 1) core policies that conform to core DRBAC, that is, policies that determine user-role and role-permission assignments, 2) hierarchical policies that conform to hierarchical DRBAC, that is, policies that determine inheritance relationships between roles, 3) SSD policies that conform to SSD DRBAC, that is, policies that determine what roles are conflicting, and 4) DSD policies that conform to DSD DRBAC, that is, policies that determine what roles to be activated in a session. A set of DRBAC policies for the banking system is given below:

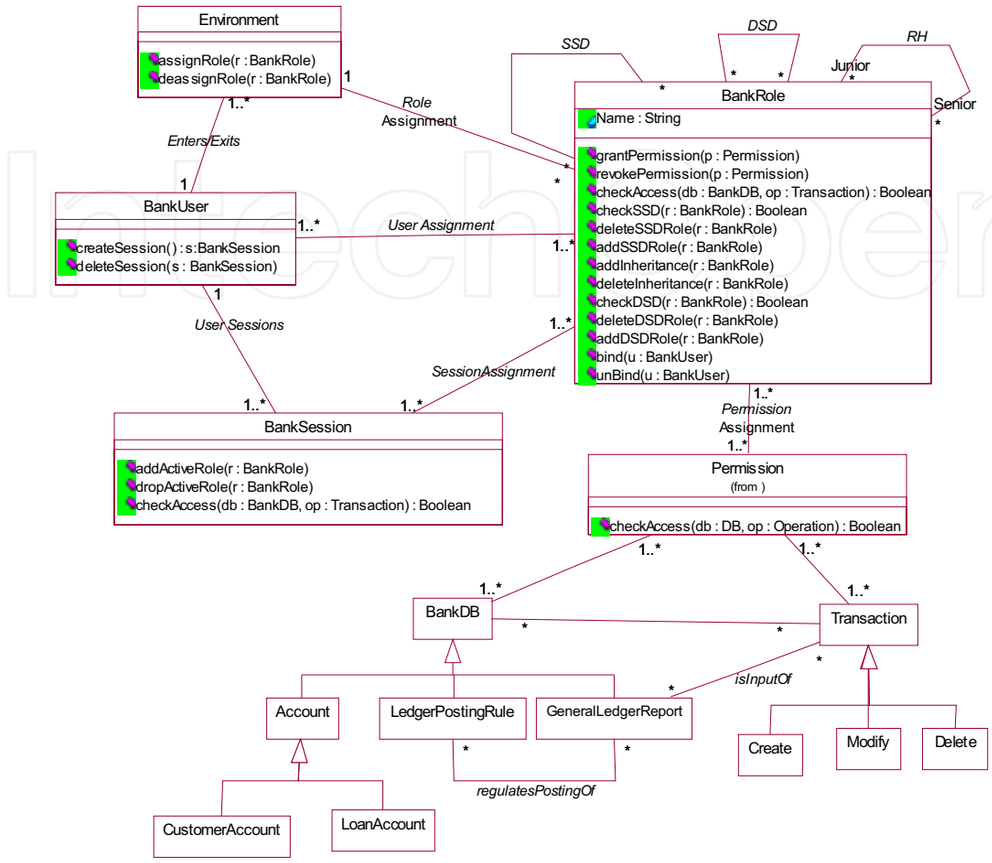


Fig. 3. A DRBAC Class Diagram for a Banking Application

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.

Fig. 4. Shows the object diagrams describing policies P1 to P5 respectively.

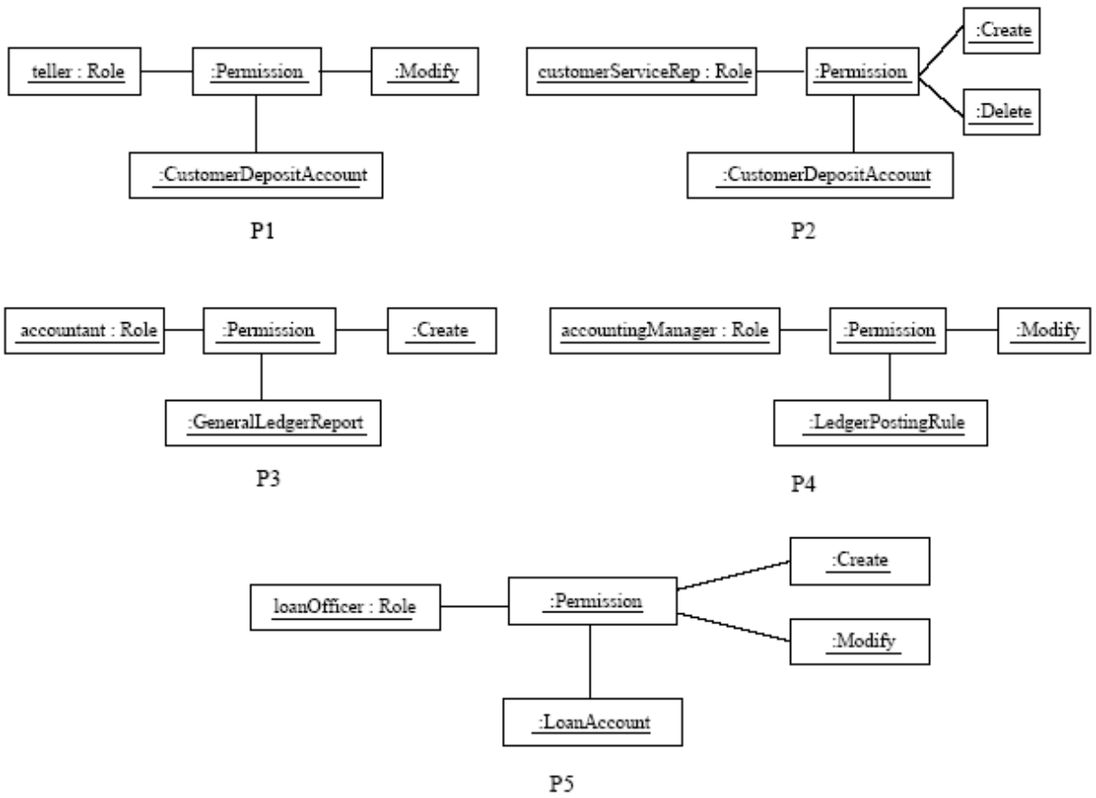


Fig. 4. Object Diagrams for Policies P1 to P5

Hierarchical policies: 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 hierarchical policies in the banking application are stated below:
H1 Customer service representative role is senior to the teller role.
H2 Accounting manager role is senior to the accountant role.
Fig. 5(a),(b) describe policies H1 and H2 respectively.

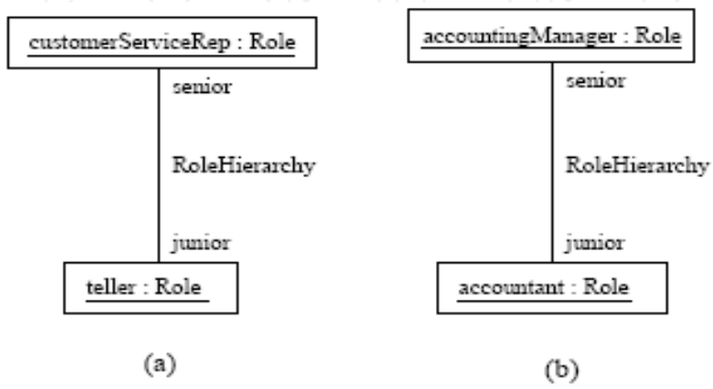


Fig. 5(a), (b) Object Diagrams for Policies H1 and H2

SSD policies: SSD policies prevent a user from being assigned to two conflicting roles. For the banking system the following pairs of roles are conflicting:
{(teller, accountant), (teller, loanOfficer),
(loanOfficer, accountant), (loanOfficer, accountingManager),
(customerServiceRep, accountingManager)}
The object diagram in Fig. 6 describes the SSD RBAC policies.

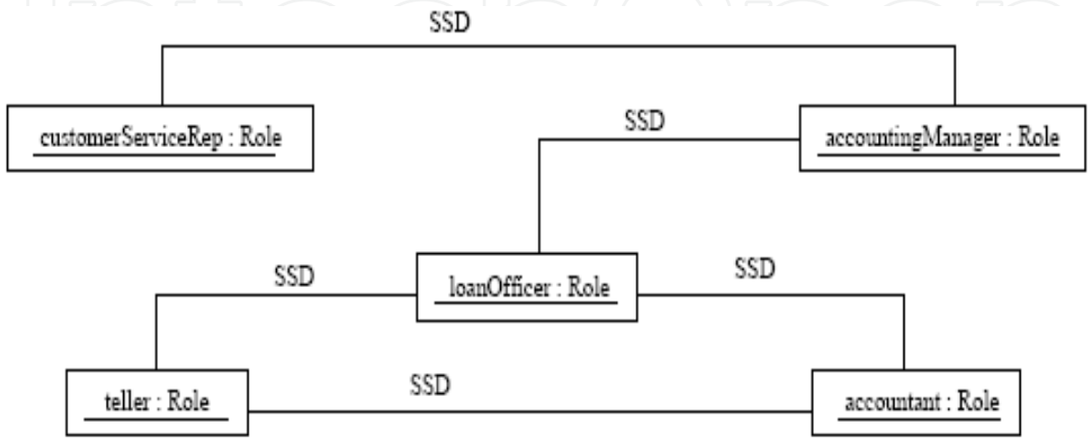


Fig. 6. Object Diagram for SSD Policies

DSD policies: DSD policies prevent a user from playing a role in a session, if another role in a DSD relation has been activated. For the banking system the following pair of roles are in DSD relation:
{(customerServiceRep, loanOfficer)} The object diagram in Fig. 7 describes the DSD DRBAC policy.

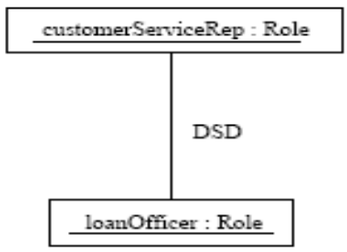


Fig. 7. Object Diagram for DSD Policy

6. Identifying Conflicts in Application-Specific DRBAC Policies

In this section we show how DRBAC violation patterns expressed as object diagram templates can be used to identify conflicts. If a violation pattern exists in an object diagram describing a policy, then a conflict exists. Fig. 8 shows object diagram templates that when instantiated produce object structures that violate DRBAC constraints. Fig. 8(a) describes structures in which a user is assigned to roles in an SSD relationship (violation of the SSD constraint). Fig. 8(b) describes structures in which two roles in an SSD relationship have a

common senior role and structures in which a senior role is in an SSD relationship with a junior role (both are violations of the hierarchical SSD constraint). Fig. 8(c) describes structures in which a user in a session activates two roles that are in a DSD relationship (a violation of the DSD constraint). 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.

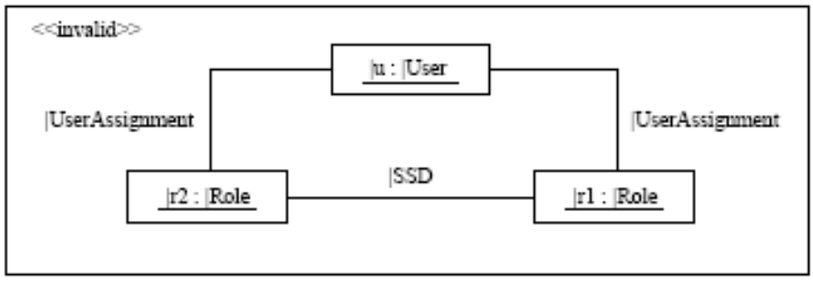


Fig. 8. (a) Violation of SSD Constraint

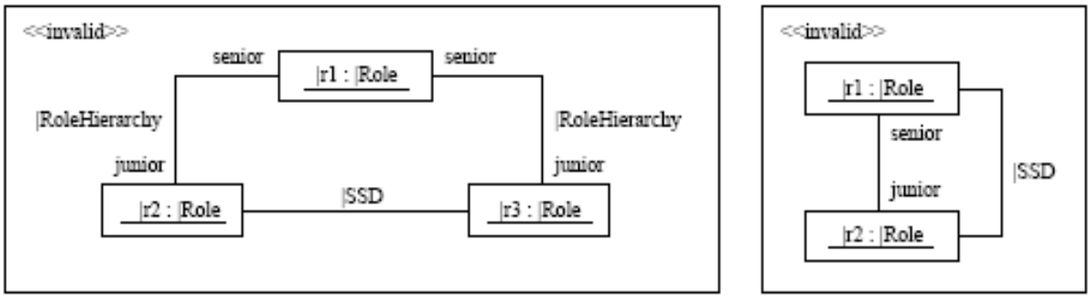


Fig. 8. (b) Violations of Hierarchial SSD Constraint

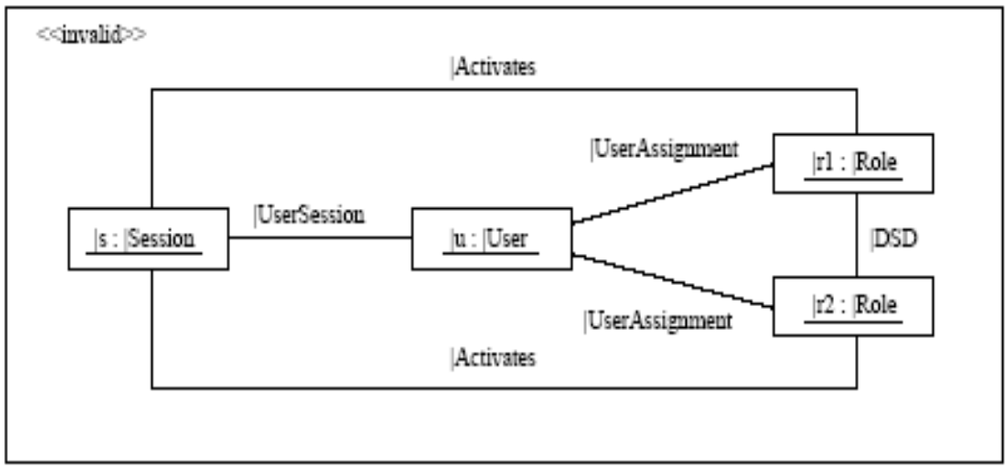


Fig. 8. (c) Violation of DSD Constraint

7. Related Work

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

Aspect-oriented programming with AspectJ has a feature of adding aspects dynamically as well as statically [14]. The main objective of writing aspects is to deal with cross-cutting concerns. It implies that there already exists some structure of module decomposition but in adding a new type of concern, related pieces of code are distributed among modules, cross-cutting the existing structure.

Although there have been efforts of designing software from the beginning based on the AOP method under the name of "early aspects"[20], the normal framework of mind for thinking aspects assumes the existing program code as a target of inserting advices to join points.

A large volume of research (e.g., see [2–4, 6, 7, 11, 12, 14] exists in the area of access control policy specification. Formal logic-based techniques (e.g., see [2–4, 6, 11, 14]) are often used to specify security policies. The use of mathematical concepts and notation that are not familiar to software developers makes them difficult to use and understand. Other researchers have used high-level languages to specify policies [12, 13, 19, 20]. Although high-level languages are easier to understand than formal logic-based approaches, they are not analyzable.

Some work has been done on modeling system security using UML. Jurjens [15] proposes UMLsec, a UML profile for modeling and evaluating security aspects based on the multi-level security model. Lodderstedt et al. propose SecureUML [17], an extension of the UML that defines security concepts. These approaches mainly focus on extending the UML notation to better reflect security concerns.

8. Conclusion

The work described in this paper focuses on specifying the dynamic role binding and access control when a role is assigned to an Object. 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. The approach of binding objects and roles have the following characteristics: Composition takes place when an object instance and a role instance are bound together; an object instance can be bound to multiple role instances residing in different environments; when an object enters an environment it is assigned a role, when it exits that environment the associated role is discarded by the object.

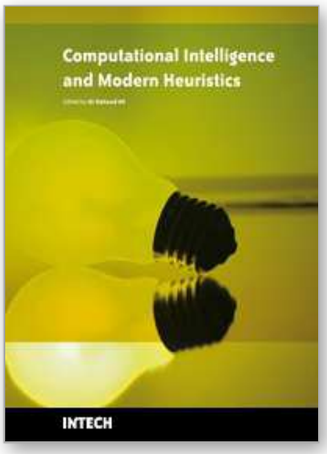
9. References

1. G.J. Ahn and R. Sandhu. Role-based Authorization Constraints Specification. ACM Transactions on Information and Systems Security, 3(4):207–226, November 2000.
2. S. Barker. Security Policy Specification in Logic. In Proceedings of the International Conference on Artificial Intelligence, pages 143–148, Las Vegas, NV, 2000.

3. 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-onthe-Lake, Canada, 2001.
4. 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.
5. R. Chandramouli. Application of XML Tools for Enterprise-Wide RBAC Implementation Tasks. In Proceedings of 5th ACM workshop on Role-based Access Control, Berlin, Germany, July 2000.
6. 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.
7. N. Damianou and N. Dulay. The Ponder Policy Specification Language. In Proceedings of the Policy Workshop, Bristol, U.K., 2001.
8. 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.
9. 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.
10. Geri Georg, Indrakshi Ray, and Robert France. Using Aspects to Design a Secure System. In Proceedings of the Interational Conference on Engineering Complex Computing Systems (ICECCS 2002), Greenbelt, MD, December 2002. ACM Press.
11. 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.
12. M. Hitchens and V. Varadarajan. Tower: A Language for Role-Based Access Control. In Proceedings of the Policy Workshop, Bristol, U.K., 2001.
13. 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.
14. 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.
15. J. Jurjens. UMLsec: Extending UML for Secure Systems Development. In Proceedings of Fifth International Conference on the Unified Modeling Language, pp. 412-425, pages 412–425, Dresden, Germany, October 2002.
16. 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.
17. T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In Proceedings of Fifth International Conference on the Unified Modeling Language, pages 426–441, Dresden, Germany, October 2002.
18. B.T. Messmer and H. Bunke. Subgraph Isomorphism in Polynomial Time. In Lecture Notes in Computer Science Graph Theory - ECCV'98, Springer-Verlag, Berlin, 1998.

19. 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.
20. 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.
21. J. E. Tidswell and T. Jaeger. An Access Control Model for Simplifying Constraint Expression. In Proceedings of 7th ACM conference on Computer and communications security, pages 154–163, Athens, Greese, November 2000.
22. G. B. andWilliam Cook. Mixin-based inheritance. In OOPSLA 1990, pages 303–311, 1990.
23. D. Bardou and C. Dony. Split objects: a disciplined use of delegation within objects. In OOPSLA '96, pages 122–137, San Jose, California, USA, Oct. 1996.
24.K.Beck and W. Cunningham. A laboratory for teaching object-oriented thinking. In OOPSLA '89, pages 1–6, 1989.

IntechOpen



Computational Intelligence and Modern Heuristics

Edited by Al-Dahoud Ali

ISBN 978-953-7619-28-2
Hard cover, 348 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

The chapters of this book are collected mainly from the best selected papers that have been published in the 4th International conference on Information Technology ICIT 2009, that has been held in Al-Zaytoonah University, Jordan in the period 3-5/6/2009. The other chapters have been collected as related works to the topics of the book.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Al-Dahoud Ali and K.Chitra (2010). Modelling Access Control with Dynamic Role Binding, Computational Intelligence and Modern Heuristics, Al-Dahoud Ali (Ed.), ISBN: 978-953-7619-28-2, InTech, Available from: http://www.intechopen.com/books/computational-intelligence-and-modern-heuristics/modelling-access-control-with-dynamic-role-binding

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen