Available online at www.sciencedirect.com
SCIENCE @ DIRECT®

Information and Software Technology 46 (2004) 575–587

INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

An aspect-based approach to modeling access control concerns

Indrakshi Ray, Robert France*, Na Li, Geri Georg

Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA

Received 25 April 2003

Abstract

Specifying, enforcing and evolving access control policies is essential to prevent security breaches and unavailability of resources. These access control design concerns impose requirements that allow only authorized users to access protected computer-based resources. Addressing these concerns in a design results in the spreading of access control functionality across several design modules. The pervasive nature of access control functionality makes it difficult to evolve, analyze, and enforce access control policies. To tackle this problem, we propose using an *aspect-oriented modeling*(AOM) approach for addressing access control concerns. In the AOM approach, functionality that addresses a pervasive access control concern is localized in an *aspect*. Other functional design concerns are addressed in a model of the application referred to as a *primary model*. Composing access control aspects with a primary model results in an application model that addresses access control concerns. We illustrate our approach using a form of Role-Based Access Control.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Access control policies; Role-based access control; Aspects; Models

1. Introduction

Access control policies are constraints that determine the type of access authorized users have to protected computer-based resources. For example, an access control policy in a banking system can stipulate that only loan managers can create and update customer loan accounts. Incorrect specification or enforcement of this policy may either prevent loan managers from creating or updating customer loan accounts or may allow unauthorized persons from update customer loan accounts. Both situations are problematic and may have serious consequences.

There is a growing awareness that security concerns need to be addressed throughout the software development cycle [19]. From a design perspective, access control policies give rise to security concerns that must be addressed in a design. Researchers [32,54,60] recommend specifying and maintaining security policies separately from the application design. 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. Independent specification of security policies presents a problem—how to integrate the policies in an application design.

Addressing access control concerns in a design of a secure system results in the spreading of access control functionality across the modules of the design. The interference of access control functionality with other application behavior can make it difficult to understand, analyze, and evolve access control functionality. Using an ad hoc approach to address pervasive access control concerns for a complex application is likely to produce a design with errors arising from the inconsistent way in which the concerns are addressed across affected design modules [27]. These errors can lead to security breaches or failures.

In this paper, we propose the use of *aspect-oriented modeling* (AOM) techniques to support systematic approaches to addressing access control concerns in a design. Aspect-oriented techniques tackle system complexity by providing support for localizing pervasive functionality in *aspects*. An AOM design model consists of aspects and a *primary model* that addresses other functional design concerns. Aspects are composed with the primary model to produce a design model that integrates the design elements

* Corresponding author. Tel.: +1-970-491-6356; fax: +1-970-491-2466.
E-mail addresses: france@cs.colostate.edu (R. France); iray@cs.colostate.edu (I. Ray); na@cs.colostate.edu (N. Li); georg@cs.colostate.edu (G. Georg).

0950-5849/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2003.10.007

in the aspects with the design elements in the primary model. This composition is called *weaving*.

In the AOM approach, aspects are patterns and thus the access control aspects can be potentially reused across different applications with similar access control requirements. The use of patterns also helps ensure that pervasive access control concerns are uniformly addressed across a design. Localizing access control concerns in aspects also facilitates evolution—changes to access control concerns (e.g. changes to security policies) can be reflected in the corresponding aspects, and the effect can be incorporated in the primary application model through weaving. The woven model produced by weaving access control aspects with a primary model can be analyzed to check whether the concerns have been adequately addressed in the design [27]. Localizing pervasive access control functionality in aspects can ease the tasks of specifying, understanding, analyzing, uniformly applying, and evolving the access control functionality.

This paper focuses only on specifying access control concerns as aspects and weaving of aspects with primary models. Analysis of the woven models are beyond the scope of this paper. The rest of the paper is organized as follows. Section 2 describes the approach to modeling access control functionality as aspects. Section 3 shows how access control aspects can be composed using primary models. Section 4 illustrates the approach using an example application. Section 5 gives an overview of related work and Section 6 concludes the paper with pointers to future directions.

2. Modeling pervasive access control functionality

In this paper, an access control aspect is represented by a pattern of structures and behaviors that reflect constraints defined by an access control policy. An access control aspect is modeled from two perspectives: the *structural* perspective and the *dynamic* perspective. The structural perspective identifies the entities constrained by the access control policy and their relationships with each other. The dynamic perspective defines the constraints that the access control policy imposes on behaviors. Our approach to describing access control aspects utilizes the *Unified Modeling Language* (UML) [63]. An access control aspect consists of (1) template forms of UML static structural diagrams (e.g. class diagrams) that provide the static structural perspective, and (2) template forms of UML dynamic diagrams (e.g. interaction diagrams) that describe the dynamic perspective.

An overview of the AOM approach proposed in this paper is shown in Fig. 1. An aspect-oriented design model consists of aspects and a primary model. In this paper, aspects are patterns describing generic access control structures and behavior. Integration of an aspect with a primary model involves (1) identifying the parts of the primary model that require access control, (2) instantiating the aspects to obtain design views, called *context-specific aspects*, that define how access control will be accomplished in the specified parts, and (3) composing the context-specific aspects with the primary model. In Fig. 1, the access control aspect is integrated with three parts of

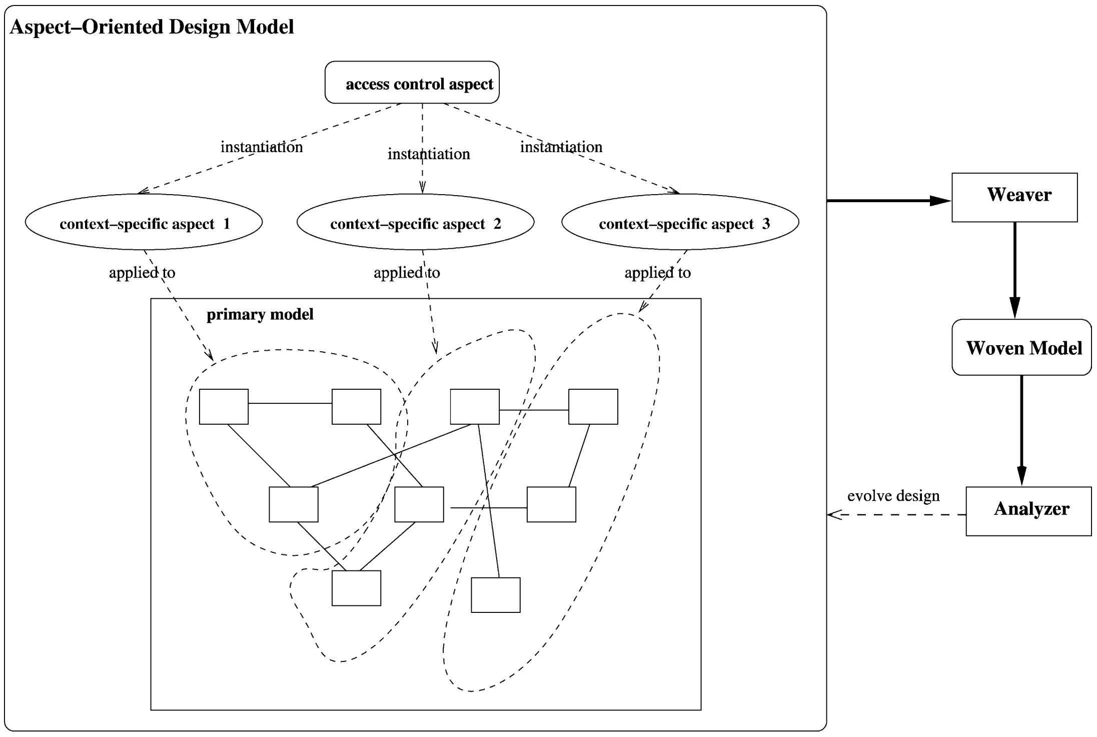

Fig. 1. An Overview of the AOM approach to modeling access control.

the primary model. The woven model produced by composing (weaving) the context-specific aspects and the primary model is analyzed to uncover emergent behaviors that result in violations to access control policies. Analysis can also focus on identifying undesirable interactions between access control functionality and other behaviors described in the primary model. Problems uncovered during analysis initiate changes to the design as indicated by the dashed arrow between the analyzer and the design model.

2.1. Modeling access control aspects as patterns

The structural perspective of an aspect is modeled by a template form of UML static structural diagram called a *Static Structure Template* (SST). The dynamic perspective is described by template forms of UML interaction diagrams called *Interaction Structure Templates* (ISTs). SSTs and ISTs are based on a pattern specification language described in previously published papers [22–24].

Role-Based Access Control (RBAC), an access control policy framework, is used to illustrate UML-based formulation of an access control aspect. Depending on the nature of the application, the RBAC will use one or more of the following components: *Core RBAC*, *Hierarchical RBAC*, *Static Separation of Duty (SSD) Relations*, and *Dynamic Separation of Duty Relations*. In this paper, we use the hierarchical SSD RBAC model which consists of the Core RBAC, hierarchical RBAC, and SSD relations. We describe these components below.

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

Hierarchical RBAC adds constraints to Core RBAC for supporting role hierarchies. Hierarchies help in structuring the roles of an organization. Role hierarchies define an inheritance relation among the roles in terms of permissions and user assignments. In other words, role $r1$ inherits role $r2$ only if all permissions of $r2$ are also permissions of $r1$ and all users of $r1$ are also users of $r2$. There are no cardinality constraints on the inheritance relationship. The inheritance relationship is reflexive, transitive and anti-symmetric.

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 relations place constraints on the assignment of users to roles, that is, membership in one role that takes part in an SSD relation prevents the user from being a member of the other conflicting role. The SSD relationship is symmetric, but it is neither reflexive nor transitive. SSD may exist in the absence of role hierarchies (referred to as SSD RBAC), or in the presence of role hierarchies (referred to as hierarchical SSD RBAC). The presence of role hierarchies complicates the enforcement of 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.

The model of hierarchical SSD RBAC shown in Fig. 2 consists of: (1) a set of users (*USERS*) where a user is an intelligent autonomous agent, (2) a set of roles (*ROLES*) where a role is a job function, (3) a set of objects (*OBS*) where an object is an entity that contains or receives information, (4) a set of operations (*OPS*) where an operation performs tasks, and (5) a set of permissions (*PRMS*) where a permission is an approval to perform operations on objects. The cardinalities of the relationships are indicated by the absence (denoting one)

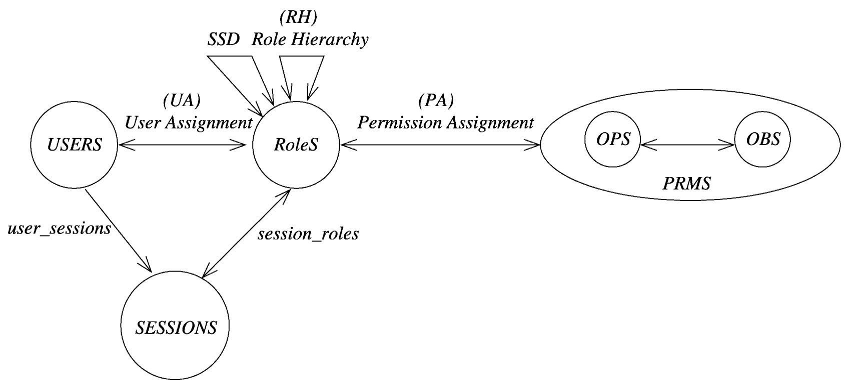

Fig. 2. Hierarchical SSD RBAC.

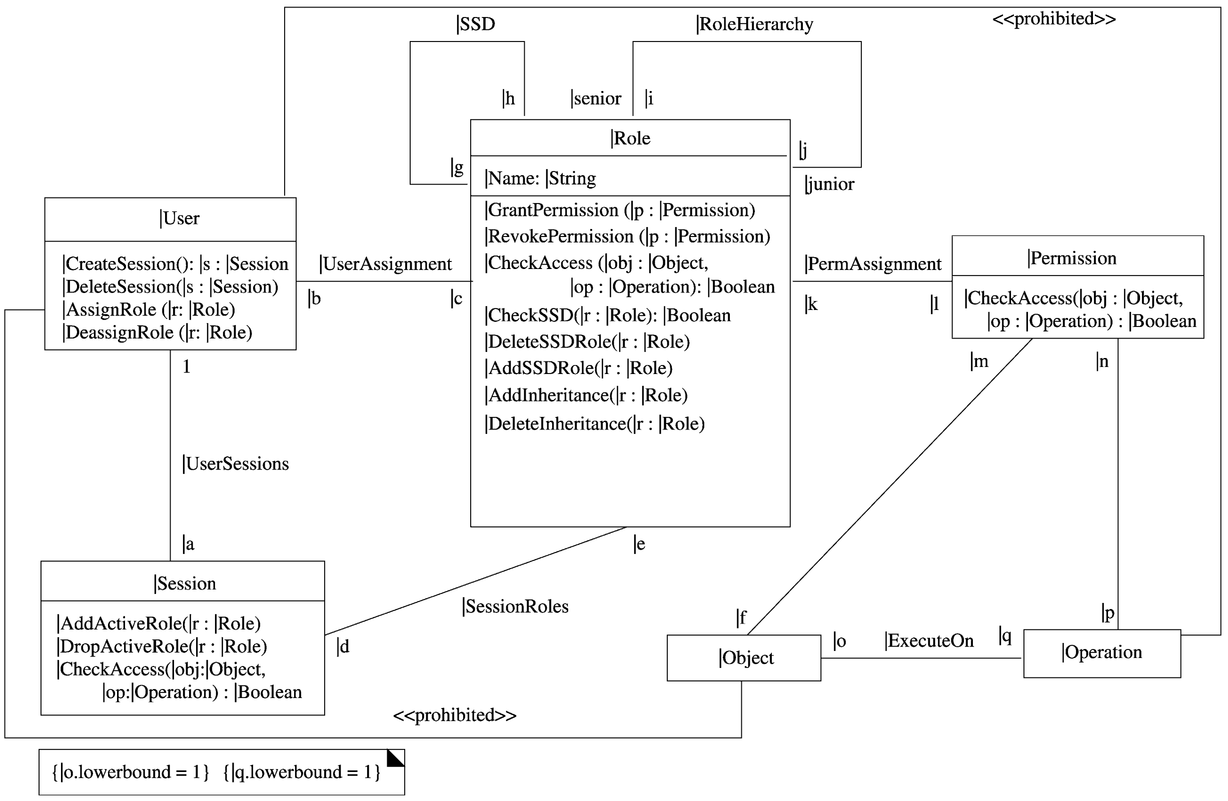
|SSD
|RoleHierarchy
<<prohibited>>
|h
|senior
|i
|Role
|j
|g
|Name: |String
|junior
|User
|GrantPermission (|p : |Permission)
|RevokePermission (|p : |Permission)
|CheckAccess (|obj : |Object,
|op : |Operation): |Boolean
|CheckSSD(|r : |Role): |Boolean
|DeleteSSDRole(|r : |Role)
|AddSSDRole(|r : |Role)
|AddInheritance(|r : |Role)
|DeleteInheritance(|r : |Role)
|CreateSession(): |s : |Session
|DeleteSession(|s : |Session)
|AssignRole (|r: |Role)
|DeassignRole (|r: |Role)
|UserAssignment
|b
|c
|PermAssignment
|k
|l
|Permission
|CheckAccess(|obj : |Object,
|op : |Operation) : |Boolean
1
|m
|n
|UserSessions
|a
|e
|Session
|AddActiveRole(|r : |Role)
|DropActiveRole(|r : |Role)
|CheckAccess(|obj:|Object,
|op:|Operation) : |Boolean
|SessionRoles
|d
|f
|Object
|o
|ExecuteOn
|q
|p
|Operation
<<prohibited>>
{|o.lowerbound = 1} {|q.lowerbound = 1}

Fig. 3. SST for the hierarchical SSD RBAC model.

or presence of arrows (denoting many) on the corresponding associations. For example, the user to session association is one-to-many, indicating that a user can be linked to zero or more sessions at any given time. All other associations shown in the figure are many-to-many. The association labeled *Role Hierarchy* defines the inheritance relationship among roles. The association labeled *SSD* specifies the roles that conflict with each other.

2.1.1. RBAC aspects: structural perspective

An SST consists of template forms of UML classifiers (e.g. classes) and relationships (e.g. associations). Template classifiers are parameterized entities, where the parameters can represent classifier names or other classifier properties. Instantiating a classifier template involves binding parameters to values. The result is an UML classifier (for example, a class or an interface). Class templates can also be associated with attribute templates and operation templates. Instantiating these templates produces attributes and operations. Template relationships (e.g. association templates) are parameterized relationships, where parameters can represent multiplicity ranges, and relationship names.

An SST for hierarchical SSD RBAC is shown in Fig. 3. Template parameters are preceded by |. Binding parameters to values representing UML model elements produces a UML class diagram referred to as a context-specific access control class diagram.

The following describes the behavior of the operations produced by the operation templates associated with the |*User* role.

- |*CreateSession*. Creates a new session and establishes a new |*UserSessions* link.[1]
- |*DeleteSession*. Deletes an existing session and removes the corresponding |*UserSessions* link.
- |*AssignRole*. Assigns a new role to the user by creating a new |*UserAssignment* link.
- |*DeassignRole*. Removes an existing role from the user by deleting the corresponding |*UserAssignment* link.
- |*GetRoles*. Returns the set of roles assigned to the user.

The behavior of operations produced by operation templates in |*Role* are given below.

- |*GrantPermission*. Gives the role a new permission by creating a new |*PermAssignment* link.
- |*RevokePermission*. Deletes an existing permission by deleting a |*PermAssignment* link.
- |*AddInheritance*. Adds an immediate inheritance by creating a |*RoleHierarchy* link.
- |*DeleteInheritance*. Deletes an immediate inheritance by deleting a |*RoleHierarchy* link.

[1] A |*UserSessions* link is an instance of an association produced by instantiating the |*UserSessions* association template.

- |*AddSSDRole*. Adds a role to the set of conflicting roles by creating a |*SSD* link.
- |*DeleteSSDRole*. Deletes a role from the existing set of conflicting roles by deleting a |*DeleteSSDRole* link.
- |*CheckSSD*. Checks whether the role is in an SSD relationship (direct or indirect) with another role.
- |*CheckAccess*. Checks whether a role has some permission.

The behavior of operations produced by operation templates in |*Session* are given below.

- |*AddActiveRole*. Activates a role in a session by creating a |*SessionRoles* link.
- |*DropActiveRole*. Deactivates a role in a session by deleting a |*SessionRoles* link.
- |*CheckAccess*. Determines whether the session has the permission to perform a given operation on a given object.

The class template |*Permission* is associated with only one operation template: a |*CheckAccess* operation checks if the permission allows the given action on the given object.

The association template between class templates |*User* and |*Object* is marked with the stereotype 《*prohibited*》. This indicates that direct associations between corresponding classes are not allowed in any class diagram generated from this template. In other words, a user should not be allowed direct access to the objects. A similar association template also exists between |*User* and |*Operation*.

Operation templates can be associated with constraint templates that produce pre- and post-condition specifications when instantiated. For example, the |*AssignRole* operation template is associated with the following constraint template:

```
context |User :: |AssignRole(r:|Role)
pre: self.|Role → excludes(r)
post: self.|Role → includes(r)
```

The template is used to generate OCL constraints in which the pre-condition holds true if the role *r* is not currently linked to the user and the post-condition is true if the role *r* is linked to the user. To obtain a constraint from the constraint template, one binds values to the template parameters. Section 4.2.1 gives an example of how constraint templates can be instantiated to produce OCL constraints.

Generic RBAC constraints can be expressed as constraint templates. Two examples are given below.

```
The set of roles activated by a user in a
session is a subset of roles assigned to
the user.
context |User
self.|Role →
includesAll(self.|Session.|Role)
```

```
The hierarchical SSD RBAC constraint
prohibits two roles in an SSD relation
from having the same senior role.
context |Role inv
let allSenior(r1) = r1.senior → union
(r1.senior → forAll(r2|allSenior(r2)))
in
self.|SSD → forAll(r1|allSenior(r1)) →
excludesAll(allSenior(self))
```

Constraint templates are also used to constrain the ranges that can be bound to multiplicity parameters of association templates. For example, the constraint template |*u.lowerbound* = 1 restricts ranges that are bound to *u* to have a lower bound of 1.

Given a primary model, an SST can be instantiated to produce a context-specific class diagram that describes the design structure used to address the access control concern in a part of the primary class diagram. In Section 4, we illustrate how this is done for a simple banking application.

2.1.2. Modeling aspects: dynamic perspective

The interaction view describes the pattern of interactions that take place and is expressed as a set of ISTs. Each IST describes a pattern characterizing a family of scenarios.

We show only a few of the ISTs for RBAC to illustrate the form of ISTs. The ISTs shown in Fig. 4 describe patterns of interactions that take place when activating and deactivating RBAC roles.

An IST consists of template forms of UML (interaction) participants and template forms of messages. Message templates have parameters representing sequence

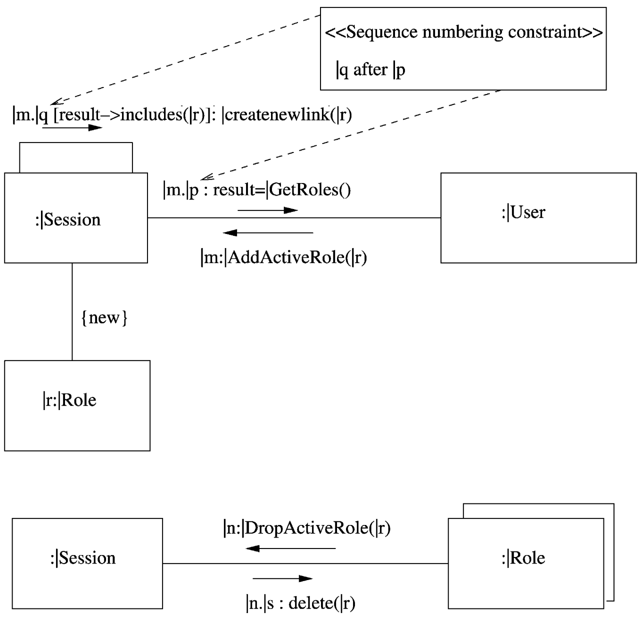

Fig. 4. |*AddActiveRole* and |*DropActiveRole* ISTs.

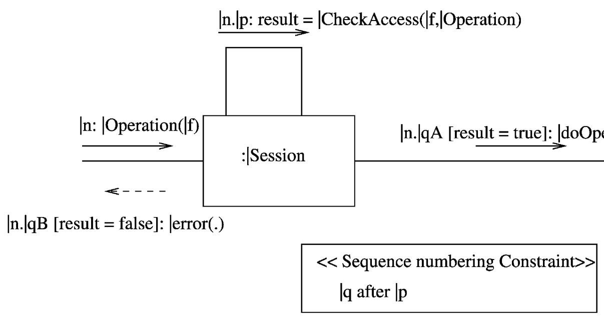
|n.|p: result = |CheckAccess(|f,|Operation)
|n: |Operation(|f)
:|Session
|n.|qA [result = true]: |doOperation()
|f:|Object
|n.|qB [result = false]: |error(.)
<< Sequence numbering Constraint>>
|q after |p

Fig. 5. |*Operation* IST.

expressions and operation calls. For example, |*m*.|*p:result* := |*GetRoles*( ) is a message template that produces the UML message 1.3: *result* := *GetMRoles*( ) when 1 is bound to |*m*, 3 to |*p* and *GetMRoles*( ) to |*GetRoles*( ).

The invocation of an |*AddActiveRole* operation causes the invocation of a |*GetRoles* operation on a |*User* participant. If the user indeed possesses this role, then a new link is created between the corresponding |*Session* and |*r* participants (indicated by the |*createnewlink* operation call parameter). The constraint '|*q* after |*p*' specifies that the |*GetRoles*( ) operation call occurs before the |*createnewlink*(|*r*) operation call. The invocation of a |*DropActiveRole* operation causes the link between a |*Session* and a |*r* participant to be deleted.

Fig. 5 shows the IST corresponding to a user invoking an operation in a |*Session* participant on a specified |*Object* participant. This causes the execution of a |*CheckAccess* operation. A |*CheckAccess* operation returns a boolean value. If the operation returns true, then the specified operation (|*doOperation*) is performed on the |*Object* participant; otherwise, an error message is returned. These two possibilities are indicated by message templates with sequence expressions |*n*.|*qA* and |*n*.|*qB* (the *A* and *B* indicate alternative branches).

Examples of interaction diagrams obtained by instantiating the templates shown in Figs. 4 and 5 are given in Section 4.

3. Composing aspects with primary models

Composing a context-specific aspect with a primary model involves merging the views of the entities and behaviors defined by the aspect with the entity and behavior views defined by the primary model. Composition can involve modifying existing elements in the primary model, adding new model elements to the primary model, and deleting primary model elements. The result of weaving is a new model (the woven model) in which information localized by the aspect is distributed across specified parts of the primary model.

In the proposed approach, composing an aspect with a primary model involves the following activities:

1. *Instantiating the aspect to obtain a context-specific aspect*. Before an aspect can be composed with a primary model, the modeler must first obtain context-specific aspects by binding values to the parameters in the aspect. The values bound to parameters are determined by the locations in the primary model in which access control is desired. The values that are bound to parameters can be model elements in the primary model or can be new design elements that are to be included in the woven model.
2. *Composing context-specific aspects with the primary model*. The views described by context-specific aspects are merged with those described by the primary model to obtain a woven model. Elements in the aspect and primary models are merged if and only if they have the same name and are instances of the same metamodel class (i.e. they have the same syntactic types). A model element in an aspect that does not have a matching element in the primary model represents a new model element that is added to the woven model. *Composition directives* can also be provided by the modeler to influence how the composition is carried out. An example of a composition directive is a dominance relationship between matching elements with different behaviors (e.g. as defined by operation specifications) that indicates that one element replaces the other in the woven model. Such directives can be used to resolve conflicts during composition. Merging the class diagram view of a context-specific aspect and a primary model proceeds as follows (all references to 'aspect' in the following refers to 'context-specific aspect')
   - All primary model elements that have the same name and type as a prohibited element in the aspect are absent from the composed model. A prohibited element is one that is marked with the stereotype 《*prohibited*》.
   - If the matching elements are classifiers, then the classifier features are merged. If a classifier feature (e.g. an attribute or operation) in the aspect does not match any features in the matching primary model classifier, then the feature is added to the classifier in the woven model.

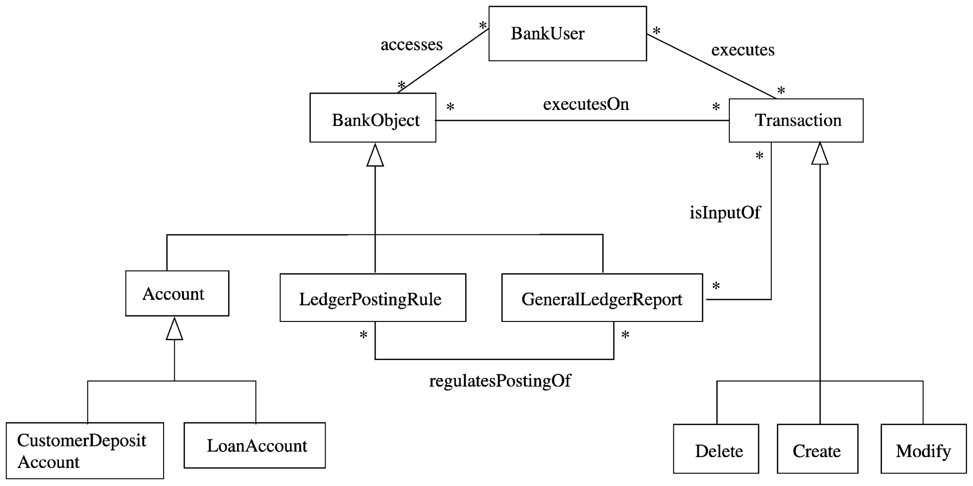

Fig. 6. Class diagram of a banking system (primary model).

- An operation in the aspect model matches an operation in the primary model if their names and signatures match. For matching operations, the modeler should indicate to the weaver, using a composition directive, how the pre- and post-conditions are to be combined (how this is indicated to the weaver is outside the scope of this paper). If this directive is not provided by the modeler, then the default is to combine the preconditions of the matching operations using the logical *or* and combine the post-conditions using the logical *and*.
- An attribute in the aspect model matches an attribute in the primary model if their names and types match. If the matching attributes are associated with constraints, then the modeler should indicate to the weaver, using a composition directive, how constraints are to be combined. The default is to connect the constraints using a logical *and*.
- A relationship in an aspect matches a relationship in the primary model if it has the same name. If matching associations have different multiplicities or role names, then the modeler must indicate to the weaver, using a composition directive, which view should dominate.

Merging of interaction views involves (1) matching aspect participants with primary model participants, (2) including aspect participants that do not match primary model participants in the woven model, and (3) merging the messages specified in the views based on composition directives provided by the modeler. The last step requires the modeler to define dominance or sequence number relationships between messages in the primary and aspect views.

4. Example application

We use a banking application adapted from Chandramouli's work [12] to illustrate the composition process.

4.1. Primary model

Bank users 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 or modify customer loan accounts, (3) modify the ledger posting rules, and (4) create a general ledger report. A class diagram for the application (the primary model) is shown in Fig. 6. Access control policies are not reflected in this class diagram.

The dynamic view of the primary model consists of a set of interaction diagrams. To limit the size of our paper, we show only one interaction diagram. A collaboration diagram of the *ModifyLoanAccount* operation is shown in Fig. 7. Access control concerns are not reflected in this collaboration diagram.

4.2. Generating context-specific RBAC aspects

The first step in composition is to instantiate the aspect to obtain a context-specific aspect. The context-specific aspects are obtained by instantiating the SSTs and the ISTs. The hierarchical SSD RBAC for the banking application can be obtained by instantiating the hierarchical SSD RBAC templates in Figs. 3–5 with the appropriate values from the primary model.

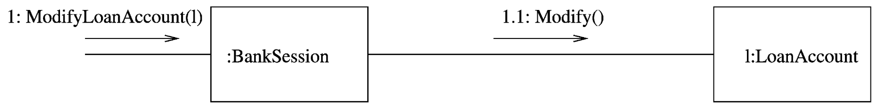

Fig. 7. An interaction diagram in the banking application (modifying a loan account).

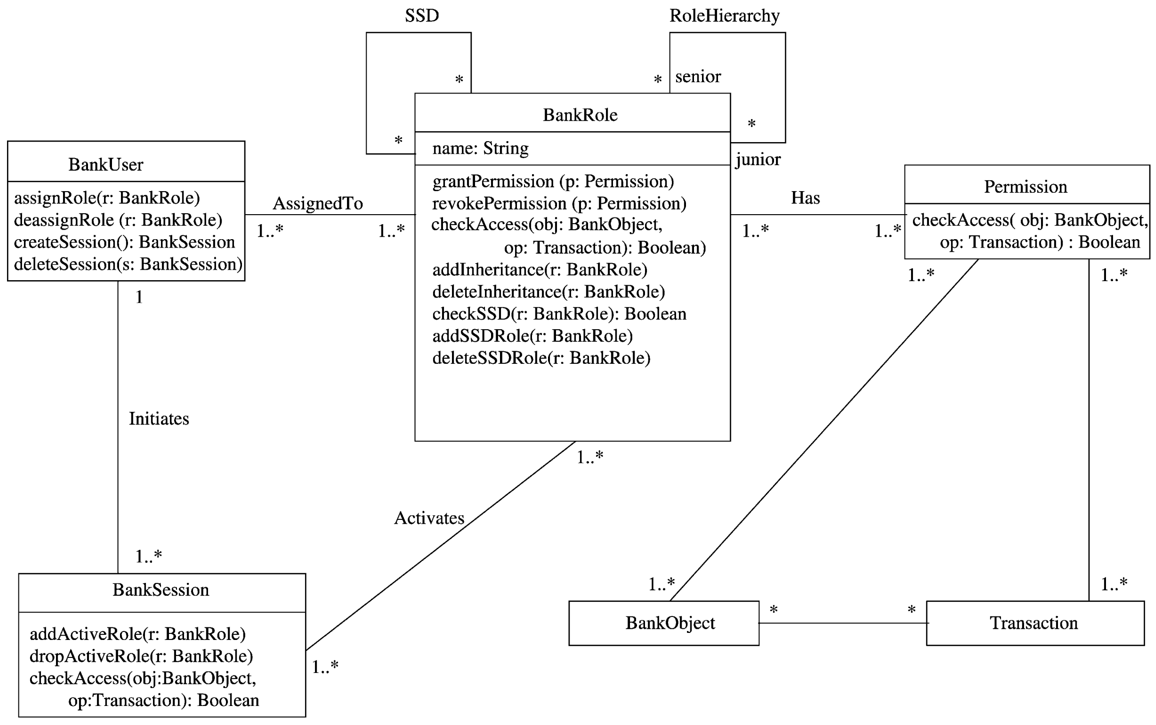
SSD
RoleHierarchy
senior
junior
BankRole
name: String
grantPermission (p: Permission)
revokePermission (p: Permission)
checkAccess(obj: BankObject, op: Transaction): Boolean)
addInheritance(r: BankRole)
deleteInheritance(r: BankRole)
checkSSD(r: BankRole): Boolean
addSSDRole(r: BankRole)
deleteSSDRole(r: BankRole)
BankUser
assignRole(r: BankRole)
deassignRole (r: BankRole)
createSession(): BankSession
deleteSession(s: BankSession)
AssignedTo
Has
Permission
checkAccess(obj: BankObject, op: Transaction) : Boolean
Initiates
Activates
BankSession
addActiveRole(r: BankRole)
dropActiveRole(r: BankRole)
checkAccess(obj:BankObject, op:Transaction): Boolean
BankObject
Transaction
1..*
1
*

Fig. 8. Context-specific RBAC aspect for the banking system.

4.2.1. Instantiating the SST

Fig. 8 is a class diagram showing a context-specific RBAC model for a banking application. The class diagram is a design view obtained from the RBAC SST that shows only the class attributes and operations needed to model the access control functionality.

In the diagram *BankUser*, *BankObject*, and *Transaction* are respectively bound to |*User*, |*Object*, and |*Operation* parameters in the RBAC template. The instantiations of the other class templates |*Role*, |*Session*, and |*Permission* are not elements in the primary model, and thus the corresponding instantiations, namely, *BankRole*, *BankSession*, and *BankPermission* are new elements that must be included in the woven model.

Constraint templates are instantiated to create OCL constraints, for example, the OCL constraint defining the pre- and post-condition for the *assignRole* operation for *BankUser* is obtained by instantiating the |*AssignRole* constraint template:

```
context BankUser ::
assignRole(r:BankRole)
pre: self.BankRole → excludes(r)
post: self.BankRole → includes(r)
```

The multiplicity parameters are also instantiated. Constraint templates that constrain the range of multiplicity parameters are instantiated to get the corresponding OCL constraints. The multiplicities at the association ends must satisfy these OCL constraints.

4.2.2. Instantiating the ISTs

The interaction diagram shown in Fig. 9 is obtained by instantiating the |*AddActiveRole* IST shown in Fig. 4.

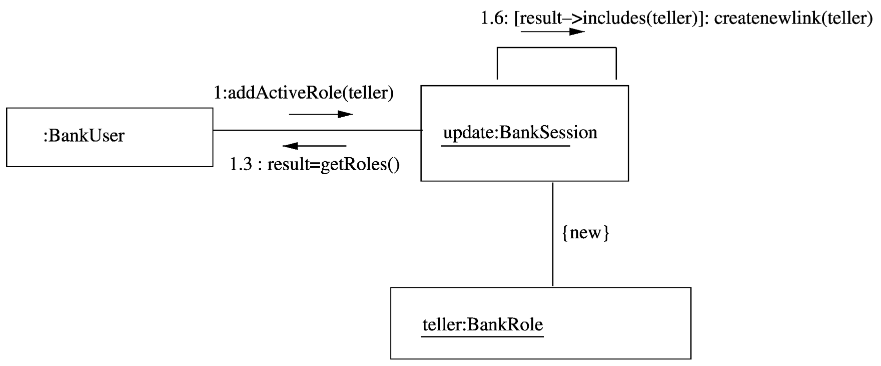

Fig. 9. Collaboration diagram obtained from the |*AddActiveRole* IST.

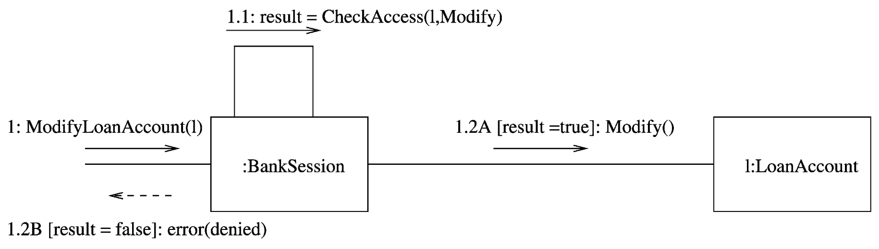
1.1: result = CheckAccess(l,Modify)
1: ModifyLoanAccount(l)
:BankSession
1.2A [result =true]: Modify()
l:LoanAccount
1.2B [result = false]: error(denied)

Fig. 10. Collaboration diagram obtained from |*Operation* IST.

The interaction diagram shows a *BankUser* activating the *teller* role in a *update* session. If the user indeed possesses this role, the teller role is activated in that session. Note that all the messages are not shown in the context-specific collaboration diagram. For example, the interactions labeled 1.1, 1.2 and 1.4, 1.5 are not shown because they are not part of the view described by the aspect. Sequence gaps in a context-specific interaction diagram arise because of the need to accommodate interactions that address other design concerns. In this example, interactions 1.1, 1.2 are required to start a logging activity and 1.4, 1.5 are required to log information.

Another collaboration diagram obtained from the |*Operation* IST (Fig. 5) is shown in Fig. 10.

4.3. Composing context-specific aspects with primary model

Having obtained the context-specific aspect, the next step is to merge this with the primary model to obtain the woven model. Comparing the class diagram of the primary model (Fig. 6) with that of the context-specific aspect (Fig. 8), we observe the following:

1. The classes *BankRole*, *BankSession*, *BankPermission* are not in the class diagram of the primary model.
2. The specializations of *BankObject* and *Transaction* are not present in the context-specific aspect.
3. The associations *AssignedTo*, *Has*, *Initiates*, and *Activates* shown in Fig. 8 are not in the class diagram of the primary model.
4. The associations *executes* and *accesses* in the primary model matches prohibited relationships.
5. The aspect classes *BankUser*, *BankRole*, *BankSession*, and *BankPermission* have operations pertaining to RBAC that are absent in the primary model.

The weaving operation takes these observations into account and produces the following woven model (see Fig. 11):

1. The classes *BankUser*, *BankObject*, and *Transaction* are merged with their corresponding counterparts in the primary model.
2. The classes *BankRole*, *BankSession*, and *BankPermission* in the context-specific aspect are included in the woven model.
3. The specializations of *BankObject* and *Transaction* in the primary model are included in the woven model.
4. The associations *AssignedTo*, *Has*, *Initiates*, *Activates* are included in the woven model.
5. New operations pertaining to RBAC (such as, *assignRole*, *grantPermission*) are added to the classes *BankUser*, *BankRole*, *BankSession*, and *BankPermission*.
6. The associations *executes* and *accesses* are absent in the woven model.

The weaving of the collaboration diagrams shown in Figs. 7 and 10 is determined by a composition directive provided by the modeler that states that the aspect model shown in Fig. 10 replaces the interaction diagram in the primary model.

5. Related work

This work is related to two distinct research areas: policy specification and aspect-oriented development. Consequently, we devote a section to each of these research areas.

5.1. Security policies

In this section, we briefly mention some work on security policies. Damianou's thesis [16] provides a comprehensive survey about the important works in this area. A large volume of research exists in the area of security policies. Some of these focus on access control models [3,6,7,9,14,29,46,55,56], some on specification of access control policies [4,5,10,13,17,33,34,36,38,49,50,53], and others in analyzing conflicts of security policies [31,43,44,45,57,58].

Formal logic-based approaches [4,5,10,13,33,38,50] are often used to specify security policies. Jajodia et al. [38] propose an authorization specification language based on stratified clause form logic. Both negative and positive authorizations can be expressed using this logic. The language also includes *integrity rules* that can be used to specify application-dependent conditions that limit the range of acceptable access control policies. Barker [4] also uses stratified clause form logic to express access control policies with special attention to RBAC. In subsequent work [5], the authors show how policies specified in stratified logic

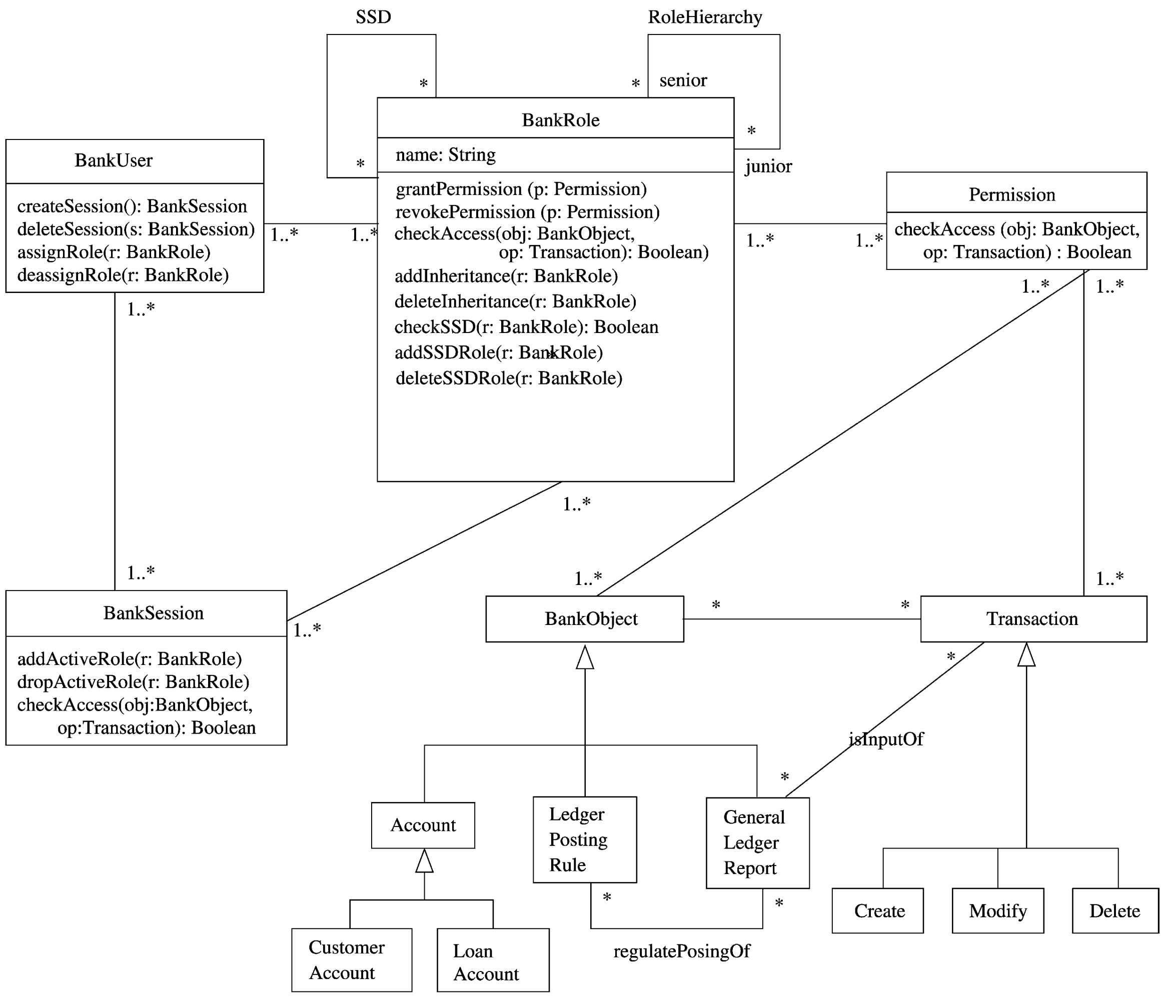
SSD
RoleHierarchy
senior
junior
BankRole
name: String
grantPermission (p: Permission)
revokePermission (p: Permission)
checkAccess(obj: BankObject, op: Transaction): Boolean)
addInheritance(r: BankRole)
deleteInheritance(r: BankRole)
checkSSD(r: BankRole): Boolean
addSSDRole(r: BankRole)
deleteSSDRole(r: BankRole)
BankUser
createSession(): BankSession
deleteSession(s: BankSession)
assignRole(r: BankRole)
deassignRole(r: BankRole)
Permission
checkAccess (obj: BankObject, op: Transaction) : Boolean
BankSession
addActiveRole(r: BankRole)
dropActiveRole(r: BankRole)
checkAccess(obj:BankObject, op:Transaction): Boolean
BankObject
Transaction
isInputOf
Account
Ledger Posting Rule
General Ledger Report
Customer Account
Loan Account
regulatePosingOf
Create
Modify
Delete
1..*
*

Fig. 11. Woven static model of RBAC and primary.

can be translated into SQL to protect a relational database from unauthorized read and update requests. Ortalo [50] describe a language for specifying security policies based on deontic logic. Researchers [33] at the Cambridge University have defined a language called Role Definition Language (RDL) based on Horn clauses. RDL is based on a set of rules that indicate the conditions under which a client may obtain a name or role. The conditions for obtaining a role depend on the credentials of the client. The notion of delegation in RDL is different in the sense that roles and not access rights are delegated. A client may delegate a role that he himself does not possess. Chen et al. [13] propose a language based on set theory for specifying RBAC state related constraints. Bertino et al. [10] extends the RBAC model with a temporal model called TRBAC. The language proposed by Bertino can specify periodic activation and deactivation of roles using periodic expressions. They can also specify temporal dependencies among role activation and deactivation using role triggers. Formal logic-based approaches, although, useful for analyzing security policies, are relatively difficult to apply.

Other researchers have used high-level languages to specify policies [34,36,49,53]. Although high-level languages are easier to understand than formal logic-based approaches, they are not analyzable. We briefly mention some important works in this area. Ribeiro et al. [53] propose a Security Policy Language for specifying authorization and obligation policies. Policies are specified using constraint rules. Tower [34] is a language for specifying RBAC policies. The policies are specified using *objects*, *privileges*, *permissions*, *users*, and *roles*. Privileges define a specific access type on an object, permissions are composed of privileges, and roles contain a set of permissions. In addition, privileges can also be associated with *conditions* and *actions*. *Conditions* limit the applicability of the privilege. *Actions* are executed when methods associated with the privileges are invoked. The *Organization for the Advancement of Structured Information Standards* (OASIS) technical committee advocates the use of XML for expressing access control policies [49]. They proposed XACML, which is an XML specification for expressing policies for information access over the Internet. The policy specification in XACML is very verbose and not aimed for human interpretation. LaSCO [36] is a graphical approach for specifying policies. The graphical format of LaSCO helps in human interpretation but is not very expressive. Ponder [17] is a specification language that allows various kinds of policies, such as, authorization, obligation, and delegation policies to be specified. Policies are specified in terms of *subject-domain*, *target-domain*, and *access-lists*. The subject-domain specifies the set of subjects that can perform the operations specified in the access-lists on the objects in the target-domain. The authors have also developed a toolkit for policy specification and deployment [18]. Steen et al. [61] propose a new language for expressing policies that can be applied over an enterprise that is modeled using UML. The language contains embedded OCL constraints. The constraints cannot specify activation/deactivation of roles or assignment of users or permissions to roles. It also does not allow for the composition of policies.

5.2. Aspect-oriented modeling

There has been much work on aspect-oriented programming [8,41,42,51,62]. A number of authors have tackled the problem of defining and weaving aspects above the programming language level (e.g. see Refs. [15,21,22,30, 48,52,59]). In the latter cases, aspect specifications are often templates, and they are generally woven by using regular expressions to match primary model elements and aspect elements. Aspect composition in these approaches usually involves wrapping additional functionality around a primary model. Proper naming and structuring of model elements is required to apply the aspect, so it is conceivable that effort must be applied to refactor existing models to correctly compose them with aspect models. The property-oriented approach used in our AOM supports a more flexible and rigorous approach to aspect definition and weaving in which primary models do not need to have syntactically equivalent names or structures in order to have aspects woven into them: Aspects are described independently of primary models as patterns; the patterns are instantiated before they are composed with the primary model. Note that, we apply our technique at the early phases. This allows for early conflict detection and resolution (via composition directives).

There has been some work on modeling dependability concerns using the UML, in particular work on modeling security concerns (e.g. see Refs. [1,2,11,20,23,28,37,39,40, 47,64,65]). These works use the UML extension mechanisms to introduce representations of dependability concerns in UML models. For example, Chan and Kwok [11] provide a design pattern for security that addresses asset and functional distribution, vulnerability, threat, and impact of loss. UML stereotypes identify classes that have particular security needs due to their vulnerability either as assets or as a result of functional distribution. Jurjens [39,40] models security mechanisms based on the multilevel classification of data in a system using an extended form of the UML called UMLsec. The UML tag extension mechanism is used to denote sensitive data. Statechart diagrams model the dynamic behavior of objects, and sequence diagrams are used to model protocols. These works are significant and complementary to our work on AOM in that they illustrate how the UML can be extended to directly represent dependability concerns. An AOM weaving process can be designed to produce extended forms of the UML that reflect the properties expressed in the aspects in a more direct manner.

The work on UMLAUT [35] is similar to our approach in that they utilize the UML, but their approach focuses only on specification of patterns. Our work goes further in that it provides support for generating models from patterns and for composing the generated models with a primary model.

The work described in this paper builds upon the notation and techniques described in previous papers [22,27]. In Ref. [27], the authors show how security concerns can be localized and then woven with models of system functionality. In Ref. [26], the authors model two independent security mechanisms as aspects and show how these two aspects (authentication and auditing) can be woven in with a primary design model. The authors also show that the order in which the aspects are woven is important. An incorrect weaving order will produce a woven model that does not meet design goals. The work described in this paper is a specialization of the AOM approach that can be used to address access control concerns.

6. Conclusion and future work

In this paper, we describe how cross-cutting access control functionality can be localized in aspects to ease evolution of policies and design. Modeling access control concerns as aspects has several benefits. It allows one to address access control concerns separately from other design concerns. This separation of concerns helps manage complexity inherent in software systems. Describing the access control aspects as patterns also makes them potentially reusable.

The AOM approach allows developers to systematically and uniformly incorporate pervasive access control functionality into a design. Composing the access control aspects with a primary model produces a woven model that can be analyzed to check whether the access control concerns have been adequately addressed in the design. Localizing access control concerns in aspects also facilitates evolution changes to access control structures and behavior can be made in the corresponding aspects, and the effect can be incorporated in the application through automated weaving.

We are currently developing an analysis technique that involves verifying the woven model against a select set of representative scenarios. The scenarios can represent malicious attacks or authorized interactions. Malicious attack scenarios are used to determine if the constraints specified in policies are sufficient to prevent the attacks from compromising protected resources, while the authorized interaction scenarios are used to determine that authorized activities are not restricted by the constraints. We will also investigate the use of model-checking techniques for statically analyzing dynamic aspects of woven models.

In separate work (e.g. see Refs. [25–27]), we have used the aspect-oriented approach to specify and weave security mechanisms into system models. The security mechanisms represented by the aspects are intended to enforce security policies. The woven model obtained by composing security mechanisms with the primary model can be checked for the satisfaction of the security policies. Using aspects to define both security policies and security mechanisms facilitates tracing of concepts expressed in policy models to concepts expressed in security mechanisms. This eases the task of verifying that the mechanisms enforce the policies. In future, we would like to develop traceability procedures that will allow one to check whether or not the mechanisms or the implementations satisfy the security policies or not.

Acknowledgements

The work of Indrakshi Ray was partially supported by the National Science Foundation under grant number IIS 0242258. The authors would like to thank the anonymous reviewers for their comments and suggestions.

References

[1] G.J. Ahn, 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), Cambridge, Massachusetts, June 2001, p. 157–162.

[2] H.A. Ali, A new model for monitoring intrusion based on Petri nets, Information Management and Computer Security 9 (4) (2002) 175–182.

[3] R.J. Anderson, A security policy model for clinical information systems, In IEEE Symposium on Security and Privacy, Oakland, CA, May 1996, p. 30–43.

[4] S. Barker, Security policy specification in logic, In Proceedings of the International Conference on Artificial Intelligence, Las Vegas, NV, 2000, p. 143–148.

[5] S. Barker, 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.

[6] J.F. Barkley, K. Beznosov, J. Uppal, Supporting relationships in access control using role based access control, In Proceedings of the Fourth ACM Workshop on Role-Based Access Control, Fairfax, VA, October 1999, p. 55–65.

[7] D.E. Bell, L.J. LaPadula, Secure computer system: unified exposition and multics interpretation, Technical Report MTR-2997, MITRE Corporation, Bedford, MA, July 1975.

[8] L. Bergmans, M. Aksit, Composing multiple concerns using composition filters, Communications of the ACM 44 (10) (2001).

[9] E. Bertino, C. Bettini, E. Ferrari, P. Samarati, An access control model supporting periodicity constraints and temporal reasoning, ACM Transactions on Database Systems 23 (3) (1998) 231–285.

[10] E. Bertino, P. Bonatti, E. Ferrari, TRBAC: a temporal role-based access control model, In Proceedings of the Fifth ACM Workshop on Role-Based Access Control, Berlin, Germany, 2000, p. 21–30.

[11] M.T. Chan, L.F. Kwok, Integrating security design into the software development process for e-commerce systems, Information Management and Computer Security 9 (2/3) (2001) 112–122.

[12] R. Chandramouli, Application of XML tools for enterprise-wide RBAC implementation tasks, In Proceedings of Fifth ACM Workshop on Role-based Access Control, Berlin, Germany, July 2000.

[13] F. Chen, R. Sandhu, Constraints for role-based access control, In Proceedings of the First ACM Workshop on Role-Based Access Control, Gaithersburg, MD, 1995.

[14] D.D. Clark, D.R. Wilson, A comparison of commercial and military computer security policies, In IEEE Symposium on Security and Privacy, Oakland, CA, May 1987.

[15] S. Clarke, J. Murphy, Developing a tool to support the application of aspect-oriented programming principles to the design phase, In Proceedings of the International Conference on Software Engineering (ICSE '98), Kyoto, Japan, April 1998.

[16] N. Damianou. A policy framework for management of distributed systems. PhD Thesis, University of London, London, UK, 2002.

[17] N. Damianou, N. Dulay, The ponder policy specification language, In Proceedings of the Policy Workshop, Bristol, UK, 2001.

[18] N. Damianou, N. Dulay, E. Lupu, M. Sloman, T. Tonouchi, Tools for Domain-based Policy Management of distributed systems, In Proceedings of the IEEE/IFIP Network Operations and Management Symposium, Florence, Italy, April 2002.

[19] P.T. Devanbu, S. Stubblebine, Engineering for security: a roadmap, In Proceedings of the International Conference on Software Engineering, Future of Software Engineering, 2000, 2000.

[20] Z. Diamadi, M.J. Fischer, A simple game for the study of trust in distributed systems, Wuhan University Journal of Natural Sciences 6 (1/2) (2001) 72–82.

[21] J.L. Fiadeiro, A. Lopes, Algebraic semantics of co-ordination or what is it in a signature?, in: A. Haeberer (Ed.), Proceedings of the Seventh International Conference on Algebraic Methodology and Software Technology (AMAST'98), Amazonia, Brasil, Lecture Notes in Computer Science, vol. 1548, Springer, Berlin, January 1999, pp. 293–307.

[22] R. France, G. Georg, Modeling fault tolerant concerns using aspects, Technical Report 02-102, Computer Science Department, Colorado State University, 2002.

[23] R. France, D. Kim, E. Song, S. Ghosh, Using roles to characterize model families, in: H. Kilov (Ed.), Practical Foundations of Business and System Specifications, Kluwer Academic Publishers, Dordrecht, 2002.

[24] R.B. France, D.K. Kim, E. Song, Patterns as precise characterizations of designs, Technical Report 02-101, Computer Science Department, Colorado State University, 2002.

[25] G. Georg, R. France, I. Ray, An aspect-based approach to modeling security concerns, In Proceedings of the Workshop on Critical Systems Development with UML, Dresden, Germany, 2002.

[26] G. Georg, R. France, I. Ray, Designing high integrity systems using aspects, In Proceedings of the Fifth IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems (IICIS 2002), Bonn, Germany, November 2002.

[27] G. Georg, I. Ray, R. France, Using aspects to design a secure system, In Proceedings of the International Conference on Engineering Complex Computing Systems (ICECCS 2002), Greenbelt, MD, ACM Press, December 2002.

[28] L. Giuri, P. Iglio, A role-based secure database design tool, In Proceedings of the 12th Annual Computer Security Applications Conference, 1996, p. 203–212.
[29] V. Gligor, Characteristics of role based access control, In Proceedings of the First ACM/NIST on Role-Based Access Control Workshop, Gaithersburg, MD, November 1995.
[30] J. Gray, T. Bapty, S. Neema, J. Tuck, Handling crosscutting constraints in domain-specific modeling, Communications of the ACM 44 (10) (2002) 87–93.
[31] N. Griffeth, H. Velthuijsen, Reasoning about goals to resolve conflicts, In Proceedings of the International Conference on Intelligent Cooperative Information Systems, Los Alamitos, California, 1993, p. 197–204.
[32] R. Grimm, B. Bershad, Providing policy-neutral and transparent access control in extensible systems, Technical Report UW-CSE-98-02-02, University of Washington, 1998.
[33] R.J. Hayton, J.M. Bacon, K. Moody, Access control in open distributed environment, In IEEE Symposium on Security and Privacy, Oakland, CA, May 1998, p. 3–14.
[34] M. Hitchens, V. Varadarajan, Tower: a language for role-based access control, In Proceedings of the Policy Workshop, Bristol, UK, 2001.
[35] W.M. Ho, F. Pennaneac'h, N. Plouzeau, UMLAUT: A framework for weaving UML-based aspect-oriented designs, In Proceedings of the Technology of Object-oriented Languages and Systems Conference (TOOLS Europe), vol. 33, IEEE Computer Society, 2000, p. 324–334.
[36] J.A. Hoagland, R. Pandey, K.N. Levitt, Security policy specification using a graphical approach, Technical Report CSE-98-3, Computer Science Department, University of California, Davis, July 1998.
[37] R. Holbein, S. Teufel, K. Bauknecht, A formal security design approach for information exchange in organisations, In Proceedings of the Ninth Annual IFIP WG 11.3 Working Conference on Database Security, Rennselaerville, New York, August 1995, p. 267–285.
[38] S. Jajodia, P. Samarati, V.S. Subrahmanian, A logical language for expressing authorizations, In IEEE Symposium on Security and Privacy, Oakland, CA, May 1997, p. 31–42.
[39] J. Jurjens, Modeling audit security for smart-card payment schemes with UMLsec, In Proceedings of the IFIP TC11 16th International Conference on Information Security (IFIP/Sec'01), Paris, France, June 2001, p. 93–107.
[40] J. Jurjens, Towards development of secure systems using UMLsec, In Proceedings of the Fourth International Conference on Fundamental Approaches to Software Engineering (FASE '01), Genova, Italy, vol. 2029 of Lecture Notes in Computer Science, 2001, p. 187–200.
[41] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W.G. Griswold, Getting started with AspectJ, Communications of the ACM 44 (10) (2001) 59–65.
[42] K. Kieberherr, D. Orleans, J. Ovlinger, Aspect-oriented programming with adaptive methods, Communications of the ACM 44 (10) (2001) 39–41.
[43] E. Lupu, M. Sloman, Conflict analysis for management policies, In Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management, San Diego, California, Chapman & Hall, London, 1997, p. 430–443.
[44] J. Michael. A formal process for testing consistency of composed security policy. PhD Thesis, George Mason University, Fairfax, Virginia, 1993.
[45] N. Minsky, V. Ungureanu, W. Wang, J. Zhang, Building reconfiguration primitives into the law of a system, In Proceedings of the International Conference on Configurable Distributed Systems, Annapolis, MD, May 1996, p. 89–97.
[46] J.D. Moffett, Control principles and role hierarchies, In Proceedings of the Third ACM/NIST on Role-Based Access Control Workshop, Fairfax, VA, October 1998.
[47] U. Nerurkar, A strategy that's both practical and generic, Dr Dobb's Journal 25 (11) (2001) 50–56.
[48] P. Netinant, T. Elrad, M.E. Fayad, A layered approach to building open aspect-oriented systems, Communications of the ACM 44 (10) (2001) 83–85.
[49] OASIS, XACML Language Proposal, Version 0.8. Technical Report, Organization for the Advancement of Structured Information Standards, 2002, Available electronically from http://www.oasis-open. org/committees/xacml.
[50] R. Ortalo, A flexible method for information systems security policy specification, In Proceedings of the Fifth European Symposium on Research in Computer Security, Louvain-la-Neuve, Belgium, Springer, Berlin, 1998.
[51] H. Ossher, P. Tarr, Using multidimensional separation of concerns to (re)shape evolving software, Communications of the ACM 44 (10) (2001) 43–50.
[52] J.A.D. Pace, M.R. Campo, Analyzing the role of aspects in software design, Communications of the ACM 44 (10) (2001) 66–73.
[53] C. Ribeiro, A. Zuquete, 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.
[54] J.H. Saltzer, M.D. Schroeder, The protection of information in computer systems, Proceedings of the IEEE 63 (9) (1975) 1278–1308.
[55] P. Samarati, S. Vimercati, Access control: policies, models and mechanisms, in: R. Focardi, R. Gorrieri (Eds.), Foundations of Security Analysis and Design (Tutorial Lectures), 2000, pp. 137–196.
[56] R. Sandhu, D. Ferraiolo, R. Kuhn, The NIST model for role-based access control: towards a unified standard, In Proceedings of the Fifth ACM Workshop on Role-based Access Control, Berlin, Germany, July 2000, p. 47–61.
[57] E. Sibley, Experiments in organizational policy representation: results to date, In Proceedings of the IEEE International Conference on Systems Man and Cybernetics, Los Alamitos, CA, IEEE Computer Society Press, 1993, p. 337–342.
[58] E. Sibley, J. Michael, R. Wexelblat, Use of an experimental policy work-bench: description and preliminary results, in: C. Landwehr, S. Jajodia (Eds.), Database Security V: Status and Prospects, Elsevier, Amsterdam, 1992, p. 47–76.
[59] A.R. Silva, Separation and composition of overlapping and interacting concerns, In OOPSLA '99 First Workshop on Multi-Dimensional separation of Concerns in Object-Oriented Systems, Denver, Colorado, November 1999.
[60] E.G. Sirer, R. Grimm, A.J. Gregory, N.R. Anderson, 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, 1998.
[61] M.W.A. Steen, J. Derrick, Formalizing ODP enterprise policies, In Proceedings of the Third International Enterprise Distributed Object Computing Conference, Mannheim, Germany, September 1999.
[62] G.T. Sullivan, Aspect-oriented programming using reflection and metaobject protocols, Communications of the ACM 44 (10) (2001) 95–97.
[63] The Object Management Group. The Unified Modeling Language. Version 1.4, OMG, formal/2001-09-67, 2001.
[64] D. Trcek, Security policy conceptual modeling and formalization for networked informaiton systems, Computer Communications 23 (17) (2000) 1716–1723.
[65] J.J. Whitmore, A method for designing secure solutions, IBM Systems Journal 40 (3) (2001) 747–768.