Enhancing UML to Model Custom Security Aspects

[Position Paper]

Jaime Pavlich-Mariscal, Laurent Michel, and Steven Demurjian

Department of Computer Science & Engineering, The University of Connecticut, Unit-2155, 371 Fairfield Road, Storrs, CT 06269- 2155. jaime.pavlich@uconn.edu, {ldm,steve}@engr.uconn.edu

Abstract. Despite its widespread usage, the Unified Modeling Language (UML) specification still lacks formal, explicit, support for access control. This paper proposes an approach to model security as a separate concern by augmenting UML with separate and new diagrams for role-based, discretionary, and mandatory access controls; collectively, these diagrams provide visual access-control aspects. Individually, each of these diagrams contain a set of *security features* that augment UML with security capabilities. The intent is to provide designers with a broad set of security features, where they can select only the features needed by their application, merge them into UML, and utilize the custom result to model security aspects. This paper presents a set of features extracted from role-based, discretionary, and mandatory access control, demonstrates their composition into a customizable security model in UML (including a formal basis), and illustrates the approach via a university application.

1 Introduction

Security has become a very important issue in the development of software applications. Definition of access control policies, along with other security requirements, must be an integral part of the software development process, to ensure that the proper level of security in an application is attained. Access control is defined as: "Limiting access to information system resources only to authorized users, programs, processes or other systems"[1]. The software development process consists of a systematic series of tasks to create a software system: requirements capture, analysis, design, coding, and testing. The scope of this research is at the design stage of the process, concentrating on modeling of access control.

To analyze the issues of modeling security, one must understand the most common *security schemes* that are used to conceptualize access control policies: *mandatory access control* (MAC) [2], *discretionary access control* (DAC)[3], and

role-based access control (RBAC)[4]. MAC is well-suited to applications where the protection of information is paramount (i.e., releasing such information would have dire national security or financial consequences). In MAC, each object is labeled with a *classification level* (e.g., top secret, secret, confidential, and unclassified) that represents the sensitivity of their information. Each subject has a *clearance level*. Security is enforced by ensuring that a subject's clearance level always *dominates* an object's classification level. DAC targets applications that are collaborative and dynamic. In DAC, permissions are defined between subjects and objects, but a subject can be granted the permission to delegate a subset of its own permissions to another user. RBAC groups permissions into independent units called *roles*, which represent the role that a user assumes in an organization. Roles, rather than permissions, are assigned to users (subjects) when they initiate an interactive session with the software system. The set of privileges granted to a user is defined by the set of permissions assigned to its corresponding role. Security schemes such as MAC, DAC, and RBAC, specify the basic semantics for access control, but they do not provide a visual language to represent this information. UML [5], the dominant software and system modeling approach, while an obvious candidate to provide security, lacks explicit support for access control. Furthermore, security is a crosscutting concern that pervades the entire application, which makes it difficult for software practitioners to adequately integrate security into software [6]. As a result, when designers wish to incorporate security concerns into an application using UML, the resulting model is very likely to have security tangled and scattered throughout the entire design.

Our proposed approach will address the above issues by extending UML with *security diagrams* to represent MAC, DAC and RBAC policies as aspects. Furthermore, the proposed approach intends to provide flexibility to the modeling of access control: as requirements vary between applications, designers do not always need all of the features present in the notation, but only a subset of them to suit their application needs. The approach decomposes MAC, DAC, and RBAC into *security features*, which represent the minimal elements of an access control policy. Designers can select specific features and combine them (according to rules and limits) in order to create a security aspect-modeling infrastructure that is suitable for their requirements. Since security features comprise a small subset of the information of an access control schema, they should be easier to understand by designers. Furthermore, they assist in tracking security requirements from models to code, reducing scattering of access-control definitions across the application, and providing a collective view of the security policy.

This paper extends our previous work on the role-slice diagram for RBAC [7] with additional diagrams for users, delegation, and mandatory access control features. Most importantly, this work applies composibility to allow custom application-level security. Section 2 describes an example that will be used to illustrate the approach. Section 3 describes security features and the process to create custom security aspect models. Section 4 compares the proposed approach with related work. Section 5 concludes the paper and reviews ongoing work.

2 The University Application Example

This paper uses a simplified example of a university application. The university application manages course, student, teacher, and public catalog information. The security requirements are as follows: teachers have assigned a set of courses, they can read and write the syllabus, and read the code of each course. Teachers can see the enrolled students in each course, access their names, and assign grades, but they cannot see in which courses students are enrolled. Students can see their grades, enrolled courses, the teachers of those courses, read the syllabus and code, but cannot see which students are enrolled in those courses, or modify any information in the system. Catalog information can be acessed by anyone; no access control is required for this information. Figure 1 shows a class diagram of the university application. *Course* keeps track of all of the courses of a university. *StudentInformation* manages information about students. *Catalog* manages the publicly-available information about courses offered at a university.

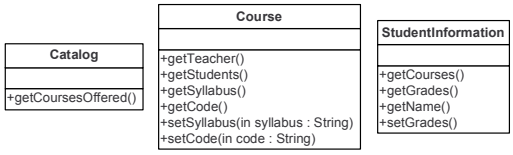


Fig. 1. Class Diagram of the University Application Example.

3 Enhancing UML with Security

The core of the approach is to extend UML with security aspect modeling capabilities. The extension comprises two elements: Security Features and Security Diagrams. *Security Features* are components that correspond to specific elements of access control schemes (e.g., positive permissions, delegation rules, MAC security properties, etc.). *Security Diagrams* provide the notation to depict security features as aspects separated from the main design of the application. Figure 2 shows an overview of the proposed security extensions to UML. The *Role Slice Diagram* (1), which is part of previous work in [7], is a visual notation for roles, positive and negative permissions (e.g., roles to methods), and role hierarchies. The *User Diagram* (2) depicts users, positive and negative permissions (roles to users), association to roles, and constraints over role assignment. The *Delegation Diagram* (4) is a notation for rules of delegation of the DAC security scheme, and includes user-delegation assignment (who is allowed to delegate), delegation authority (can delegate), and pass-on delegation authority (can delegate the ability to delegate). *MAC Features* (3) provide the constructs for the three security diagrams to depict Mandatory Access Control rules. Each extension is associated with a set of *Security Features*, which are building blocks that correspond to specific elements of access control schemes (e.g., roles, permissions, delegation rules, classifications, clearances, etc.). Designers choose a subset of features, and perform a composition (6) between their meta-models and the UML meta-model

(5) to yield an augmented meta-model (7). To create a design model for the entire application (including all of the security and non-security concerns), the composite meta-model (7) is instantiated (8) into a *Main Design* (9) that is the design of the non-security concerns and *Security Aspects* (10) that conform the access control policy for the application.

The main theme of this paper is the definition of the *infrastructure* (i.e., meta-models and policy permissions) required to model security aspects. Section 3.1 describes example security aspects (10) of the access control policy of the university application and the way to depict them using security extensions (1) through (4). Section 3.2 describes the creation of its infrastructure using composition (6) of security features.

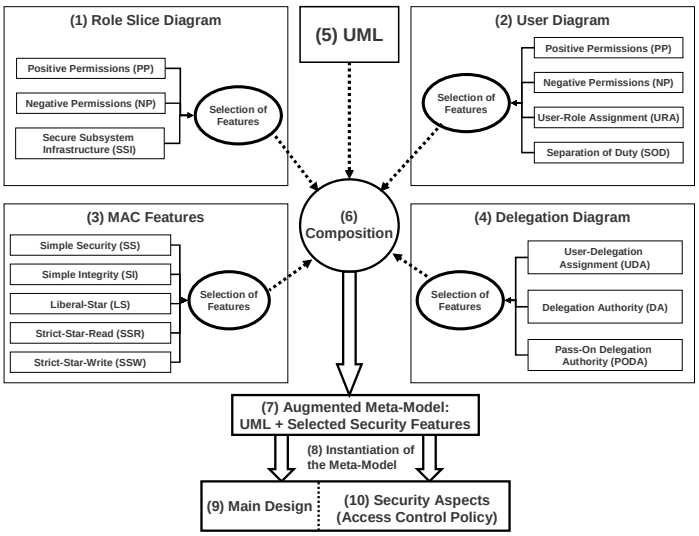


Fig. 2. Overview of the Proposed Approach.

3.1 Modeling Security Aspects

To model a security policy, designers must identify three key components: subjects, objects, and permissions. *Subjects* are the entities that require access to the system. The system contains a set of *objects* that are the entities that require protection against subjects. For the proposed approach, class methods (operations) are the objects in the system that require protection. *Permissions* determine which operations can access each subject in the system. Formally, this is represented as follows:

$Subject$: A set of subjects.

$Operation$: A set of operations, i.e., the methods of classes.

$P \subseteq Subject \times Operation$: A set of permissions, where $\langle s, op \rangle \in P$ iff. subject s is allowed to invoke operation op.

To model the requirements of the university application, designers must choose an access control structure that represents the three sets above, and satisfies the security requirements. The university application has two kinds of users, each one with different permissions: teachers and students. A role-based policy is a good alternative to group users according to their similarities. To assign permissions to roles, designers have two alternatives: assign positive permissions explicitly, or use mandatory access control rules. For this example, assume that designers choose MAC rules, assigning clearances to users, and classifications to operations, and allowing a subject to access an operation only if its clearance is greater than or equal to the classification of the operation. Some operations that would be allowed by a MAC-based policy may not be permitted according to requirements, so designers can also decide to use negative permissions to explicitly deny them.

Figure 3 shows a role-slice diagram enhanced with MAC that represents the roles and permissions for the university application. The *Secure Subsystem*, depitcted as a package with the stereotype ≪SecureSubsystem≫, comprises all of the operations in the system that require access control. The secure subsystem also defines their classifications (unclassified (u), confidential (c), secret(s), or top-secret (ts)), and their access mode (read or write). Roles Teacher and Student appear as packages with the stereotype ≪roleSlice≫. They have assigned a clearance and negative permissions (operations with the stereotype ≪neg≫). Roles are connected to the secure subsystem, meaning that role permissions must be a subset of the operations referenced by the secure subsystem. Figure 4 shows a user diagram that depicts users as packages with the stereotype ≪user≫; and, users' assigned roles as dependencies with the stereotype ≪roleAssignment≫.

3.2 Creating the Infrastructure to Model Security Aspects

To enable UML to define security aspect models such as the ones in Fig. 3 and Fig. 4, and to precisely specify the permissions of a security policy (i.e., set P), the proposed approach uses *Security Features*, which represent the structure and semantics of the minimal components of a policy. Recall that Fig. 2 showed the set of proposed security features. The *Secure Subsystem Infrastructure (SSI)* represents the operations in the system that require access control. *User-Role Assignment (URA)* represents users, roles and their associations. *Positive Permissions (PP)* and *Negative Permissions (NP)* represent what subjects can or can't do in a system. *Separation of Duty (SOD)* constrains user-role assignment, defining which roles cannot be simultaneously assigned to one user [4]. *User Delegation Assignment* (UDA), *Delegation Authority (DA)* and *Pass-On Delegation Authority (PODA)* provide delegation capabilities. The remaining MAC features provide the base elements for security-level policies (i.e., classifications and clearances), and different security properties to constrain subject read and/or write access to operations based on clearances and classifications.

A security feature has two main elements: a meta-model, defined with the Meta-Object Facility(MOF) [8] that can be composed into the UML meta-model using

the PackageMerge relation [5]; and, an authorization constraint, which is a relation that defines the way that permissions (i.e., elements of P) are obtained from instances of the feature's meta-model. One reason to partition meta-models across security features is to simplify their use by designers. According to [9] the UML specification is difficult to understand because of its complex and large meta-model. Using features as a semantic and structural unit that encompasses small portions of the security meta-model may make it easier for designers to understand them. Another important advantage (although not directly related to modeling) is that security features improve code generation, because only the features chosen by designers are transitioned into code. Overall, each security feature realizes one particular security requirement, helping designers to track security requirements down to models and code, and providing the means to make changes without impacting the entire security of the application.

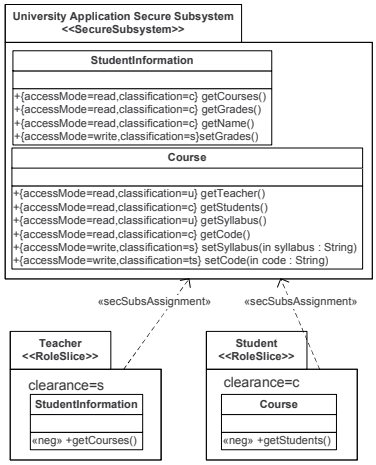


Fig. 3. A Role-Slice Diagram.

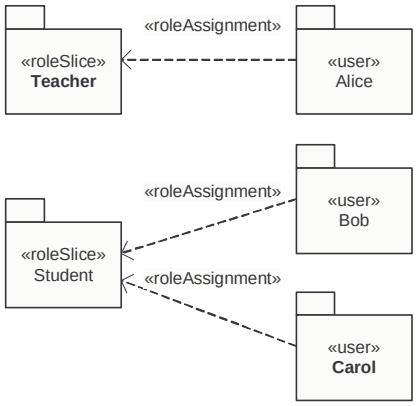
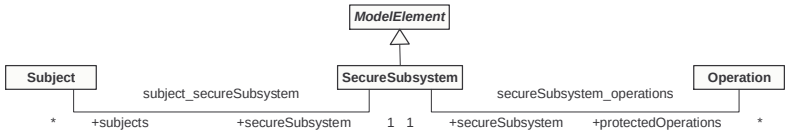


Fig. 4. A User Diagram.

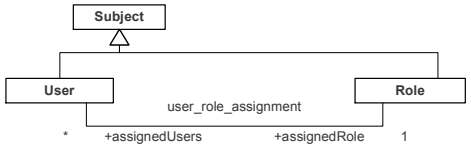
To enable UML to represent the security policy of the university application, designers choose a subset of security features. The SSI feature, shown in Fig. 5 enables the secure subsystem shown in Fig. 3, and its meta-model associates subjects with meta-class *SecureSubsystem* that references the set of operations that require access control. To enable users, roles, and user-role associations of Fig. 4, designers choose security feature URA, shown in Fig. 6. URA's meta-model defines meta-classes $User$ and *Role* as children of $Subject$, which implies that users and roles can also be assigned permissions. URA's authorization constraint allows users to access all of those operations that are allowed to their assigned roles. To define a MAC-based policy, designers use features Simple Security (SS) and Simple Integrity (SI), shown in Fig. 7. Both features have the same meta-model that assigns a clearance to subjects (relation $subject\_clearance$), a classification to operations (relation $operation\_classification$), and an access mode (i.e., read or write) to operations. Feature SS allows access to *read*-operations where the subject's clearance is greater than or equal to the operation's classification. Fea-

ture SI allows access to *write*-operations where the subject's clearance is greater than or equal to the operation's classification. To explicitly deny access to chosen operations, designers incorporate the Negative Permissions (NP) feature, shown in Fig. 8, whose meta-model defines the relation *subject_denied_operations* between subjects and operations, and whose authorization constraint defines a set of permissions NP that includes all of those tuples $\langle s, op\rangle$ where subject s and operation op are associated by relation *subject_denied_operations*.



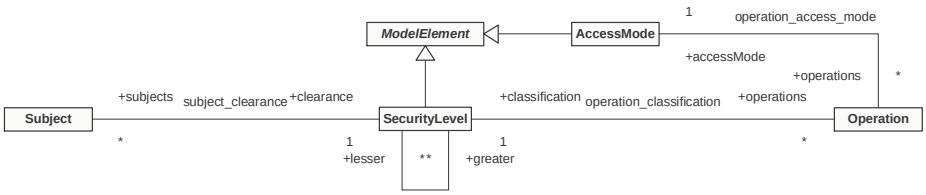
$$SSI = \{\, \langle s, op\rangle \,|op \in s.secureSubsystem.protectedOperations \,\}$$

Fig. 5. Meta-Model of the SSI Feature



$$URA = \{\, \langle u, op\rangle \mid \langle u.assignedRole, op\rangle \in P \,\}$$

Fig. 6. Meta-Model of the URA Feature

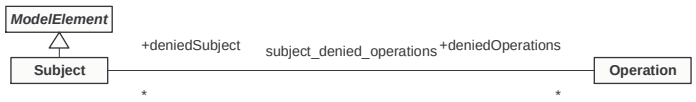


$$SS = \{\, \langle s, op\rangle \,|s.clearance \geq op.classification \wedge op.accessMode.name = \text{ read} \,\}$$
$$SI = \{\, \langle s, op\rangle \,|s.clearance \geq op.classification \wedge op.accessMode.name = \text{ write} \,\}$$

Fig. 7. SS and SI Features

Figure 9 shows meta-model SSI+SS+SI+NP+URA (meta-class $ModelElement$ not shown for space reasons), which corresponds to (7) in Fig. 2, and is the result of the composition of all of the chosen meta-models, and the UML meta-model (simplified for space purposes; it provides meta-classes $Class$, $Attribute$, and $Operation$). A composition obtained from PackageMerge relations is the union of the contents of the participating meta-models. If there are two meta-models having a meta-class with the same name, the composite meta-model will contain a meta-class with that name, and its attributes, associations, and operations will

be the union of the attributes, associations, and operations of the corresponding meta-classes in the original meta-models.



$$NP = \{\langle s, op\rangle \,|op \in s.deniedOperations\}$$

Fig. 8. Meta-Model of the NP Feature

To combine authorization constraints, designers use standard set operators. The result will determine the final set of permissions, i.e., set P, based on the security diagrams of Fig. 3 and Fig. 4. For this example, assume that designers choose to combine authorization constraints as follows:

$$P = SSI^C \cup ((SI \cup SS) \backslash NP)$$

The final set of permissions for this custom policy includes all of the tuples $\langle s, op\rangle$ where op does not belong to the secure subsystem (SSI^C): operations that do not require access control are authorized by default (in this example, there is only one such operation, $getCoursesOffered$ from class $Catalog$, which is not part of the secure subsystem). All of the tuples referenced by SI and SS are also included in P, except those referenced by NP, i.e., all of the operations that satisfy MAC simple security and simple integrity properties are authorized, unless they are explicitly denied.

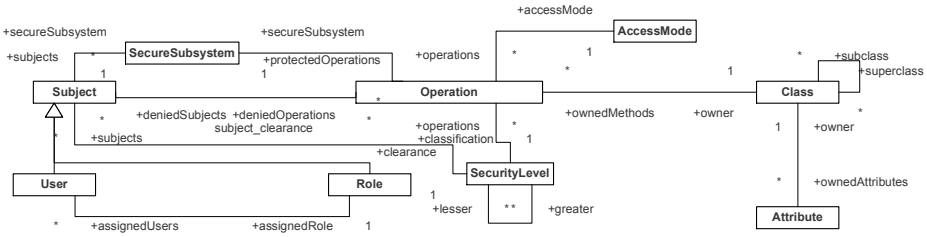


Fig. 9. Merged Meta-Model of SSI+SS+SI+NP+URA.

4 Related Work

This research is seeking to include security as part of the UML. In the UML area, there have been many research efforts that involve security from different perspectives: UMLsec [10], is an extension to UML to represent and verify security concerns such as: fair exchange, secrecy/confidentiality, secure information flow, and secure communication links. The UMLSec approach does not explicitly address access control. SecureUML [11] is an approach that extends the UML meta-model to define RBAC policies. AuthUML [12] models RBAC policies using use cases and Horn clauses to represent the security information and

to check its consistency. AuthUML and SecureUML focus on only one access control scheme (RBAC). In contrast, the work proposed herein augments UML with RBAC, MAC, and DAC capabilities. Ray et al. [13] compose access control behavior into an application by using aspect-oriented modeling techniques in UML. However, they do not provide a visual notation for security models. In contrast, the work proposed herein extends the UML notation to represent access control policies. The work by Doan et al. [14] extends use-case, class, and sequence diagrams with tagged values representing access-control attributes, such as classification and clearance levels, lifetimes (legal time intervals for accessing elements in the model), etc. Doan's approach does not provide a collective view of security at a model level. As a result, access control definitions (i.e., objects, subjects and permissions) are scattered throughout the design. The approach proposed in this paper solves this problem providing diagrams that concentrate into a separate unit the security concerns of the application.

5 Conclusions and Ongoing Work

This paper has presented an approach to compose features from different security schemes, to represent custom security aspects using specialized diagrams. This includes an expansion of UML to include role-based, discretionary, and mandatory access controls, via new UML diagrams for roles, user authorizations, and delegation, and MAC features that are applicable to multiple diagrams. As a result, designers are able to represent access control aspects with UML-based diagrams and an underlying scheme that combines RBAC, MAC and DAC. The unification of these three security schemes provides designers with a broader set of options to define security aspects than each scheme separately. To our knowledge, no other approach integrates RBAC, MAC and DAC into a set of security-specific UML-based diagrams separated from the main design. The use of security features should increase flexibility to cope with changes in requirements, providing compositionality of the underlying security capabilities, which make it possible to add security features without affecting the non-security aspects of the design. The usage of existing UML mechanisms to realize this approach (MOF and PackageMerge) should facilitate the integration of the proposed approach with tools relying on standard practices for software development, specifically UML CASE tools. Overall, we anticipate that this work will eventually yield an improved secure-software-engineering process with security aspects incorporated as an integral part of the software design and implementation process.

Ongoing research is defining additional security features and diagrams. As given in Fig. 2, the diagrams and features represent an initial core set of capabilities; as the work proceeds, there will be changes and refinements. In terms of composition, the model as given in Fig. 9 represents one option; part of the ongoing work is to define allowable combinations of security features (since not all combinations make sense), and to prove that the composition of features is secure. In addition to security support at the design level, past work [7] has explored

the transition to the implementation stage via the generation of aspect-oriented security enforcement code for the role-slice diagram (see Fig. 3). Part of the ongoing work explores this code-generation capability for all of the security aspects (and their composition), formalizing the transition from security features into aspect-oriented code, and ellaborating proofs of correctness of the synthesized code. The prototyping effort includes the implementation of meta-model composition and the notation for security diagrams (delegation and user diagrams, since the role-slice diagram has been implemented). The prototype will serve to validate the work proposed herein, and compare it against other approaches for modeling and implementing security.

References

1. Telecom, A.: Glossary 2000. t1.523-2001 (2001)
2. Bell, D., LaPadula, L.: Secure Computer Systems: Mathematical Foundations Model. Technical report, Mitre Corporation (1975)
3. Liebrand, M., E.H.J.P.C., Ting, T.C.: Role delegation for a distributed, unified RBAC/MAC. In: Proceedings of Sixteenth Annual IFIP WG 11.3 Working Conference on Data and Application Security. (2002)
4. Ferraiolo, D., Sandhu, R., Gavrila, S., D., K., Chandramouli, R.: Proposed NIST Standard for Role-Based Access Control. ACM Transactions on Information and System Security **4** (2001) 224–274
5. Object Management Group: UML 2.0 superstructure. Technical report, Object Management Group (2005)
6. De-Win, B., Piessens, F., Joosen, W., Verhanneman, T.: The importance of the separation-of-concerns principle in secure software engineering (2002)
7. Pavlich-Mariscal, J., Doan, T., Michel, L., Demurjian, S., Ting, T.: Role Slices: A Notation for RBAC Permission Assignment and Enforcement. In: Proceedings of 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security. (2005)
8. Object Management Group: Meta object facility (MOF) core specification. version 2.0. Technical report, Object Management Group (2006)
9. France, R.B., Ghosh, S., Dinh-Trong, T., Solberg, A.: Model-driven development using uml 2.0: Promises and pitfalls. Computer **39**(2) (2006) 59
10. Jürjens, J.: UMLsec: Extending UML for Secure Systems Development. In: Proceedings of the 5th International Conference on The Unified Modeling Language. (2002)
11. Basin, D., Doser, J., Lodderstedt, T.: Model Driven Security. In: Engineering Theories of Software Intensive Systems. Springer (2005)
12. Alghathbar, K., Wijesekera, D.: AuthUML: a three-phased framework to analyze access control specifications in use cases. In: Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering. (2003)
13. Ray, I., Li, N., Kim, D., France, R.: Using Parameterized UML to Specify and Compose Access Control Models. In: In Proceedings of the 6th IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems. (2003)
14. Doan, T., Michel, L., Demurjian, S., Ting, T.: Stateful Design for Secure Information Systems. In: Proceedings of 3rd International Workshop on Security in Information Systems (WOSIS05). (2005)