SecureUML: A UML-Based Modeling Language for Model-Driven Security[⋆]

Torsten Lodderstedt, David Basin, and Jürgen Doser

Institute for Computer Science, University of Freiburg, Germany
`{tolo,basin,doser}@informatik.uni-freiburg.de`

Abstract. We present a modeling language for the model-driven development of secure, distributed systems based on the Unified Modeling Language (UML). Our approach is based on role-based access control with additional support for specifying authorization constraints. We show how UML can be used to specify information related to access control in the overall design of an application and how this information can be used to automatically generate complete access control infrastructures. Our approach can be used to improve productivity during the development of secure distributed systems and the quality of the resulting systems.

1 Introduction

Security plays a central role in the development and functioning of many large-scale distributed software systems, like those for electronic commerce. However, an analysis of today's software development processes reveals that the careful engineering of security into the overall system design is often neglected. Security features are typically built into an application in an ad-hoc manner or are only integrated later during the system administration phase. There are several reasons for this. First, security is a "horizontal" aspect of software development that affects nearly every component of an application and its integration into the software development process is not well understood. Second, there is a lack of tools supporting security engineering. Third, the integration of security into a system by hand is difficult and errors often arise due to the lack of experience of the individual developers. These developers are generally not security experts and they need concrete guidelines for constructing secure applications. Overall, the post-hoc, low-level integration of security has a negative impact on the quality of resulting applications.

We present a methodology for modeling access control policies and their integration into a model-driven software development process and show how this methodology can help avoid the kinds of problems mentioned above. Our methodology is based on *SecureUML*, a modeling language designed to integrate information relevant to access control into application models defined with the Unified Modeling Language (UML) [9].

[⋆] This work has been supported by the German "Bundesministerium für Wirtschaft und Technologie" under the reference number IT-MM-01MS107. The authors are responsible for the content of this publication.

The integration of security engineering into a model-driven software development approach has the following advantages. To begin with, security requirements can be formulated and integrated into system designs at a high level of abstraction. In this way, it becomes possible to develop security aware applications that are designed with the goal of preventing violations of a security policy. For example, a database query can be designed so that users can only retrieve those data records that they are allowed to access. Furthermore, the model information can be used to detect and to correct design errors or to verify the correctness of the mapping between requirements and their realization in a design. Moreover, access control infrastructures can be generated from SecureUML models and thereby prevent errors during the realization of access control policies and enable the technology independent development of secure systems.

The work described here is part of ongoing research to develop a complete model-driven approach for developing secure e-commerce systems. While SecureUML currently focuses on access control, future research will extend the scope of the language to cover other security aspects, like digital signatures.

A prerequisite for the approach taken here is the existence of a modeling language with an extensible syntax, a sufficiently precise semantics, and CASE tool support. UML fulfills these requirements. We show how a modeling language for specifying access control policies can be defined as an extension of UML. Because of its visual notation and the possibility to define designs at a high abstraction level, UML is well suited for designing secure systems. Therefore, SecureUML enables even developers without a strong security background to develop secure systems.

Our language is based on an extended model for *role-based access control* (RBAC). RBAC is a well-established access control model with widely-recognized advantages, e.g. as explained in [10], and it is supported by a large number of software platforms. However, RBAC lacks, in general, support for expressing access control conditions that refer to the state of a system, e.g. the state of a protected resource, parameter values, date or time. To cover such cases, we introduce the concept of *authorization constraints*. An authorization constraint is a precondition for granting access to an operation. We define such constraints using the *Object Constraint Language* (OCL).

SecureUML offers significant design flexibility because it combines the simplicity of a graphical notation for RBAC with the power of logical constraints on models. Simple policies can be expressed using role-based permissions and more complicated requirements can be specified by adding authorization constraints. The resulting combination is quite powerful; for example, it is possible to base access decisions on dynamically changing data like time or to support concepts like “object ownership”.

As a proof of concept we have implemented a prototypical generator for the component architecture Enterprise JavaBeans (EJB) [11]. Our prototype demonstrates that it is possible to generate security infrastructures for access control based on SecureUML models, including role definitions, method permissions, user-role assignments and authorization constraint implementations.

We proceed as follows: In Section 2, we explain the foundations of our work. This includes the underlying access control model RBAC, the component architecture EJB, the Unified Modeling Language and an introduction to the concept of model-driven software development. We give an overview of SecureUML in Section 3 and explain our metamodel and notation in Sections 4 and 5. In Section 6, we draw conclusions and discuss related and future work.

2 Background

2.1 Role-Based Access Control

We use role-based access control (RBAC) as the underlying security model of our modeling language. RBAC is a model for access control where users and their privileges are decoupled by roles. This decoupling is not only conceptually useful, it also leads to significantly compacter access control policy descriptions.

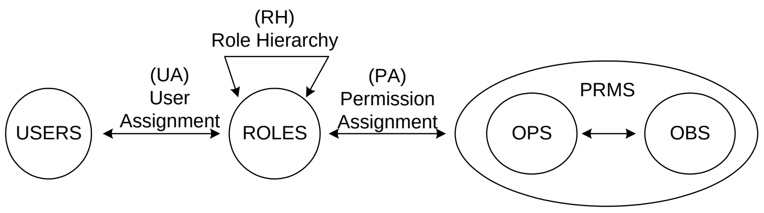

Fig. 1. Role-Based Access Control

Figure 1 shows the data model we use as foundation of our modeling language. It is based on the standard for RBAC as proposed in [4]. The model consists of five data types: users (USERS), roles (ROLES), objects (OBS), operations (OPS) and permissions (PRMS). A user[1] is defined as a person or a software agent. A role is a job or function within an organization. It combines all privileges needed to fulfill the respective job or function. Privileges are expressed in terms of the permissions assigned to a role by entries to the relation Permission Assignment. A permission represents the authorization to execute an operation on one or more protected objects or resources. An object in this context is a system resource or a set of resources that are protected by the security mechanism. An operation is an action on a protected object that can be initiated by a system entity. The types of operations depend on the type of the protected objects. In a file system, for example, there might be permissions to read, write or execute files. The assignment of roles to users is defined by the relation User Assignment. The relation Role Hierarchy defines an inheritance relationship between roles. A relation r1 inherits r2 implies that all permissions of role r2 are also permissions of role r1.

RBAC is well suited as a foundation for the modeling of access control for several reasons. The concept of role-based permissions is close to the domain vocabulary used to define security in organizations. Therefore, it can ease the expression of requirements relevant for access control during analysis as well as promote their realization in the design. Roles can be used to decouple the design of the application access control policy

[1] To simplify the presentation, we omit a comprehensive model of users and groups.

and its administration. This opens the possibility of developing application access control policies in the context of a model-driven process. Finally, note that many modern software platforms support the RBAC model. Thus, it will be possible to directly generate access control infrastructures for these platforms from application models expressed with SecureUML.

2.2 Enterprise JavaBeans

We use Enterprise JavaBeans (EJB) as an example of a component architecture in our prototype and in this paper. EJB is widely used in the industry for developing distributed systems. It is an industry standard with strong security support, which is implemented by a large number of application servers. Due to lack of space, we only describe the basic concepts of EJB, focusing on access control. For more information we refer to the EJB standard as defined in [11].

The access control model of EJB is RBAC, where the protected resources are the methods accessible by the interfaces of an EJB. An access control policy is mainly realized by using *declarative access control*. This means that the access control policy is configured in the so-called deployment descriptors of an EJB component. The security subsystem of the EJB environment is responsible for enforcing this policy on behalf of the components. The following example shows the definition of a permission that authorizes the role `AdminRole` to execute the method `withdraw` on the component `Account`.

```
<method-permission>
  <role-name>AdminRole</role-name>
  <method>
    <ejb-name>Account</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>withdraw</method-name>
  </method>
</method-permission>
```

As illustrated by this example, such permissions are defined at the level of particular methods. In general, for realistic applications the information needed to specify a comprehensive access control policy is quite voluminous. Therefore, there is the inherent danger of inadmissible simplifications due to oversights or shortcuts taken by developers. Suppose a security policy grants a role the permission to access some methods of an EJB. A correct realization would be to define one method-permission element with one method element for each of these methods. To save time, a developer might define just one method permission granting the role full access to all methods of the EJB. This is likely to cause security holes as well as inconsistencies between a security policy and its realization. We see the modeling of security policies at a higher abstraction level and the automatic generation of the corresponding deployment descriptors as a promising solution to this problem.

In addition to declarative access control, EJB offers the possibility of implementing access control decisions within the business logic of components. This mechanism is called *programmatic access control*.

2.3 Unified Modeling Language

We use the Unified Modeling Language (UML) as the foundation of our work for several reasons: UML is the de-facto standard for object-oriented modeling. Many modeling tools support UML and a great number of developers are familiar with the language. Hence, our work enables these users to develop access control policies using an intuitive, graphical notation.

UML offers the possibility of extending the modeling language using well-defined extensibility constructs that are packaged in a so-called UML Profile. In our work, we use *stereotypes* to define new types of model elements and *tagged values* to introduce additional attributes on metamodel types. UML also provides a metamodel and a logical language, the Object Constraint Language (OCL), to define constraints on model elements. The SecureUML metamodel defines authorization constraints as a special kind of UML constraint and uses OCL as the expression language. In this way, we are able to utilize existing tools and concepts for the definition and analysis of constraints in the context of access control.

2.4 Model-Driven Software Development

Model-driven software development is an approach where software systems are defined using models and constructed, at least in part, automatically from these models. A system can be modeled at different levels of abstraction or from different perspectives. The syntax of every model is defined by a metamodel.

Systems for model-driven software development can be seen as a new generation of visual programming languages. The metamodel defines the syntax of the modeling languages, a model plays the role of the source code, and the generator replaces the compiler. Using this approach, it is possible to generate automatically large amounts of source code and other artifacts, e.g. deployment descriptors and make files, based on relatively concise models. This improves the productivity of the development process as well as the quality of the resulting systems. It is also a large step towards the platform independent design of systems.

The *Object Management Group* (OMG) is working on a standard architecture for model-driven software development called *Model Driven Architecture* (MDA). Our language conforms to the upcoming standard as described in [8]. We define a reference model based on this proposal, which is shown in Figure 2 as a UML class diagram.

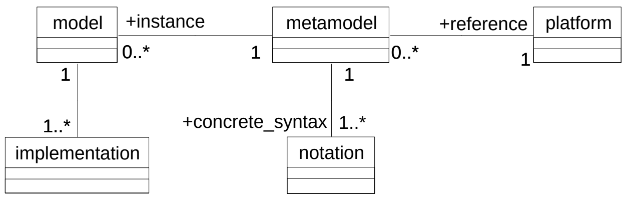

Fig. 2. Reference model for model-driven systems

The types and relationships have the following meaning: A model represents a software system at an appropriate level of abstraction or from a certain perspective. One or more implementations are generated from a model. A metamodel defines the syntax of a class of models. Every metamodel refers to a particular platform, called its reference platform (reference). A platform is an execution environment for software systems, like the Java platform. The semantics of a metamodel is defined through transformation rules that map every language construct to constructs in the reference platform, e.g. that a UML class is transformed to a Java class. There are one or more notations for every metamodel. A notation defines the concrete format used to represent models, which are instances of the respective metamodel. There are textual as well as graphical notations. A UML Profile is an example of a graphical notation to be used in a UML tool.

3 SecureUML Overview

In this section we give an overview of the goals and the structure of SecureUML. We explain the metamodel and the notation in detail in Sections 4 and 5.

SecureUML is a modeling language that defines a vocabulary for annotating UML-based models with information relevant to access control. It is based on the model for RBAC as defined in Section 2.1, with additional support for specifying authorization constraints. SecureUML defines a vocabulary for expressing different aspects of access control, like roles, role permissions and user-role assignments. Due to its general access-control model and extensibility, SecureUML is well suited for business analysis as well as design models for different technologies. Our goal is to use this language as part of another modeling language, called the *host language*, to cover access control aspects. In this way, different models at different abstraction levels can be annotated with access control information using the same syntax and a compatible semantics.

The structure of the modeling language conforms to the reference model for model-driven systems defined in Section 2.4. The *metamodel* defines the abstract syntax of the language, i.e. the structure of a model representation that is independent of particular notation. Our *notation* used to enrich class models in UML is defined as a *UML Profile*.

To begin with, the semantics of the language is defined using informal transformation rules referring to the standard model given in Section 2.4, e.g. a role in the model is transformed to a single entry of the set ROLES. The usage of the standard model for RBAC enables us to give SecureUML a general semantics independent of any particular technology. Afterwards, when the language is included into a host language, the semantics is refined. In our work, refinement means that the specification of the language semantics is made more precise and is adapted to the vocabulary used by the security platform of the host language, e.g. a role in the model is transformed to a single *deployment descriptor element of type* security-role. Informal, imperative, or declarative techniques could be used for the refinement. The resulting language is called a *SecureUML dialect*.

Suppose, for example, we want to use SecureUML to annotate platform dependent design models used to generate executable systems. In this case, the semantics can be refined by using imperative generation rules that define the transformation of the syntactical elements of the model to constructs of the target platform. In this context, it is also possible to parameterize the language to meet the needs for the particular environment using a so-called *resource type model* (see below).

4 SecureUML Metamodel

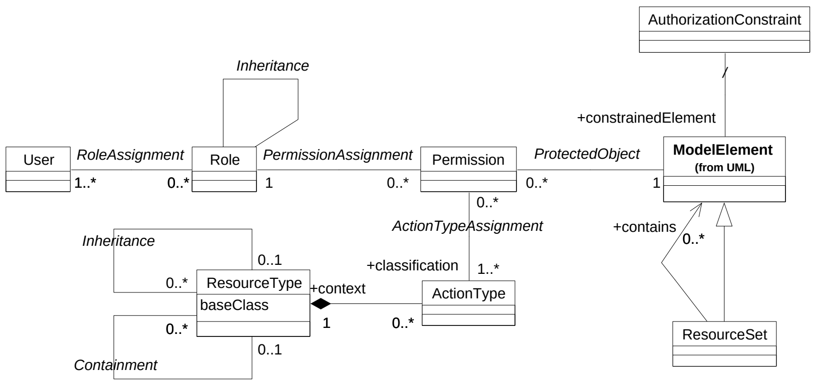

Fig. 3. SecureUML Metamodel

The SecureUML metamodel, shown in Figure 3, is defined as an extension of the UML metamodel. The concepts of RBAC are represented directly as metamodel types. We introduce the new metamodel types User, Role and Permission as well as relations between these types. Due to the design goal given in Section 3, protected resources are represented in a different way. Instead of defining a dedicated metamodel type to represent them, we allow every UML model element to take the role of a protected resource. Additionally, we introduce the type ResourceSet, which represents a user defined set of model elements used to define permissions or authorization constraints.

A Permission is a relation object connecting a role to a ModelElement or a ResourceSet. The semantics of a permission is defined by the ActionType elements used to classify the permission (see the association ActionTypeAssignment). Every ActionType represents a class of security relevant operations on a particular type of protected resource. A method with the security relevant action `execute` or an attribute with the actions `change` and `read` are examples of this. In our modeling language, there is a corresponding action type for every class of such actions. Action types may also represent more conceptual classes of operations at a higher abstraction level. A class may contain methods and attributes and we can attach a permission to the class with an action type `read`. This action type might represent the permission to invoke all side effect free methods and to read the values of all attributes of this class. Action types give the developer a vocabulary to express permissions at a level close to the domain vocabulary.

The set of action types available in the language can be freely defined using ResourceType elements. A ResourceType defines all action types available for a particular metamodel type. The connection to the metamodel type is represented by the attribute baseClass, which holds the name of a type or a stereotype. The set of resource types and their action types, and the definition of their semantics on a particular platform, define the resource type model for the platform.

An AuthorizationConstraint is a part of the access control policy of an application. It expresses a precondition imposed on every call to an operation of a particular resource, which usually depends on the dynamic state of the resource, the current call, or the environment. Suppose, we want to define an access control condition stating that the access to the method `makeAppointment()` on the class `Calendar` is limited to business hours only. To achieve this goal, an authorization constraint is created whose expression refers to the resource's local time. In an operational view, a violation of such a constraint might result in an exception signaling the denial of access. AuthorizationConstraint is derived from the UML core type Constraint. Such a constraint is attached either directly or indirectly, via a permission, to a particular model element representing a protected resource[2].

5 UML Profile for SecureUML

We illustrate the UML Profile using an example for EJB. We define a simple modeling language for EJB as the host language and present a small scheduler application expressed in this language. We show how the host language is enhanced by a resource type model for EJB and explain the semantics of the SecureUML language constructs as well as the refined semantics for the projection to EJB.

5.1 Example: The SecureUML dialect for EJB

In our EJB modeling language, EJB components are modeled as UML classes with the stereotype «ejb» (see Figure 5). An EJB class may contain methods and attributes. A method in the model causes the generation of a business method stub for the EJB component. Attributes are mapped to a member holding the state of the attribute and two access methods get<AttributeName> and set<AttributeName>. The EJB standard requires an EJB component to implement some default methods accessible for clients, for instance, a "finder" method that finds instances by their primary key and methods to delete instances. Since these methods must always be present in the EJB implementation, they can be omitted in the model and are generated automatically. We show in our example how access control information can be expressed even for such "invisible" methods.

We define three resource types and corresponding action types for EJB components, methods and attributes. As shown in Figure 4, a ResourceType is defined as a class with the stereotype «secuml.resourceType» and an ActionType is a class with the stereotype «secuml.actionType» . All action types belonging to a resource type are embedded in this type as nested classes. The semantics of the action types are given in the context of their usage in the example.

[2] The standard UML association between Constraint and ModelElement is used to attach the constraints.

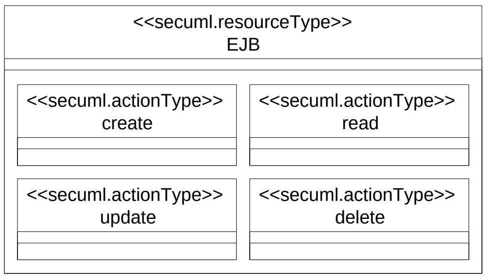

Fig. 4. resource type definition for EJB components

The resource type model is included in every design model defined with the EJB language. In this way, action types can directly be referenced by any permission in the model.

5.2 Scheduler Example

The example application, shown in Figure 5, consists of two components: `Calendar` and `Entry`. A calendar may contain several entries, each representing an appointment with a start and end date and a location. Every entry is owned by a user whose name is stored in the attribute `owner`. The additional constructs shown in the diagram are used to express access control information. We cover all SecureUML constructs in the following subsections.

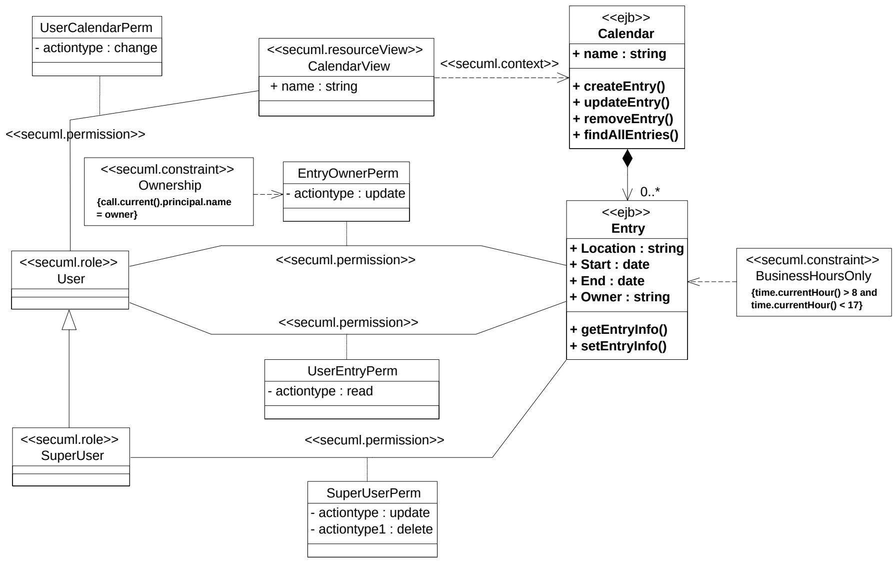

Fig. 5. Example: Secure Scheduler

5.3 Roles

A role is represented by a UML class with the stereotype «secuml.role». In our example, the two roles `User` and `SuperUser` are modeled. In the projection to EJB, each role model element is transformed to a role definition in the deployment descriptor as shown in the following example code:

```
<security-role>
  <role-name>User</role-name>
</security-role>
```

An inheritance relationship between roles is represented by a standard UML generalization relationship. This relation results in an entry for the relation Role Hierarchy in the general semantics. In our example, the role `SuperUser` is derived from the role `User`. Since there is no direct representation of a role hierarchy in EJB, all permissions generated for a super-role are also generated for its sub-roles.

5.4 Permissions

A permission is drawn as an association class with the stereotype «secuml.permission». In the simplest case, the association is bound to a type representing one or more protected resources. An action type used to classify the permission is referenced as the type of an attribute of the permission class. The name of this attribute is not interpreted. The number of referenced action types is unlimited, but all action types must be compatible with the type of the model element referenced by the permission. The necessary type information is defined in the corresponding resource type. Each assigned action type represents a set of permissions in terms of the reference platform. If multiple action types are assigned to a model permission, the union of these sets is taken.

In the example shown in Figure 5, the permission `UserEntryPerm` defines a permission for the role `User` to access the EJB `Entry` with the action type `read`. In the transformation to the EJB security platform, this means that method permissions for all side-effect free methods of the referenced EJB will be generated. These are the standard finder methods, the read-methods of all attributes, and all methods explicitly marked as side-effect free.[3] In this example, six method-permission entries are generated in the deployment descriptor.

```
<method-permission>
  <role-name>User</role-name>
  <method>
    <ejb-name>Entry</ejb-name>
    <method-intf>Home</method-intf>
    <method-name>findByPrimaryKey</method-name>
  </method>
   ...
</method-permission>
```

[3] The standard UML-Attribute BehavioralFeature.isQuery is used for that purpose. In the example model, the method `Entry.getEntryValue` is marked as a query.

This example demonstrates the expressiveness of action types as well as the possibility of expressing permissions for “invisible” methods. Although the finder method `findByPrimaryKey` is not represented in the model, a corresponding permission is generated since this method belongs to the class of side-effect free methods selected by the action type `read`.

Since it is syntactically impossible to address single attributes or methods using the construct above, we complement action types with *resource views*. This is a realization of the metamodel type ResourceSet that defines a subset of features of a particular type. It is used as target of permissions as well as authorization constraints. A resource view is modeled as a UML class with the stereotype «secuml.resourceView». The context type is referenced using a dependency with stereotype «secuml.context». For every feature of the context type contained in the set defined by the resource view there is a feature contained in the resource view model element with the same signature. In our example (see Figure 5), the resource view `CalendarView` references the single attribute `name` of the type `Calendar`. The permission `UserCalendarPerm` defines a permission for the role `User` on that attribute with the action type change. In the generated EJB infrastructure, this will result in a method permission granting the role `User` the permission to execute the set-method of the attribute `name`, i.e. the method `setName(String)`.

5.5 Authorization Constraints

An authorization constraint is defined as a UML constraint with the stereotype «secuml.constraint» as shown in Figure 5. We anticipate that the same authorization constraint is, in most cases, imposed on several or all methods of a type. Therefore, our notation is optimized for such cases. Authorization constraints are expressed using OCL where the *context* element of the expression is omitted and is computed as follows. The authorization constraint is bound to a type using a UML dependency. In terms of the metamodel this means that all, or a subset of, the methods of the type are restricted by the constraint. We call this the “affected methods” of the constraint. The UML notation is transformed to an internal view by creating a single OCL constraint for every affected method, including a corresponding *context* element for that method (see below).

Authorization constraints have access to the application model as well as to the respective platform model. The platform model defines an abstract API to access information in the execution environment. We have defined types to access both the current time and date and information from the security system, like the principal name and the roles of a caller. These types are defined in a default model, which is part of every application model. During the generation of authorization constraint expressions, all calls to the abstract API are substituted by API calls of the concrete platform.[4]

Authorization constraints can be directly bound to a class or a resource view using a UML dependency. In this case, we speak of a “class bound” constraint. Such an authorization constraint represents preconditions for every method generated for the type or the subset of methods defined by the resource view. An application is given in Figure 5 by the authorization constraint `BusinessHoursOnly`. It restricts the access to

[4] We omit the platform model due to space restrictions.

all attributes and methods of the EJB component `Entry` to legal business hours. The expression

```
time.currentHour() > 8 and time.currentHour() < 17
```

is transformed to standard OCL preconditions for every affected method as shown below for one of these methods:

```
context Entry::getEntryInfo():EntryInfo
pre: time.currentHour() > 8 and time.currentHour() < 17
```

An authorization constraint can also be bound to a permission. This form of authorization constraints is designated as a “permission bound” constraint. Such a constraint defines additional restrictions on this permission. Logically, the role permission and the constraint are conjoined. The resulting condition is used as a precondition to all methods affected by the role permission. Figure 5 shows an example constraint bound to a permission.

We want to restrict the access to all update methods on `Entry` objects to their respective owners. We define the role permission `EntryOwnerPerm` with the action type `update` and attach the constraint `Ownership`. The action type `update` limits the permission to all set-methods of attributes and all methods that are not side-effect free. The constraint `Ownership` uses the platform model to access the name of the current caller and compares it to the value of the attribute `owner`.

```
call.current().principal.name = owner
```

The permission and the authorization constraint are transformed to the following combined precondition.

```
context Entry::setLocation(newValue:String):void
pre: call.current().principal.isInRole("User") and
    call.current().principal.name = owner
```

The role permission is transformed to an OCL expression validating the role using the class `call` of the platform model and is conjoined with the expression of the constraint `Ownership`.

For both the analysis of the model and system generation, it is useful to construct one predicate per protected resource formalizing the access control conditions defined by all permissions or authorization constraints for that particular protected resource. This overall access control predicate is defined as a OCL precondition built by the following rule:

$$(permExpr_1 \textbf{ or } ... \textbf{ or } permExpr_n) \textbf{ and } constrExpr_1 \textbf{ and } ... \textbf{ and } constrExpr_n$$

That is, all permission expressions are disjunctively combined. The resulting term is conjoined with all class bound authorization constraints affecting the particular method. The following example shows the complete access control precondition for the method `Entry::setEntryInfo` that is affected by the permissions `EntryOwnerPerm`, `SuperUserPerm` and the authorization constraint `BusinessHoursOnly`.

```
context Entry::setEntryInfo(EntryInfo):void
pre: (call.current().principal.isInRole("SuperUser") or
(call.current().principal.isInRole("User") and
 call.current().principal.name = owner) and
(time.currentHour() > 8 and time.currentHour() < 17)
```

5.6 User-Role-Assignments

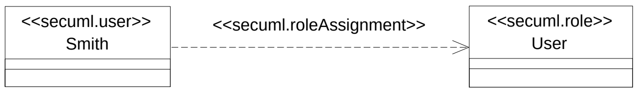

Fig. 6. User-Role-Assignment

Users are represented as classes with the stereotype «secuml.user» as shown in Figure 6. The assignment of a role to a user is defined using a dependency relationship with the stereotype «secuml.roleAssignment». Note that the EJB specification [11] does not prescribe a standard format for assigning roles to users; therefore each EJB product uses a proprietary syntax. Below, we give an example for the product BEA WebLogic Server [1].

```
<security-role-assignment>
  <role-name>User</role-name>
  <principal-name>Smith</principal-name>
</security-role-assignment>
```

6 Conclusion

6.1 Discussion

We have presented SecureUML, a modeling language designed for integrating the specification of access control into application models. The language builds on the access control model of RBAC with additional support for specifying authorization constraints. We have shown how SecureUML can be used in the context of a model-driven software development process to generate access control infrastructures. In this way, productivity during the development and the quality of the resulting systems can be improved.

We have validated the concepts presented in this paper using a prototypical generator for EJB. With this prototype, it is possible to generate EJB applications with full configured role-based access control, including role definitions, method permissions, user-role assignments and authorization constraints. We used ArcStyler^{TM} as a tool for modeling and generation, which is an MDA compliant development environment for component-based systems based on Rational Rose^{TM}. We used the template-based generator of ArcStyler^{TM} (see [5]) to implement our generation rules.

With our prototype, we developed case studies for a banking and a scheduling application and the latter is used as example in this paper. Our experiments have shown the feasability of expressing access control policies with UML in a manner that is concise and precise. The information necessary to realize such a policy in an EJB infrastructure is at a lower level of abstraction and thus significantly more detailed. For example, the generated access control policy for the scheduling application consists of about 220 lines of deployment descriptor source and 50 lines of Java code.

In particular, the concept of action types contributes to the problem solution. Permissions can be expressed in an intuitive vocabulary, like “a role is granted *read* access to a component”. Furthermore, permissions are adapted dynamically to changes of the object model due to the filter effect of action types. This reduces, in comparison to explicit sets like resource views, the possibility of inconsistencies between the application model and the access control policy model.

We want to emphasize that due to its general access control model, SecureUML is well suited for defining access control policies for every security architecture supporting role-based or programmatic access control. The language is adapted to a particular security architecture by defining a corresponding resource type model that identifies the model element types representing protected resource for this architecture and its actions relevant to access control. Based on the resource type model, the semantics of the language is refined in terms of the target architecture. In this paper we have shown this approach on the example of the EJB security architecture.

Our work also shows that OCL is well suited for formalizing authorization constraints. OCL expressions can refer to all application model types and allow considerable flexibility in defining constraints. Since OCL is a first-order language, constraints can be incorporated into a formal analysis of UML models (see [2]).

6.2 Related Work

In the area of security modeling with UML, Epstein and Sandhu propose in [3] a UML based notation for access control. In contrast to our approach, they do not cover the generation of secure systems from models. Apart from this, there are similarities, but also differences, in the notation. In particular, their concept of a *set handler* is similar to our resource view. In contrast, we complement resource views with action types. These are used to define dynamic filters over sets of operations. Epstein and Sandhu furthermore propose to define constraints for single roles or operations in an informal notation. We use formal constraints expressed in OCL, which can be bound to a set of operations or permissions.

Jürjens proposes in [7] a concept for specifying requirements on confidentiality and integrity in analysis models based on UML. The underlying security models are Multi-Level Security and Mandatory Access Control. In contrast, our work focuses on the design phase and builds on RBAC as a security model.

In the area of models for RBAC, Jaeger proposes in [6] the introduction of constraints in RBAC that restrict the assignment relationship between roles and users. This mechanism can be used to realize the concept of “separation of duty”. Such constraints are also defined in [4]. Our authorization constraints are a complementary concept used for restricting the execution of an operation in a system state dependent way. Furthermore, an authorization constraint as proposed in this paper is a UML modeling construct and not an extension to the RBAC model in general.

6.3 Future Work

Currently our approach focuses on static design models, which are relatively close to the implementation. It is worth considering whether the efficiency of the development process of secure applications can be improved by annotating models at a higher level of abstraction (e.g. analysis) or by annotating dynamic models, e.g. state machines. Moreover, some critical questions concerning the development process are still open, e.g. how are roles and permissions identified? Beyond that, the current prototype does not yet demonstrate the platform independence of our concepts.

Future work will focus on modeling security requirements and design information using dynamic UML models. Furthermore, the development process for secure systems starting with the initial analysis up to the complete secure system design will be investigated. In this context, we will examine the possibility of propagating security requirements between analysis and design models and ways to verify the compatibility of requirements and design information given at different levels.

References

1. BEA Systems, Inc. *Programming WebLogic Enterprise JavaBeans*, 2002. `http://e-docs.bea.com/wls/docs61/pdf/ejb.pdf`.
2. A. D. Brucker and B. Wolff. A Proposal for a Formal OCL Semantics in Isabelle/HOL. In C. Muñoz, S. Tahar, and V. Carreño, editors, *TPHOLs 2002*, LNCS. Springer-Verlag, 2002.
3. P. Epstein and R. Sandhu. Towards a UML based approach to role engineering. In *Proceedings of the fourth ACM workshop on Role-based access control*, pages 135–143. ACM Press, 1999.
4. 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 System Security (TISSEC)*, 4(3):224–274, 2001.
5. Interactive Objects Software GmbH. *ArcStyler Extensibility Guide*, 2002. `http://www.io-software.com/as_support/docu/extensibility_guide.pdf`.
6. T. Jaeger. On the increasing importance of constraints. In *Proceedings of the fourth ACM workshop on Role-based access control*, pages 33–42. ACM Press, 1999.
7. J. Jürjens. Towards development of secure systems using UMLsec. In H. Hussmann, editor, *Fundamental Approaches to Software Engineering, 4th International Conference, Proceedings*, LNCS, pages 187–200. Springer, 2001.
8. T. Koch, A. Uhl, and D. Weise. Model Driven Architecture. Technical report, Interactive Objects Software GmbH, 2002. `http://cgi.omg.org/cgi-bin/doc?ormsc/02-01-04.pdf`.
9. Object Management Group. *OMG Unified Modeling Language Specification, Version 1.4*, 2001. `http://www.omg.org/technology/documents/formal/uml.htm`.
10. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
11. Sun Microsystems, Inc. *Enterprise JavaBeans Specification, Version 2.0*, 2001. `http://java.sun.com/ejb/docs.html`.