Complex Systems Informatics and Modeling Quarterly
CSIMQ, Issue 3, July, 2015, Pages 35-62
Published online by RTU Press, https://csimq-journals.rtu.lv
http://dx.doi.org/10.7250/csimq.2015-3.03
ISSN: 2255-9922 online

A Model-driven Role-based Access Control for SQL Databases

Raimundas Matulevičius and Henri Lakk

Institute of Computer Science, University of Tartu, J. Liivi 2, 50409 Tartu, Estonia

rma@ut.ee, henri.lakk@gmail.com

Abstract. Nowadays security has become an important aspect in information systems engineering. A mainstream method for information system security is Role-based Access Control (RBAC), which restricts system access to authorised users. While the benefits of RBAC are widely acknowledged, the implementation and administration of RBAC policies remains a human intensive activity, typically postponed until the implementation and maintenance phases of system development. This deferred security engineering approach makes it difficult for security requirements to be accurately captured and for the system's implementation to be kept aligned with these requirements as the system evolves. In this paper we propose a model-driven approach to manage SQL database access under the RBAC paradigm. The starting point of the approach is an RBAC model captured in SecureUML. This model is automatically translated to Oracle Database views and instead-of triggers code, which implements the security constraints. The approach has been fully instrumented as a prototype and its effectiveness has been validated by means of a case study.
Keywords: Model-driven security, Role-based Access Control, SecureUML, PL/SQL, updatable view, instead-of trigger.

1 Introduction

Security engineering is an engineering discipline within system engineering "concerned with lowering the risk of intentional unauthorized harm to valuable assets to level that is acceptable to the system's stakeholders by preventing and reacting to malicious harm, misuse, threats, and security risks" [16]. Developing a secure system correctly is difficult and error-prone. It is not enough to ensure correct functioning of security mechanisms used; they cannot be "blindly" inserted into a security-critical system. It is observed in [19], [42] and [43] that while functional requirements are generally analysed during *requirements engineering* and *design* stages, security considerations often arise most usually during *implementation* or *maintenance* stages. Firstly, this means that security engineers get little feedback about the secure functioning of the system in practice, since security violations are kept secret for fear of harming an organisation's reputation. Secondly, security risks are very hard to calculate: security-critical systems are characterised by the fact that the occurrence of a successful attack at one point in time on a given system increases the likelihood that the attack will be launched subsequently at another system's point. This is a serious hindrance to secure system development, since the early consideration of security (e.g., at the requirements and/or design stages) allows engineers to envisage threats, their consequences, and design countermeasures; and thus design alternatives, which do not offer a sufficient security level, can be discarded.

We propose to solve the above problem by using model driven architecture (MDA). MDA prescribes the system development process based on models [33]. The models, the simplified representations of reality, can be looked at from different perspectives (e.g., problem domains, architectural solutions), studied for different purposes (e.g., analysis of problems, evaluation of architectural solutions), and their evolution and transformation can address different objectives (e.g., integration of technical concepts, transformations between different modelling languages). Security modelling languages support understanding of the security concerns through discovery of all the important security requirements and relevant domain properties. In other words, they only support validity criteria with respect to the stakeholder needs. However, in general, they provided limited support for transformation of the security model to code. Thus, it requires an additional effort of system developer to *verify* the implemented security concerns. In nowadays information systems, databases still remain the key technology to gather, store, and manage the business data. The database logical structure is defined with the standard query language (SQL). Although transformation of a structural data model (e.g., expressed in UML class diagram or Entity Relationship diagram) to SQL code is highly supported by the variety of the modelling tools, we did not observe that graphical security models could be translated to mission-critical constraints. Typically, the implementation of these constraints remains a programmer's job. However, this is a labour intensive activity and requires a thorough *validation* of the code.

In this paper we present a set of *transformation templates* that help to translate the *security model* expressed in SecureUML [4], [25], to *security constraints* based on database views and instead-of triggers. These security constraints are applied to the SQL database schema (which could be also generated from, e.g., the UML class diagram) to enforce the *role-based access control rules* to the *secured data*. By our approach we, firstly, remove necessity to verify security concerns at the implementation level, because all the security complexity is modelled during the system design stage. Secondly, the necessity of the code validation is also abandoned, because the code is automatically generated from the SecureUML security models.

Although the transformation templates and examples given in the paper are Oracle DBMS specific, the approach is not limited to it. Oracle was chosen as the base system by our industrial partner. The generated code can be applied to any relational database that supports views instead-of triggers, and packages. But using the generated code on other than Oracle databases may require some modifications depending on the differences in the syntax.

In order to validate our proposal we compared two security models: one directly created in PL/SQL [15], another created in SecureUML and then transformed into database views and instead-of triggers code.The second model had higher quality than the first one.

This paper is organised as follows: Section 2 gives the background for our research. It focuses on the principles of Role-based Access Control and introduces the SecureUML language. In Section 3 we discuss how to transform the data model to the data SQL code, and how to translate the security model into authorisation constraints using database views and instead-of triggers. Section 4 presents a case study. Finally, Section 5 discusses the contribution of presented research work and concludes the paper.

2 Background

2.1 Role-based Access Control

The core RBAC model [14], [40] is shown in Figure 1. It includes five major concepts: *Users*, *Roles*, *Objects*, *Operations*, and *Permissions*. A *User* is defined as a human being but this concept could also be extended to machines, networks, or intelligent autonomous agents. A *Role* is a job function within the context of an organisation. Some associated semantics include authority and responsibility conferred on the user assigned to the role. *Permission* is an approval to perform an operation on one or more protected objects. An *Operation* is an executable image

of a program, which upon invocation executes some function for the user. Hence, the operation types and secured objects depend on the type of the system where they are implemented. *User assignment* and *permission assignment* are many-to-many relationships. The first describes how users are assigned to their roles. The second characterises the set of privileges assigned to a role.

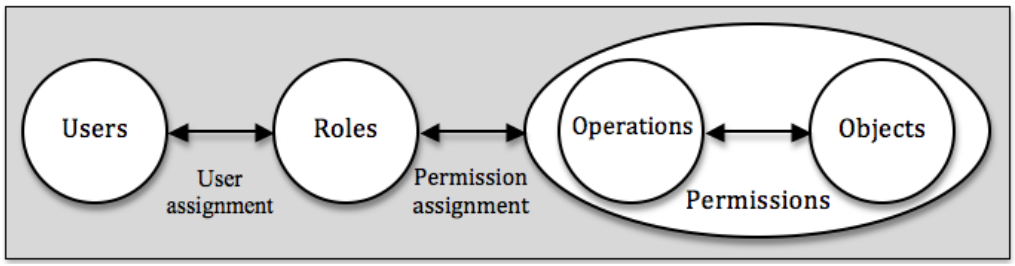

Figure 1. The core RBAC model (adapted from [14] and [40])

The basic concept of RBAC is that *users* are assigned to *roles*, *permissions* are assigned to *roles*, and *users* acquire *permissions* by being members of *roles*. The same *user* can be assigned to many *roles* and a single *role* can have many *users*. Similarly, for *permissions*, a single *permission* can be assigned to many *roles* and a single *role* can be assigned to many *permissions*.

2.2 SecureUML

A modelling language is characterised through three major components abstract syntax, concrete syntax, and semantics. We will discuss each of these components in order to present SecureUML [4], [27].

2.2.1 SecureUML Abstract Syntax

An abstract syntax of SecureUML [4], [25] is organised as a UML class diagram and is displayed in Figure 2. It adapts the principles of the RBAC model, and introduces concepts like *User*, *Role*, and *Permission* as well as relationships *RoleAssignment* and *PermissionAssignment*. Here secured objects and operations are expressed through protected objects, which are modelled using the standard UML constructs (e.g., see concept of *ModelElement*). In addition, *ResourseSet* represents a user defined set of model elements used to define permissions and authorisation constraints.

The semantics of *Permission* is defined through *ActionType* elements used to classify permissions. Here every *ActionType* represents a class of security-relevant operations on a particular type of protected resource. In Figure 3 we introduce four specific security actions: *Select*, *Update*, *Insert*, and *Delete*, which we will define later in Section 4.

An *AuthorisationConstraint* expresses a precondition imposed to every call to an operation of a particular resource. This precondition usually depends on the dynamic state of the resource, the current call, or the environment. The authorisation constraint is attached either directly or indirectly to a particular model element that represents a protected resource.

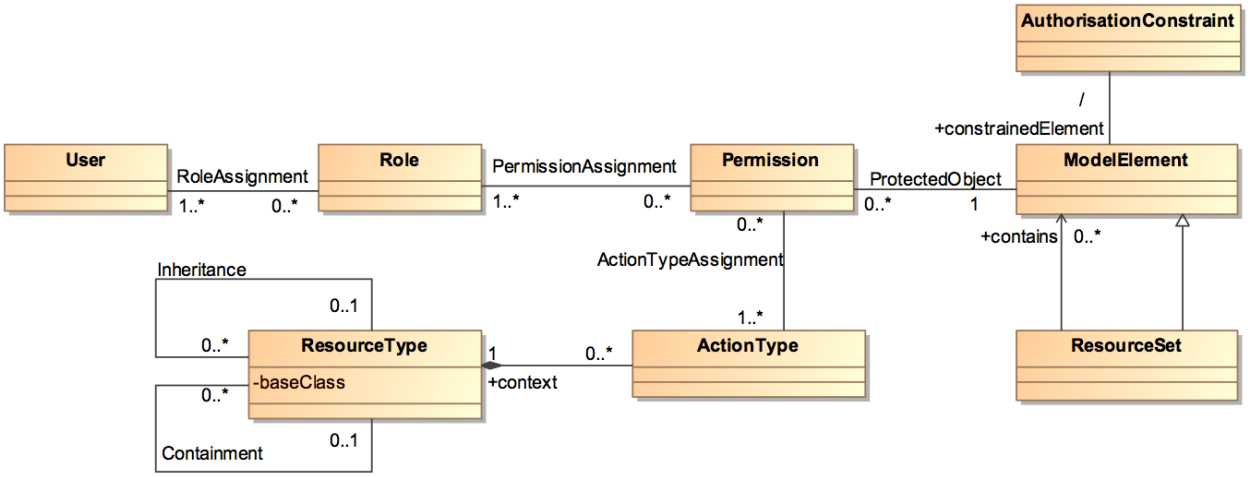

Figure 2. SecureUML meta-model (adapted from [4] and [25])

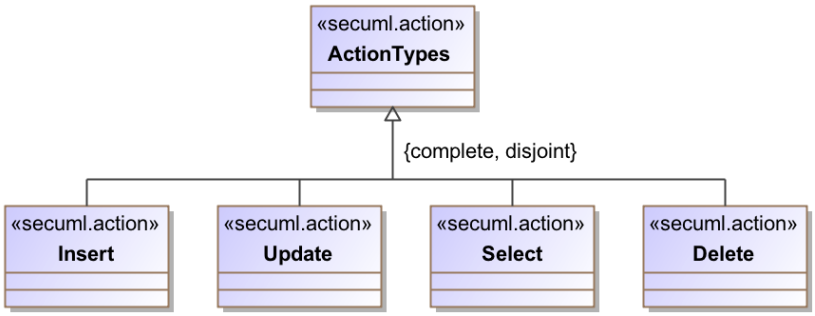

Figure 3. Action types for the security permission

2.2.2 SecureUML Concrete Syntax

At the concrete syntax level SecureUML is a "lightweight extension" of UML, namely, through stereotypes, tagged values, and constraints. The stereotypes are defined for the classes and relationships in the class diagrams are specifically oriented to the RBAC terminology. In Figure 4 we illustrate the SecureUML concrete syntax through the *Meeting Scheduler* example [13].

In Figure 4 we define a secure resource *Meeting*, which is characterised by a *place* where, and *time* when a meeting should be organised. These data needs to be secured from unintended audience. Thus, a certain restriction on changing the resource state (changing the value of the attributes *place* and *time*) needs to be defined for the roles *MeetingInitiator* and *MeetingParticipant*.

Association class *InitiatorPermission* characterises three actions allowed for the *MeetingInitiator*:

- action *enterMeetingDetails* (of type *Insert*) defines that *MeetingInitiator* can enter *place* and *time* by executing operation *setTimePlace*() (see class *Meeting*);
- action *changeMeetingInformation* (of type *Update*) allows changing *place* and *time* of the *Meeting* by executing operation *changeTimePlace*() (see class *Meeting*);
- action *deleteMeetingAgreement* (of type *Delete*) permits deleting *place* and *time* of the *Meeting* by executing operation *deleteTimePlace*() (see class *Meeting*).

Similarly, the association class *ParticipantPermission* defines a restriction for the *MeetingParticipant* role. It introduces an *action getMeetingDetails* (of type *Select*) that says that only *MeetingParticipant* can perform *viewTimePlace*().

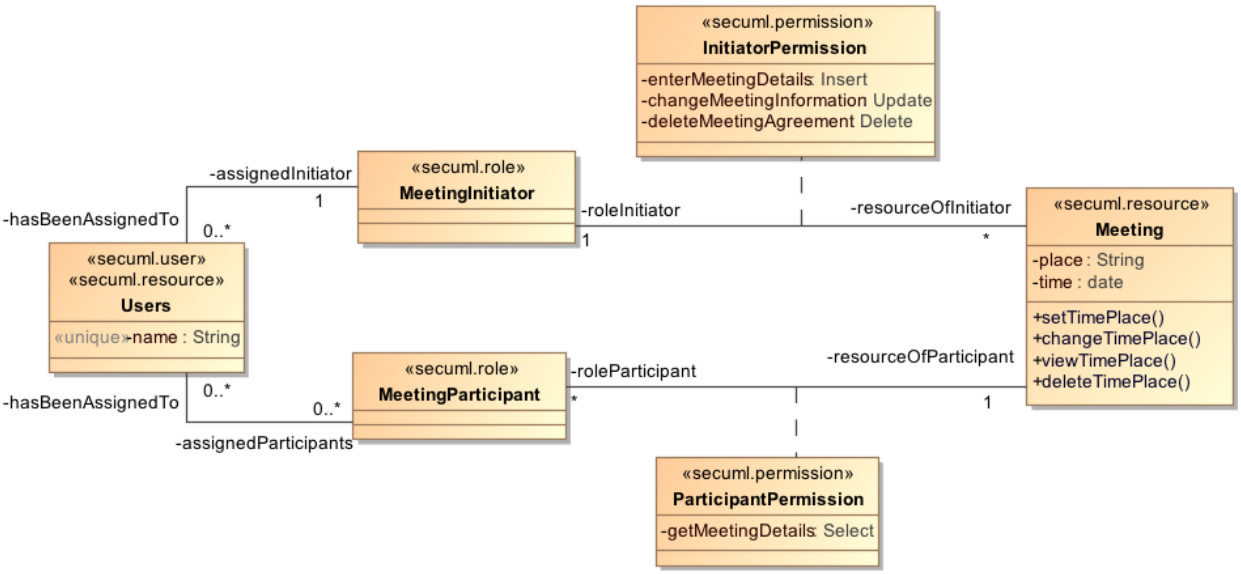

Figure 4. Meeting Scheduler with SecureUML

To strengthen these four permissions we define authorisation four constraints, written in object constraint language (OCL) [45]:

AC#1:

```
context MeetingAgreement::setTimePlace():void
pre: self.roleInitiator.hasBeenAssignedTo=caller
```

AC#2:

```
context MeetingAgreement::changeTimePlace():void
pre: self.roleInitiator.hasBeenAssignedTo=caller
```

AC#3:

```
context MeetingAgreement::deleteTimePlace():void
pre: self.roleInitiator.hasBeenAssignedTo=caller
```

AC#4:

```
context MeetingAgreement::viewTimePlace():void
pre: self.roleParticipant.hasBeenAssignedTo=caller
```

Authorisation constraint AC#1 means that operation *setTimePlace*() can be executed by a user set defined as variable caller, that is assigned to be *MeetingInitiators*. Similarly, the authorisation constraint AC#2 defines restriction for operation *changeTimePlace*(), AC#3 for operation *deleteTimePlace*(), and AC#4 for operation *viewTimePlace*().

2.2.3 SecureUML Semantics

In [4] semantics of Secure UML is formalised to satisfy two purposes: (*i*) to define a declarative access control decisions that depend on static information, namely, the assignments of users and permissions to roles and (*ii*) to support implementation-based access control decisions that depend on dynamic information, namely, the satisfaction of authorisation constraints in the current system state. Similarly to [1] and [11] we discuss the conceptual SecureUML semantics for the system modelling purpose. We utilise the RBAC model to define semantics of the SecureUML constructs [27] [28] as illustrated in Table 1.

Some mappings between RBAC and SecureUML are understood as a lexical correspondence. For instance, the SecureUML classes with the stereotype *<<securml.user>>* correspond to the RBAC *users*, *<<securml.role>>* — to the RBAC *roles*, and *<securml.permission>>* — to the

RBAC *permissions*. These SecureUML constructs and RBAC concepts are similar according to their textual expression and also their semantic application (see *Meeting Scheduler* in Figure 4).

Table 1. Correspondence between RBAC concepts and SecureUML constructs

RBAC concepts	**SecureUML construct**	***Meeting Scheduler* example**
Users (concept)	Class stereotype *<<securml.user>>*	Class *Users*
User assignment (relationship)	Association between classes with stereotypes *<<securml.user>>* and *<<securml.role>>*	Association relationship [*hasBeenAssignedTo–assignedinitiator*] and [*hasBeenAssignedTo–hasAssignedParticipant*]
Roles (concept)	Class stereotype *<<securml.role>>*	Classes MeetingInitiator and MeetingParticipant
Permission assignment (relationship)	Association class stereotype *<<securml. permission>>*	Operations of association classes *InitiatorPermissions* and *ParticipantPermissions*
Objects (concept)	Class stereotype *<<securml. resource >>*	Class *Meeting*
Operations (concept)	Operations of a class with stereotype *<<securml.resource>>*	Operations *setTimePlace()*, *changeTimePlace()*, and *viewTimePlace()*
Permissions (concept)	Authorisation constraint	AC#1, AC#2, and AC#3

The SecureUML classes with the stereotype *<<securml.resource>>* are used to define objects that need some security protection. This corresponds to the RBAC concept *objects.* Since the value of attributes (that characterise the state of the protected resources) could be changed by the resource (object) operations, these operations (that belong to the classes with the stereotype *<<securml.resource>>*) are understood as the RBAC operations.

To define the RBAC *user assignment* we use the association link between classes with stereotypes *<<securml.user>>* (e.g., *Users*) and *<<securml.role>>* (e.g., *MeetingInitiator*). We define class Users with stereotype *<<securml.user>>* and link this class to the roles (classes with stereotype *<<securml.role>>*) using association links. We map these association relationships to the RBAC *user assignment* relationship. Finally, we define that the SecureUML *authorisation constraints* characterise the RBAC *permissions*. The SecureUML association class with stereotype *<<securml.permission>>* is introduced for this purpose, too.

2.3 Role-Based Access Control in Relational Databases

Like many other available databases (e.g., DB2, MySQL, PostgreSQL, SQLite and others), Oracle DBMS implements the notion of roles [23], [35] and includes the support for administration of the access control state. Oracle has provided security at the table level and, to some extent, at the column level. Privileges may be granted to allow or restrict users to access only some tables or columns. There are two kinds of privileges in Oracle: (*i*) system privileges (e.g., privilege to create new roles) and (*ii*) object privileges (e.g., privilege to insert new records into a table). An object privilege identifies an object, which is either a table or a view, and an access mode, which is one of the following: *select*, *insert*, *update* or *delete*.

Object-level privileges satisfy many requirements, but in some cases they are not granular enough to meet the security requirements that are associated with a company's data. A classic example arises from Oracle's traditional human resources demonstration tables. The employee table contains information about all the employees in the company, but a department manager should only be able to see information about employees in his department. This requirement is not solvable with object-level privileges and therefore is usually implemented in a higher (e.g. business or presentation) tier.

In [39] object-level permissions are used with the old desktop applications, where the applications are connected with the database, but the number of users is limited. This two-tier architecture results in a model, where the user is working with the application layer that interfaces directly with the database layer. This means that the database would directly identify

the computer, the user transactions, and the user herself. Thus, it becomes possible to authorise user and follow up single user transactions in order to discover signs of intrusion as all the transactions of the same user are passed via the same connection.

However, nowadays, web applications are executed at the browsers by sending request to the Web server, which performs transactions to/from the database. As the result of this three (or more) – tier architecture (also called *pooling*), the database is able to identify neither who has accessed the data nor the transaction of the same user. The web application does not open or close a connection before/after each request, but uses a connection pool to store the connections. Using such a connection pool a large number of users can be satisfied with few database connections. However, regarding the database security, the principle of minimal privilege is violated and every connected user has an access to the same data. Such a situation results in the horizontal (i.e., access to the data of other users) and vertical (i.e., access to the department's data) privileges escalation. "Although many advances have been made in developing secure applications, trusting applications, which are developed under time constraints by developers, which are not security experts, presents a large risk to the database and therefore databases are threatened by these applications" [39].

3 Data Role-based Access Control using Model-driven Security

Our proposal to model and implement security policy for the data is discussed in this section and presented in Figure 5. In Section 3.1 we present the transformation of a *data model*, expressed as the UML class diagram, to the data code represented in the SQL code. In Section 3.2 we show how to transform a *security model* defined using SecureUML into the *security constraints* represented with database views and instead-of triggers. Both the SQL code and the security constraints are intertwined together when executing them on the Oracle relational database management system. Finally, in Section 3.3 we overview software tools that we used to support our proposal. (Figure 5).

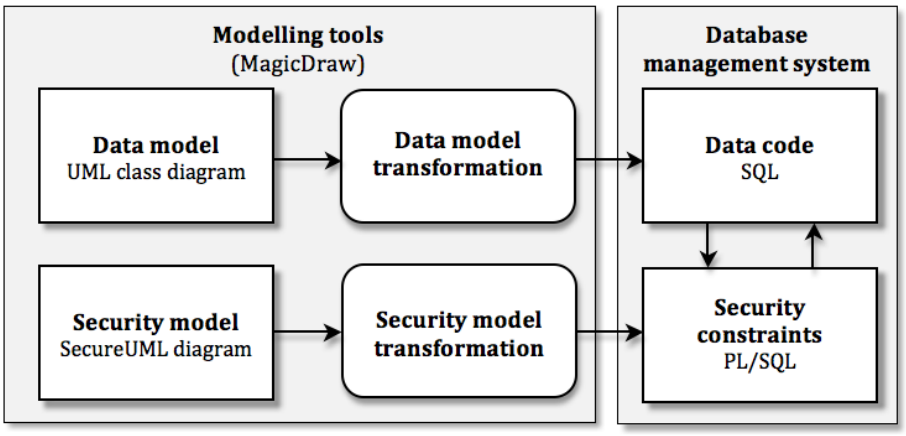

Figure 5. Data and Security Model Transformation

3.1 Data Model Transformation

Transformation of the logical data model to the data model SQL code is supported by majority of the UML modelling tools. Typically this transformation consists of two steps (see Figure 6). The logical data model (e.g., Figure 7) expressed in the UML class diagram is translated to the physical data model. The classes in the physical model (Figure 8) are equipped with the

stereotype <<*table*>> indicating that they represent database tables. Each table is complemented with a primary key attribute (e.g., see Users attribute <<PK>>-id:integer).

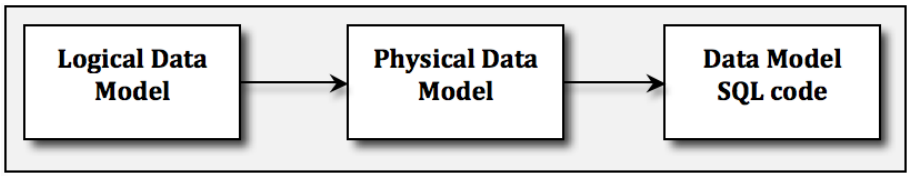

Figure 6. Data Model transformation

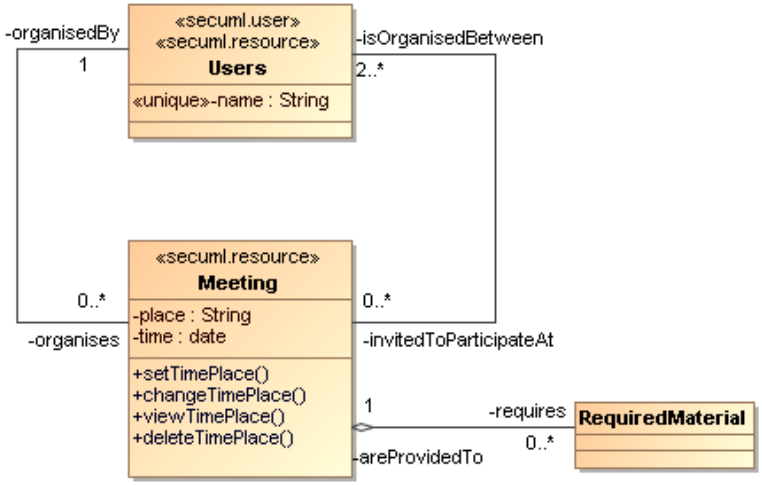

Figure 7. Logical Data Model

The associations of the logical data model are transformed into database foreign keys (i.e., dependencies with stereotype <<*FK*>>). One-to-many associations (e.g., between Meeting and RequiredMaterial) are transformed into a single foreign key. Many-to-many associations from the logical data model are transformed into the physical data model by introducing a new table (e.g., Users_Meeting).

The physical data model (Figure 8) is translated to data definition language (DDL, Figure 9) class by class: for every class with the stereotype <<*table*>> a CREATE TABLE statement is generated. The attribute names and types are transformed respectively into table column names and data types.

3.2 Security Model Transformation

Following the *Meeting Scheduler* example (see Section 2.2) we will illustrate how the RBAC policy defined in the SecureUML security model is transformed into database views and instead-of triggers, which implement the security constraints. In Figure 10 a *secured resources* – objects of class Meeting – are equipped with a stereotype <<*securml.resource*>>. Two roles (classes MeetingInitiator and MeetingParticipant carrying the stereotype <<*securml.role*>>) have different set of *permissions* (association classes MeetingInitiator and MeetingParticipant) to access and modify Meeting values (place and time). Two authorisation constraints – InitiatorAuthConstraint and ParticipantAuthConstraint– restrict the permissions of the defined roles. These constraints are expressed in PL/SQL to simplify the model transformation to the security constraints. The procedures InitiatorAuthConstraint(self.id) and ParticipantAuthConstraint(self.id) are provided in Appendix A and they correspond to authorisation constrainTs AC#1-4. Appendix A also

includes procedure sec.is_role(argument), used to identify role for which security action is granted.

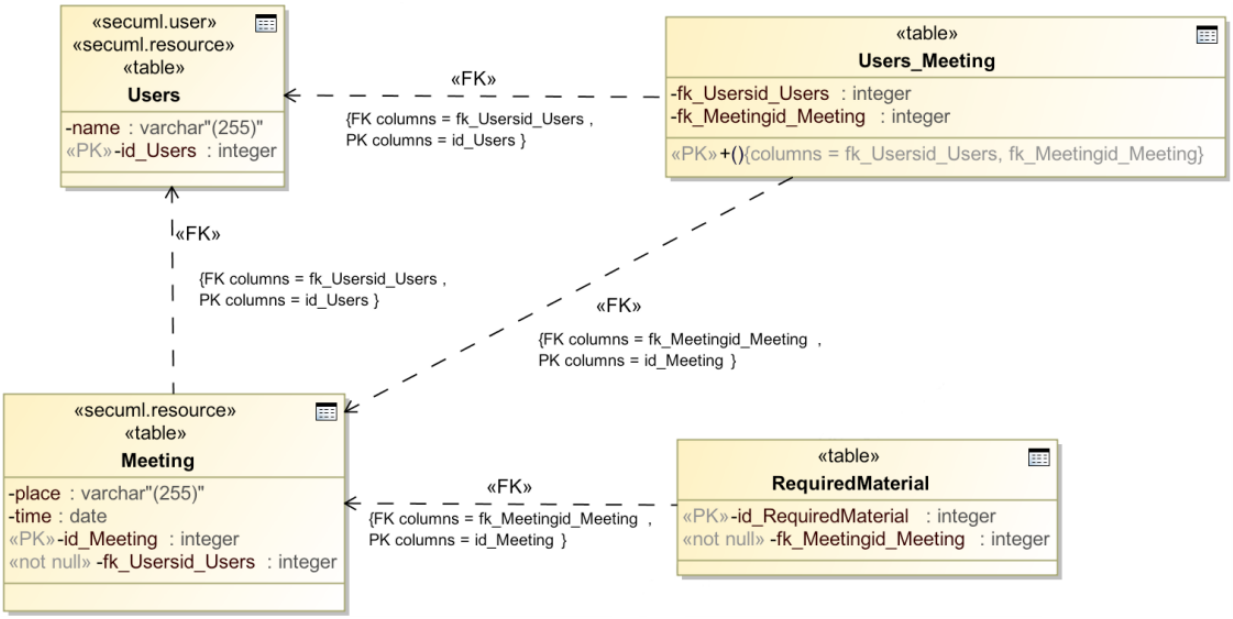

Figure 8. Physical Data Model

```
CREATE SEQUENCE Users_SEQ;

CREATE SEQUENCE Meeting_SEQ;

CREATE SEQUENCE RequiredMaterial_SEQ;

CREATE TABLE Users (
  name VARCHAR (255),
  id_Users INTEGER PRIMARY KEY);

CREATE TABLE Meeting (
  place VARCHAR (255),
  time DATE,
  id_Meeting INTEGER PRIMARY KEY,
  fk_Usersid_Users INTEGER NOT NULL,
  FOREIGN KEY(fk_Usersid_Users) REFERENCES Users (id_Users));

CREATE TABLE RequiredMaterial (
  id_RequiredMaterial integer PRIMARY KEY,
  fk_Meetingid_Meeting INTEGER NOT NULL,
  FOREIGN KEY(fk_Meetingid_Meeting) REFERENCES Meeting (id_Meeting));

CREATE TABLE Users_Meeting (
  fk_Usersid_Users INTEGER,
  fk_Meetingid_Meeting INTEGER,
  PRIMARY KEY(fk_Usersid_Users, fk_Meetingid_Meeting),
  FOREIGN KEY(fk_Meetingid_Meeting) REFERENCES Meeting (id_Meeting),
  FOREIGN KEY(fk_Usersid_Users) REFERENCES Users (id_Users));
```

Figure 9. Database SQL data definition script

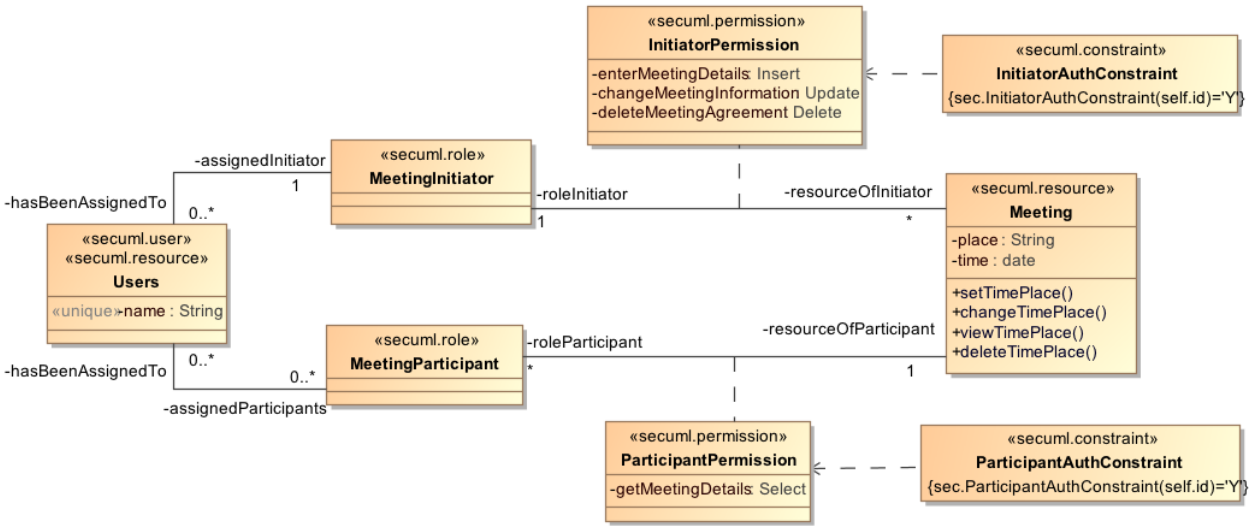

Figure 10. Meeting Scheduler Security Model

In Figure 11 we present how the SecureUML security model is transformed automatically into the PL/SQL security authorisation constraints. Using Velocity[1] template language we have developed security transformation rules and adapted them to the transformation templates (see Appendix B). These rules specify four security actions (see Figure 3) performed on the secured table: (i) *Insert*, for entering new data; (ii) *Update*, for changing the existing data; (iii) *Select*, for viewing existing data; and (iv) *Delete*, for deleting data. By applying the transformation templates the SecureUML security model is systematically translated to database views and instead-of triggers, which implement security authorisation constrains on the secured data. We will illustrate this translation in the Meeting Scheduler example.

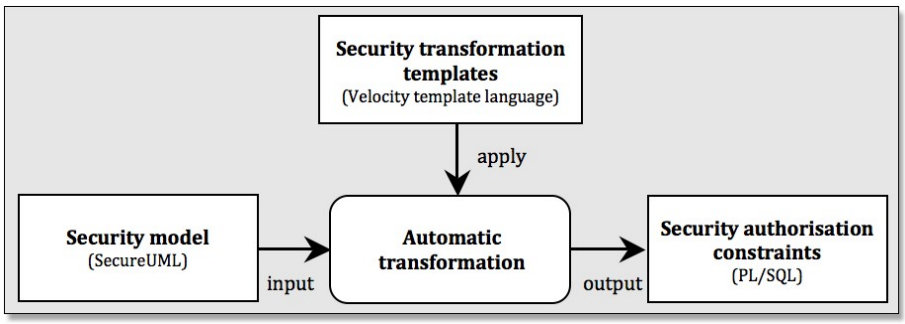

Figure 11. Security model transformation

3.2.1 Select Authorisation Constraint

The *Select* authorisation constraint specifies the data that can be viewed at the runtime. At the data level the secure resources are interpreted as database tables as discussed in Section 3.1. As illustrated in Figure 12, a database view is created for a secured resource (e.g., Meeting), which in SecureUML diagram (see Figure 10) carries stereotype *<<securml.resource>>*. The created view corresponds to the same schema structure, but the unique name is created adding suffixes

[1] http://velocity.apache.org/engine/devel/user-guide.html

"*_v*" to the table names. For instance, for secured resource Meeting we define its data view Meeting_v, which has two secured attributes place and time[2].

For every secured attribute (e.g., name and place) the Select statement performs a conditional check on a Boolean expression. If the check returns TRUE, the attribute value is selected and allowed to view; otherwise the NULL result is provided.

The Boolean expression is a combination of implicit and explicit constraints. The implicit constraint checks the Role assigned to a runtime user. The Role is captured from the SecureUML class with stereotype *<<securml.role>>* and through its association class (carrying stereotype *<<securml.permission>>*) with the secured resource. In Figure 10 (see association class InitiatorPermission) we can see that only MeetingInitiator (as expressed using the authorisation constraint) is allowed to perform Select action on the Meeting's place and data. In the view declaration (see Figure 12) the implicit constraint is defined as function sec.is_role(argument)[3], where argument is the name of the Role (e.g., MeetingInitiator) participating in the *Select* action.

The explicit constraint is defined in the SecureUML classes with the stereotype *<<securml.constraint>>*. These are connected to the security permission and applied on the secured resource. They become part of the Boolean expression, as illustrated in Figure 12 (see, sec.ParticipantAuthConstraint(self.id)).

```
-- Imported common-sql.vtl
CREATE OR REPLACE VIEW Meeting_v
AS
  SELECT
    CASE
      WHEN sec.is_role('MeetingParticipant')     = 'Y' AND
           sec.ParticipantAuthConstraint(self.id) = 'Y'
      THEN self.place
      ELSE CAST (NULL AS VARCHAR2 )
    END AS place ,
    CASE
      WHEN sec.is_role('MeetingParticipant')     = 'Y' AND
           sec.ParticipantAuthConstraint(self.id) = 'Y'
      THEN self.time
      ELSE CAST (NULL AS DATE )
    END AS TIME
  FROM Meeting self
  WHERE sec.is_role('MeetingParticipant')    = 'Y'
  AND sec.ParticipantAuthConstraint(self.id) = 'Y'
/
```

Figure 12. Transformed *Select* authorisation constraint

3.2.2 Delete Authorisation Constraint

The *Delete* authorisation constraint (see Figure 13) specifies, which data a user, assigned to a valid role, can remove. The *Delete* constraint is implemented through an instead of delete trigger (which is assigned a unique name Meeting_delete_trigger) on the database view as described in Section 3.2.1. When an SQL *DELETE* statement is executed on a row of the view, the trigger is executed instead of this statement. The trigger can result in success or failure. In the first case, if

[2] For the technical details we need to define the third attribute, which is the primary key of the secured table, as illustrated in Figure 9. However, none of the security constraints are defined on this attribute, thus, we leave it aside from the discussion.

[3] The implementation of this function may vary depending on the authentication method used (e.g., LDAP or password authentication).

there is no security violation and if the targeted data are not referenced from other sources, the requested data are deleted; otherwise the action results in an exception.

To delete the targeted data, these need to be selected based on their primary key (e.g., res.ID=:OLD.ID in Figure 13). Similarly to the Select constraint, a Boolean expression to check the condition for data deletion consists of the implicit constraints (e.g., sec.is_role(MeetingInitiator), captured from the class MeetingInitiator and association class InitiatorPermission) and explicit constraints (e.g., sec.InitiatorAuthConstraint(self.id) = 'Y', captured from the class InitiatorAuthConstraint, carrying <<*securml.constraint*>>).

```
-- Imported common-sql.vtl
CREATE OR REPLACE TRIGGER Meeting_delete_trg
  INSTEAD OF DELETE ON Meeting_v
  REFERENCING OLD AS OLD
  FOR EACH ROW
DECLARE
  self Meeting%ROWTYPE;
  ex_denied EXCEPTION;
BEGIN
  SELECT * INTO self FROM Meeting res WHERE res.ID = :OLD.ID;
  IF sec.is_role('MeetingInitiator') = 'Y' AND
     sec.InitiatorAuthConstraint(self.id) = 'Y'
  THEN
    DELETE FROM Meeting tbl WHERE tbl.ID = :OLD.ID;
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END
/
```

Figure 13. Transformed *Delete* authorisation constraint

3.2.3 Insert Authorisation Constraint

The *Insert* authorisation constraint (Figure 14) specifies, which new data a User, assigned to a valid Role, can insert into a table.

```
-- Imported common-sql.vtl
CREATE OR REPLACE TRIGGER Meeting_sec_insert_trg
  INSTEAD OF INSERT ON Meeting_v
  REFERENCING NEW AS NEW
  FOR EACH ROW
DECLARE
  ex_denied EXCEPTION;
BEGIN
  IF sec.is_role('MeetingInitiator') = 'Y' AND
     sec.InitiatorAuthConstraint(self.id) = 'Y'
  THEN
    INSERT INTO Meeting ( place , TIME)
    VALUES ( :NEW.place , :NEW.time);
  ELSE
    RAISE ex_denied;
  END IF;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
/
```

Figure 14. Transformed *Insert* authorisation constraint

It is implemented through an instead of insert trigger (e.g., Meeting_sec_insert_trg) on the secured resource. Execution of the trigger results in success or in the failure. If there is no security violation and if new inserted values are valid, trigger will give a positive result. Otherwise the update action will result in an exception.

The *Insert* authorisation constraint is row-based because it is not possible[4] to insert only parts of a row into a table. This means that before the actual *insert* is performed, a Boolean security constraint is checked. The Boolean constraint consists of the implicit role constraint (e.g, *sec.is_role(MeetingInitiator),* captured from the SecureUML class MeetingInitiator, and association class InitiatorPermission, and explicit constraints (e.g., sec.InitiatorAuthConstraint(self.id) = 'Y', captured from the class InitiatorAuthConstraint).

3.2.4 Update Authorisation Constraint

The *Update* authorisation constraint (Figure 15) specifies, which data a User, assigned to a valid Role, can change. Changing the data does not include *inserting* new (as defined in Section 3.2.3) or *deleting* existing (see Section 3.2.2) data. The constraint also calls the selection action (see Section 3.2.1), because before updating the targeted data, these need to be selected from the data table.

The *Update* authorisation constraint is implemented through an instead of update trigger (e.g., Meeting_sec_update_trg) on the secured resource. Execution of the trigger results in the success (if there is no security violation and if the new values for the updated data are valid) or in the failure (otherwise).

Like previous constraints, *Update* is also created on the view (e.g., Meetign_v) of the secured resource. Checking the data (that are to be changed) values is performed on this view (for example, variables self and :NEW refer to the data values before and after the update is performed), thus, securing the actual data before the security action is finished. When the check (see, util.null_eg(:NEW.place, self.place) != 'Y') for self and :NEW values give positive answer, then the checking for the implicit (see, sec.is_role('MeetingInitiator')='Y') and explicit (e.g., sec.InitiatorAuthConstraint(self.id)='Y') expressions takes part. Both implicit and explicit constraints are captured from the SecureUML model. Satisfying all these conditions results in the value change for the attribute of the secured resource (e.g., self.time := :NEW.time). After the success of the update action on the view, the actual values are updated as well (see, UPDATE Meeting res SET ROW=self WHERE res.ID = self.ID).

3.3 Tool Support

Nowadays there exist different software tools[5] to support model-driven development (MDD) using UML at the different levels of abstraction for various modelling goals. To illustrate and implement the proposal discussed in previous sections we have selected MagicDraw[6]. This selection was influenced by the facts that, in addition to covering UML 2.0 diagrams, MagicDraw supports database modelling, business process modelling, and various XML standards. For instance, MagicDraw supports a comprehensible and systematic translation of the *logical data representation* into the *physical data representation*, and then the generation of the database SQL code, as described in Section 3.1.

[4] In relational databases, table columns can be complemented with default values, which will be used when the value of the column is not specified in the insert statement; however, this is not supported in the current transformation templates.

[5] http://www.objectsbydesign.com/tools/umltools_byProduct.html

[6] http://www.magicdraw.com/

```
-- Imported common-sql.vtl
CREATE OR REPLACE TRIGGER Meeting_sec_update_trg
  INSTEAD OF UPDATE ON Meeting_v
  REFERENCING NEW AS NEW OLD AS OLD
  FOR EACH ROW
DECLARE
  self Meeting%ROWTYPE;
  ex_denied EXCEPTION;
BEGIN
  SELECT * INTO self FROM Meeting res WHERE res.ID = :OLD.ID;
  IF util.null_eq (:NEW.place, self.place) != 'Y'
  THEN
    IF sec.is_role('MeetingInitiator') = 'Y' AND
       sec.InitiatorAuthConstraint(self.id) = 'Y'
    THEN
      self.place := :NEW.place;
    ELSE
      RAISE ex_denied;
    END IF;
  END IF;
  IF util.null_eq (:NEW.time, self.time) != 'Y'
  THEN
    IF sec.is_role('MeetingInitiator') = 'Y' AND
       sec.InitiatorAuthConstraint(self.id) = 'Y'
        -- Permission from InitiatorPermission
    THEN
      self.time := :NEW.time;
    ELSE
      RAISE ex_denied;
    END IF;
  END IF;

  UPDATE Meeting res SET ROW = self WHERE res.ID = :OLD.ID;
EXCEPTION
  WHEN ex_denied THEN
    raise_application_error (-20000, 'Access denied!');
END;
/
```

Figure 15. Transformed *Update* authorisation constraint

Our proposal focuses on the transformation rules to translate the security model expressed in SecureUML to the security authorisation constraints. To generate these transformation rules adapted to the transformation templates, we use a *report generation* mechanism implemented in MagicDraw. It supports capturing, of every UML (SecureUML) model entity through the Velocity template language. Although originally Velocity template language is created to reference objects from the Java code and to embed their dynamic context into websites, the Velocity engine was adapted for the MagicDraw tool to capture information from UML diagrams. In our case we capture security related entities and transform them into database views and instead-of triggers automatically, as illustrated in Section 3.2 for *insert* (Figure 12), *select* (Figure 13), *delete* (Figure 14), and *update* (Figure 15) actions. Using modelling tools, like MagicDraw, together with Velocity interpreter supports translation of the design models (e.g., SecureUML) to the implementation code (e.g., database views and instead-of triggers) following the defined guidelines and transformation rules for the OCL-based authorisation constraints.

As the final step, the generated SQL (i.e., data model and security authorisation constraints) code could be intertwined together executing them on the database management system, such as SQL*Plus[7] developed by Oracle.

[7] http://download.oracle.com/docs/cd/B19306_01/server.102/b14357.pdf

4 Validation

In order to validate the performance of the proposal, we have performed a case study, where we compare the quality of two security models. In the first case the developers have coded the security constraints manually. In the second case designers have created a SecureUML model, which was translated to the security authorisation constraints using our proposal. This case study is reported in [29] and [30]. In this paper we provide the summary of the major findings.

In order to compare both security models we have applied the semiotic quality framework (SEQUAL) reported in [20] and [21] that defines the assessment of the model quality by distinguishing between quality goals and means to achieve these goals. More specifically we have analysed the following quality types:

- *Semantic* quality – a correspondence between the model and its semantic domain. Quality was considered with respect to
  - *semantic completeness*: everything that the software is supposed to do is included in the model;
  - *semantic correctness*: a model should represent something that is required to be developed;
  - *traceability*: the origin of the model and its content should be identifiable;
  - *annotation*: a reader is easily able to determine, which elements are most likely to change; and
  - *modification:* the structure and the content are easy to change.
- *Syntactic* quality – a correspondence between a model and modelling language. It was defined in terms of
  - *syntactic validity*: the grammatical expressions used to create a model should be a part of the modelling language and
  - *syntactic completeness*: all grammar constructs and their parts are present in the model.
- *Pragmatic* quality – a correspondence between a model and its technical and social interpretation. It was expressed through
  - *cross-referencing*: different pieces of model content are linked together;
  - *organisation*: the model content should be arranged so that a reader could easily locate information and logical relationships among the related information;
  - *understandability*: a reader is able to understand the model with minimum explanations, and
  - *executability*: there should exist technology capable of inputting the model and resulting in its implementation.
  -

Each qualitative property was instantiated with the objective metric. The case study findings are summarised in Table 2. We observe that the SecureUML model is better evaluated than then the PL/SQL security model, especially along the qualitative properties that characterise the validity (e.g., *semantic completeness*, *semantic correctness*, *annotation*, and especially *understandability*) of the model. Regarding the model verification (e.g., *executability* and *syntactic completeness*), we found both models assessed at about the same level. The design and precise details of the case study are provided in [29] and [30].

Within this case study we certainly acknowledge that we have assessed both security models using a certain set of qualitative properties and corresponding measures. This might, however, affect the validity of conclusions, because if any other qualitative properties are applied, it might result in different outcome. But this threat is rather limited since these qualitative properties are theoretically sound and their selection is based on the previous experience.

Table 2. Quality assessment results (the highest score is in bold)

SEQUAL quality types	Qualitative property	Measure	PL/SQL security model	SecureUML security model
Semantic quality	Semantic completeness	Percentage of the RBAC domain coverage	42,86%	**100%**
	Semantic correctness	Percentage of security related statements	7,69%	**100%**
	Traceability	Number of traced links	0	0
	Annotation	Number of annotation elements	0	**8**
	Modifiability	Time spent to modify	Not known	**5-10** minutes
Syntactic quality	Syntactic validity	Number of syntactically invalid statements	**0**	2
	Syntactic completeness	Number of syntactically incomplete statements	0	0
Pragmatic quality	Understandability	Number of explanations	More than 45 minutes	**10-15** minutes
	Organisation	Number of elements for model organization	2	**4**
	Cross referencing	Number of cross-reference links	1	**3**
	Executability	Tools to execute the model	Yes	Yes

5 Discussion and Future Work

In this paper we have presented the approach for the model-driven RBAC for SQL databases. We illustrate how the SecureUML model could be translated to the database views and instead-of triggers implementing the security authorisation constraints following the transformation templates developed in the Velocity language. We observe that our approach facilitates comparatively easier modification, understandability, and communication of the security solutions. In this section we conclude our discussion by situating our work into the state of the art and by highlighting the future work.

5.1 Model-driven Security

The literature reports on a number of case studies, [32], [33] and [44] analysing different characteristics of the model-driven development. Mostly these studies focus on the benefits and on the infrastructure needed for the model-driven development. Also in [9], [26] and [32] we observe that security models facilitate automatic code generation. We also argue that the security models should be prepared with the high-quality modelling language [33] that ensures the model semantic completeness and tools [26] that guarantee model syntactic validity and syntactic completeness. Only then one could expect that model-driven security could yield a higher productivity with respect to a traditional development [32].

We identified one case study, performed in [9], reporting on the SecureUML practical application. It was observed that although the security models are integrated with the data models, the security design remains independent, reusable, and evolvable. In our case study we noted that semantic correctness of SecureUML is comparatively high since the representation is oriented only to the security concerns. We also observe that the SecureUML model is easier modifiable, which leads to the model evolvability. Like in [9] we identified that the SecureUML models are understandable at least to readers who are familiar with UML. This can improve communication of security solutions to project stakeholders [26].

5.2 Security Modelling Languages

For security-critical systems MDA facilitates security consideration from early stages in the development process and provides a seamless guidance through the development stages [19]. Model driven security (MDS) could be supported using different modelling languages. On one hand, at the various development stages and for different stakeholders' purposes, security can be

addressed using various models: like *goal models* created with *i** [46], Tropos [7], or KAOS [10], [22]; *data models* created with ER [8] or UML [38]. These modelling languages are not specifically designed with security in mind and, thus, their support for security is weak.

On another hand, there exist *security modelling languages* specifically dedicated to analysis and modelling of system security concerns. For example, abuse frames [24] suggest means to consider security during early requirements engineering stage. Secure *i** [12] addresses security trade-offs. KAOS extension to security [22] was augmented with anti-goal models designed to elicit attackers' rationales. Tropos has been extended with the notions of ownership, permission, and trust [17]. Another version of Secure Tropos [34] defines security through security constraints. Abuse cases [31], misuse cases [42], [43], and mal-activity diagrams [41] are the extensions for the modelling languages from the UML family. Another two UML extensions (through the stereotypes, tagged values, and constraints) towards security are UMLsec [19] and SecureUML [4] [25]. Those languages are, basically, used to address security concerns during the system design stage.

5.3 RBAC and Security Modelling Languages

In [2] the survey of security modelling languages shows that SecureUML does not explicitly model security criteria (such as confidentiality, integrity, and availability), but it focuses on modelling the security solutions applying the RBAC technique. With SecureUML, a modeller can define assets; however, the language does not allow expressing attacks or harms to these assets. In [18] Jayaram and Mathur investigate how the practice of software engineering blends with the requirements of secure software. The work describes a two-dimensional relationship between the software lifecycle stages and modelling approaches used to engineer security requirements. A part of the study is dedicated to the RBAC modelling using SecureUML. Authors indicate that SecureUML is suggested as the means to specify access control policies; however, it cannot describe protected resources (system design), thus, it has to be used in conjunction with the base modelling language; similarly as we illustrate in Section 3. Furthermore, we go beyond the scope of these surveys by developing the transformation templates to implement the RBAC solutions.

5.4 RBAC for SQL Databases

In [37] Oh and Park propose a model-driven approach to manage RBAC policies on top of SQL databases. The paper specifically focuses on a task-RBAC model, whereby permissions are assigned to tasks and tasks are assigned to roles. In contrast, our approach does not require the notion of task – which may or may not be relevant depending on the application domain. Our approach is based on an established security modelling language, namely SecureUML, whereas the approach in [37] is based on the combination of non-standard diagram types, namely organisation diagrams, information object diagrams and task diagrams. Thus, it can be argued that our approach is more generally applicable.

Temporal RBAC models [5] allow designers to capture time-sensitive access control policies, e.g., a user only has the access to certain resources during a specified period of time. In [3] Barker *et al* sketch a method to transform temporal RBAC policies, specified in a logic-based notation, into PL/SQL code, but their code generation method is incomplete – it only deals with specific types of temporal RBAC constraints. Our approach differs from the above one in that it takes SecureUML models as an input. Extending SecureUML with temporal constraints and enhancing the PL/SQL generation method accordingly is a direction for the future work.

Some researchers had addressed the issue of generating code for RBAC models in the context of data warehouses. In [6] Blanco *et al* present a QVT transformation for generating code for the Microsoft SSAS platform from RBAC models defined in terms of an ad hoc security meta-

model. This and other similar works on secure data warehouses do not deal with data updates as these updates are done offline. Also, the security models differ from those that we deal with, since their models deal with features specific to data warehouses such as dimensions and measures. Finally, their code generation methods do not target SQL platforms.

5.4 Future Work

In [44] Staron identifies five conditions for the successful adoption of the model driven development technology. He stresses the *maturity of the modelling technology* and *maturity of the related methods*. He also speaks about the *process compatibility* and the necessity for the *core language-engineering expertise*. Finally, he stresses the importance of the *goal-driven adoption process*.

Following [44] we see possible improvement for our proposal. For example, a *mature security modelling method* needs to be introduced in order to guide discovery of the security requirements and to support security quality assurance through project planning. A possible candidate could be adoption of the security risk management methods, e.g., ISSRM [11]. This would improve traceability and also record rationales for security decisions.

Model driven security analysis should be *compatible with the working processes*. We plan to perform another case study where we would investigate the quality of processes to develop security models at the design stage (e.g., using SecureUML or other modelling language) to compare it to the quality of processes to develop security models at the implementation stages.

Oracle databases support fine-grained access control using Virtual Private Database [36] (VPD), which enables data access control by users with the assurance of physical data separation. The next step for the transformation is to take advantage of the VPD and compare the performance of the two approaches (i.e., using views with instead-of triggers and using VPD).

Finally, we need to support a *goal-driven process*, where we would define goals to introduce security model-driven development systematically. In this paper we specifically focussed on the security policy for the data model. Our future goal is to develop transformation rules that would facilitate implementation of the security concerns at the system and software application, and presentation levels.

Acknowledgment

This research was started while the first author was at the Software Technology and Applications Competence Centre and the second author was at Logica Estonia (acquired by the Canada-based CGI Group). The work was funded by ERDF via Enterprise Estonia and by Logica Estonia. We thank Prof. Marlon Dumas for the discussion, his advises, and support.

References

[1] V. Anaya, G. Berio, M. Harzallah, P. Heymans, R. Matulevičius, A. L. Opdahl, H. Panetto, and M. J. Verdech, "The unified enterprise modelling language – overview and further work," in Computers in Industry, Elsevier Science Publication, vol. 61, no. 2, pp. 99-111, 2010. Available: http://www.sciencedirect.com/science/article/pii/S0166361509002061

[2] A. Bandara, H. Shinpei, J. Jurjens, H. Kaiya, A. Kubo, R. Laney, H. Mouratidis, A. Nhlabatsi, B. Nuseibeh, Y. Tahara, T. Tun, H. Washizaki, N. Yoshioka, and Y. Yu, Security Patterns: Comparing Modelling Approaches, Department of Computing Faculty of mathematics, Computing Technology, The Open University, Tech. Rep. No 1009/06, 2009.

[3] S. Barker, P. Douglas, and T. Fanning, "Implementing RBAC policies," in Research Directions in Data and Applications Security, E. Gudes, S. Shenoi, Eds., Kluwer Academic Publishers Group, 2003, pp. 27-36. http://dx.doi.org/10.1007/978-0-387-35697-6_3

[4] D. Basin, J. Doser, and T. Lodderstedt, “Model driven security: from UML models to access control infrastructure,” in ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 15(1), pp. 39-91, 2006. http://dx.doi.org/10.1145/1125808.1125810

[5] E. Bertino, P. A. Bonatti, and E. Ferrari, “TRBAC: A Temporal Role-based Access Control Model,” in Transactions on Information and Security Systems (TISSEC), ACM, vol. 4, no. 3, pp. 191-233, 2001. http://dx.doi.org/10.1145/501978.501979

[6] C. Blanco I. G.-R. de Guzman, E. Fernandez-Medina, J. Trujillo, and M. Piattini. “Automatic generation of secure multidimensional code for data warehouses: an MDA approach,” in Proc. OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems, 2008, pp. 1052-1068. http://dx.doi.org/10.1007/978-3-540-88873-4_9

[7] P. Bresciani, A. Perini P. Giorgini, F. Giunchiglia, and J. Mylopoulos, “Tropos: an agent-oriented software development methodology,” in Autonomous Agents and Multi-Agent Systems, Springer, 2004. http://dx.doi.org/10.1023/b:agnt.0000018806.20944.ef

[8] P. P. Chen, “The Entity-Relationship Model: Towards a Unified View of Data,” in ACM Transactions on Database Systems, vol. 1(1), pp. 9-36, 1976. http://dx.doi.org/10.1145/320434.320440

[9] M. Clavel, V. Silva, C. Braga, and M. Egea. “Model-driven security in practice: an industrial experience,” in Proc. the 4th European Conference on Model Driven Architecture: Foundations and Applications, Springer-Verlag, 2008, pp. 326-337. http://dx.doi.org/10.1007/978-3-540-69100-6_22

[10] A. Dardenne, A. van Lamsweerde, and S. Fickas, “Goal-directed requirements acquisition,” in Science of Computer Programming, vol. 20, North Holland, pp. 3-50, 1993. http://dx.doi.org/10.1016/0167-6423(93)90021-g

[11] E. Dubois, P. Heymans, N. Mayer, and R. Matulevičius, “A systematic approach to define the domain of information system security risk management,” in Intentional Perspectives on Information Systems Engineering, S. Nurcan, C. Salinesi C. Souveyet J. Ralyte, Eds., Springer Heidelberg, 2010, pp. 289-306. http://dx.doi.org/10.1007/978-3-642-12544-7_16

[12] G. Elahi and E. Yu. “A goal oriented approach for modeling and analyzing security trade-offs,” in Parent et al. Eds., Proc. the 26th International Conference on Conceptual Modelling, 2007. http://dx.doi.org/10.1007/978-3-540-75563-0_26

[13] M. S. Feather, S. Fickas, A. Finkelstein, and A. van Lamsweerde, “Requirements and specification exemplars. automated software engineering,” vol. 4, pp. 419-438, 1997. http://dx.doi.org/10.1023/a:1008680612960

[14] 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), vol. 4, no. 3, pp. 224-274, 2001. http://dx.doi.org/10.1145/501978.501980

[15] S. Feuerstein and B. Pribly, “Oracle PL/SQL Programming,” in O'Reilly Media Inc, 4th edition, 2005.

[16] D. G. Firesmith. “Engineering safety and security related requirements for software intensive systems,” ICSE 2007 tutorial, Minneapolis, USA:IEEE, 2007. http://dx.doi.org/10.1109/icsecompanion.2007.35

[17] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. “Modeling security requirements through ownership, permision and delegation,” in Proc. the 13th IEEE International Conference on Requirements Engineering, IEEE Computer Society, 2005. http://dx.doi.org/10.1109/re.2005.43

[18] K. R. Jayaram and A. P. Mathur, “Software engineering for secure software – State of the art: A survey,” Dept. of Computer Sciences & CERIAS, Purdue University, Tech. Rep. CERIAS TR 2005-67, 2005.

[19] J. Jurjens, “Secure systems development with UML,” Springer Berlin Heidelberg, 2005. http://dx.doi.org/10.1007/b137706

[20] J. Krogstie. “A semiotic approach to quality in requirements specifications,” in Proc. IFIP 8.1 working Conf. on Organisational Semiotics, 2001, pp. 231-249. http://dx.doi.org/10.1007/978-0-387-35611-2_14

[21] J. Krogstie, “Using a semiotic framework to evaluate uml for the development for models of high quality,” in K. Siau, T. Halpin, Eds., Unified Modelling Language: System Analysis, Design and Development Issues, IDEA Group Publishing, 1998, pp. 89-106. http://dx.doi.org/10.4018/978-1-930708-05-1.ch006

[22] A. van Lamsweerde. “Elaborating security requirements by construction of intentional anti-models,” in Proc. the 26th International Conference on Software Engineering, IEEE Computer Society, 2004, pp. 148-157. http://dx.doi.org/10.1109/icse.2004.1317437

[23] N. Li and Z. Mao. “Administration in role-based access control,” in Proc. the 2nd ACM Symposium on Information, Computer and Communications Security, ASIACCS '07, ACM, 2007, pp. 127-138.b http://dx.doi.org/10.1145/1229285.1229305

[24] L. Lin, B. Nuseibeh, D. Ince, and M. Jackson. “Using abuse frames to bound the scope of security problems,” in Proc. the 12th IEEE International Conference on Requirements Engineering, IEEE Computer Society, 2004, pp. 354-355. http://dx.doi.org/10.1109/icre.2004.1335698

[25] T. Lodderstedt, D. Basin, and J. Doser. “SecureUML: A UML-based modeling language for model-driven security,” in Proc. the 5th International Conference on The Unified Modeling Language, LNCS, vol. 2460, Springer-Verlag, 2002, pp. 426-441. http://dx.doi.org/10.1007/3-540-45800-x_33

[26] A. MacDonald, D. Russell, and B. Atchison. “Model-driven Development within a Legacy System: An Industry Experience Report,” in Proc. the 2005 Australian Software Engineering Conference (ASWEC’05). IEEE Computer Science, 2005. http://dx.doi.org/10.1109/aswec.2005.32

[27] R. Matulevičius and M. Dumas. “A comparison of SecureUML and UMLsec for role-based access control,” in: Databases and Information Systems: The 9th Conference on Databases and Information Systems, Riga, LV, 2010. J. Barzdins, M. Kirikova, Eds., Latvia: University of Latvia Press, 2010, pp. 171-185.

[28] R. Matulevičius and M. Dumas, “Towards model transformation between SecureUML and UMLsec for role-based access control,” J. Barzdins, M. Kirikova, Eds., in Databases and Information Systems VI. IOS Press, pp. 339-352, 2011.

[29] R. Matulevičius, H. Lakk, M. Lepmets, and A. Sisask. “Comparing quality of security models: a case study,” Proc. ADBIS 2010 workshop MDASD 2010, University of Novi Sad Press, Serbia, 2010.

[30] R. Matulevičius, H. Lakk, and M. Lepmets, “An approach to assess and compare quality of security models,” in Computer Science and Information Systems, ComSIS Consortium, vol. 8, no 2, pp. 447-476, 2011. http://dx.doi.org/10.2298/csis101231014m

[31] J. McDermott and C. Fox. “Using abuse case models for security requirements analysis,” in Proc. the 15th Annual Computer Security Applications Conference, 1999. http://dx.doi.org/10.1109/csac.1999.816013

[32] The Middleware Company, Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach: Productivity Analysis, MDA Productivity case study, 2003.

[33] M. de Miguel, J. Jourdan, and S. Salicki. “Practical experiences in the application of MDA” in Proc. the 5th International Conference on The Unified Modeling Language, Springer-Verlag, 2002, pp. 128-139. http://dx.doi.org/10.1007/3-540-45800-x_11

[34] H. Mouratidis. “Analysing security requirements of information systems using Tropos,” in Proc. 1st Annual Conference on Advances in Computing and Technology, 2006, pp. 55-64.

[35] A. Nanda and S. Feuerstein, Oracle PL/SQL for DBAs. O'Reilly Media, Inc. 2005.

[36] K. Nishida and S. Duvvuri. (2009). Row Level Security with BI Publisher Enterprise. An Oracle white paper, Oracle Communications [Online]. (last check 26.07.2015) Available: http://www.oracle.com/technetwork/middleware/bi-publisher/overview/wp-oracle-bip-row-level-security-132091.pdf

[37] S. Oh and S. Park. “Enterprise model as a basis of administration on role-based access control,” in Proc. the 3rd International Symposium on Cooperative Database Systems for advanced Applications (CODAS 2001), 2001, pp. 150-158. http://dx.doi.org/10.1109/codas.2001.945161

[38] OMG, Unified Modeling language: Superstructure, version 2.0, format/05-07-04, 2005.

[39] A. Roichman and E. Gudes, “DIWeDa - detecting intrusions in web databases,” in Data and Applications Security XXII, LNCS 5094, 2008, pp. 313-329. http://dx.doi.org/10.1007/978-3-540-70567-3_24

[40] R. S. Sandhu and E. J. Coyne, “Role-based access control models,” in Computer, pp. 38-47, 1996. http://dx.doi.org/10.1109/2.485845

[41] G. Sindre. “Mal-activity diagrams for capturing attacks on business processes,” in Proc. Working Conference on Requirements Engineering: Foundation for Software Quality, Springer-Verlag Berlin Heidelberg, 2007, pp. 355-366. http://dx.doi.org/10.1007/978-3-540-73031-6_27

[42] G. Sindre and A. L. Opdahl. “Template for misuse case de- scription,” in Proc. the International Workshop Requirements Engineering: Foundation for Software Quality (REFSQ 2001), 2001.

[43] G. Sindre and A. L. Opdahl, “Eliciting security requirements with misuse cases,” in Requirements Eng. vol. 10, no. 1, Springer-Verlag, 2005. http://dx.doi.org/10.1007/s00766-004-0194-4

[44] M. Staron. “Adopting model driven software development in industry – a case study at two companies,” in the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2006), Springer-Verlag, 2006, pp. 57-72. http://dx.doi.org/10.1007/11880240_5

[45] J. Warner and A. Kleippe, “The object constraint language, second edition, getting your models ready for MDA,” Addison-Wesley, 2003. http://dx.doi.org/10.5381/jot.2003.2.6.r1

[46] E. Yu. “Towards modeling and reasoning support for early-phase requirements engineering,” in Proc. RE’97, IEEE Computer Society, 1997, pp. 226. http://dx.doi.org/10.1109/isre.1997.566873

Appendix A

All functions in this appendix are part of a package named sec. This package contains security related functions needed by the views and instead-of triggers. As our templates do not yet support OCL the authorisation constraints in A.1 and A.2 are written by hand in order to simplify the security model. The function is_role (See A.3) is used in the generated views and instead-of triggers to limit access to users in the specified roles, but the implementation is not limited to the given example.

A.1 InitiatorAuthConstraint

The InitiatorAuthConstraint shown in Figure A1 is an authorisation constraint, which is placed on MeetingInitiator actions (i.e. *Insert*, *Update* and *Delete*) on the Meeting resource. It corresponds to the OCL-defined authorisation constraints – AC#1, AC#2, and AC#3 – presented in Section 2.2.

```
FUNCTION InitiatorAuthConstraint (
   pi_organisedBy   IN   Meeting.organisedby%TYPE
)
   RETURN VARCHAR2
IS
   s_user_name   Users.NAME%TYPE;
BEGIN
   -- Select the orgeniser's name
   SELECT Users.NAME
     INTO s_user_name
     FROM Users
    WHERE Users.ID = pi_organisedBy;

   -- Is the current user the organiser?
   IF s_user_name = sec.get_username
   THEN
      RETURN 'Y';
   ELSE
      RETURN 'N';
   END IF;
END InitiatorAuthConstraint;
```

Figure A1. PL/SQL constraint InitiatorAuthConstraint

A.2 ParticipantAuthConstraint

The ParticipantAuthConstraint shown in Figure A2 is an authorisation constraint, which is placed on MeetingParticipant action *Select* on the Meeting resource. It corresponds to the OCL-defined authorisation constraint AC#4, presented in Section 2.2.

```
FUNCTION ParticipantAuthConstraint (
   pi_meetingId IN Meeting.ID%TYPE
)
   RETURN VARCHAR2
IS
   i_participation   NUMBER;
BEGIN
   -- If count = 1 then the user is a participant
   SELECT COUNT (*)
     INTO i_participation
     FROM Users, MeetingParticipant
    WHERE Users.ID = MeetingParticipant.isOrganisedBetween
      AND Users.NAME = sec.get_username
      AND MeetingParticipant.invitedToParticipateAt = pi_meetingId;

   IF i_participation = 1
   THEN
      RETURN 'Y';
   ELSE
      RETURN 'N';
   END IF;
END ParticipantAuthConstraint;
```

Figure A2. PL/SQL constraint ParticipantAuthConstraint

A.3 is_role

The is_role constraint shown in Figure A3 is an authorisation constraint, which is implicitly placed on resources and on their actions. This constraint limits access to only one specified role. The implementation given here is presented only as an example and the actual implementation is not limited in any way, as it is usable inside the generated views and instead-of triggers. The

function uses Oracle database context demo_context, which stores the user's role. If they match the letter 'Y' for "Yes" is returned, otherwise letter 'N' for "No" is returned.

```
FUNCTION is_role(
    p_role VARCHAR2)
  RETURN VARCHAR2
IS
BEGIN
  IF upper(sys_context('demo_context', 'role')) = upper(p_role)
  THEN
    RETURN 'Y';
  ELSE
    RETURN 'N';
  END IF;
END is_role;
```

Figure A3. PL/SQL constraint is_role

Appendix B

In this Appendix we present four transformation rules for *Insert*, *Update*, *Delete*, and *Select* security actions that help translating the SecureUML model to the PL/SQL code. All these rules are written in Velocity template languages and are used to generate PL/SQL security constraints from the SecureUML model. For the readability purpose the layout of the rules is slightly modified. The templates are allowed only for academics and research purpose.

B.1 Transformation Rules for Insert Action

```
#parse("common-sql.vtl")##
#foreach($resource in $Class)
  #if($report.containsStereotype($resource,"secuml.resource"))
    CREATE OR REPLACE TRIGGER #if($secureSchema.length>0)${secureSchema}.#end##if
${resource.name}_sec_insert_trg
       INSTEAD OF INSERT
       ON #if($secureSchema.length>0)${secureSchema}.#end##if
${resource.name}_v
       REFERENCING NEW AS NEW
       FOR EACH ROW
    DECLARE
       ex_denied   EXCEPTION;
    BEGIN
    #set($prefixClause="IF")
    #foreach($permission in $report.getRelationship($resource))
      #if($report.containsStereotype($permission,"secuml.permission"))
        #getPermissionAssignedRole($permission)
        #if($assignedRole)
          #foreach($attribute in $permission.ownedAttribute)
            #if ($attribute.type.name == "Insert")
              $prefixClause #hasRole($assignedRole) ##
              #set($prefixClause="OR")
              #getParsedPermissionConstraints($permission,":NEW") -- From $permission.name
            #end##if Insert
          #end##foreach attribute
        #end##if hasCorrectRole
      #end##if secuml.permission
    #end##foreach permission
    #if($prefixClause!="IF")
      THEN
        INSERT INTO ${resource.name} (
      #countColumns($resource)
      #set($columnsDoneCount=0)
      #foreach($column in $resource.ownedAttribute)
```

```
          #if(!$column.getAssociation())
            ${column.name}##
            #set($columnsDoneCount=$columnsDoneCount+1)
            #if($columnsDoneCount < $columnCount)##
              ,
            #else

            #end##if hasMore
          #end##if !association
        #end##foreach column
        ) VALUES (
        #set($columnsDoneCount=0)
        #foreach($column in $resource.ownedAttribute)
          #if(!$column.getAssociation())
            :NEW.${column.name}##
            #set($columnsDoneCount=$columnsDoneCount+1)
            #if($columnsDoneCount < $columnCount)##
              ,
            #else

            #end##if
          #end##if !association
        #end##foreach column
        );
        ELSE
          RAISE ex_denied;
        END IF;
      EXCEPTION
        WHEN ex_denied
        THEN
      #end##if
        raise_application_error (-20000, 'Access denied!');
    END;
    /
    #end##if
  #end##foreach class
```

B.2 Transformation Rules for Update Action

```
  #parse("common-sql.vtl")##
  #foreach($resource in $Class)
    #if($report.containsStereotype($resource,"secuml.resource"))
      CREATE OR REPLACE TRIGGER #if($secureSchema.length>0)${secureSchema}.#end##if
  ${resource.name}_sec_update_trg
        INSTEAD OF UPDATE
          ON #if($secureSchema.length>0)${secureSchema}.#end##if
  ${resource.name}_v
          REFERENCING NEW AS NEW OLD AS OLD
        FOR EACH ROW
        DECLARE
          self    #if($protectedSchema.length>0)${protectedSchema}.#end##if
  ${resource.name}%ROWTYPE;
          ex_denied   EXCEPTION;
        BEGIN
          SELECT *
            INTO self
            FROM #if($protectedSchema.length>0)${protectedSchema}.#end##if
  ${resource.name} res
           WHERE res.ID = :OLD.ID;

        #countColumns($resource)
        #set($columnsDoneCount=0)## -- not used at the moment
        #foreach($column in $resource.ownedAttribute)
          #if(!$column.association)
            IF util.null_eq (:NEW.${column.name}, self.${column.name}) != 'Y' -- $column.name
updated
            THEN
```

```
          #set($prefixClause="IF")
          ###getPermissions($resource,"Update", "self")
          #findImmediatePermissions($resource, "Update", "res", $prefixClause)
          ## -- BEGIN Find all resource views
          #foreach($dep in $Dependency)##
            #foreach($target in ${dep.target})##
              #set($isTarget=false)##
              #if($target == $resource)##
                #set($isTarget=true)##
              #end## -- if
              #if($isTarget)##
                ## -- BEGIN Find all resource view select permissions
                #foreach($resourceView in ${dep.client})##
                  #if($report.containsStereotype($resourceView,"secuml.resourceView"))##
                    #set($isStillTarget = false)##
                    #foreach($viewAtt in $resourceView.ownedAttribute)##
                      #if ($viewAtt.name == $column.name)##
                        #set ($isStillTarget = true)##
                      #end##-- if
                    #end##-- foreach owned attribute
                    ## --if still is target
                    #if($isStillTarget)##
                      ## --all permissions for the resource view
                      #foreach($permission in $report.getRelationship($resourceView))##
                        #if($report.containsStereotype($permission,"secuml.permission"))##
                          #getPermissionAssignedRole($permission)##
                          #if($assignedRole)##
                            #foreach($attribute in $permission.ownedAttribute)##
                              #if($attribute.type.name == "Update")##
                                $prefixClause                          #hasRole($assignedRole)
#getParsedPermissionConstraints($permission,"res")##
                                #set($prefixClause="OR")
                              #end##-- if Update
                            #end##-- foreach attribute
                          #end##-- if hasCorrectRole
                        #end##-- if secuml.permission
                      #end##-- foreach permission
                    #end##-- is still a target
                    ## -- END Find all resource view select permissions
                  #end##-- isTarget #2
                #end##-- if secuml.resourceView
              #end##-- foreach resourceView
            #end##-- foreach target
          #end##-- foreach dep
          ## -- END Find all resource views
          #if($prefixClause!="IF")
              THEN
                self.${column.name} := :NEW.${column.name};
              ELSE
          #end##if
                RAISE ex_denied;
              END IF;
            END IF;
          #set($columnsDoneCount=$columnsDoneCount + 1)## -- not used!
        #end##if !association
      #end##foreach

          UPDATE #if($protectedSchema.name.lenght>0)${protectedSchema.name}.#end##if
  ${resource.name} res
             SET ROW = self
           WHERE res.ID = :OLD.ID;
       EXCEPTION
          WHEN ex_denied
          THEN
             raise_application_error (-20000, 'Access denied!');
       END;
  /
```

```
  #end##if
#end##foreach resource
```

B.3 Transformation Rules for Delete Action

```
#parse("common-sql.vtl")##
#set($secumlResource="secuml.resource")##
#set($secumlPermission="secuml.permission")##
#foreach($resource in $Class)
  #if($report.containsStereotype($resource,$secumlResource))
    CREATE OR REPLACE TRIGGER #if($secureSchema.length>0)${secureSchema}.#end##if
${resource.name}_delete_trg
       INSTEAD OF DELETE
       ON #if($secureSchema.length>0)${secureSchema}.#end##if
${resource.name}_v
       REFERENCING OLD AS OLD
       FOR EACH ROW
    DECLARE
       self    #if($protectedSchema.length>0)${protectedSchema}.#end##if
${resource.name}%ROWTYPE;
       ex_denied   EXCEPTION;
    BEGIN
       SELECT *
         INTO self
         FROM #if($protectedSchema.length>0)${protectedSchema}.#end##if
${resource.name} res
        WHERE res.ID = :OLD.ID;

    #set($prefixClause="IF")
    #foreach($permission in $report.getRelationship($resource))
      #if($report.containsStereotype($permission,$secumlPermission))
        #getPermissionAssignedRole($permission)
          #if($assignedRole)
            #foreach($att in $permission.ownedAttribute)
              #if ($att.type.name == "Delete")
                $prefixClause #hasRole($assignedRole) ##
                #set($prefixClause="OR")
                #getParsedPermissionConstraints($permission, ":OLD")
              #end##if Delete
            #end##if assignedRole
          #end##if
        #end##foreach
      #end
      #if ($prefixClause!="IF")
        THEN
        DELETE FROM #if($protectedSchema.length>0)${protectedSchema}.#end##if
${resource.name} tbl
            WHERE tbl.ID = :OLD.ID;
        ELSE
          RAISE ex_denied;
        END IF;
      EXCEPTION
        WHEN ex_denied
        THEN
      #end##if
          raise_application_error (-20000, 'Access denied!');
      END;
      /
  #end##is secuml.resource
#end##resource
```

B.4 Transformation Rules for Select Action

```
#parse("common-sql.vtl")##
```

```
#foreach($resource in $Class)##
  #if($report.containsStereotype($resource,"secuml.resource"))
    CREATE OR REPLACE VIEW #if($secureSchema.lenght>0)${secureSchema}.#end##if
${resource.name}_v
    AS
       SELECT
    ## -- BEGIN find own attributes
    #countColumns($resource)
    #set($columnsDoneCount=0)
    ## -- END find own attributes
    #foreach($column in $resource.ownedAttribute)##
      #set($initialKeyword="CASE WHEN")
      #if(!$column.getAssociation())
        #findImmediatePermissions($resource, "Select", "res", $initialKeyword)
        #if($initialKeyword=="CASE WHEN")
          -- No immediate permissions
        #end##if
        ## -- BEGIN Find all resource views
        #foreach($dep in $Dependency)##
          #foreach($target in ${dep.target})##
            #set($isTarget=false)##
            #if($target == $resource)##
              #set($isTarget=true)##
            #end## -- if
            #if($isTarget)##
              ## -- BEGIN Find all resource view select permissions
              #foreach($resourceView in ${dep.client})##
                #if($report.containsStereotype($resourceView,"secuml.resourceView"))##
                  #set($isStillTarget = false)##
                  #foreach($viewAtt in $resourceView.ownedAttribute)##
                    #if ($viewAtt.name == $column.name)##
                      #set ($isStillTarget = true)##
                    #end##-- if
                  #end##-- foreach owned attribute
                  ## --if still is target
                  #if($isStillTarget)##
                    ## --all permissions for the resource view
                    #foreach($permission in $report.getRelationship($resourceView))##
                      #if($report.containsStereotype($permission,"secuml.permission"))##
                        #getPermissionAssignedRole($permission)##
                        #if($assignedRole)##
                          #foreach($attribute in $permission.ownedAttribute)##
                            #if($attribute.type.name == "Select")##
                              $initialKeyword #hasRole($assignedRole) ##
                              #set($initialKeyword="OR")
                              #getParsedPermissionConstraints($permission,"res")##
                            #end##-- if Insert
                          #end##-- foreach attribute
                        #end##-- if hasCorrectRole
                      #end##-- if secuml.permission
                    #end##-- foreach permission
                  #end##-- is still a target
                  ## -- END Find all resource view select permissions
                #end##-- isTarget #2
              #end##-- if secuml.resourceView
            #end##-- foreach resourceView
          #end##-- foreach target
        #end##-- foreach dep
        ## -- END Find all resource views
        #if($initialKeyword!="CASE WHEN")
          THEN self.${column.name}
          ELSE CAST (NULL AS #getOracleSqlType($column))
          END
          #set($columnsDoneCount=$columnsDoneCount+1)
        #else
          CAST (NULL AS #getOracleSqlType($column))
        #end##if
        AS ${column.name}##
```

```
        #if($columnsDoneCount<$columnCount)##
          ,
        #else

        #end##if
      #end##-- if not associatioin
    #end##-- foreach column

     FROM #if($protectedSchema.lenght>0)${protectedSchema}.#end##if
${resource.name} self
    #getPermissions($resource,"Select", "res", "WHERE");
/
  #end##-- if secuml.resource
#end##-- foreach resource
```