U N I V E R S I T Y OF T A R T U

Faculty of Mathematics and Computer Science

Institute of Computer Science

Henri Lakk

Model-Driven Role-Based Access Control for Databases

Master's Thesis (30 ECTS)

Supervisor: Raimundas Matulevičius, PhD

Author:	.....	"..." May 2012
Supervisor:	.....	"..." May 2012
Allowed to defend		
Professor:	.....	"..." May 2012

Tartu 2012

Abstract

With the constant march towards a paperless business environment, database systems are increasingly being used to hold more and more sensitive information. This means they present an increasingly valuable target for attackers. A mainstream method for information system security is Role-based Access Control (RBAC), which restricts system access to authorised users. However the implementation of the RBAC policy remains a human intensive activity, typically, performed at the implementation stage of the system development. This makes it difficult to communicate security solutions to the stakeholders earlier and raises the system development cost, especially if security implementation errors are detected.

The use of connection pooling in web applications, where all the application users connect to the database via the web server with the same database connection, violates the the principle of minimal privilege. Every connected user has, in principle, access to the same data. This may leave the sensitive data vulnerable to SQL injection attacks or bugs in the application.

As a solution we propose the application of the model-driven development to define RBAC mechanism for data access at the design stages of the system development. The RBAC model created using the SecureUML approach is automatically translated to source code, which implements the modelled security rules at the database level. Enforcing access-control at this low level limits the risk of leaking sensitive data to unauthorised users.

In out case study we compared SecureUML and the traditional security model, written as a source code, mixed with business logic and user-interface statements. The case study showed that the model-driven security development results in significantly better quality for the security model. Hence the security model created at the design stage contains higher semantic completeness and correctness, it is easier to modify and understand, and it facilitates a better communication of security solutions to the system stakeholders than the security model created at the implementation stage.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today 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. However the implementation of the RBAC policy remains a human intensive activity, typically, performed at the implementation stage of the system development. This makes it difficult to communicate security solutions to the stakeholders earlier and raises the system development cost, especially if security implementation errors are detected.

With the explosion in web-based commerce and information systems, databases have been drawing ever closer to the network perimeter [16]. This is a necessary consequence of doing business in the Web - the customers need to have access to the information systems via web servers, so the web servers need to have access to the databases. As the result of this, the databases dare closer to the attackers. With the trend towards a paperless business environment, database systems are increasingly holding more and more sensitive information, which makes them a valuable target for attackers.

Furthermore, in old desktop applications the RBAC was sufficient [27]. The applications where connected with the database but the number of users was limited. This two-tier architecture results in a model where the user is working with an application layer that interfaces directly with the database layer. This means the database would directly identify the computer, the user transactions and the user. 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. But, today, 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, the database is not able to identify neither who has accessed the data not the transaction of the same

user. The web application does not open nor close a connection before/ after each request but uses a connection pool to store connections. Using such a connection pool a large number of users can also be satisfied with few database connections.

However, regarding the database security, the principle of minimal privilege is violated, and every connected user has access to the same data. Such a situation results in horizontal (i.e., access to the data of other user) and vertical (i.e., access to a department's data) privileges escalation. This may leave the sensitive data vulnerable to SQL injection attacks or bugs in the application. *"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"* [27].

In this thesis we propose the application of the model-driven development to define RBAC mechanism for data access at the design stages of the system development. The RBAC model created using the SecureUML approach is automatically translated to the implementation with database views and triggers.

Our case study showed that the model-driven security development results in significantly better quality for the security model. Hence the security model created at the design stage contains higher semantic completeness and correctness, it is easier to modify and understand, and it facilitates a better communication of security solutions to the system stakeholders than the security model created at the implementation stage.

The thesis is divided into four parts. In the first part we give an overview of model-driven architecture, model-driven development and model-driven security. We will describe several security modelling languages (Missuse cases, UMLSec, MAL Activity, SecureUML) and compare these languages and explain the choice to use SecureUML to model RBAC. We compare modelling tools and explain the choice to use NoMagic MagicDraw in our project. We describe the Oracle database and the access control and give an overview of the Velocity language and template engine. Finally we give guidelines on modelling security constraint for a database using secureUML and propose conventions for modelling to make the security models more easy to read.

In the second part we provide a brief summary of the published (and submitted) papers. In the third part we will give the conclusion and list the future work of this project. Finally in part four we provide our published and submitted papers.

Part I

Background

Chapter 2

Model-Driven Development

In this chapter we give an overview of model-driven architecture (MDA), model-driven development (MDD) and model-driven security (MDS).

2.1 Model-Driven Architecture

The Model-Driven Architecture (MDA)[1, 12] is a development paradigm that aims to separate business and application logic from technology evolution. It helps to build code quickly, in a middle ware independent, well designed, consistent and maintainable fashion. The aim of MDA is to the separation of technology dependent concepts from independent concepts. This solution limits the problems of platform dependencies and portability of the software and thereby reduces the costs. The separation is supported at model level to avoid platform dependencies in all phases of the life cycle. The transformation mappings from Platform-Independent Model (PIM) to Platform-Specific Model (PSM) support to reduce the cost of adaptation of the subsystems to different platforms.

MDA proposes solutions to automate the software development process. The main objective is the reduction of the time to market based on tool support for the refinement of models and code generation. This approach reduces development errors because it reduces the manual development process and provides support to reuse the best-known solutions. In this development process, the tools can provide support for the integration of different software development phases based on the transformation of models of different phases. The tool support provides a constructive method based on models with the combination of concerns at modelling level.

Models provide support for different types of problems: i) description of concepts, ii) validation of these concepts based on checking and analysis

techniques, iii) transformation of models and generation of code, configurations, and documentation. Separation of concerns avoids confusion because of the combination of different types of concepts. MDA introduces solutions for the specialisation of the models for specific concerns and for the interconnection of concerns based on models. This approach reduces the complexity of models by the specialisation of modelling activities. It improves communication between stakeholders using the models to support the interchange of information.

The main idea of the MDA paradigm is a development process with the following steps:

1. Secure business requirements for an application.

2. Develop UML diagrams for the domain model, independent of any particular technology (J2EE, Microsoft .NET, CORBA, etc.). This UML model represents the core business services and components. This UML model would be the same regardless of whether to use J2EE or .NET.

3. Build UML diagrams for the application, specific to a particular technology (such as J2EE, for example). This UML model will have elements that are technology-specific, such as specific J2EE design patterns. This Platform Dependent Model can be built manually, or much of id can be generated using an MDA tool and hand-tune only the pieces of it that require customisation.

4. Finally, generate the application code using an MDA tool. Instead of writing the application by hand based on the UML model, the majority of the code is generated from the UML diagrams. In the case of J2EE, the MDA tool would generate most of the servlets, JSPs and EJBs. The developers would then be left to fill in any details that could not be modelled using UML, such as business logic.

As mentioned above MDA enables to generate code from UML models. It was also possible pre-MDA. Rational Rose, for example, can generate Java classes from a UML model. The key advancement of MDA is that it enable to go from a platform independent, high-level design all the way to platform specific code that is fairly complete. There are several particular points to note:

- MDA starts from a higher level of abstraction than other design processes. The top-level model (PIM) is very abstract; just entities and services.

- The PSM is a complete description of the application in the form of meta-data. At that level it is possible to enhance the design with technology specific features (e.g. custom finders for EJB entity beans) without touching Java code.

- The code generated from the PSM is close to a complete application. Many tools generate code from some kind of model (such as Middlegen or XDoclet), but they give only pieces of an application. They are not comprehensive because they do not start from a complete model of the application.

- The algorithms that generate PSM from PIM, and code from PSM, are intended to be configurable by the architect.

The suggested benefits of MDA are:

- **Faster development time.** Generating code rather than handwriting each file, saves the "busy work" required to write the same files over and over again. For example, in the J2EE world, it is sometimes needed to write six or more files to create just one EJB component. Most of this can be automated with a clever code generation tool.

- **Architectural advantages.** When using MDA, the system is modelled using UML—not just by modelling Java classes, but high-level domain entities as well. This procedure forces the developer to actually think about the architecture and object model behind the system, rather than simply diving into coding, which many developers still do. Software engineering principles have proven that designing the system first, will reduce the possibility of introducing architectural flaws into the system later on in the development life cycle.

- **Improved code consistency and maintainability.** Most organisations have problems keeping their application architectures and application code consistent in their projects. Some developers will use well-accepted design patterns, while others will not. By using an MDA tool to generate the code with a consistent algorithm, rather than writing it by hand, all developers get the ability to use the same underlying design patterns, since the code is generated in the same way each time. This is a significant advantage from the maintenance perspective. For example, developers at organisations that subscribe to an MDA approach to development will all be able to understand each other's code more easily because they will be levering the same design paradigm and language.

- **Increased portability across middle-ware vendors.** If it is needed to switch between middle-ware platforms (for example, switching between J2EE, .NET or CORBA), the PIM is reusable. From the PIM, one should be able to regenerate the code for the new target platform. While not all of the code can be regenerated automatically, the ability to regenerate a large proportion of one application certainly would save time over having to rewrite it all from scratch.

2.2 Model-Driven Development

Model Driven Software Development (MDD)[31, 28] is a new trend in utilising models in software development. MDD refers to the systematic use of models and model transformations as primary engineering artefacts throughout the entire software life cycle. This, in practice often means that the code is generated from models and thus the models need to be used efficiently and effectively. The vision of MDD requires shifting the focus of estimations, analyses, or evaluations from code to models. For example, the initial complexity analyses should be done based on models or test planning and development should be done based on models.

MDD raises the level on abstraction at which developers create software by simplifying and formalising the various activities and tasks that make up the software development process. The idea is to automate the process of creating new software and to facilitate evolution in a rapidly changing environment by using model transformations.

In MDD models are no longer simple mediums form describing software systems or facilitating inter-team communication. Models are part of the development process, and even the code is managed as a model. Using MDD, a software system is obtained through the definition of different models at different abstraction layers. Models of a certain abstraction layer are derived from models of the upper abstraction layer, by means of automatic model transformations.

The way [31] in which the companies can adopt MDD is presented in Figure 2.2.1 which shows an adaptation of the modelling spectrum by Brown [6]. The left hand side of the spectrum represents the traditional development without graphical modelling - the code is the main artefact. The right hand side of the spectrum represents the opposite of it, the code playing a secondary role and the development is done solely using models (e.g. utilising executable modelling techniques). The model centric approach is an ambitious goal of MDD as it still is based on code while the models are the main artefacts. Most (or all, if possible) of the code is generated from models; the

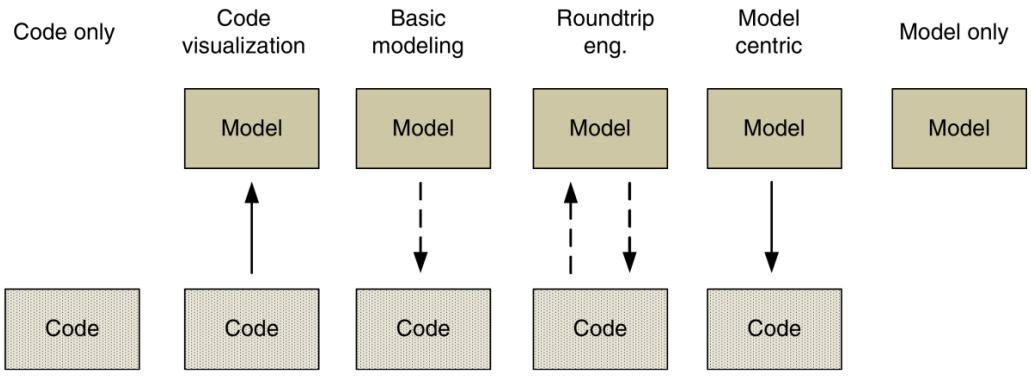

Figure 2.2.1: MDD adoption spectrum, adapted from [6]

developers, however, are given a possibility to add the code and synchronise it with models. The fact that the code can be altered after it is generated and it can be synchronised is close to the idea of round-trip engineering, where the code and the model coexist and one is synchronised once the other is updated. Such a usage scenario can be seen as an advanced usage of models which is the extension of the idea of basic modelling. The basic modelling represents a situation when models are used as a documentation and as basic (usually architectural only) sketches of the software to be built. The models and the code coexist but the code is the main artefact which is used in the course of software development. In the code visualisation scenario the code is the main artefact; models are generated automatically and are not used to develop software, but to provide means of understanding the code.

2.3 Model-Driven Security

For security-critical systems MDA facilitates security consideration from early stages in the development process and provides a seamless guidance through the development stages [13]. Model-Driven Security (MDS) [7] is a recently proposed specialisation of the MDA approach. In MDS designers specify system models along with their security requirements. Using tools, these models are automatically generated into system architectures, including complete and configured access control infrastructures. It is argued that this approach *"bridges the gap between security analysis and the integration of access control mechanism into end systems"*. MDS integrates security models with system design models and thus produces a new kind of model, security design models.

2.4 Conclusion

The Model-Driven Architecture (MDA) is a development paradigm that separates business and application logic. The business logic is expressed in platform independent models, which can be transformed with existing mappings into platform dependent models. The suggested benefits of MDA are faster development time, better design of the system, improved consistency and maintainability and increased portability across middle-ware vendors.

Model Driven Software Development (MDD) utilises models in software development. In MDD systematic use of models and model transformations is the primary engineering artefacts throughout the entire software life cycle. This, in practice often means that the code is generated from models and thus the models need to be used efficiently and effectively. The idea is to automate the process of creating new software and to facilitate evolution in a rapidly changing environment by using model transformations.

In MDS designers specify system models along with their security requirements. Using tools, these models are automatically generated into system architectures, including complete and configured access control infrastructures.

Chapter 3

Security Modelling Languages

In this chapter we will describe several security modelling languages (misuse cases, UMLSec, MAL Activity, SecureUML). We will compare these languages and explain the choice to use SecureUML to model role-bases access control.

3.1 Misuse Cases

Use cases document functional requirements of a system by exploring the scenarios in which the system may be used [4]. Scenarios are useful for eliciting and validating functional requirements, but are less suited for determining security requirements which describe behaviours not wanted in the system. Similar to anti- goals, misuse cases are a negative form of use cases and thus are use cases from the point of view of an actor hostile to the system. They are used for documenting and analysing scenarios in which a system may be attacked. Once the attack scenarios are identified, countermeasures are then taken to remove the possibility of a successful attack. Although misuses cases are not entirely design-oriented as they represent aspect of both problems and solutions, they have become popular as a means of representing security concerns in system design. Worth noting is that they are limited by the fact that they are based only on scenarios. Completeness of requirements analysed through scenarios is not guaranteed as other scenarios by which the security of a system could be exploited may be left out.

Figure 3.1.1 shows some of the use cases and misuse case in the bank account example. Use cases are represented as clear ellipses while misuse cases are represented with the shaded ellipses. The «threatens» stereotype implies that the given misuse case is a threat to the satisfaction of the requirements of the corresponding use case.

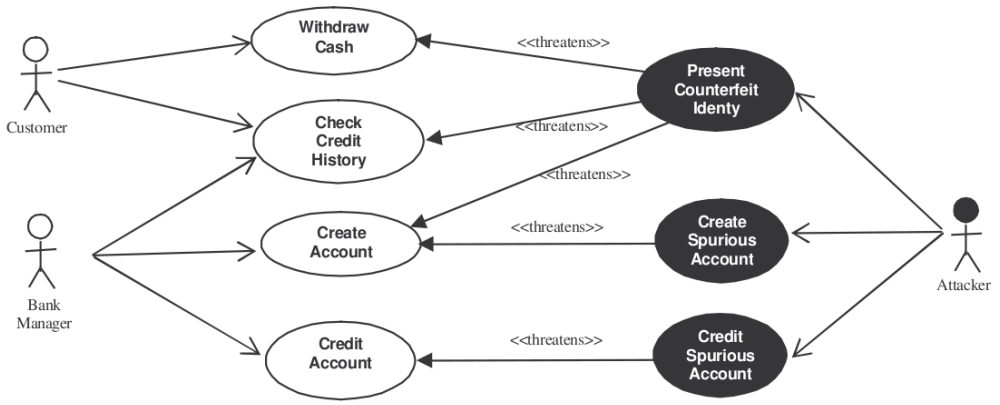

Figure 3.1.1: Use case and misuse cases in the banking example [4]

As illustrated in Figure 3.1.1, the security threats described in misuse cases are based on the functional requirements described in use cases. For example, the create account use case can be threatened by the create spurious account and present counterfeit identity misuse cases. The attacker in both misuse cases could be a malicious bank manager. An untrustworthy bank manager could also fraudulently transfer funds from a customer account to the spurious account

3.2 UMLSec

UMLSec[4] is an extension of UML which allows an application developer to embed security-related functionality into a system design and perform security analysis on a model of the system the system to verify that it satisfies particular security requirements. Security requirements are expressed as constraints on the behaviour of the system and the system design may be specified either in a UML specification or annotated in source code. UMLsec assumes that requirements have already been elicited and there exists some system design to satisfy them. Its objective is to establish whether the system design satisfies security properties. The design is then progressively refined to ensure that it satisfies security requirements.

A major purpose of security modelling is to define mechanisms to satisfy security criteria, such as confidentiality and integrity. To support this activity UMLSec is defined as a UML profile extension using stereotypes, tagged values and constraints. Constraints specify security requirements. Threat specifications correspond to actions taken by the adversary. Thus, different threat scenarios can be specified based on adversary strengths. A

subset of UMLsec are the role-based access control stereotype – «rbac» – its tagged values and constraints. This stereotype enforces RBAC in the business process specified in the activity diagram. It has three associated tags {protected}, {role}, and {right}. The tag {protected} describes the states in the activity diagram, the access to whose activities should be protected. The {role} tag may have as its value a list of pairs (actor, role) where actor is an actor in the activity diagram, and role is a role. The tag {right} has as its value a list of pairs (role, right) where role is a role and right represents the right to access a protected resource. The associated constraint requires that the actors in the activity diagram only perform actions for which they have the appropriate rights.

3.3 MAL Activity

The idea of Mal(icious)-Activity Diagrams[29] is to use the same syntax and semantics as for ordinary UML Activity Diagrams, only with the addition of the following:

- Malicious activities, shown with icons that are the inverse of normal activity icons
- Malicious actors, indicated with swim-lanes where the actor name is shown as inverse (i.e., white text on black background).
- Malicious decision boxes (i.e., where the decision is made with a malicious purpose) shown as the inverse of normal decision boxes.

3.4 SecureUML

The SecureUML meta-model [18] based on the RBAC model, is shown in Figure 3.4.1. It defines the abstract syntax to annotate UML diagrams with information pertaining to access control. The meta-model introduces concepts like `User`, `Role`, and `Permission` as well as relationships between them. Protected resources are expressed using the standard UML elements (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 (e.g., read, change, delete, and etc) on a particular type of protected resource. On another hand a `ResourceType` defines all action

types available for a particular meta-model type. An `AuthorisationConstraint` is a part of the access control policy. It 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, via permissions, to a particular model element representing a protected resource.

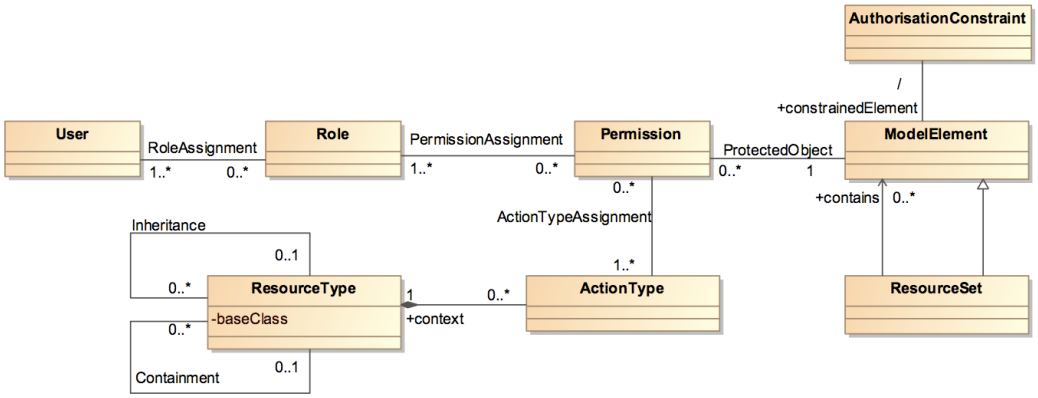

Figure 3.4.1: SecureUML meta-model (adapted from [17, 5])

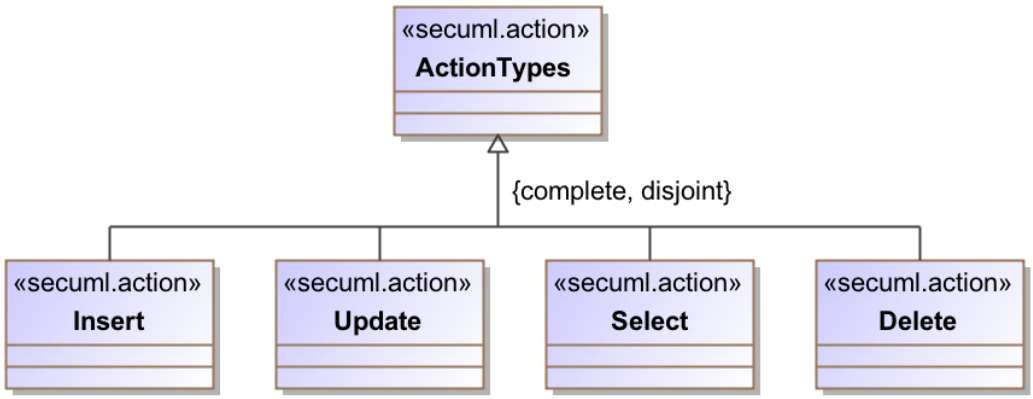

Figure 3.4.2: Action types for the security permission

At the concrete syntax level SecureUML is a "lightweight extensions" of UML, namely through stereotypes, tagged values and constraints. The stereotypes are defined for the classes and relationships in the class diagrams and are specifically oriented to the terminology of the RBAC model: «`securml.user`», «`securml.role`», «`securml.permission`», «`securml.actionType`»,

«secuml.resource», and others. The concrete syntax of SecureUML is presented in Figure 3.4.3 by illustrating it on the Meeting Scheduler example.

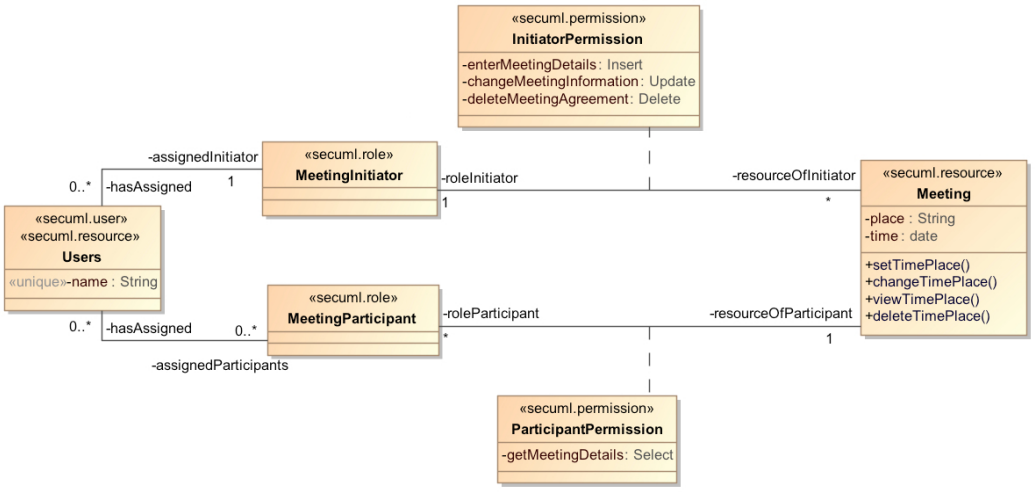

Figure 3.4.3: Meeting Scheduler with SecureUML

In this Figure we define a secure resource Meeting, which is characterised by a place where, and a 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) of this resource needs to be defined for the role `MeetingInitiator` and role `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, association class `ParticipantPermission` defines a restriction for the `MeetingParticipant` role. It introduces an action `getMeetingDetails` (of type `Select`) that says that only `MeetingParticipant` can view `Meeting` place and time. This is done through operation `viewTimePlace()` defined in

class `Meeting`. To strengthen these four permissions we define authorisation four constraints, written in Objec Constraint Language (OCL):

```
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
```

Authorisation constraint `AC#1` means that operation `setTimePlace()` (of class `Meeting`) can be executed (enter time and place), by a user set defined as variable caller, that are assigned to be `MeetingInitiators`. Similarly, the authorisation constraint `AC#2` defines restriction for operation `changeTimePlace()` and the authorisation constraint `AC#3` defines restriction for operation `deleteTimePlace()`.

```
AC#4:
   context MeetingAgreement::viewTimePlace():void
   pre: self.roleParticipant.hasBeenAssignedTo=caller
```

Authorisation constraint `AC#4` defines a permission to execute operation `viewTimePlace()` for the set of users (e.g., `caller`) that are assigned for the role `MeetingParticipant`.

3.5 Conclusion

Misuse cases and MAL-activity diagrams address security concerns through negative scenarios executed by the attacker. These modelling approaches could be applied to model RBAC in a system, however they are rather general than specific. However the languages SecureUML and UMLsec actually, contain targeted concepts for RBAC[21].

Both SecureUML and UMLsec are applicable to model RBAC solutions. They have means to address the RBAC concepts and relationships. The strong feature of SecureUML is the explicit definition of permissions through authorisation constraints using OCL. On another hand at the methodological level SecureUML only focus on the solution domain[18]. The consequences of using SecureUML is a solution to an access control problem in access rights to resource are assigned to roles and users are assigned to roles with specific authorization constraints [4]. UMLsec provides means to identify

and consider system risks, determine system vulnerabilities, and also develop solutions (RBAC is one of them) to mitigate the identified risks[18].

Although both approaches originate from UML, SecureUML and UMLsec focus on different modelling perspectives to define security policies. UMLsec is used to model dynamic characteristics of RBAC , thus it relies heavily on activity diagrams. SecureUML is used to model static characteristics of RBAC, thus, it is applied in the class diagrams[18]. Based on this, we have chosen to model the RBAC for databases using SecureUML.

Chapter 4

Tools

In this chapter we introduce the tools used in the project. First we compare modelling tools and explain the choice to use NoMagic MagicDraw in our project. In the second part we describe Oracle database and the access control. Finally we give an overview of the Velocity language and template engine.

4.1 Modelling Tools

In this section we give an overview of the modelling tools we considered to use in this project. Most of the overview of the tools is based on their official web pages. We compare the tools and explain the choice to use MagicDraw further on in this project.

4.1.1 Obeo Acceleo

Acceleo[1] is a pragmatic implementation of the Object Management Group (OMG) MOF Model to Text Language (MTL) standard. Acceleo [26] is an open source project, licensed under the Eclipse Public License (EPL) started in the French company Obeo. Acceleo is guaranteed to work on all PC-type computers on which Eclipse itself can be used. Acceleo is distributed as free software (also called "Open Source") and is licensed under EPL[2] (Eclipse Public Licence). Acceleo features:

- Complete integration with both the Eclipse environment and the EMF framework.

[1] http://www.eclipse.org/acceleo/
[2] http://www.eclipse.org/legal/epl-v10.html

- Code/model Synchronization.
- Incremental generation.
- Smooth adaptation for any industrial projects.
- Ease of update and management of templates.
- Syntax highlighting, auto-completion and error detection.

4.1.2 IBM Rational Rose Data Modeler

Rational Rose Data Modeler[3] is a visual modelling tool that makes it possible for database designers, analysts, architects, developers and anyone else on development team to work together, capturing and sharing business requirements, and tracking them as they change throughout the process. It provides the realization of the ER methodology using UML notation to bring database designers together with the software development team. With UML, the database designer can capture information like constraints, triggers and indexes directly on the diagram rather than representing them with hidden properties behind the scenes. Rational Rose Data Modeler gives allows to transfer between object and data models and take advantage of basic transformation types such as many-to-many relationships. This tool provides an intuitive way to visualize the architecture of the database and how it ties into the application.

The main features of the Data Modeler are:

- Enables database designers to visualize how the application accesses the database, so problems are escalated and resolved before deployment.
- Enables the creation of the object models, data models and data storage models and provides the ability to map logical and physical models to flexibly evolve database designs into the application's logic.
- Supports round-trip engineering between the data model, object model and defined data language (DDL) file/database management system (DBMS) and offers transformation synchronization options (synchronization between data model and object model during transformation).
- Offers a data model-object model comparison wizard, supports forward engineering of an entire database at a time, and integrates with other

[3] `http://www-01.ibm.com/software/awdtools/developer/datamodeler/features`

IBM Rational Software Development life-cycle tools Provides the ability to integrate with any Source Code Controle (SCC)compliant version control system, including IBM Rational ClearCase®.

- Provides Web publish models and reports to improve communication across the extended team.

- Operating systems supported: HP-UX, Solaris (Sun Microsystems), Windows family.

The licence for Rational Rose Data Modeler with twelve month of support costs from 1316€ to 2794€ (excluding taxes), depending on the nature of the licence (i.e floating or authorized user licence, initial fix term licence).

4.1.3 Sparx Enterprise Architect

Enterprise Architect[30] (EA) is a Computer Aided Software Engineering(CASE) tool for the design and construction of software systems. EA supports the UML 2.0 specification, which describes a visual language by which maps or models of a project can be defined. EA is a progressive tool that covers all aspects of the development cycle, providing full traceability from initial design phase through to deployment and maintenance. It also provides support for testing, maintenance and change control.

Some of the key features of Enterprise Architect are:

- Create UML model elements for a wide range of purposes.

- Place those elements in diagrams and packages.

- Create connectors between elements.

- Document the elements you have created.

- Generate code for the software you are building.

- Reverse engineer existing code in several languages.

EA can forward and reverse engineer C++, C#, Delphi, Java, PHP, VB.NET and Visual Basic classes, synchronize code and model elements, and design and generate database elements. High quality documentation can be quickly exported from your models in industry standard .RTF format and imported into Word for final customization and presentation. Enterprise Architect supports all UML 2.0 models/diagrams. It is possible to model business

processes, web sites, user interfaces, networks, hardware configurations, messages and more. Capture and trace requirements, resources, test plans, defects and change requests. From initial concept to maintenance and support, Enterprise Architect has the features to design and manage development and implementation.

The licence for Enterprise Architect varies from 95$ to 335$ per user, depending on the edition, amount purchased and the nature of the licence (i.e. floating or standard licence). Enterprise Architect has three editions:

- **Corporate** - supports large, collaborating teams with security and remote DBMS access.
- **Professional** - full featured UML modelling for work-groups, analysts and developers.
- **Desktop** - comprehensive UML modelling tool for individual analysts.

4.1.4 NoMagic MagicDraw

MagicDraw [25] is a visual UML modeling and CASE tool with teamwork support. Designed for Business Analysts, Software Analysts, Programmers, QA Engineers, and Documentation Writers, this dynamic and versatile development tool facilitates analysis and design of Object Oriented (OO) systems and databases. It provides the industry's best code engineering mechanism (with full round-trip support for Java, C#, C++, WSDL, XML Schema, and CORBA IDL programming languages), as well as database schema modeling, DDL generation and reverse engineering facilities.

MagicDraw is available in several licences:

- **Personal Edition** contains UML diagramming capabilities, including full UML 2 support and extensibility features, basic reporting functionality, and image export. Exported files are stored in XMI format. All model elements can be accessed via the MagicDraw Open API. This edition has everything needed to draw, edit, and publish UML models. Personal Edition is available only in a standalone version and is not designed for use with MagicDraw Teamwork Server.
- **Standard Edition** provides all of the Features of Personal Edition and adds Web Application Extension (WAE) content, and Robustness diagrams. Standard Edition also adds model analysis and facilitation features, customizable and extendable patterns, integrations with most popular IDEs, and a set of predefined model templates and UML profiles. Standard Edition supports UNISYS XMI and the latest Model

Driven Architecture (MDA) tool offerings. UNISYS XMI diagramming extensions allow the interchange of MagicDraw models with other UML modelling tools. Standard Edition is available in standalone, floating and mobile license versions and is fully compatible with MagicDraw Teamwork Server. Standard Edition is ideally suited for analysts and architects who need various model extensions and modelling facilitations.

- **Architect Edition** is specially packaged to provide the optimal price and technical features necessary for architects who do not need the full capabilities of the Enterprise Edition. This edition combines the common functionality of the Standard Edition together with some options from the Enterprise Edition: advanced modelling facilitations and analysis, reverse engineering and code generation for DDL, WSDL, CORBA IDL and XML. Architects have less need for IDE integrations as well as Java and C++ code engineering, so these capabilities are not included.

- **Professional Edition** is built on the Standard Edition capabilities and is available in one of three programming language specific versions: Java, C++ and C#. In addition to the Standard Edition features, Professional Edition adds code generation and reverse engineering functionality. Depending on the language version selected, the user will receive:

  - Java version - Code engineering for Java, Java bytecode. Integration with Java IDEs.
  - C++ version - Code engineering for C++.
  - C# version - Code engineering for C#, CIL (MSIL).

  Professional Edition is ideal for anyone who wants to generate code from an existing model or create a UML model from an existing project.

- **Enterprise Edition** represents the top of the line in the MagicDraw family of products. Enterprise Edition combines all of the functionality of the Personal and Standard Editions, and all three versions of the Professional Edition, into a comprehensive state-of-the-art UML programming solution. In addition Enterprise Edition features code engineering and diagramming functionality in CORBA IDL, EJB, WSDL and XML schema. For working with DB structure, Enterprise Edition not only provides code engineering and diagramming, but also provides structure retrieval via JDBC.

- **Community Edition** is free for developers working on non-commercial projects. It has a minimal functionality set and only the class diagram has no limitations. Other diagrams allow saving a project with up to 25 use cases, states, classifier roles, action states, instances, nodes, and components. Community Edition is designed for creating static structure models when XMI output is needed and it is ideally suited for Open Source projects. Printing and image export capabilities are also included.

- **Reader Edition** is made for reading and previewing UML models created with MagicDraw and it is free of charge. It is useful for sharing ideas expressed in UML with partners, colleagues, or clients, who do not have a copy of MagicDraw. Printing and image export capabilities are also included. Since MagicDraw version 14.0, Reader Edition has the ability to open and review Teamwork Server projects.

The licence fees depend on the licence and licence type and are presented in Table 4.1.

Licence Type	**Commercial Licence**			**Academic Licence**
	Standalone	**Mobile**	**Floating**	**Standalone**
Personal Edition	156€	N/A	N/A	53€
Standard Edition	531€	631€	849€	211€
Professional Edition	956€	1061€	1586€	319€
Architect Edition	1249€	1374€	1906€	411€
Enterprise Edition	1694€	1906€	2699€	581€

Table 4.1: Pricing of MagicDraw

4.1.5 Comparison

To choose a suitable modelling tool for the project, we compared four modelling tools having one free ware (Obeo Acceleo) and three commercial (IBM Rational Rose Data Modeler, Sparx Enterprise Architect, NoMagic MagicDraw) modelling tools.

The main criteria for the choice where:

- **Price** - Our industrial partner was already using Sparx Enterprise Architect as their modelling tool. Our goal was to investigate if we can use this tool to use existing licences or find an other suitable modelling tool with licence fees comparable with Enterprise Architect.

- **Functionality** - many modelling tools support UML and code generation from models, but due to the nature of our project custom code generation is needed.

The tools where chosen from the well-known vendors of modelling tools, which supported UML.

In the pricing criteria we ordered the tools based on the additional cost to the project and availability of the licence. Our first choise was Enterprise Achitect due to the fact that the licences where already available to be used by our industrial partner. The second choice for the tool was Obeo Acceleo, which is a free software. The first two choices would not add any additional costs to the project. The third and the fourth tools where accordingly NoMagic MagicDraw and IBM Rational Rose Data Modeler. This order, starting with Enterprise Architect, was used to investigate the usability in our project.

Sparx Enterprise Architect allows to generate RTF (Rich Text Format) and HTML (Hyper Text Mark-up Language) custom reports of the models, but it lacks the ability to generate complex reports and thus was eliminated from the choice. Based on tutorials found in the user manuals[26, **?**] of Acceleo and MagicDraw's Velocity template engine were suitable for the project. Rational Rose was eliminated from the choice by being the most expensive tool while having two cheaper alternatives available.

Ultimately MagicDraw was chosen as the tool for the project. The choice was made based on the simplicity of Velocity templates compared to the transformation engine provided by Obeo Acceleo.

4.2 Oracle DBMS

We have chosen to use Oracle DBMS in this work, because it is probably the most popular database server, with the largest share of the market [16]. It's used in most vertical market areas for a range of storage need such as financial records, human resources, billing and so on. One of the reasons for this is that Oracle was an early player in the RDBMS area and it provided versions of its database that ran on most operating systems; and it still does.

4.2.1 Oracle PL/SQL

Oracle PL/SQL [10, 3] is a completely portable, high-performance transaction processing language. Oracle PL/SQL stands for Procedural Language extension to the Standard Query Language (SQL). PL/SQL was introduced by Oracle Corporation to overcome some limitations in SQL and provide a

more complete programming solution to build mission-critical applications which run against the Oracle database. PL/SQL is an embedded language and was not designed to be used as a standalone language. It is intended to be invoked from within a host environment. The language ensures that the programs can stay entirely within the operating-system independent Oracle environment.

One of the important aspects of the language is its tight integration with SQL. This means the programs do not rely on intermediate software (e.g. Open DataBase Connectivity (ODBC) or Java DataBase Connectivity (JDBC)) to run SQL statements. PL/SQL provides among other features control flow, exception handling and advanced data types.

4.2.2 Data access control in DBMS

In this section we describe the Oracle DBMS role based access control and give an overview how to implement fine grained access control using Oracle Virtual Private Database and using views.

4.2.2.1 DBMS Role Based Access Control

The Oracle DBMS implements the notion of roles [15, 22] since early 1990s, and it includes support for administration of the access control state. Oracle has, for years, provided security at the table level and, to some extent, at the column level. Privileges may be granted to allow or restrict users (or groups of users) to access only some tables or columns. Privileges may be granted to specific users to insert only into certain tables while allowing them to select from other tables. For example, a user John can be granted `select` access to the table `Meeting` owned by Bob, which allows John to select any row from the table, but not to update, delete, or insert.

There are two kinds of privileges in Oracle: system privileges and object privileges. There are over 100 system privileges in Oracle 10g. For example, the `create role` system privilege allows one to create a new role, `drop any role` allows to drop any role, `grant any role` allows to grant any role to a user or another role. 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` and `delete`. Oracle's permission management is a hybrid of Discretionary Access Control (i.e. each object has an owner who exercises primary control over the object [8]) and RBAC. Privileges can be granted to users and to roles. And roles can be granted to roles and to users. A system privilege or a role can be granted "`with admin option`". If a user is granted a role with `admin option`, then we say the user has administrator

power over the role. This enables the user to grant the role to other users and roles as well as to revoke the role from other users or roles. A role r_1 can also be granted to another role r_2 with `admin option`, in which case any user that is a member of r_2 has administrator power over r_1 . A user can create a role if he has the `create role` system privilege and the role to be created does not already exist. When a role is created, the creator will be automatically granted the role with `admin option`. This enables the creator to further grant the role to any other role or user.

Object-level privileges satisfy many requirements, but sometimes they are not granular enough to meet the various security requirements that are often 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 departmental manager should only be able to see information about employees in his department.

4.2.3 Fine-Grained Access Control

Database fine-grained access control allows to limit or grant access to database objects in more detail than object based RBAC. It allows to specify access rules to database table rows and columns. There are two alternatives for fine grained access control in Oracle RDBMS, using: Virtual Private Database or database views.

4.2.3.1 Using Virtual Private Database

The Virtual Private Database (VPD) [23] enables data access control by user with the assurance of physical data separation. For example the VPD can ensure that on-line banking customers see only their own accounts. The web-hosting companies can maintain data of multiple companies in the same Oracle database, while permitting each company to see only its own data. Within the enterprise, the VPD results in lower costs of ownership in deploying applications. Security can be built once, in the data server, rather than in each application that accesses data. Security is stronger, because it is enforced by the database, no matter how a user accesses data. Security is no longer bypassed by a user accessing a reporting tool or new report writer.

VPD's row-level security (RLS) [14, 22] allows you to restrict access to records based on a security policy implemented in PL/SQL. A security policy simply describes the rules governing access to the data rows. This process is done by creating a PL/SQL function that returns a string. The function is then registered against the tables, views, or synonyms you want to protect

by using the `DBMS_RLS` PL/SQL package. When a query is issued against the protected object, Oracle effectively appends the string returned from the function to the original SQL statement, regardless of how that statement was executed., thereby filtering the data records. If you create the condition in such a way that it excludes all rows that should not be seen by a user, you will effectively be establishing security at the row level. Oracle's automatic application of the predicate to a user's SQL statement is a key aspect of what makes RLS secure and comprehensive.

At a high level, RLS [22] consists of three main components:

- **Policy** - a declarative command that determines when and how to apply restrictions on a user's access during queries, insertions, deletions, updates, or combinations of these operations. For example, you may want only `UPDATE`s to be restricted for a user, while `SELECT`s remain unrestricted, or you may want to restrict access for `SELECT`s only if the user queries a certain column (e.g., SALARY), not others.

- **Policy function** - a stored function that is called whenever the conditions specified by the security policy are met.

- **Predicate** - a string that is generated by the policy function, and then transparently and automatically appended by Oracle to the `WHERE` clause of a user's SQL statements.

Oracle Database 10g offers a new feature to VPD called Column Sensitive VPD[14]. The objective of this feature is to invoke the security policy when a specific column is referenced. It is possible to mask values (e.g. with `NULL`), when the user does not have access to the data. An other option is to hide an entire row, when it contains a value which is not accessible to the user.

According to Oracle Database 10g Release 2 Licensing information [2] VPD is available only for Enterprise Eddition. As we have only Oracle 10g Standard Edition at our disposal, we are unable to use this feature in this thesis.

4.2.3.2 Using Views

Oracle has supported database views for many years[14]. Views can be used to solve many challenges, and views can be a tremendous security tool. Views can hide columns, mask data values and aggregate data to remove personally identifiable information for maintaining privacy.

Views are database objects and access to them occurs at the object level. However, privileges on the view are separate and distinct from the privileges

on the underlying objects the view accesses. Allowing users access to a view and not to the underlying objects is an effective security technique for insulating your sensitive data.

Consider the example where the user Bob wishes to allow certain users to see which meeting rooms are booked at a certain time. He does not have to allow access to the `Meeting`, `Users`, `Users_Meeting` tables. He can simply create a view that contains the information based on the tables. Granting access to the view then allows users to retrieve this summary data while simultaneously maintaining separate security for the underlying objects.

This is an excellent technique in cases where privacy needs to be maintained. The view could easily be showing a medical researcher the number of patients that have been diagnosed with a certain illness. Likewise, the view could show a bank manager the number of customers with certain financial status. As the actual names are hidden by the view, any sensitive information that can be derived by correlating the department, diagnosis, or financial status with an individual is prevented.

Views are an ideal tool for providing column-level security (CLS). CLS has three possible definitions:

- **Preventing access to the column** - this means that the column is inaccessible to the users. The column values should not just be hidden - the column should not exist. Then, by granting the users access to the view and not to the underlying table, you have successfully removed user access to the sensitive column data. This is largely a security by design solution - the security was done prior to developing and deploying any application code. This may or may not be a possible solution for existing designs and fielded applications. For many applications, views can replace tables because many applications have no bias for querying directly against tables or directly against views. However, a challenge may exist if an application can only access tables or if an application is already written and you expect it to go against a specifically defined table. In the last case, you may be able to rename the table and create the view with the name the table originally had.

- **Masking and controlling the values of a column** - this means that some but not all of the column values are accessible. For the values that are not to be seen, you can mask the values returned to the user. For example, you may elect to return the string "NOT AUTHORISED" or return the value zero when a user queries a column to which you wish to hide its real value. Another masking option is to return a `null`. Returning `null` is a good choice because they are a standard value for data that does not exist.

- **Controlling access to the values within a column**. Consider an example where users can access only the place of the meeting where they are the initiators or participants. Users should be prohibited from accessing the location of other meetings. Because the user has access to certain locations of the meetings, it is not possible, to hide the entire column. To meet this CLS requirement, a view can be used with a function that masks the values of the salary column. Views with functions are an effective column-level security technique. The functions can return different values for different rows based on a policy decision that is implemented within the function.

For data updates, there exist cases, where a user cannot issue direct updates on the view. This could break your applications resulting in an error like "`ORA-01733: virtual column not allowed here`". This is an easy problem to solve. Oracle provides Instead-of triggers for performing DML operations on complex views. You can simply create an Instead-of trigger for this view. When you do this, you want to ensure the trigger's behaviour is consistent with the security policy provided by the view.

One particularly useful aspect of views is that they provide row-level security. Row-level security, sometimes referred to as fine-grained access control, ensures that security is applied not only to the object (for example, a database table) but also to each row within the object. When combined with a check constraint, views in this manner are simple to understand, implement, and manage.

This method of fine-grained access control is not limited to Oracle database, and could be implemented on any relational database system, which supports views and instead-of triggers on them. Nevertheless we will use Oracle specific syntax throughout the thesis.

4.3 Velocity

Velocity[24] is a Java-based template engine that processes templates and references Java objects to produce output documents. A basic Velocity template can contain static text, layouts, conditional statements and place holders for each referenced Java object. When a template is being read by Velocity, conditional statements will be processed and place holders will be replaced with the value from the referenced Java objects. It is possible to use Velocity, for example, to generate web pages, email, SQL, and XML documents. A template in Velocity is a text file that tells Velocity how the output should look like.

Two of the major components of Velocity are[11]:

- complete language for manipulating content including loops and conditionals
- Access to Java object methods

4.4 Conclusion

In this chapter we introduced the main tools used in this thesis. We started by comparing four modelling tools Obeo Acceleo (See Section 4.1.1), IBM Rational Rose (See Section 4.1.2), Sparx Enterprise Architect (See Section 4.1.3) and NoMagic MagicDraw (See Section 4.1.4). Based on the price and functionality MagicDraw was chosen to be used in this project. It is not the cheapest of the alternatives, but enables easily to create model transformations into code using its Velocity language based template engine.

We gave an overview of the Oracle Database. This included the Oracle specific PL/SQL language, which is a procedural language to complement the SQL language, Oracle database RBAC and two alternatives (i.e. Using database views with triggers and using Oracle Virtual Private Database (VPD)) to implement fine-grained access control on the database. Although implementing fine-grained access control using VPD is the standard way in Oracle, we chose to implement it with database views and triggers due to the fact that VPD is not available in Oracle Standard edition.

Finally we introduced the Velocity template language. This language is used to transform the SecureUML models into code, which can be executed on the database system.

Chapter 5

Modelling Guidelines

In this chapter we give guidelines on modelling security constraint for a database using SecureUML. We also provide some conventions for modelling to make the security models more easy to read. In the following we will assume the structure (e.g. tables and columns with their exact names) of the underlying database is known (i.e. the database exists or has been modelled). A primary key for every database table is mandatory, for simplicity we assume the primary key column is called uniquely `id` for every table.

The following sections are ordered in a logical way, like a security architect would most likely design the security. First we define security resources, which need to be secured. We will continue by defining security roles followed by permissions which associate the roles and resources. Finally we will define conditional constraints on the permissions.

5.1 Defining Stereotypes

SecureUML is not a part of UML, this means we need to define the required stereotypes. As the first step we need to define the following stereotypes:

- `«secuml.resource»` - stereotype for resources (i.e. the objects which need to be secured).
- `«secuml.resourceView»` - stereotype for resource view (i.e. a subset of a resource view).
- `«secuml.role»` - stereotype for roles.
- `«secuml.permission»` - stereotype for permissions.
- `«secuml.constraint»` - stereotype for constraints.

Some modelling tools like MagicDraw allow to import elements from other projects. To avoid repeatedly redefining the stereotypes for every new SecureUML project, it is recommended to define a reusable project, containing only the stereotypes, which can be easily imported.

5.2 Using separate diagrams

When modelling large systems it is useful to use several diagrams. This is essential with SecureUML, due to the fact that every permission between a secure resource and a role adds at least one association and an association class to the model and probably one or many constraint classes. This can easily result in a complex and difficult for humans to read diagram with tens or even hundreds of classes. To keep the model simple to read, we recommend to use at least one diagram for each pair of role and secure resource.

To further more enhance the readability of the diagrams, the classes with different stereotypes should be placed on every diagram the same way. We propose the layout for the diagram as follows: role on the left side , permission and their constraints on them in the middle and resource whit its constraints on the right hand side.

5.3 Defining a Secure Resource

The first thing in defining the security model we have to select the objects which need to be secure. In a database these objects are tables.

A resource in SecureUML is a class with the `«secuml.resource»` stereotype. To define a resource on a database table the resource has to be given the same name as the table and all the columns with the exact names and types have to be added as attributes of the class. In principle, the table objects in the data model can be reused as the resource by adding the stereotype, but in this case we recommend to make a copy of the object.

Resource views can be used to apply permissions on a subset of a resource (i.e. in our case, apply them only on some of the tables columns). Resource views can be applied to only one resource with a dependency association. Resource views may only have some or all the attributes as the underlying resource. A resource view has a stereotype `«secuml.resourceView»`.

5.4 Defining Roles

Roles are modelled as classes with the «secuml.role» stereotype. The main property of the role object is the name. The name is the only property of the role class which is used while generating the actual code, this means roles may have an arbitrary number of attributes and methods, but they will be ignored.

5.5 Permissions

Permissions are assigned to roles on resources. This is modelled as an association class with the «secuml.permission» stereotype between role and resource objects.

Permissions on a database security model may have up to four attributes - one for each action type. The action types (i.e. `Insert`, `Update`, `Delete`, `Select`) specify the for which action the permission is applied to. The name of the permission class and the names of the attributes are not used in the code generation process, thus may be given arbitrary descriptive names. For readability we recommend to the "Permission" suffix to every permission association class name (e.g. `InitiatorPermissioin`).

5.6 Defining Constraints

Constraints allow more precise access control with the model. Constraints may be applied to resources or permissions. The constraints are classes with the «secuml.constraint» stereotype and are applied as dependencies to the appropriate classes with the direction from the constraint towards the object. The actual constraints are specified on the constraint class and are written in PL/SQL. The reference `self` represents the database table row being currently accessed (i.e. being updated, deleted, inserted or selected). This reference with a correct column name, separated by a dot ".") may be used inside the PL/SQL constraint (e.g `self.username != 'administrator'`).

For readability We recommend to use the "Constraint" suffix for every constraint name (e.g. `ParticipantAuthConstraint`).

5.7 Conclusion

In this chapter we gave guidelines on how to model SecureUML security constraints. As a first thing the stereotypes of all the SecureUML extension have

to be defined or imported from an other existing SecureUML project. The SecureUML specific objects role, constraint, resource and resource view are based on class diagram classes. Permissions are association classes, between roles and resources or resource views.

To avoid difficult to read models, the security model should be divided into separate diagrams for each pair of role and secured resource. The secured resources (e.g. tables in databases) have to be placed with their constraints on the right hand side of the diagram. The role's have to be placed on the left side of the diagram. The permissions between roles and secured resources and their constraints should be placed in the middle of the diagram.

Part II

Contribution

Chapter 6

Contribution

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. On the one hand practitioners might not be aware of the approaches that help represent security concerns at the early system development stages. On the other hand a part of the problem might be that there exists only limited support to compare different security development languages and especially their resulting security models.

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

In "A Model-driven Role-based Access Control for SQL Databases " we present an approach for the model-driven RBAC for the 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 security models facilitate automatic code generation. We also argue that the security models should be prepared with the high-quality modelling language that ensures the model semantic completeness, and tools 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. In our case study we note 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. We identify that the SecureUML models are understandable at least to readers who are familiar with UML. This might ease communication of security solutions to project stakeholders.

6.2 Comparing Quality of Security Models: A Case Study

In "Comparing Quality of Security Models: A Case Study " [20] we compare the quality of two security models, which propose a solution to the industrial problem. One model is created using PL/SQL, a procedural extension language for SQL; another model is prepared with SecureUML, a model driven approach for security. We result in significantly better quality for the SecureUML security model: it contains higher semantic completeness and correctness, it is easier to modify and understand, and it facilitates a better communication of security solutions to the system stakeholders than the PL/SQL model.

A result review was performed together with the developers of the security models. Firstly, the developers noted that the overall quality of both models could be improved if to take into account these evaluation results. For example the traceability, annotation, and understandability of the PL/SQL model could be easily improved using code comments. However, the developers acknowledged that this is not the case in the common practice or the code comments, even if they are present, are not sufficient. On another hand to improve syntactic validity of the SecureUML model we could write the authorisation constraints in OCL instead of SQL. Secondly, developers provided few remarks regarding some qualitative properties. For instance, semantic completeness could be improved by presenting concrete instances in the models. This means hard coding in the PL/SQL model and object presentation in the SecureUML model. However, this neglects the principle of generosity in modelling. On one hand, a tool used to make the PL/SQL model, does not support hyper-linking. Although there exist several PL/SQL editing tools (e.g., Oracle SQLDeveloper or Quest Software Toad for Oracle, actually used by our industrial partner) that supports cross-references between various model elements, but these were not used in this case study.

On another hand, developers also indicated that PL/SQL grammar principles, the ones, which allow expressing procedures (e.g., PROCEDURE meeting_permissions in Fig. 4) and referring to them from the main code, could also be seen as textual cross-referencing. Thus, we estimate this qualitative property as partial for the PL/SQL model.

6.3 An Approach to Assess and Compare Quality of Security Models

In "An Approach to Assess and Compare Quality of Security Models" [19] we propose a systematic approach to assess quality of the security models. To illustrate validity of our proposal we investigate three security models, which present a solution to an industrial problem. One model is created using PL/SQL, a procedural extension language for SQL; another two models are prepared with SecureUML and UMLsec, both characterised as approaches for model-driven security. The study results in a higher quality for the later security models. These contain higher semantic completeness and correctness, they are easier to modify, understand, and facilitate a better communication of security solutions to the system stakeholders than the PL/SQL model.

Our approach to systematic approach to compare quality of security models is based on the instantiation of the SEQUAL framework. To illustrate the performance of our proposal we have executed a cases study, where we have compared quality of three security models. One model is prepared at the implementation stage using PL/SQL; other two models are developed at the system design stage using SecureUML and UMLsec. We resulted in (i) a higher quality for the SecureUML security model regarding UMLsec and PL/SQL; and (ii) higher quality for the UMLsec security model regarding PL/SQL. Thus, it suggests that practitioners should consider security analysis at the earlier stages (at least design or maybe even requirements engineering) of the software system developing. However we also note that executability of the UMLsec model is worse than executability of the PL/SQL model. Thus, if one wishes to create executable models he would prefer PL/SQL (or SecureUML) instead of UMLsec. Our comparison also identifies important directions for improvement of the security analysis at the early stages. For example, a mature security modelling method needs to be introduced in order to guide discovery of the early security requirements and to support security quality assurance through overall project planning. This would allow improving the traceability qualitative property, also facilitating recording of the rationales for security decisions. Another concern includes

development and improvement of the modelling tools (e.g., MagicDraw and Velocity interpreter) that would support the translation of the design models (e.g., SecureUML) to the implementation code (e.g., PL/SQL). For instance, we need to define guidelines and transformation rules for the OCL-based authorisation constraints. This would also improve the syntactic validity of the SecureUML model. On the other hand executability of the UMLsec security model is not supported at all – this might result in that practitioners would select the PL/SQL language instead. For the successful adoption by practitioners, model driven security analysis should be compatible with the working processes.

6.4 Conclusion

These papers make up the main contribution of this research project. In "A Model-driven Role-based Access Control for SQL Databases " we present the transformation rules from SecureUML into Oracle database statements. Based on these rules, we have developed Velocity templates, which do the transformation automatically. In "An Approach to Assess and Compare Quality of Security Models" do a survey, comparing the quality of two models. Compared to human written application source code containing security constraints, we result in significantly better quality for the SecureUML security model. Finally in "An Approach to Assess and Compare Quality of Security Models" we propose an approach to estimate the quality of models. This approach is illustrated by comparing SecureUML, UMLsec and PL/SQL code.

Part III

Conclusions

Chapter 7

Conclusions and Future Work

7.1 Limitations

As described in Chapter 5 and Appendix A "A Model-driven Role-based Access Control for SQL Databases " a primary key value for the underlying protected table is mandatory. This requirement may be an obstacle applying our method on existing systems.

Using our security constraints on foreign key values may cause problems with data integrity. It is possible to model security constraints on a foreign key column and on the referenced table such, that although the user can see the foreign key values, but not the actual records in the referenced table. This situation would not be possible without the security constraints and would result in an `ORA-02291: integrity constraint (string.string) violated - parent key not found` exception in Oracle databases.

The biggest concern of the generated code is the performance. If the generated access-control reduces the performance of the whole system (i.e. by making the database into a bottleneck), the software developers would probably discard the benefits of MDS and implement the RBAC by hand (as they have done countless times in previous projects). The most serious impact on performance is coming from the usage of security functions in the database view definition. This could result in a situation, where the database engine is not able to indexes when running queries. As the functions rely on some meta-data (e.g. user role or username), defining function indexes is not possible.

7.2 Conclusions

Today, as the databases are holding ever increasingly more sensitive data and are are accessible though web applications, they represent a valuable target for attackers. In this thesis we propose the application of the model-driven (MDD) development to define RBAC mechanism for data access at the design stages of the system development. The RBAC model created using the SecureUML approach is automatically translated to the implementation with database views and triggers with the transformation templates presented in "A Model-driven Role-based Access Control for SQL Databases " (See appendix A).

The Model-Driven Architecture (MDA) is a development paradigm that separates business and application logic. The business logic is expressed in platform independent models, which can be transformed with existing mappings into platform dependent models. The suggested benefits of MDA are faster development time, better design of the system, improved consistency and maintainability and increased portability across middle-ware vendors.

Model Driven Software Development (MDD) utilises models in software development. In MDD systematic use of models and model transformations is the primary engineering artefacts throughout the entire software life cycle. This, in practice often means that the code is generated from models and thus the models need to be used efficiently and effectively. The idea is to automate the process of creating new software and to facilitate evolution in a rapidly changing environment by using model transformations.

In MDS designers specify system models along with their security requirements. Using tools, these models are system architectures, including complete, configured access control infrastructures, are automatically generated.

Misuse cases and MAL activity diagrams address security concerns through negative scenarios executed by the attacker. These modelling approaches could be applied to model RBAC in a system, however they are rather general than specific. However the languages SecureUML and UMLsec actually, contain targeted concepts for RBAC[21].

Both SecureUML and UMLsec are applicable to model RBAC solutions. They have means to address the RBAC concepts and relationships. The difference between these languages is that SecureUML is used to model static characteristics of RBAC, thus, it is applied in the class diagrams and UMLsec is used to model dynamic characteristics of RBAC , thus it relies heavily on activity diagrams[18]. Based on this, we are using SecureUML together with OCL to model static RBAC and to be used in this project.

We have chosen to use MagicDraw as our modelling tool based on the

price and functionality. It is not the cheapest of the alternatives (Obeo Acceleo, IBM Rational Rose, Sparx Enterprise Architect), but it enables easily to create model transformations into code using its Velocity language based template engine.

PL/SQL is an Oracle database specific language, complements the SQL language. The Oracle database has two alternatives (i.e. Using database views with triggers and using Oracle Virtual Private Database (VPD)) to implement fine-grained access control. Although implementing fine-grained access control using VPD is the standard way, we chose to implement it with database views and triggers due to the fact that VPD is not available in Oracle Standard edition and not supported by other database engines.

The Velocity template language captures the information from the SecureUML model which then is used to transform into code (database views and instead-of triggers). The code can be executed on the database to enforce the security constraints.

In this chapter we gave guidelines on how to model SecureUML security constraints. As a first thing the stereotypes of all the SecureUML extension have to be defined or imported from an other existing SecureUML project. The SecureUML specific objects role, constraint, resource and resource view are based on class diagram classes. Permissions are association classes, between roles and resources or resource views. To avoid difficult to understand models we propose a simple convention, how to model the security constraints using SecureUML.

7.3 Future Work

Staron [31] 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 [31] we see necessary 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 [9]. This would improve traceability, also record rationales for security decisions.

Another concern is development of the modelling tools (e.g., MagicDraw and Velocity interpreter) that would support a translation of the design mod-

els (e.g., SecureUML) to the implementation code (e.g., database views and instead-of triggers). For instance, we need to define guidelines and transformation rules for the OCL-based authorisation constraints.

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

Oracle databases support fine-grained access control using Virtual Private Database [23] (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).

To adapt the solution in an industrial project, it is essential to analyse the performance impact. The goal of the survey is to estimate the scalability of the solution and if it is practically usable.

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.

ROLLIPÕHINE LIGIPÄÄSU KONTROLL ANDMEBAASIDELE

Henri Lakk

Resümee

Liikudes üha enam paberivaba äri suunas, hoitakse üha enam tundlikku informatsiooni andmebaasides. Sellest tulenevalt on andmebaasid ründajatele väärtuslik sihtmärk. Levinud meetod andmete kaitseks on rollipõhine ligipääsu kontroll (*role-based access control*), mis piirab süsteemi kasutajate õiguseid vastavalt neile omistatud rollidele. Samas on turvameetmete realiseerimine arendajate jaoks aeganõudev käsitöö, mida teostatakse samaaegselt rakenduse toimeloogika realiseerimisega. Sellest tulenevalt on raskendatud turva vajaduste osas kliendiga läbirääkimine projekti algfaasides. See omakorda suurendab projekti reaalsete arenduskulude kasvamise riski, eriti kui ilmnevad turvalisuse puudujäägid realisatsioonis.

Tänapäeva veebirakendustes andmebaasi ühenduste puulimine (*connection pooling*), kus kasutatakse üht ja sama ühendust erinevate kasutajate teenindamiseks, rikub vähima vajaliku õiguse printsiipi. Kõikidel ühendunud kasutajatel on ligipääs täpselt samale hulgale andmetele, mille tulemusena võib lekkida tundlik informatsioon (näiteks SQLi süstimine (*SQL injection*) või vead rakenduses).

Lahenduseks probleemile pakume välja vahendid rollipõhise ligipääsu kontrolli disainimiseks tarkvara projekteerimise faasis. Rollipõhise ligipääsu kontrolli modelleerimiseks kasutame UML'i laiendust SecureUML. Antud mudelist on võimalik antud töö raames valminud vahenditega genereerida koodi, mis kontrollib ligipääsu õiguseid andmebaasi tasemel. Antud madaltaseme kontroll vähendab riski, et kasutajad näevad andmeid, millele neil ligipääsu õigused puuduvad.

Antud töös läbiviidud uuring näitas, et mudelipõhine turvalisuse arendamise kvaliteet on kõrgem võrreldes programmeerijate poolt kirjutatud koodiga. Kuna turvamudel on loodud projekteerimise faasis on selle semantiline täielikkus ja korrektsus kõrge, millest tulenevalt on eda kerge lugeda ja muuta ning seda on lihtsam kasutada arendajate ja klientide vahelises suhtluses.

Bibliography

[1] Model driven development for j2ee utilizing a model driven architecture (mda) approach. Technical report, The Middleware Company, 2003.

[2] *Oracle Database Licensing Information 10g Release 2 (10.2).* `http://docs.oracle.com/cd/B19306_01/license.102/b14199.pdf`, 2010.

[3] Shashaanka Agrawal, Cailein Barclay, Eric Belden, Dmitri Bronnikov, Sharon Castledine, Thomas Chang, Ravindra Dani, Chandrasekharan Iyer, Susan Kotsovolos, Neil Le, Warren Li, Bryn Llewellyn, Valarie Moore, Chris Racicot, Murali Vemulapati, John Russell, Guhan Viswanathan, and Minghui Yang. *PL / SQL User's Manual and Reference.* Oracle, 2005.

[4] Arosha Bandara, Hayashi Shinpei, Jan Jurjens, Haruhiko Kaiya, Atsuto Kubo, Robin Laney, Haris Mouratidis, Armstrong Nhlabats, Bashar Nuseibeh, Yasuyuki Tahara, Thein Tun, Hironori Washizaki, Nobukazi Yoshioka, and Yijun Yu. Security patterns: Comparing modeling approaches. Technical report, Department of Computing,Faculty of Mathematics, Computing and Technology,The Open University, 2009.

[5] David A. Basin, J"urgen Doser, and Torsten Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, 2006.

[6] Alan Brown. An introduction to model driven architecture. *The Rational Edge*, Part I: MDA and today's systems:13–17, February 2004.

[7] Manuel Clavel, Viviane Silva, Christiano Braga, and Marina Egea. Model-driven security in practice: An industrial experience. In *Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA '08, pages 326–337, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] Stephen Dranger, Robert Sloan, and Jon Solworth. The complexity of discretionary access control. In Hiroshi Yoshiura, Kouichi Sakurai, Kai Rannenberg, Yuko Murayama, and Shinichi Kawamura, editors, *Advances in Information and Computer Security*, volume 4266 of *Lecture Notes in Computer Science*, pages 405–420. Springer Berlin / Heidelberg, 2006. 10.1007/11908739_29.

[9] Éric Dubois, Patrick Heymans, Nicolas Mayer, and Raimundas Matulevičius. A systematic approach to define the domain of information system security risk management. In Selmin Nurcan, Camille Salinesi, Carine Souveyet, and Jolita Ralyté, editors, *Intentional Perspectives on Information Systems Engineering*, pages 289–306. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-12544-7_16.

[10] Steven Feuerstein and Bill Pribly. *Oracle PL/SQL Programming*. O'Reilly Media Inc, 4th edition edition, 2005.

[11] Joseph D. Gradecki and Jim Cole. *Mastering Apache Velocity*. Wiley Publishing, Inc., 2003.

[12] Jean-Marc J'ez'equel, Heinrich Hußmann, and Stephen Cook, editors. *UML 2002 - The Unified Modeling Language, 5th International Conference, Dresden, Germany, September 30 - October 4, 2002, Proceedings*, volume 2460 of *Lecture Notes in Computer Science*. Springer, 2002.

[13] Jan Jürjens and Pasha Shabalin. Tools for secure systems development with UML: Security analysis with atps. In Maura Cerioli, editor, *Fundamental Approaches to Software Engineering*, volume 3442 of *Lecture Notes in Computer Science*, pages 305–309. Springer Berlin / Heidelberg, 2005. 10.1007/978-3-540-31984-9_23.

[14] David Knox. *Effective Oracle Database 10g Security by Design*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2004.

[15] Ninghui Li and Ziqing Mao. Administration in role-based access control. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, ASIACCS '07, pages 127–138, New York, NY, USA, 2007. ACM.

[16] David Litchfield, Chris Anley, John Heasman, and Bill Grindlay. *The Database Hacker's Handbook: Defending Database Servers*. John Wiley & Sons, 2005.

[17] Torsten Lodderstedt, David A. Basin, and J"urgen Doser. SecureUML: A UML-based modeling language for model-driven security. In *UML*, pages 426–441, 2002.

[18] Raimundas Matulevicius and Marlon Dumas. A comparison of SecureUML and UMLsec for role-based access control. In *Databases and Information Systems*, pages 171–185. University of Latvia Press, Riga, Latvia, July 2010.

[19] Raimundas Matulevicius, Henri Lakk, and Marion Lepmets. An approach to assess and compare quality of security models. *Comput. Sci. Inf. Syst.*, 8(2):447–476, 2011.

[20] Raimundas Matulevicius, Marion Lepmets, Henri Lakk, and Andreas Sisask. Comparing quality of security models: A case study. In Mirjana Ivanovic, Bernhard Thalheim, Barbara Catania, and Zoran Budimac, editors, *ADBIS (Local Proceedings)*, volume 639 of *CEUR Workshop Proceedings*, pages 95–109. CEUR-WS.org, 2010.

[21] Raimundas Matulevičius and Marlon Dumas. Towards model transformation between SecureUML and UMLsec for role-based access control. In *Proceedings of the 2011 conference on Databases and Information Systems VI: Selected Papers from the Ninth International Baltic Conference, DB&IS 2010*, pages 339–352, Amsterdam, The Netherlands, The Netherlands, 2011. IOS Press.

[22] Arup Nanda and Steven Feuerstein. *Oracle PL/SQL for DBAs*. O'Reilly Media, Inc., 2005.

[23] Kanichiro Nishida and Shankar Duvvuri. *Row Level Security with BI Publisher Enterprise*. Oracle, January 2009. Available online.

[24] No Magic, Inc. *Creating a Report Template to Use in MagicDraw*, version 16.6 edition, 2010.

[25] No Magic, Inc. *MagicDraw User's Manual*, version 16.6 edition, 2010.

[26] Obeo, `http://www.acceleo.org/doc/obeo/en/acceleo-2.6-user-guide.pdf`. *Acceleo User Guide*.

[27] Alex Roichman and Ehud Gudes. Fine-grained access control to web databases. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, SACMAT '07, pages 31–40, New York, NY, USA, 2007. ACM.

[28] Pablo S'anchez, Ana Moreira, Lidia Fuentes, Joao Ara'ujo, and Jos'e Magno. Model-driven development for early aspects. *Inf. Softw. Technol.*, 52:249–273, March 2010.

[29] Guttorm Sindre. Mal-activity diagrams for capturing attacks on business processes. In Pete Sawyer, Barbara Paech, and Patrick Heymans, editors, *Requirements Engineering: Foundation for Software Quality*, volume 4542 of *Lecture Notes in Computer Science*, pages 355–366. Springer Berlin / Heidelberg, 2007.

[30] Sparx Systems, `http://www.apexnet.com.ar/downloads/EAUserGuide5.0.pdf`. *Enterprise Architect Version 5.0 User Guide*, July 2005.

[31] Miroslaw Staron. Adopting model driven software development in industry – a case study at two companies. In *Model Driven Engineering Languages and Systems*, pages 57–72, 2006.

Part IV

Publications

A

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

Publication:

Raimundas Matulevičius and Henri Lakk, A Model-driven Role-based Access Control for SQL Databases, *Journal of Database Management*, Under review, 2012

A

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" (Firesmith, 2007). 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 (Sindre and Opdahl 2001, 2005, Jurjens, 2005), 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 on 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 point. This is a serious hindrance to secure system development, since the early consideration of security (e.g., at the requirements and/or

design stage) allows engineers to envisage threats, their consequences and design countermeasures. Then design alternatives, that do not offer a sufficient security level, could be discarded.

One possible suggestion to solve the above problem is an approach called model driven architecture (MDA). MDA provides a solution for the system development process based on models (de Miguel et al., 2002). The models, the simplified representations of reality, could 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 could address different objectives (e.g. integration of technical concepts, transformations between different modelling language).

For security-critical systems MDA facilitates security consideration from early stages in the development process and provides a seamless guidance through the development stages (Jurjens, 2005). Model driven security (MDS) could be supported using different modelling languages. On the 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** (Yu, 1997), Tropos (Bresciani Et al., 2004) or KAOS (Dardenne et al., 1993, van Lamsweerde, 2004); *data models* created with ER (Chen, 1976) or UML (OMG, 2005). These modelling languages are not, specifically, designed with security in mind and, thus, their support for security is weak.

On the other hand there exist *security modelling languages* specifically dedicated to analysis and modelling of system security concerns. For example, abuse frames (Lin et al., 2004) suggest means to consider security during early requirements engineering stage. Secure *i** (Elahi, et al., 2007) addresses security trade-offs. KAOS' extension to security (van Lamsweerde, 2004) was augmented with anti-goal models designed to elicit attackers' rationales. Tropos has been extended with the notions of ownership, permission and trust (Giorgini et al., 2005). Another version of Secure Tropos (Mouratidis, 2006) defines security through security constraints. Abuse cases (McDermott and Fox, 1999), misuse cases (Sindre and Opdahl, 2001, 2005) and mal-activity diagrams (Sindre, 2007) 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 (Jurjens, 2005) and SecureUML (Lodderstedt et al., 2002; Basin et al., 2006). Those languages are, basically, used to address security concerns during the system design stage. All the 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 support validity criteria with respect to the stakeholder needs. However, in general, they provided limited support to transform 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 ER diagram) to SQL code is highly supported by the variety of the modelling tools, we did not observe how 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 required 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 (Lodderstedt et al., 2002; Basin et al., 2006) 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, for example, 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 database may require some modifications depending on the differences in the syntax.

In order to validate our proposal we perform a cases study, where we compare two security models: one directly created in PL/SQL (Feuerstein, et al., 2005), another created in SecureUML and, then, transformed into database views and instead-of triggers code using our proposal. We result in better quality for the second model.

This paper is organised as follows: Section 2 gives the background for our research. It focusses 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 our contribution and concludes the paper.

2. BACKGROUND

2.1. Role-based Access Control

The core RBAC model 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 includes 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 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.

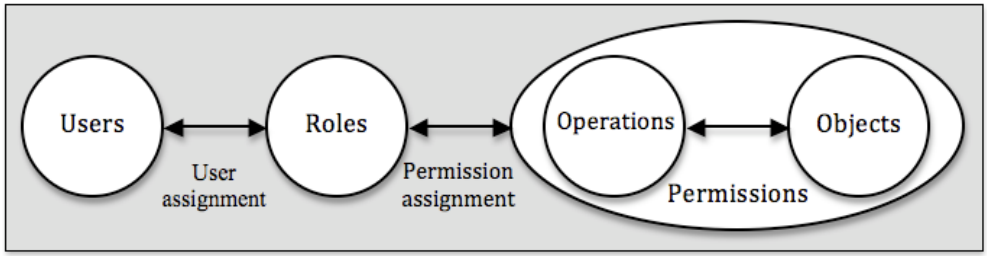

Figure 1. The core RBAC model (adapted from (Sandhu and Coyne, 1996; Ferraiolo et al., 2001))

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 (Lodderstedt et al., 2002; Basin et al., 2006).

2.2.1. SecureUML Abstract Syntax

An abstract syntax of SecureUML (Lodderstedt et al., 2002; Basin et al., 2006) is organised as a UML class diagram and 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.

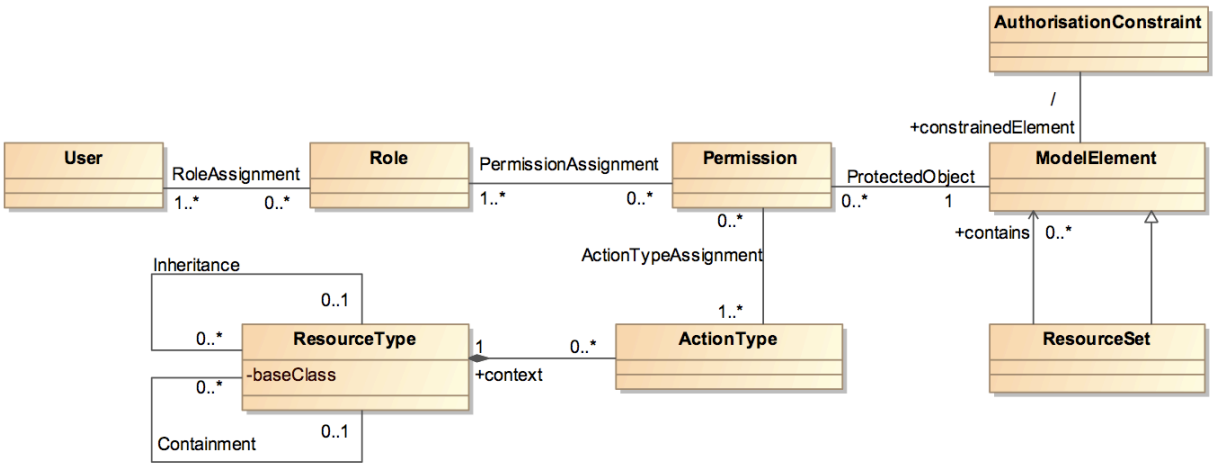

Figure 2. SecureUML meta-model (adapted from (Lodderstedt et al., 2002; Basin et al., 2006)

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. On the other hand a *ResourceType* defines all action types available for a particular meta-model type.

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.

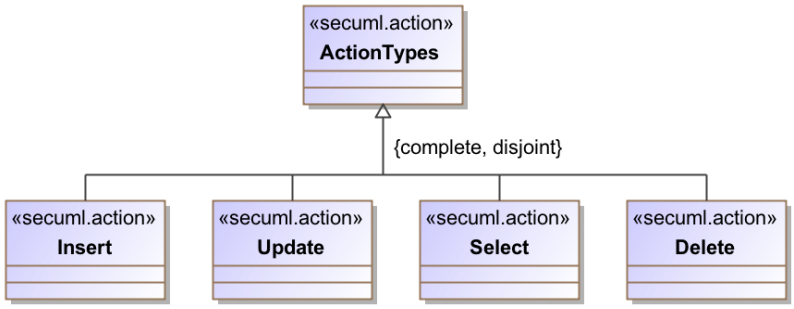

Figure 3. Action types for the security permission

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 and are specifically oriented to the RBAC terminology. In Figure 4 we illustrate the SecureUML concrete syntax through the *Meeting Scheduler* example (Feather et al, 1997).

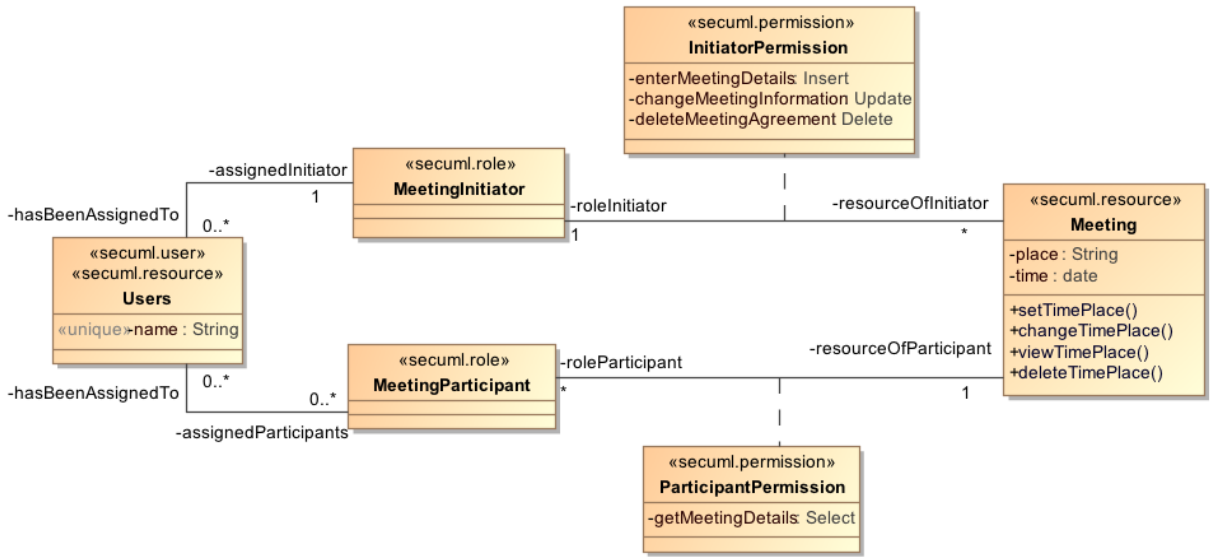

Figure 4. Meeting Scheduler with SecureUML

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*().

To strengthen these four permissions we define authorisation four constraints, written in object constraint language (OCL) (Warner and Kleippe, 2003):

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 are 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 (Basin et al. 2006) 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 (Anaya et al., 2010; Dubois et al., 2010) we discuss the conceptual SecureUML semantics for the purpose of system modelling. We utilise the RBAC model to define semantics of the SecureUML constructs (Matulevičius and Dumas, 2010, 2011) as illustrated in Table 1.

Some mappings between RBAC and SecureUML are understood as a lexical correspondence. For example, 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 a class Users with a 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 a 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 (Nanda Feuerstein, 2005; Li and Mao, 2007), and includes 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 is requirement is not solvable with object-level privileges and, therefore, is usually implemented in a higher (e.g. business or presentation) tier.

In (Roichman *et al*., 2007) 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 an application layer that interfaces directly with the database layer. This means 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 not able to identify neither who has accessed the data not the transaction of the same user. The web application does not open nor close a connection before/ after each request but uses a connection pool to store connections. Using such a connection pool a large number of users can also 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 horizontal (i.e., access to the data of other user) and vertical (i.e., access to a 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" (Roichman *et al*., 2007).

3. DATA ROLE-BASED ACCESS CONTROL USING MODEL-DRIVEN SECURITY

Our proposal to model and implement security policy for the data is presented in Figure 5. It consists of two major steps. Firstly, we present a transform a *data model*, expressed as the UML class diagram, to the data code represented in the SQL code. Secondly, we propose a way 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. In Section 3.3 we overview the software tools used to support our proposal.

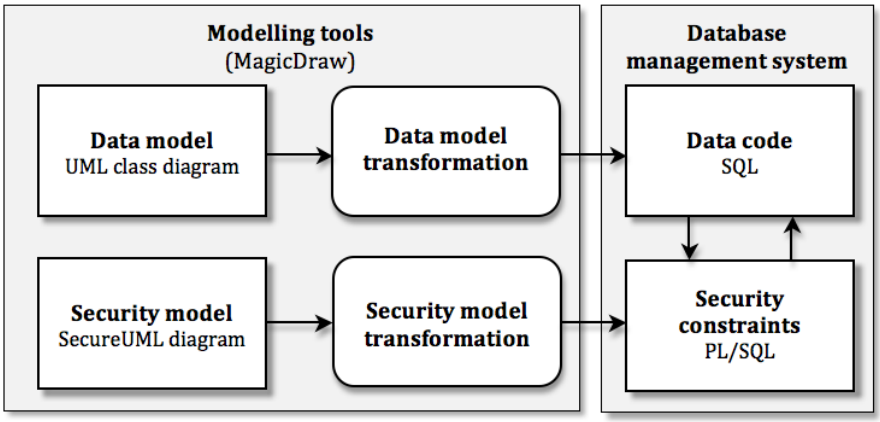

Figure 5. Data and Security Model Transformation

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).

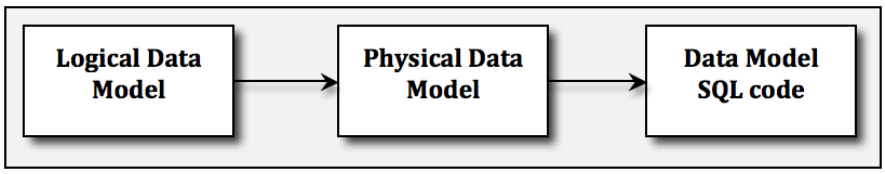

Figure 6. Data Model Transformation

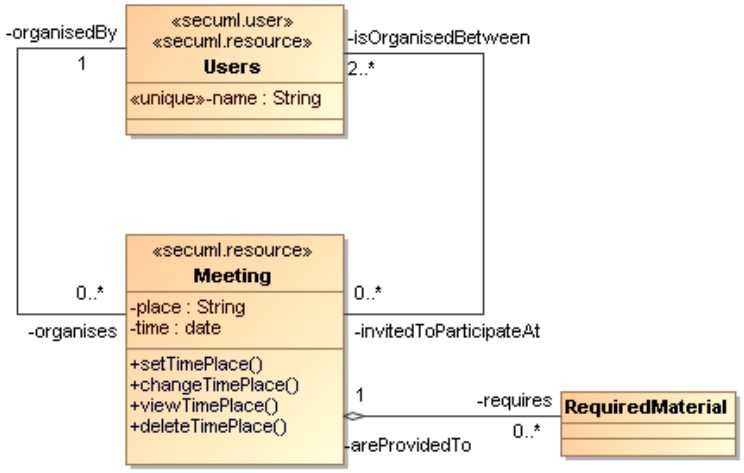

Figure 7. Logical Data Model

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 columns names and data types.

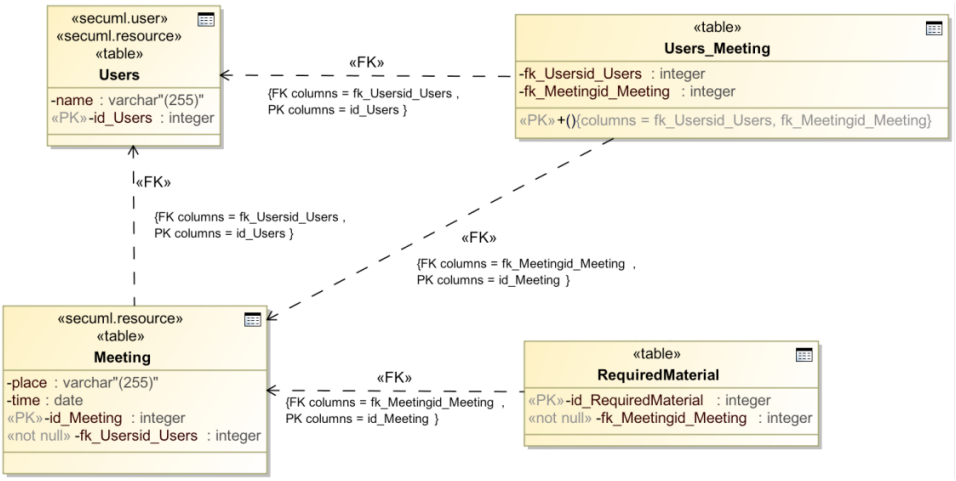

Figure 8. Physical Data Model

```
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

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 constrains AC#1-4. Appendix A also includes procedure sec.is_role(argument), used to identify role for which security action is granted.

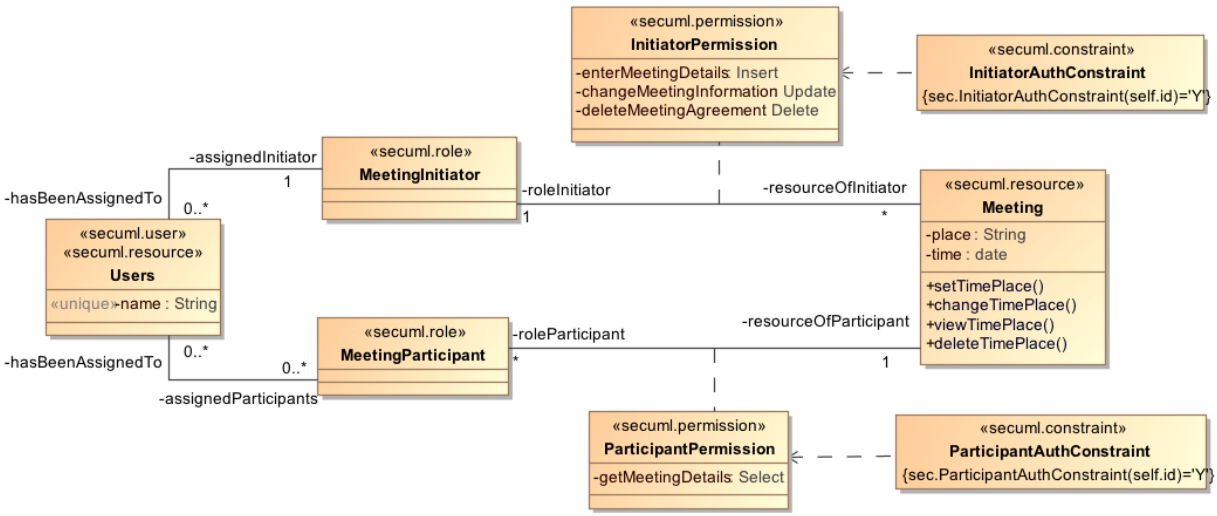

Figure 10. Meeting Scheduler Security Model

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.

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

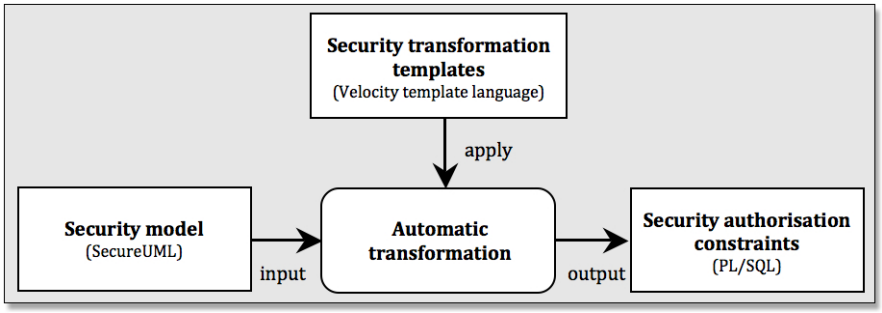

Figure 11. Security model transformation

3.2.1. Select *authorisation constraint*

The ***Select*** authorisation constraint specifies the data that could be viewed at the runtime. At the data level the secure resources are interpreted as database tables as discusses 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 "*_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 name[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 a 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 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)).

[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).

```
-- 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 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 (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>>*) parts.

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. 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 failure. On one hand if there is no security violation and if the new inserted values are valid, trigger will give a positive result. On another hand 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 the 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 an explicit constraints (e.g. sec.InitiatorAuthConstraint(self.id) = 'Y', captured from the class InitiatorAuthConstraint).

```
-- 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

```
-- 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

[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.

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 date, these need to be selected from the data table.

```
-- 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

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 a success (if there is no security violation and if the new values for the updated data are valid) or in a 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 of 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 check for the implicit (see, sec.is_role('Meeting')='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 level of abstraction for various modelling goals. To illustrate and implement our proposal 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 a generation of the database SQL code, as described in Section 3.1.

Our proposal focusses on the transformation rules to translate the security model expressed in SecureUML to the security authorisation constrains. To generate these transformation rules adapted to the transformation templates, we use a *report generation* mechanism implemented in MagicDraw. It supports capturing, every entity of the UML (SecureUML) model 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 the websites, the Velocity engine was adapted for the MagicDraw tool to capture information from the UML diagrams. In our case we capture security related entities and transform them into database views and instead-of triggers, as illustrated in Section 3.2 for *insert* (Figure 12), *select* (Figure 13), *delete* (Figure 14), and *update* (Figure 15) actions.

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.

4. A CASE STUDY

In order to validate the performance of our proposal, we have performed a case study, where we compare a 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 (Matulevičius *et al.,* 2010; 2011). 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) developed by Krogstie (1998, 2001) that defines the assessment of the model quality

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

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

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

by distinguishing between quality goals and means to achieve these goals. More specifically we have analysed the following quality types:

- *Semantic* (i.e., correspondence between the model and its semantic domain) quality was considered with respect to semantic completeness, semantic correctness, traceability, annotation, and modification;
- *Syntactic* (correspondence between a model and modelling language) was defined in terms of syntactic validity and syntactic completeness; and
- *Pragmatic* (correspondence between a model and its technical and social interpretation) quality was expressed through cross-referencing, organisation, understandability, and executability.

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 rather assessed at the same level. The design and precise details of the case study are provided in (Matulevičius *et al.,* 2010; 2011).

Within this case study we certainly acknowledge that we have assessed both security models using a certain set of qualitative properties (and their appropriate measures). This might, however, affect the conclusion validity, because if any other qualitative properties were 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. Assessment Results of the Semantic Quality (the higher 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 an approach for the model-driven RBAC for the 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 (de Miguel *et al.,* 2002; Staron, 2006; the Middleware, 2003) 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. Similarly to (Clavel *el al*., 2008; MacDonald *et al*., 2005; the Middleware, 2003) 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 (de Miguel *et al.,* 2002) that ensures the model semantic completeness, and tools (MacDonald *et al.,* 2005) 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 (the Middleware, 2003).

We identified one case study performed by Clavel *et al*. (2008), 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 note 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 (Clavel *el al*., 2008) we identify that the SecureUML models are understandable at least to readers who are familiar with UML. This might ease communication of security solutions to project stakeholders (MacDonald *et al.,* 2005).

5.2. RBAC and Security Modelling Languages

In (Bandara *et al*., 2009) the survey of security modelling languages shows that SecureUML does not explicitly model security criteria (such as confidentiality, integrity, and availability), but it focusses 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. Jayaram and Mathur (2005) 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 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 a 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.

5.3. RBAC for SQL Databases

Oh and Park (2001) propose a model-driven approach to manage RBAC policies on top of SQL databases. The paper specifically focusses on a task-RBAC model, whereby permissions are

assigned to tasks and tasks are assigned to roles. In contrast, our approach does not require a 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 (Oh and Park, 2001) is based on a 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 (Bertino *et al*, 2001) allow designers to capture time-sensitive access control policies, such as the fact that a user only has access to certain resources during a specified period of time. Barker *et al* (2003) 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 as input SecureUML models. Extending SecureUML with temporal constraints and enhancing the PL/SQL generation method accordingly is a direction for future work.

Other related work has addressed the issue of generating code for RBAC models in the context of data warehouses. Blanco *et al* (2008) 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

Staron (2006) 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 Staron (2006) we see necessary 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 (Dubois *et al.*, 2010). This would improve traceability, also record rationales for security decisions.

Another concern is development of the *modelling tools* (e.g., MagicDraw and Velocity interpreter) that would support a translation of the design models (e.g., SecureUML) to the implementation code (e.g., database views and instead-of triggers). For instance, we need to define guidelines and transformation rules for the OCL-based authorisation constraints.

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

Oracle databases support fine-grained access control using Virtual Private Database (Nishida and Duvvuri, 2009) (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. 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. **Anaya V., Berio G., Harzallah M., Heymans P., Matulevičius R., Opdahl A. L., Panetto H., Verdech M. J. (2010)** The Unified Enterprise Modelling Language – Overview and Further Work, Computers in Industry, Elsevier Science Publication, Vol 61, No 2, 99-111
2. **Bandara, A., Shinpei, H., Jurjens, J., Kaiya, H., Kubo, A., Laney, R., Mouratidis, H., Nhlabatsi, A., Nuseibeh, B., Tahara, Y., Tun, T., Washizaki, H., Yoshioka, N., Yu, Y. (2009)** Security Patterns: Comparing Modelling Approaches. Technical Report No 1009/06, Department of Computing Faculty of mathematics, Computing Technology, The Open University
3. **Barker S., Douglas P., Fanning T. (2003)** Implementing RBAC Policies. In Gudes E., Shenoi S. (eds) Research Directions in Data and Applications Security, Kluwer Academic Publishers Group, pp. 27-36
4. **Basin, D., Doser, J., Lodderstedt, T. (2006)** Model Driven Security: from UML Models to Access Control Infrastructure. ACM Transactions on Software Engineering and Methodology (TOSEM), 15 (1), 39--91.
5. **Bertino E., Bonatti P. A., Ferrari E. (2001)** TRBAC: A Temporal Role-based Access Control Model. In Transactions on Information and Security Systems (TISSEC), ACM, 4 (3), 191-233
6. **Blanco C., de Guzman I. G.-R., Fernandez-Medina E., Trujillo J., Piattini M. (2008)** Automatic Generation of Secure Multidimensional Code for Data Warehouses: an MDA Approach. Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems, pp. 1052-1068
7. **Bresciani P., Perini A., Giorgini P., Giunchiglia F., Mylopoulos J. (2004)** Tropos: An Agent-Oriented Software Development Methodology, Autonomous Agents and Multi-Agent Systems, Springer.
8. **Chen P. P. (1976)**. The Entity-Relationship Model: Towards a Unified View of Data. ACM Transactions on Database Systems, 1(1), 9-36.
9. **Clavel, M., Silva, V., Braga, C., Egea, M. (2008)** Model-driven Security in Practice: an Industrial Experience, In Proceedings of the 4th European Conference on Model Driven Architecture: Foundations and Applications, Springer-Verlag, pp. 326--337
10. **Dardenne A., van Lamsweerde A., Fickas S. (1993)** Goal-Directed Requirements Acquisition. Science of Computer Programming Vol. 20, North Holland, 1993, pp. 3-50.
11. **de Miguel M., Jourdan J., Salicki S. (2002)** Practical Experiences in the Application of MDA. In Proceedings of the 5th International Conference on The Unified Modeling Language, Springer-Verlag, 128--139

12. **Dubois, E.; Heymans, P.; Mayer, N.; Matulevičius, R. (2010)** A Systematic Approach to Define the Domain of Information System Security Risk Management. Nurcan, S.; Salinesi C.; Souveyet C.; Ralyte, J. (Eds.). Intentional Perspectives on Information Systems Engineering, 289-306, Springer Heidelberg, ISBN: 978-3-642-12543-0
13. **Elahi, G., Yu, E. (2007)** A Goal Oriented Approach for Modeling and Analyzing Security Trade-Offs, In: Parent et al. (eds.), Proceedings of the 26th International Conference on Conceptual Modelling
14. **Feather, M. S., Fickas, S., Finkelstein, A. and van Lamsweerde A. (1997)** Requirements and Specification Exemplars. Automated Software Engineering, 4, 419-438.
15. **Ferraiolo D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R. (2001)** Proposed NIST Standard for Role-based Access Bontrol. ACM Transactions on Information and System Security (TISSEC), 4(3), 224--274
16. **Feuerstein, S., Pribly, B. (2005)** Oracle PL/SQL Programming. O'Reilly Media Inc, 4th edition edition
17. **Firesmith D. G. (2007)** Engineering Safety and Security Related Requirements for Software Intensive Systems, ICSE 2007 tutorial, Minneapolis
18. **Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N. (2005)** Modeling Security Requirements Through Ownership, Permision and Delegation. In Proceedings of the 13th IEEE International Conference on Requirements Engineering, IEEE Computer Society
19. **Jayaram, K.R., Mathur, A.P. (2005)** Software Engineering for Secure Software – State of the Art: a Survey. Technical report CERIAS TR 2005-67, Department of Computer Sciences & CERIAS, Purdue University
20. **Jurjens J. (2005)** Secure Systems Development with UML, Springer-Verlag Berlin Heidelberg
21. **Krogstie, J. (2001)** A Semiotic Approach to Quality in Requirements Specifications. In Proceedings of IFIP 8.1 working Conf. on Organisational Semiotics, 231--249.
22. **Krogstie, J. (1998)** Using a Semiotic Framework to Evaluate UML for the Development for Models of High Quality. In: Siau, K., Halpin, T. (eds.) Unified Modelling Language: System Analysis, Design and Development Issues, IDEA Group Publishing, pp. 89--106
23. **van Lamsweerde, A. (2004)** Elaborating Security Requirements by Construction of Intentional Anti-models. In Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society 148--157
24. **van Lamsweerde A. (2009)** Requirements Engineering: From System Goals to UML Models to Software Specifications, Wiley
25. **Li N., Mao Z. (2007)** Administration in role-based access control. In Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security, ASIACCS '07, ACM, 127--138
26. **Lin, L., Nuseibeh, B., Ince, D., Jackson, M. (2004)** Using Abuse Frames to Bound the Scope of Security Problems. In Proceedings of the 12th IEEE International Conference on Requirements Engineering, IEEE Computer Society 354--355
27. **Lodderstedt, T., Basin, D., Doser, J. (2002)** SecureUML: A UML-based Modeling Language for Model-driven Security. In Proceedings of the 5th International Conference on The Unified Modeling Language, LNCS, vol. 2460 Springer-Verlag, 426--441
28. **MacDonald, A., Russell, D., Atchison, B. (2005)** Model-driven Development within a Legacy System: An Industry Experience Report. In Proceedings of the 2005 Australian Software Engineering Conference (ASWEC'05). IEEE Computer Science
29. **Matulevičius, R., Dumas, M. (2010)** 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, Latvia; July 5-7, 2010. (Eds.) Barzdins, J.; Kirikova, M.. Latvia: University of Latvia Press, Riga, Latvia, 2010, 171 - 185.

30. **Matulevičius, R., Dumas, M. (2011)** Towards Model Transformation between SecureUML and UMLsec for Role-based Access Control. Barzdins, J., Kirikova, M. (Eds.). Databases and Information Systems VI (339 - 352). IOS Press
31. **Matulevičius, R.; Lakk, H.; Lepmets, M., Sisask, A., (2010)** Comparing Quality of Security Models: a Case Study, Proceedings of the ADBIS 2010 workshop MDASD 2010, University of Novi Sad Press, Serbia
32. **Matulevičius, R.; Lakk, H.; Lepmets, M. (2011)** An Approach to Assess and Compare Quality of Security Models. Computer Science and Information Systems, ComSIS Consortium, 8 (2), 447 - 476.
33. **McDermott, J., Fox, C. (1999)** Using Abuse Case Models for Security Requirements Analysis. In Proceedings of the 15th Annual Computer Security Applications Conference
34. **The Middleware Company (2003)** Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach: Productivity Analysis, MDA Productivity case study
35. **de Miguel, M., Jourdan, J., Salicki, S. (2002)** Practical Experiences in the Application of MDA. In Proceedings of the 5th International Conference on The Unified Modeling Language, Springer-Verlag, 128--139
36. **Mouratidis, H. (2006)** Analysing Security Requirements of Information Systems using Tropos. In Proceedings 1st Annual Conference on Advances in Computing and Technology 55--64
37. **Nanda A., Feuerstein S. (2005)** Oracle PL/SQL for DBAs. O'Reilly Media, Inc.
38. **Nishida K. and Duvvuri S. (2009)** Row Level Security with BI Publisher Enterprise. An Oracle white paper, Oracle Communications, URL (last check 28.12.2011): http://www.oracle.com/technetwork/middleware/bi-publisher/overview/wp-oracle-bip-row-level-security-132091.pdf
39. **Oh S., Park S. (2001)** Enterprise Model as a Basis of Administration on Role-based Access Control. In Proceedings of the 3rd International Symposium on Cooperative Database Systems for advanced Applications (CODAS 2001), pp. 150-158
40. **OMG (2005)** Unified Modeling language: Superstructure, version 2.0, format/05-07-04
41. **Sandhu R. S., Coyne E. J. (1996)** Role-based Access Control Models, Computer, 38-47
42. **Sindre, G. (2007)** Mal-activity Diagrams for Capturing Attacks on Business Processes. In Proceedings of the Working Conference on Requirements Engineering: Foundation for Software Quality, Springer-Verlag Berlin Heidelberg 355--366
43. **Sindre G., Opdahl A. L. (2001)** Template for Misuse Case De- scription. In Proceedings of the International Workshop Re- quirements Engineering: Foundation for Software Quality (REFSQ 2001)
44. **Sindre G., Opdahl A. L., (2005)** Eliciting Security Requirements with Misuse Cases. Requirements Eng. 10 (1), Springer-Verlag.
45. **Staron, M. (2006)** 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 57--72
46. **Warner, J., Kleippe, A. (2003)** The Object Constraint Language, second edition, Getting Your Models Ready for MDA. Addison-Wesley
47. **Yu E. (1997)** Towards Modeling and Reasoning Support for Early-phase Requirements Engineering, Proc. RE'97, IEEE Computer Society, pp. 226

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 virews 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 sec.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
```

Comparing Quality of Security Models: A Case Study

B

Publication:

Raimundas Matulevičius and Henri Lakk and Marion Lepmets and Andreas Sisask, Comparing Quality of Security Models: a Case Study, *Proceedings of the ADBIS 2010 workshop MDASD 2010*, University of Novi Sad Press, Serbia, 2010

Comparing Quality of Security Models: A Case Study

Raimundas Matulevičius[1, 2], Marion Lepmets[1, 3], Henri Lakk[4], and Andreas Sisask[4]

[1] Software Technology and Application Competence Center,
Ülikooli 8, 51003 Tartu, Estonia
[2] Institute of Computer Science, University of Tartu,
J. Liivi 2, 50409 Tartu, Estonia
rma@ut.ee
[3] Institute of Cybernetics, Tallinn University of Technology,
Akadeemia 21, Tallinn, Estonia
marion.lepmets@ttu.ee
[4] Logica, Sobra 54, Tartu, Estonia
{henri.lakk,andreas.sisask}@logica.com

B

Abstract. System security is an important artefact. However security is typically considered only at an implementation stage nowadays in industry. This makes it difficult to communicate security solutions to the stakeholders earlier and raises the system development cost, especially if security implementation errors are detected. In this paper we compare the quality of two security models, which propose a solution to the industrial problem. One model is created using PL/SQL, a procedural extension language for SQL; another model is prepared with SecureUML, a model driven approach for security. We result in significantly better quality for the SecureUML security model: it contains higher semantic completeness and correctness, it is easier to modify and understand, and it facilitates a better communication of security solutions to the system stakeholders than the PL/SQL model.

Keywords: Model-driven security development, Modelling quality, PL/SQL, SecureUML

1 Introduction

Nowadays, computer software and systems play an important role in different areas of human life. They deal with different type of information including the one (e.g., bank, educational qualification, and health records) that must be secured from the unintended audience. Thus, ensuring system security is a necessity rather than an option. Security analysis should be performed throughout the whole system development cycle starting from the early stages (e.g., requirements engineering and system design) and leading to the late stages (e.g., implementation and testing). However this is not the case in practice [7] [20], where security is considered only when the system is about to be implemented (e.g., at implementation stage) or deployed (e.g., at installation stage). This is a serious limitation to the secure system development, since the early stages are the place where security requirements should be discovered and communicated among stakeholders, security trade-offs should be

considered, and security concerns should be clearly differentiated among different system aspects (e.g., data, functionality, and etc).

In this work we report a case study carried out at the Software Technology and Application Competence Centre in Estonia, where quality of two security models is compared following the semiotic quality framework [8] [9]. One security model is created using PL/SQL [18], a procedural programming language, another – SecureUML [1] [11], a language for the model-driven security development. Both models define a role-based access control [5] on the *data model* provided to us by our industrial partner. The following research question is considered:

Which security model – PL/SQL or SecureUML – *is of a better quality?*

Our study results in a high quality of the SecureUML security model, which is typically created at the requirements engineering and design stages of the systems development. The structure of the paper is as follows: in Section 2 we introduce the general RBAC model and the quality framework used to evaluate security models. In Section 3 we present our case study design. Section 4 presents the evaluation of the security models. In Section 5 we discuss the results, conclude our study, and present some future work.

2 Theory

The security models analysed in this paper present the security policy expressed through the role-based access control (RBAC) mechanism. In this section we briefly present the RBAC domain. Then we introduce and instantiate the framework used to assess the quality of the security models.

2.1 Role-based Access Control

The standard RBAC model is provided in [5]. Its basic concepts are illustrated in Fig. 1. The main elements of this model are *Users*, *Roles*, *Objects*, *Operations*, and *Permissions*. A *User* is typically defined as a human being or a software agent. A *Role* is a job function within the context of an organisation. Role refers to authority and responsibility conferred on the user assigned to this role. *Permissions* are approvals to perform one or more *Operations* on one or more protected *Objects*. An *Operation* is an executable sequence of actions that can be initiated by the system entities. An *Object* is a protected system resource (or a set of resources). Two major relationships in this model are *User assignment* and *Permission assignment*. *User assignment* relationship describes how users are assigned to their roles. *Permission assignment* relationship characterises the set of privileges assigned to a *Role*.

Two security models described in Section 3, define a security policy based on the RBAC domain. Thus, we will analyse model correspondence to the RBAC domain, when considering their quality in Section 4.

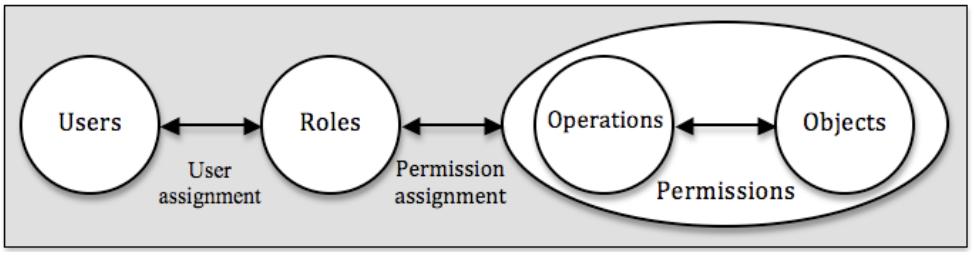

Fig. 1. Role-based access control model (adapted from [5])

2.2 Modelling Quality

Evaluations of a model quality [19] could be performed (*i*) using detailed qualitative properties or (*ii*) through general quality frameworks. A systematic survey of these approaches could be found in [17]. In this study we combine both approaches: firstly, we follow guidelines of the *semiotic quality* (SEQUAL) framework [8], [9] to select the quality types of interest. Secondly, we identify a set of qualitative properties that are used to compare two security models.

The SEQUAL framework (Fig. 2) is an extension of the Lindland *et al*, (1994) quality framework [10], which includes discussion on syntax, semantics and pragmatics. It adheres to a constructivistic world-view that recognises model creation as part of a dialog between participants whose knowledge changes as the process takes place. The framework distinguishes between quality goals and means to achieve these goals. *Physical quality* pursues two basic goals: externalisation, meaning that the explicit knowledge ***K*** of a participant has to be externalised in the model ***M*** by the use of a modelling language ***L***; and internalisability, meaning that the externalised model ***M*** can be made persistent and available, enabling the stakeholders to make sense of it. *Empirical quality* deals with error frequencies when reading or writing ***M***, as well as coding and ergonomics when using modelling tools. *Syntactic quality* is the correspondence between ***M*** and the language ***L*** in which ***M*** is written. *Semantic quality* examines the correspondence between ***M*** and the domain ***D***. *Pragmatic quality* assesses the correspondence between ***M*** and its social as well as its technical audiences' interpretations, respectively, ***I*** and ***T***. *Perceived semantic quality* is the correspondence between the participants' interpretation ***I*** of ***M*** and the participants' current explicit knowledge $\boldsymbol{K_S}$. *Social quality* seeks agreement among the participants' interpretations ***I***. Finally, *organisational quality* looks at how the modelling goals ***G*** are fulfilled by ***M***. In the second case the major quality types include physical, empirical, syntactic, semantic, pragmatic, social and organisational quality.

2.3 Quality Framework Application

Although SEQUAL provides fundamental principles to evaluate model quality, it remains abstract. We need to adapt it in order to evaluate two security models analysed in our case study. Although being influenced by the overall theoretical background of the SEQUAL framework, in our study we specifically focus only on three quality types, namely *semantics*, *pragmatics*, and *syntax*. Furthermore, based on

our experience of assessing the requirements engineering tools [13], development guidelines [6], goal modelling languages and models [14], we select a set of qualitative properties, that instantiates SEQUAL for the *security model* assessment.

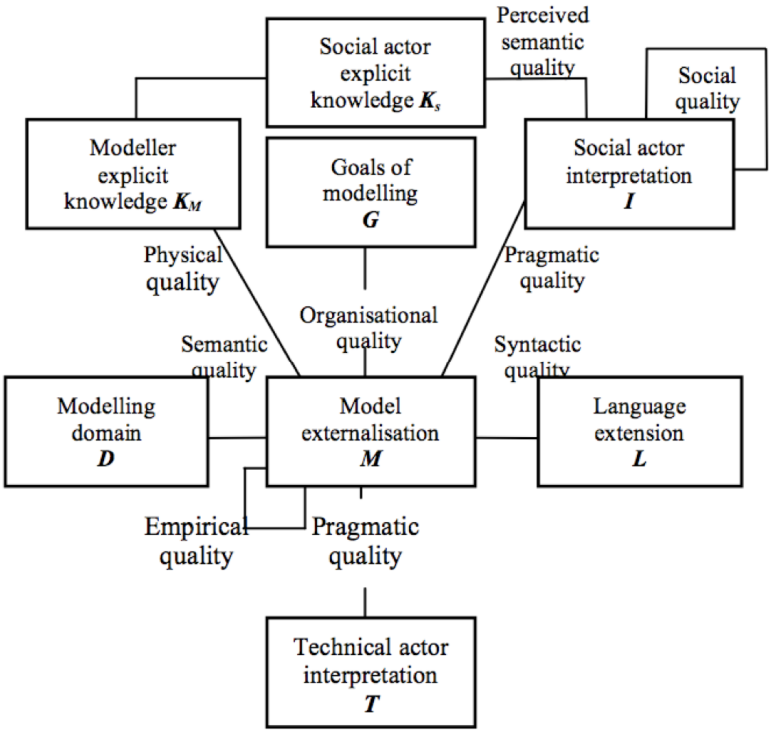

Fig. 2. The SEQUAL framework (adapted from [8][9])

Semantic quality is a correspondence between a model and its semantic domain. We define that the model should be:

- *Semantically complete*. It means that everything that the software is supposed to do is included in the model. With respect to the security domain, we say that the security model should include concepts corresponding to the RBAC domain.
- *Semantically correct*. It means that a model should represent something that is required to be developed. With respect to the security domain this qualitative property requires separation between data- and security-related concerns – only the security-related knowledge is required in the security model.
- *Traced.* It requires that the origin of the model and its content should be identifiable. The security model should clearly present the rationale why different security solutions are included in the model.
- *Achievable*. It determines that there exists at least one implementation/application that correctly implements the model.
- *Annotated*. It means that a reader is easily able to determine which elements are most likely to change. This is especially important in the security model because system security policy might be often changed.
- *Modifiable*. It means that the structure and the content are easy to change. When security policies change it should be easy to change the security concerns quickly in the model.

The last two qualitative properties are important when new system security policies are introduced. Knowing the place and being able to implement the new security concerns quickly might substantially reduce the overall system maintenance cost.

Syntactic quality is a correspondence between a model and a modelling language. The major goal of the syntactic quality is syntactic correctness. Thus, the model should be:

- *Syntactically valid.* It means that the grammatical expressions used to create a model should be a part of the modelling language;
- *Syntactically complete*. It means that all grammar constructs and their parts are present in the model.

To test the syntactic correctness of the security models we need to investigate the concrete syntax of the languages used to create these models.

Pragmatic quality is a correspondence between a model and an interpretation of social and technical audience. With respect to the social actors we say that the model should be:

- *Cross-referenced*. It means that different pieces of model content are linked together;
- *Organised*. It means that the model content should be arranged so that a reader could easily locate information and logical relationships among the related information;
- *Understandable*. It means that a reader is able to understand the model with minimum explanations.

The social audience of security model are typically security engineer, but it also includes the system analysts, software developers, stakeholders (actors who pay for the development of the secure system), and even direct users, who should also be involved in the security requirements definition process.

For the technical model interpretation we define that the model should be *executable,* meaning that there should exist technology capable of inputting the model and resulting in its implementation.

3 Research method

In this section we will introduce a case study carried out to compare two security models. We will define a case study design. Next, we will present our research subjects – the two security models.

3.1 Design

Our research method presented in Fig. 3, is pretty straightforward. Firstly, we formulated the research question (see Section 1). Then two researchers experienced in modelling quality analysis, system and security modelling, performed the investigation of two security models presented using PL/SQL and SecureUML

languages. The researchers applied the qualitative properties following the SEQUAL framework (Section 2.3), and recorded their observations on the model quality. The results were communicated to the model developers in order to verify the correctness. Finally, the results are summarised.

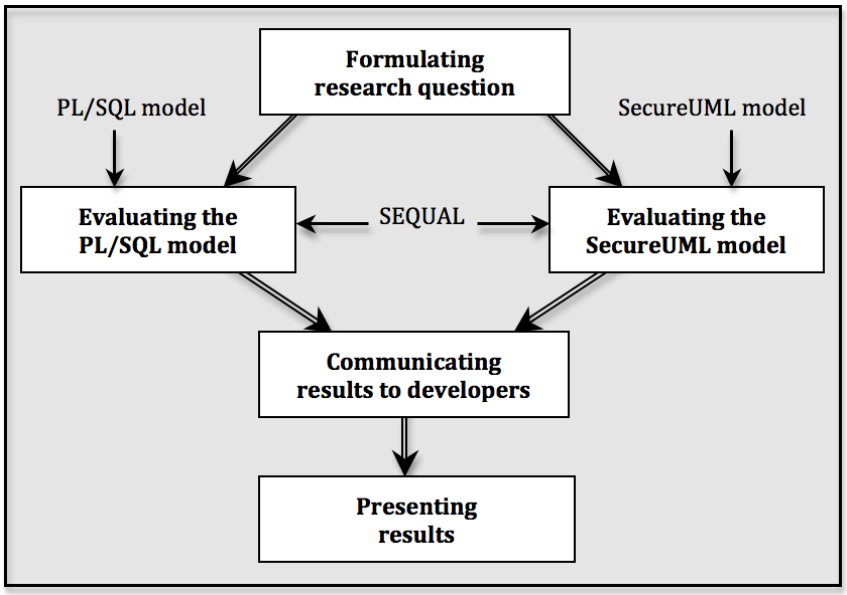

Fig. 3. A case study design

3.2 Research Subject

As mentioned above the research subject includes two security models; one created with PL/SQL, another – with SecureUML. Both models were prepared to solve the same problem. The actual data and security models used in this case study could not be presented here due to the privacy concerns of our industrial partner. But here we include an extract of a *meeting management subsystem* [4]. This example closely corresponds to the industry model used in the assessment. Our observations are the same for both cases.

In our example users are allowed adding information about new meetings and viewing information about all existing meetings. But one can delete or change meeting information if and only if he/she is an owner (e.g., meeting initiator) of this meeting. We will present the PL/SQL and in SecureUML models for this problem.

PL/SQL Security Model. The first security model is created using PL/SQL [18], which is an Oracle Corporation's procedural extension language for SQL and the Oracle relational database. The model was prepared using the *EditPlus*[1] tool. In order to receive a running application one needs to compile the PL/SQL source code.

In the industrial case the security description included two text-based (PL/SQL code) pages. In Fig. 4 we illustrate a procedure that describes a permission defined on the meeting. Here we see that if a certain condition (e.g., a user is a meeting *owner*

[1] http://www.editplus.com/

and the meeting end date has not yet passed) holds, it is possible to edit meeting attributes (e.g., *start*, *end*, *location*, and *owner*); otherwise editing is not allowed.

```
PROCEDURE meeting_permissions
IS
BEGIN
   IF :meeting.owner = sec.get_username() AND :meeting.end > SYSDATE
   THEN
      DO.item_edit_yes('meeting.start');
      DO.item_edit_yes('meeting.end');
      DO.item_edit_yes('meeting.location');
      DO.item_edit_yes('meeting.owner');
      DO.item_enable('meeting.delete_meeting');
   ELSE
      DO.item_edit_no('meeting.start');
      DO.item_edit_no('meeting.end');
      DO.item_edit_no('meeting.location');
      DO.item_edit_no('meeting.owner');
      DO.item_disable('meeting.delete_meeting');
   END IF;
   DO.item_enable('meeting.new_meeting');
END;
```

Fig. 4. Excerpt of the PL/SQL security model

SecureUML Security Model. The second security model is created in SecureUML [1], [11], which is a model-driven security approach that follows RBAC guidelines. The model was prepared using *MagicDraw*[2]. The overall SecureUML model (the industrial case) included around eight permissions on the secured resource for each security action (e.g., *update*, *select*, *delete*, and *insert* of information). In Fig. 5 we present an excerpt related to the meeting management subsystem.

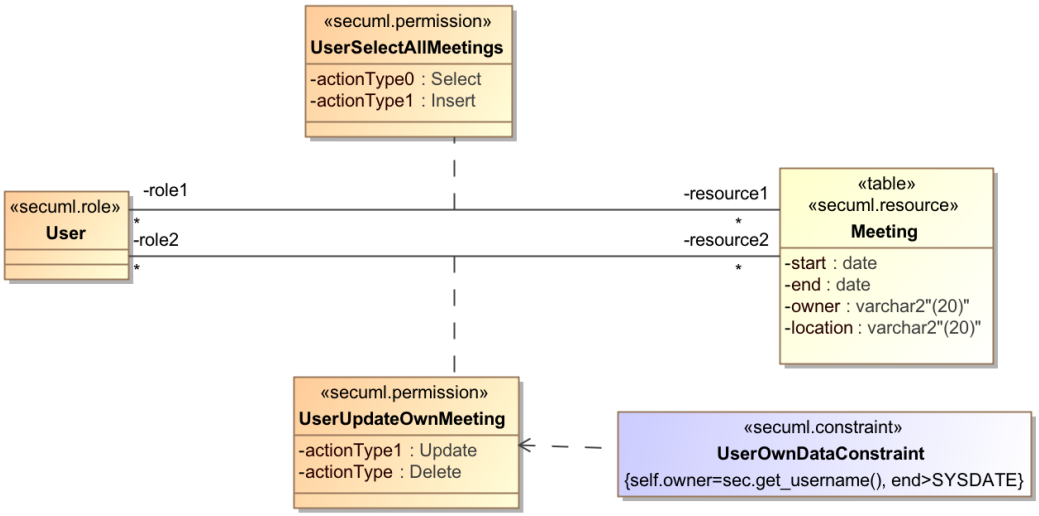

Fig. 5. Excerpt of the SecureUML security model

Here two security permissions (e.g., *UserSelectAllMeetings* and *UserUpdateOwnMeeting*) are defined for the role *User* over the resource *Meeting*.

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

Similarly like in the PL/SQL model, an authorisation constraint *UserOwnDataConstraint* defines that only an *owner* is allowed to *update* or *delete* meeting information if the meeting date has not yet passed.

In order to receive an executable application, the SecureUML model is automatically transformed to the PL/SQL code. For example, Fig. 6 illustrates the example of a new *owner* assignment before the *update* action is executed. The assignment is performed if the security condition (defined under *UserOwnDataConstraint* in Fig. 5) holds. The transformed PL/SQL code is compiled to a running application.

```
   IF util.null_eq (:NEW.owner, :OLD.owner) != 'Y' -- owner updated
   THEN
      IF 1!=1
         OR sec.is_role('User') = 'Y' AND self.owner = sec.get_username() AND
self.end > SYSDATE -- Permission from UserUpdateOwnMeetings
      THEN
         self.owner := :NEW.owner;
      ELSE
         RAISE ex_denied;
      END IF;
   END IF;
```

Fig. 6. Excerpt of the SecureUML model transformation to PL/SQL code

In the case study we have selected to analyse the model created using SecureUML, but not its PL/SQL transformation. The reason is that we intend to analyse the model, which is editable by system engineers directly.

4 Evaluating Security Models

This section presents our analysis of two security models introduced in Section 3.2. We address syntactic, semantic and pragmatic quality types through the qualitative characteristics, presented in Section 2.3. But first we discuss threats to the result validity.

4.1 Threats to Validity

In our case study *only* two evaluators assessed the security models according to their knowledge and experience. This certainly raises the level of subjectivity and influences the *internal validity* of the case study. To mitigate this threat the evaluation results were communicated to the model developers.

In our case the SEQUAL framework was instantiated with a *certain* set of qualitative properties. This certainly affects the *conclusion validity*, because if any other qualitative properties were applied, it might result in different outcome. But this threat is rather limited because these qualitative properties are theoretically sound and the selection is based on the previous experience as presented in Section 2.3.

We applied the ordinal scale (e.g., *high*, *partial*, and *low*) to assess the qualitative properties of the models. This influences *construct validity* because different readers

might interpret the assigned property values differently. On another hand we could use the interval scales for each qualitative property (also reducing some subjectivity). For example, semantic completeness could be expressed as a ratio between the number of RBAC concepts that are possible to present using the modelling language, and the total number of RBAC concepts (see Section 2.1). Similarly, annotation could be addressed through counting annotated elements in the model. However, the construction of the interval scale was not the purpose of our case study – we rather were concerned about the feasibility to assess the security model quality and to learn about the quality of PL/SQL or SecureUML security models in general.

In this case study we analysed *only* two different security models and these models were quite limited in their size. This might influence the *external validity* by a fact, that different results might be received if some other security models (created either using PL/SQL, SecureUML, or any other modelling language) would be analysed. However our research subject is a solution to an industry problem; thus we believe that our analysis is generalisable in similar situations.

Finally we try to avoid a use of single type of measuring that might affect the *construct validity*. As shown in the case study design (Fig. 3) the evaluation of the security models is followed with the communication of the received results to the models developers. This certainly reduces a risk of the mono-interpretation.

4.2 Quality of the PL/SQL Model

Semantic quality. *Semantic completeness* is assessed through a model correspondence to the RBAC domain (see Section 2.1). The PL/SQL model focuses primarily on the presentation of the security *permissions*, which are defined as the *attributes* of *objects* that need to be secured. However it does not explicitly define on which *operations* the security permissions are placed. In addition the PL/SQL model does not express explicitly *objects* themselves, *users*, and their *roles*. This knowledge is defined in the data model and not in the security model. This results in partial *semantic completeness*.

The *semantic correctness* of the PL/SQL model is low, because it does not separate the data and programmable concerns from the security concerns. For example in the PL/SQL model we can observe assignment of different programmable variables and definition of the user interface components (e.g., *DO.item_enable('meeting.new_meeting')* is enabling the item of the user interface).

The PL/SQL model is not *traced* – this means that origin and rationale for the security decisions are not provided in the model. We were not able to check *achievability* property of the PL/SQL model. The reason is that it was not possible to get the security requirements in order to confront its application correctness. The PL/SQL model is not *annotated*, thus it is difficult to determine which elements are most likely to change. The model is also difficult to *modify* because the same security concern is addressed in several places of the model.

Syntactic quality. The PL/SQL model is of high *syntactic validity* and *completeness*, because the model is created using the PL/SQL language, a programmable language.

Syntactically this model is correct because otherwise it would not be possible to compile it to the application.

Pragmatic quality. We found the PL/SQL model of low *understandability*. In fact we asked model developers to explain us different security solutions presented in this model. The *organisation* of the model is also low, because there are no means that would support finding security information or defining relationships between related security solutions. The PL/SQL model is presented as a plain-text source code, thus it does not contain any hyperlinks that would *cross-reference* related security concerns (but also see Section 5.1). Finally, the *executability* of the PL/SQL model is high. It is possible to compile this model through the Oracle database management system resulting in a running application.

4.3 Quality of the SecureUML Model

Semantic quality. SecureUML is developed to design the RBAC-based solutions [5]. This means that SecureUML fully corresponds to the semantic domain, thus resulting in high *semantic completeness*. We also identify high *semantic correctness*, because only security solutions are presented in the SecureUML model.

In the SecureUML model we did not observe any rationale for security decisions, thus it results in a low *traced* property. Like in the PL/SQL model, we were not able to check the *achievability* of the SecureUML model because the security requirements are not available. On another hand the *achievability* of the SecureUML model is high with respect to its implementation. This model is automatically transformed to the PL/SQL code thus resulting in the direct correlation between design and implementation.

The Secure UML model is partially *annotated*. This annotation is achieved through SecureUML stereotypes (e.g., <<secuml.permission>>, <<secuml.role>>, etc.) and class names given to the *permissions* (e.g., *UserSelectAllMeetings* and *UserUpdateOwnMeeting*) and the *authorisation constraints* (e.g., *UserOwnDataConstraint*). These class names are not directly used in the transformation of the model to code, but they provide additional information to the model reader. They also identify the places in the model where security policy is most likely to be changed.

The SecureUML model is *modifiable.* The model implies a certain presentation pattern – *Role-Permission-Resource*, which facilitates the changing of the model.

Syntactic quality. In the current model of the SecureUML we can identify a case of *syntactic invalidity*. For instance the SecureUML documentation [1] [11] identify that *authorisation constraints* need to be written in OCL. However in this model the SQL-based authorisation constraints are used (e.g., see class *UserOwnDataConstraint* constraint *{owner=sec.get_username(), end>SYSDATE}*). On another hand the model is *syntactically complete* – it includes only UML extensions and their relationships proposed by the authors of SecureUML [1] [11].

Pragmatic quality. The Secure UML model is well *understood* by those readers familiar with the UML modelling notation. This also opens the way to communicate this model to a larger audience, including various project stakeholders, potential direct users of the system, systems analysers, and developers. Our personal experience is that this model is quite intuitive and did not require big effort to understand it.

The SecureUML model consists of several diagrams. It is also supported by a modelling tool, which simplifies managing the model itself. For example the tool provides the content table where the model diagrams and all model elements are listed. In addition it is possible to prepare a navigation map diagram (see Fig. 7) that assembles the logical relationship between different diagrams, thus keeping this model both *organised* and *cross-referenced.*

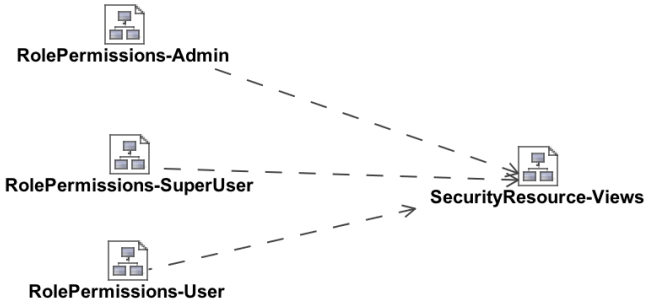

Fig. 7. SecureUML content diagram

The SecureUML model is *executable*: there exists a number of the transformation rules defined using the Velocity[3] language (interpretable by MagicDraw tool) that transform the model to PL/SQL code, which could be executed through Oracle database management system.

4.4 Comparison

Table 1 summarises the assessment results for both PL/SQL and SecureUML models. We see that three qualitative properties (i.e., model properties of being *traced*, *syntactically complete*, and *executable*) score equally. One property – *syntactic validity* – is found to be better in the PL/SQL model. The eight remaining properties (i.e., *semantic completeness*, *semantic correctness*, *achievability*, *annotation*, *modifiability*, *understandability*, *organisation*, and *cross-referencing*) are evaluated to be higher in the SecureUML model.

5 Discussion and Conclusion

Our case study results in a higher quality for the SecureUML security model. In this section we present a discussion on these results. Firstly, we communicate our finding

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

with the developers of the security models. Next, we situate our findings into the state of the art. Finally, we present the future work.

Table 1. Quality of the security models. Quality is evaluated as *High*, *Partial*, and *Low*. *NA* means – assessment is not available due to the lack of data

Quality types and qualitative property		PL/SQL model		SecureUML model	
		Score	**Comments**	**Score**	**Comments**
Semantic quality	**Semantically complete**	Partial	It focuses on security permissions. Presentation of other RBAC constraints is limited.	High	SecureUML is based on the RBAC model. Semantically it is possible to present all RBAC concepts.
	Semantically correct	Low	Data, programming and security concerns are intermixed.	High	He security and data modelling concerns are separated.
	Traced	Low	Rationale is not given.	Low	Rationale is not given.
	Achievable	NA	Security requirements were not obtained.	NA // High	Security requirements were not obtained // No errors were observed in the transformed PL/SQL code
	Annotated	Low	Nothing observed.	Partial	SecureUML stereotypes, Class names given to *permissions* and *authorisation constraints*
	Modifiable	Low	Changing one security concern requires several changes in the model.	High	Model contains a structured way to express and change security concerns.
Syntactic quality	**Syntactically valid**	High	The model is compiled to the application.	Partial	SQL (and not OCL) is used for *authorisation constraints*
	Syntactically complete	High	The model is compiled to the application.	High	The model includes the UML extensions for the SecureUML.
Pragmatic quality	**Understandable**	Low	The model had to be explained by developers.	High	The model is intuitive and could be used for the communication purpose among various stakeholders.
	Organised	Low	Search for the related security concerns is not supported.	High	Content table supported by a tool.
	Crossed-referenced	Partial	Plain text does not contain any hyperlinks. Procedure definition might be seen as textual cross-references.	High	A diagram – navigation map, containing cross-referenced links between different diagrams.
	Executable	High	Oracle database management system.	High	Transformation templates supporting model translation to PL/SQL code.

5.1 Communicating Results to Developers

A result review was performed together with the developers of the security models. Firstly, the developers noted that the overall quality of both models could be improved if to take into account these evaluation results. For example the *traceability*, *annotation*, and *understandability* of the PL/SQL model could be easily improved using code comments. However, the developers acknowledged that this is not the case in the common practice or the code comments, even if they are present, are not sufficient. On another hand to improve *syntactic validity* of the SecureUML model we could write the authorisation constraints in OCL instead of SQL.

Secondly, developers provided few remarks regarding some qualitative properties. For instance, *semantic completeness* could be improved by presenting concrete instances in the models. This means hard coding in the PL/SQL model and object presentation in the SecureUML model. However, this neglects the principle of generosity in modelling.

On one hand, a tool used to make the PL/SQL model, does not support hyper-linking. Although there exist several PL/SQL editing tools (e.g., *Oracle SQLDeveloper* or *Quest Software Toad for Oracle*, actually used by our industrial partner) that supports cross-references between various model elements, but these were not used in this case study. On another hand, developers also indicated that PL/SQL grammar principles, the ones, which allow expressing procedures (e.g., `PROCEDURE meeting_permissions` in Fig. 4) and referring to them from the main code, could also be seen as textual *cross-referencing*. Thus, we estimate this qualitative property as partial for the PL/SQL model.

5.2 Related Work

We found none studies that would compare quality of (security) models prepared using different modelling approaches. However, the literature reports on a number of case studies [12] [15] [16] analyzing different characteristics of the *model-driven development*. Mostly these studies focus on the benefits and on the infrastructure needed for the model-driven development. Similarly to [2] [12] [15] we observe that SecureUML model facilitates automatic code generation – the SecureUML security model is *executable* through its generation to PL/SQL code (see Section 3.2). We also argue that the security models should be prepared with the high-quality modelling language [16], ensuring the model *semantic completeness*, and tools [12], guaranteeing 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 [15].

We identified only one case study performed by Clavel *et al* [2], reporting on the SecureUML application in practice. Here authors observe that although security models are integrated with the data models, the security design remains independent, reusable and evolvable. In our work we also observe that *semantic correctness* of SecureUML model is high, because only security aspect are described in this model. We also observe that SecureUML model is *modifiable*, which means the first step towards model evolvability. Like in [2] we identify that the SecureUML model is

understandable at least to readers who are familiar with UML. This might ease communication of requirements and design solutions to project stakeholders [12]. Finally, Clavel *et al* [2] identify that SecureUML is expressive enough to model the RBAC policy defined in the requirements document. However, we were not able to analyse *achievability* property because our industrial partner did not provide us security requirements documents.

5.3 Future Work

Our future work includes a definition of a framework that would facilitate the *adoption* of the model-driven security approach in practice [21]. For instance an organisation should have modelling *tools* (e.g., MagicDraw and Velocity interpreter) that would support developing and applying security model transformation rules. Also the organisation should adopt a mature *security modelling method* that should include the early security requirements discovery, security quality assurance, and overall project planning.

For the successful adoption, organisation's working *processes* should also be compatible with model-driven security. Our future work includes performing another case study where we would compare *quality of processes* to develop security models using PL/SQL and SecureUML.

The organisation should have an *expertise for security language engineering*. This includes knowledge about how to combine the existing software tools and security modelling approaches together. For instance we need to define guidelines and transformation rules for the OCL-based authorisation constraints. This would also improve the *syntactic validity* of the SecureUML model.

Finally an organisation should follow a *goal-driven process* for defining goals to introduce security model-driven development. Examples of this paper focuses on the security policy for the data model. Our next goal is to develop transformation rules that would facilitate implementation of the security concerns at the system application and presentation levels.

Acknowledgments. This research is funded by Logica and the European Regional Development Funds through the Estonian Competence Centre Programme and through the Estonian Center of Excellence in Computer Science, EXCS.

References

1. Basin, D., Doser, J., Lodderstedt, T.: Model Driven Security: from UML Models to Access Control Infrastructure. ACM Transactions on Software Engineering and Methodology (TOSEM), 15 (1), 39--91 (2006)
2. Clavel M., Silva V., Braga C., Egea M.: Model-driven Security in Practice: an Industrial Experience, In: Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications, Springer-Verlag, pp. 326--337, (2008)
3. Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Srimani, P., Ta, A., Theofanos, M.: Identifying and Measuring

Quality in a Software Requirements Specification. In: Proceedings of the 1st International Software Metrics Symposium, pp. 141--152 (1993)
4. Feather, M.S., Fickas, S., Finkelstein, A., van Lamsweerde A.: Requirements and Specification Exemplars. Automated Software Engineering, 4: 419--438 (1997)
5. Ferraiolo D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST Standard for Role-based Access Bontrol. ACM Transactions on Information and System Security (TISSEC), 4(3), 224--274 (2001)
6. Hakkarainen S., Matulevičius R., Strašunskas D., Su X. and Sindre G.: A Step Towards Context Insensitive Quality Control for Ontology Building Methodologies. In Proceedings of the CAiSE 2004 Open INTEROP-EMOI Workshop, pp. 205--216, (2004)
7. Jurjens. J.: Secure Systems Development with UML. Springer-Verlag Berlin Heidelberg, (2005)
8. Krogstie, J.: A Semiotic Approach to Quality in Requirements Specifications. In: Proc. IFIP 8.1 working Conf. on Organisational Semiotics, pp. 231--249 (2001)
9. Krogstie, J.: Using a Semiotic Framework to Evaluate UML for the Development for Models of High Quality. In: Siau, K., Halpin, T. (eds.) Unified Modelling Language: System Analysis, Design and Development Issues, IDEA Group Publishing, pp. 89--106 (1998)
10. Lindland O. I., Sindre G., Sølvberg A.: Understanding Quality in Conceptual Modelling. IEEE Software, 11(2), pp. 42--49 (1994).
11. Lodderstedt, T., Basin, D., Doser, J.: SecureUML: A UML-based Modeling Language for Model-driven Security. In: Proceedings of the 5th International Conference on The Unified Modeling Language, LNCS, vol. 2460, pp. 426--441. Springer-Verlag (2002)
12. MacDonald A., Russell D., Atchison B.: Model-driven Development within a Legacy System: An Industry Experience Report. In: Proceedings of the 2005 Australian Software Engineering Conference (ASWEC'05). IEEE Computer Science (2005)
13. Matulevičius, R.: Process Support for Requirements Engineering: A Requirements Engineering Tool Evaluation Approach. PhD theses. Norwegian University of Science and Technology (2005)
14. Matulevičius R., Heymans P.: Comparison of Goal Languages: an Experiment. In Proceedings of the Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2007), Trondheim, Norway, Springer-Verlag, pp 18--32 (2007)
15. The Middleware Company: Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach: Productivity Analysis, MDA Productivity case study (2003)
16. de Miguel M., Jourdan J., Salicki S.: Practical Experiences in the Application of MDA. In: Proceedings of the 5th International Conference on The Unified Modeling Language, Springer-Verlag, pp. 128--139 (2002)
17. Moody D.L.: Theoretical and Practical Issues in Evaluating the Quality of Conceptual Models: Current State and Future Directions. Data and Knowledge Engineering 55 (3): 243--276 (2005)
18. Morris-Murthy L.: Oracle9i: SQL, with an Introduction to PL/SQL. Course Technology, (2003)
19. Piattini, M., Genero, M., Poels, G., Nelson J.: Towards a Framework for Conceptual Modelling Quality. In: Genero, M., Piattini, M., Calero, C. (eds.) Metrics for Software Conceptual Models, pp. 1--18. Imperial College Press, London (2005)
20. Sindre, G., Opdahl, A.L.: Eliciting Security Requirements with Misuse Cases. Requirements Engineering Journal 10(1) pp. 34--44 (2005)
21. Staron M.: Adopting Model Driven Software Development in Industry – A Case Study at Two Companies. In: 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2006), pp. 57--72. Springer-Verlag (2006)

An Approach to Assess and Compare Quality of Security Models

C

Publication:

Raimundas Matulevicius, Henri Lakk, and Marion Lepmets, An Approach to Assess and Compare Quality of Security Models. *Computer Science and Information Systems, ComSIS Consortium*, 2011.

DOI:10.2298/CSIS101231014M

An Approach to Assess and Compare Quality of Security Models

Raimundas Matulevičius[1], Henri Lakk[1], and Marion Lepmets[2]

[1] Institute of Computer Science, University of Tartu,
J. Liivi 2, 50409 Tartu, Estonia
rma@ut.ee, henri.lakk@gmail.com
[2] Centre for Public Research Henri Tudor – SSI
29 Av. John F. Kennedy, L-1855 Luxembourg,
Marion.Lepmets@tudor.lu

Abstract. System security is an important artefact. However security is typically considered only at implementation stage nowadays in industry. This makes it difficult to communicate security solutions to the stakeholders earlier and raises the system development cost, especially if security implementation errors are detected. On the one hand practitioners might not be aware of the approaches that help represent security concerns at the early system development stages. On the other hand a part of the problem might be that there exists only limited support to compare different security development languages and especially their resulting security models. In this paper we propose a systematic approach to assess quality of the security models. To illustrate validity of our proposal we investigate three security models, which present a solution to an industrial problem. One model is created using PL/SQL, a procedural extension language for SQL; another two models are prepared with SecureUML and UMLsec, both characterised as approaches for model-driven security. The study results in a higher quality for the later security models. These contain higher semantic completeness and correctness, they are easier to modify, understand, and facilitate a better communication of security solutions to the system stakeholders than the PL/SQL model. We conclude our paper with a discussion on the requirements needed to adapt the model-driven security approaches to the industrial security analysis.

Keywords: model-driven security development, modelling quality, PL/SQL, secureUML, UMLsec.

1. Introduction

Nowadays, computer software and systems play an important role in different areas of everyday life. They deal with different type of information including the one (e.g., bank, educational qualification, and health records) that must be secured from the unintended audience. Thus, ensuring system security is a necessity rather than an option. Security analysis should be performed

throughout the whole system development cycle starting from the early stages (e.g., requirements engineering and system design) and leading to the late stages (e.g., implementation and testing). However this is not the case in practice [13], [32] where security is considered only when the system is about to be implemented (e.g., at implementation stage) or deployed (e.g., at installation stage). This is a serious limitation to the secure system development, since it is the early stages where security requirements should be discovered and communicated among stakeholders, security trade-offs should be considered, and security concerns should be clearly differentiated among different system aspects (e.g., data, functionality, and etc).

One possible suggestion to solve the above problem is an approach called model driven architecture (MDA). MDA provides a solution for the system development process based on models [5] that are the simplified representations of reality. Although MDA is certainly useful for the general-purpose system and software development [14], [20], [33], [34], the current state of the art gives little evidence (we identified only one study – [3]) on how model driven security (MDS) could help developers to improve the security definition and implementation process.

A part of the problem could be a lack of the systematic support to assess the security development languages both at the systems modelling and system implementation stages. In this paper we have proposed a systematic approach to evaluate quality the security models following the instantiation of the Semiotic Quality (SEQUAL) framework [15] [16]. To validate our proposal we have performed a case study (carried on at the Software Technology and Application Centre in Estonia), where we compare quality of the security model prepared using PL/SQL [9] (a procedural programming language), and quality of the security model prepared using MDS approaches, namely SecureUML [2], [19] and UMLsec [11]. All the security models define a role-based access control [8] on the *data model* provided to us by our industrial partner. Our case study results in a higher quality for the security models, created at the requirements engineering and design stages of the systems development. However we also highlight a set of requirements that are necessary to fulfil in order the MDS approaches were applicable in practice.

The structure of the remaining paper is as follows: in Section 2 we introduce the background of our research. We present the general RBAC model, the quality framework, and the approaches that help express system security concerns. In Section 3 we introduce an approach to assess quality of the security models. Next in Section 4 we illustrate the application of our proposal to evaluate quality of three languages, namely PL/SQL, SecureUML and UMLsec. Hence, we list our observations regarding model semantic, syntactic and pragmatic quality types. Finally, in Section 5 we discuss the results against the related work, and we also conclude our study.

2. Background

In this section we provide the background for our study. Firstly, we discuss the principles of the role-based access control. Secondly, we survey an evaluation framework that helps to assess model quality. Finally, we discuss development languages to represent system security.

2.1. Role-based Access Control

In this work we adapt the core role-based access control (RBAC) model [8]. This model defines a minimum set of concepts and relationships in order to define a role-based access control system. 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*.

The basic concepts of the RBAC model are illustrated in Fig. 1. The main elements of this model are *Users*, *Roles*, *Objects*, *Operations*, and *Permissions*. A *User* is typically defined as a human being or a software agent. A *Role* is a job function within the context of an organisation. Role refers to authority and responsibility conferred on the user assigned to this role. *Permissions* are approvals to perform one or more *Operations* on one or more protected *Objects*. An *Operation* is an executable sequence of actions that can be initiated by the system entities. An *Object* is a protected system resource (or a set of resources). Two major relationships in this model are *User assignment* and *Permission assignment*. *User assignment* relationship describes how users are assigned to their roles. *Permission assignment* relationship characterises the set of privileges assigned to a *Role*.

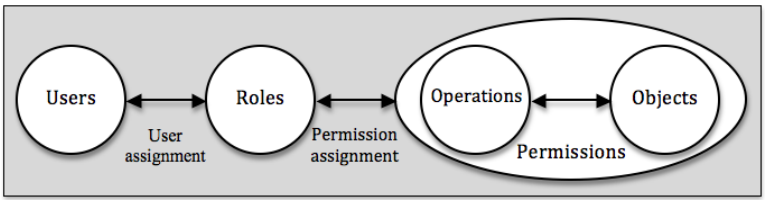

Fig. 1. Role-based Access Control Model (adapted form [8])

In Section 3 we propose an assessment of the quality for security models. There, the RBAC model suggests the criteria that help to judge about the model semantic properties as we illustrate in Section 4.

2.2. Modelling Quality

Evaluations of a model quality [30] could be performed (*i*) using detailed qualitative properties or (*ii*) through general quality frameworks. A systematic survey of these approaches could be found in [28]. In this study we combine both approaches: firstly, we follow guidelines of the *semiotic quality* (SEQUAL) framework [15], [16] to select the quality types of interest. Secondly, we identify a set of qualitative properties that are used to compare two security models.

The SEQUAL framework (Fig. 2) is an extension of the Lindland *et al*, (1994) quality framework [18], which includes discussion on syntax, semantics and pragmatics. It adheres to a constructivistic world-view that recognises model creation as part of a dialog between the participants whose knowledge changes as the process takes place. The framework distinguishes between quality goals and means to achieve these goals. *Physical quality* pursues two basic goals: externalisation, meaning that the explicit knowledge ***K*** of a participant has to be externalised in the model ***M*** by the use of a modelling language ***L***; and internalisability, meaning that the externalised model ***M*** can be made persistent and available, enabling the stakeholders to make sense of it. *Empirical quality* deals with error frequencies when reading or writing ***M***, as well as coding and ergonomics when using modelling tools. *Syntactic quality* is the correspondence between ***M*** and the language ***L*** in which ***M*** is written. *Semantic quality* examines the correspondence between ***M*** and the domain ***D***. *Pragmatic quality* assesses the correspondence between ***M*** and its social as well as its technical audiences' interpretations, respectively, ***I*** and ***T***. *Perceived semantic quality* is the correspondence between the participants' interpretation ***I*** of ***M*** and the participants' current explicit knowledge $\boldsymbol{K_S}$. *Social quality* seeks agreement among the participants' interpretations ***I***. Finally, *organisational quality* looks at how the modelling goals ***G*** are fulfilled by ***M***. In the second case the major quality types include physical, empirical, syntactic, semantic, pragmatic, social and organisational quality.

2.3. System Security

In order to define the system security policy in a systematic way it is important to understand the need for security within an organisation. One of the possible ways is to apply the security risk management process [26]. This process begins with the identification of the secure assets and the determination of the security objectives (in terms of *confidentiality*, *integrity*, and *availability*). During the next step security risks and their harm to the secured assets and their security objectives, are identified. Once the risk assessment is performed, risk treatment decisions (e.g., risk avoidance, risk reduction, risk transfer or risk retention) are taken. Following these decisions, the developers formulate the security requirements in order to mitigate the identified risks. Security requirements are, finally implemented into the security controls.

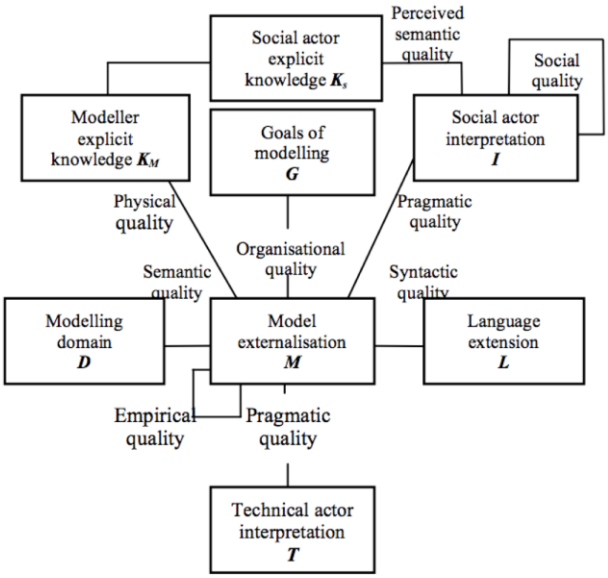

Fig. 2. The SEQUAL framework (adapted from [15], [16])

In order to support security modelling various research groups have proposed a variety of different approaches. For instance abuse frames [17] suggest means to consider security during early requirements engineering stage. Secure *i** [6] addresses security trade-offs. KAOS' extension to security [35] was augmented with anti-goal models designed to elicit attackers' rationales. Tropos has been extended with the notions of ownership, permission and trust [10]. Another version of Secure Tropos [29] defines security through the security constraints. Abuse cases [27], misuse cases [32] and mal-activity diagrams [31] are the extensions for the modelling languages from the UML family. Another UML extension (through the stereotypes, tagged values and constraints) towards security is UMLsec [13]. This language is, basically, used to address the security concerns during the system design stage. Although the majority of those approaches contribute to a proper definition of the security requirements, but they discuss little on how these security requirements should be implemented into the security controls.

Furthermore there is little support to assess these languages before their actual application to solve problems of system and software development. Thus, in this paper we propose a systematic approach, which could guide evaluation of the security languages through the hands-on testing. To illustrate application of the approach we have executed a case study where we have selected three languages – PL/SQL [9], SecureUML [2], [19], UMLsec [13]. We have investigated how these languages could contribute to the implementation of the security controls. More specifically we use these three approaches to define a role based access control (RBAC) policy for the data that needs to be secured.

3. An Assessment of Quality for Security Models

In this paper we introduce a systematic and hands-on-based approach to assess and compare quality of the security models. Our proposal consists of six steps as illustrated in Fig. 3. During the first step one needs to define the evaluation goal. With respect to the security models, the assessment goal could be understanding of the nature of the security needs, learning about the scope of the security models, learning about the quality of the security models, comparing different security models according to the quality criteria identified in the second step and similar.

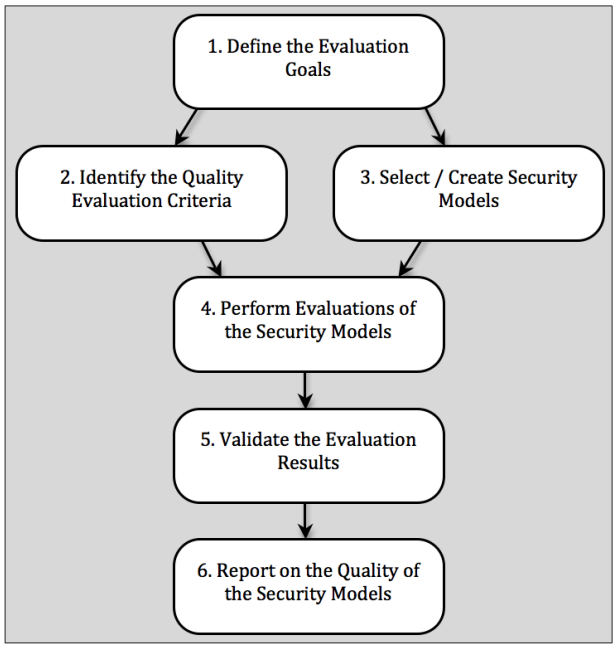

Fig. 3. An Assessment of Quality for Security Models

The second and the third steps of our proposal could be executed in parallel. The second step is identification of the quality evaluation criteria. Although, as illustrated in Section 2.2, the SEQUAL framework provides fundamental principles to evaluate model quality, firstly, it remains abstract, and, secondly, it is dedicated to the models of the general purpose, but not to the security models. As we show in Section 4.2, we select a set of qualitative properties that instantiates SEQUAL for the *security model* assessment based on the literature [4], [15] and on our experience of assessing the requirements engineering tools [21], development guidelines [11], goal modelling languages and models [24].

As discussed in Section 2.3, the security concerns could be represented using different languages. Thus, depending on the goal defined in the first

step, one needs to select or to create security models, which quality will be executed assessed in the subsequent steps.

The fourth step is about performance of the evaluation of the selected/created (in step 3) security models. This includes the investigation of the models and assignment of the subjective and objective values to the predefined (in step 2) model measures.

Expressing security quality is not an easy task. Thus we introduce the fifth step where evaluators have to validate the quality evaluation results. This typically means consultation of the received measures to the experts or to the model developers (see for instance Section 4.5.2). The final step of the security model assessment is the summary and report on the evaluation results.

In Section 4 we are reporting on a case study where we use our proposal to assess quality of three security models, created using PL/SQL [9], SecureUML [2] [19] and UMLsec [13].

4. A Case Study

Two researchers have followed the steps of the assessment of the quality for security models. They have defined the evaluation goals, identified the quality evaluation criteria and created the security models for evaluation. The model assessment results were communicated to the model developers in order to validate their correctness. The overall application of the method is illustrated in the following subsections.

4.1. Defining the Evaluation Goals

The goal of this case study is twofold:

- Firstly, we are interested in learning about the quality of the security models created using different languages. More specifically we will compare the models created at the software system design stage and software system implementation stage. In both cases our model will be defining the role-based access control polity for the system data.
- Secondly, we are interested in performance and feasibility of the method introduced in Section 3. Through the case study we will record our observations on the method application.

4.2. Identifying the Quality Evaluation Criteria

Although being influenced by the overall theoretical background of the SEQUAL framework, in our study we specifically focus only on three quality types, namely *semantics*, *pragmatics*, and *syntax*. Hence we will introduce a

set of measures in order to understand the quality of the security models. In fact in [25] we have already defined a set of subjective measures that helped us to address the model quality by its relative level (there we applied the ordinal scale consisting of *Low*, *Partial*, and *High* values). In this work we extend the quality model by introducing measures that allow developers to estimate quality quantitatively. The instantiation of the SEQUAL framework for the security model is illustrated in Fig. 4 and presented below.

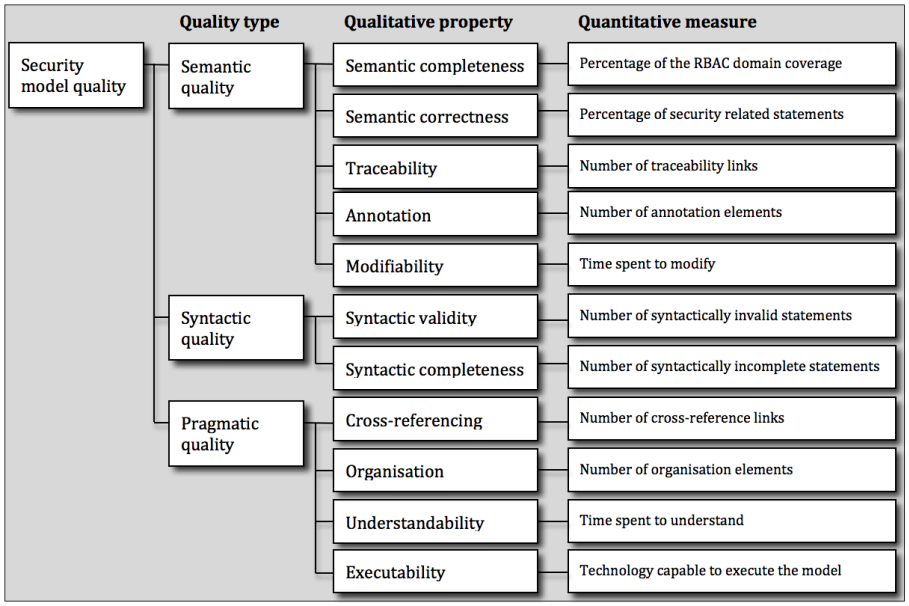

Fig. 4. Instantiation of the SEQUAL framework

Semantic quality is a correspondence between a model and its semantic domain. We assess semantic quality through the following qualitative properties and their measures:

Semantic completeness. It means that everything that the software is supposed to do is included in the model. With respect to the security domain, we say that the security model should include concepts corresponding to the RBAC domain, which is presented in Section 2.1. The *Percentage of the RBAC domain coverage* is calculated as a division between the number of RBAC concepts presented in the model and the number of RBAC concepts.

Semantic correctness. It means that a model should represent something that is required to be developed. With respect to the security domain this qualitative property requires separation between data- and security-related concerns – only the security-related knowledge is required in the security model. *Percentage of security related statements* describe the degree of security statements with respect to the overall model is.

Traceability. It requires that the origin of the model and its content should be identifiable. The security model should clearly present the rationale why different security solutions are included in the model. We define a measure *Number of traceability links*, which characterise a count of links traced to the origin of the model.

Annotation. It means that a reader is easily able to determine which elements are most likely to change. This is especially important in the security model because system security policy might be changed often. A measure of *Number of annotation elements* gives the count of annotations used in the model.

Modifiability. It means that the structure and the content are easy to change. When security policies change it should be easy to change the security concerns quickly in the model. To estimate modifiability we define a measure of *Time spent to modify*. It indicates how long it takes to change security policy in the system.

The last two qualitative properties are important when the new system security policies are introduced. Knowing the place and being able to implement the new security concerns quickly might substantially reduce the maintenance cost of overall system.

Syntactic quality is a correspondence between a model and a modelling language. The major goal of the syntactic quality is syntactic correctness. The following qualitative properties and their measures are defined:

Syntactic validity. It means that the grammatical expressions used to create a model should be a part of the modelling language. The measure defined for this qualitative property is a *Number of syntactically invalid statements*. If the value for this measure is higher the syntactical validity of the model is worse.

Syntactic completeness. It means that all grammar constructs and their parts are present in the model. We define a measure *Number of syntactically incomplete statements*. Similarly to syntactic validity measure, the syntactic completeness estimates high if *Number of syntactically incomplete statements* results in null.

To test the syntactic correctness of the security models we need to investigate the concrete syntax of the languages used to create these models.

Pragmatic quality is a correspondence between a model and an interpretation of social and technical audience. The social audience of security model is typically security engineer, but it also includes the system analysts, the software developers, the stakeholders (actors who pay for the development of the secure system), and even the direct users, who should also be involved in the security requirements definition process. With respect to the social actors we define the following qualitative properties and their measures:

Understandability. It means that a reader is able to understand the model with minimum explanations. To estimate the understandability of the security model we can count number of the explanations needed for the social audience. On the other hand here we define a measure *Time spent to understand* the model.

Cross-referencing. It means that the different pieces of model content are linked together. A measure of *Number of cross-reference links* provides a count of cross-referenced links between model components.

Organisation. It means that the model content should be arranged so that a reader could easily locate information and logical relationships among the related information. This could be done by the table of content, division of the model to different sections/chapters, inclusion of the glossary and similar. A measure of *Number of organisation elements* returns a count for the elements, which could help in arrangement of the logical information.

For the technical model interpretation we define that the model should be estimated according to *executability* property*,* meaning that there should exist technology capable of inputting the model and resulting in its implementation. The existence of technology is characterised by a measure *Technology capable to execute the model*.

4.3. Selecting / Creating Security Models

In order to understand the quality of the security models we have selected three languages: PL/SQL [9], SecureUML [2] [19], and UMLsec [13]. We have applied these languages to create the models following the RBAC policy. In fact in our models we were solving the industrial problem; however the actual data and security models could not be presented here due to the privacy concerns of our industrial partner. But here we include an extract of a *meeting scheduling system* [7]. This example closely corresponds to the industry models used in the assessment. Our observations are the same for the industrial problem and for the meeting scheduler system.

Security problem. *Meeting scheduling system* [7] is described as follows: there is a need to organise a *top-secret* meeting in the way that only intended users would know when the meeting starts and ends, what meeting owner and location are. In our example users are allowed adding information about new meetings and viewing information about all existing meetings. But one can delete or change meeting information if and only if he/she is an owner (e.g., meeting initiator) of the meeting. We will present solutions to this problem in the PL/SQL, SecureUML and UMLsec security models.

PL/SQL. Oracle PL/SQL is a procedural language extension [9] to the standard query language (SQL). PL/SQL was introduced by Oracle Corporation to overcome some limitations of SQL and to provide a more complete implementation solution to develop the mission-critical applications, which run on the Oracle database. PL/SQL is an embedded language and could not be used as a standalone language. The language ensures that the programs can stay entirely within the operating-system independent Oracle environment. One of the important aspects of the language is its tight integration with SQL. This means the programs do not rely on intermediate software (e.g. Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC)) in order to run SQL statements. Among other features,

PL/SQL deals with control flows, exception handling, and advanced data types.

```
PROCEDURE meeting_permissions
IS
BEGIN
  IF sec.is_role('User')
  THEN
    DO.item_enable('meeting.start');
    DO.item_enable('meeting.end');
    DO.item_enable('meeting.location');
    DO.item_enable('meeting.owner');
    DO.item_enable('meeting.insert_button');

    IF :meeting.owner = sec.get_username AND
       :meeting.END > SYSDATE
    THEN
      DO.item_edit_yes('meeting.start');
      DO.item_edit_yes('meeting.end');
      DO.item_edit_yes('meeting.location');
      DO.item_edit_yes('meeting.owner');
      DO.item_enable('meeting.update_button');
      DO.item_enable('meeting.delete_button');
    ELSE
      DO.item_edit_no('meeting.start');
      DO.item_edit_no('meeting.end');
      DO.item_edit_no('meeting.location');
      DO.item_edit_no('meeting.owner');
      DO.item_disable('meeting.update_button');
      DO.item_disable('meeting.delete_button');
    END IF;
  ELSE
    DO.item_disable('meeting.start');
    DO.item_disable('meeting.end');
    DO.item_disable('meeting.location');
    DO.item_disable('meeting.owner');
    DO.item_disable('meeting.insert_button');
    DO.item_disable('meeting.update_button');
    DO.item_disable('meeting.delete_button');
  END IF;
END;
```

Fig. 5. Excerpt of the PL/SQL security model

The PL/SQL security model is prepared using the *EditPlus*[1] tool. In general the security model consists of the library that accumulates different security procedures written in PL/SQL. In our example this library contains three procedures that define different security policies for three RBAC roles – *Admin*, *SuperUser*, and *User*. For example in Fig. 5 we illustrate a procedure of *meeting_permissions* that describes a set of permissions, which are defined on the meeting for one RBAC role, called *User* (e.g., the role is checked through the condition *if sec.is_role('User')*). Here we see that if a certain condition (e.g., a user is a meeting *owner* and the meeting end date has not yet passed) holds, it is possible to edit meeting attributes (e.g., *start*, *end*, *location*, and *owner*); otherwise editing is not allowed. In order to receive a running application one needs to compile the PL/SQL source code.

[1] http://www.editplus.com/

SecureUML. The SecureUML modelling language [2] [19] adapts the RBAC model. At the concrete syntax level SecureUML is a "lightweight extensions" of the UML, namely through stereotypes, tagged values and constraints. It introduces the concepts and the stereotypes for *User*, *Role*, and *Permission* as well as the relationships between them (*RoleAssignment* and *PermissionAssignment*). Here the secured objects and the operations are expressed through the protected objects, which are modelled using the standard UML elements.

The semantics of *Permission* is defined through *ActionType* elements used to classify permissions. Here every *ActionType* represents a class of security-relevant operations (e.g., specific security actions: *select*, *change*, *insert*, and *delete*) on a particular type of protected resource. An *AuthorisationConstraint* is a part of the access control policy. It 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, via permissions, to a particular model element representing a protected resource.

The SecureUML security model was prepared using *MagicDraw*[2]. The overall model consists of five diagrams. A top-level diagram is a content diagram as shown in Fig. 6. Other four diagrams present four aspects of the security model. For instance, diagram *SecurityResource-Views* describes the data, which need to be secured, diagrams *RolePermissions-Admin*, *RolePermissions-SuperUser*, and *RolePermissions-User* present the security permissions with respect to the roles Admin, SuperUser, and User.

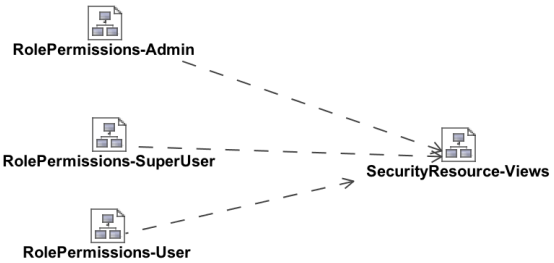

Fig. 6. SecureUML content diagram

In Fig. 7 we present an excerpt of the *Meeting Scheduling* system (*User permissions*). Here two security permissions (e.g., *UserSelectAllMeetings* and *UserUpdateOwnMeeting*) are defined for the role *User* over the resource *Meeting*. Similarly like in the PL/SQL model, an authorisation constraint *UserOwnDataConstraint* defines that only an *owner* is allowed to *update* or *delete* meeting information if the meeting date has not yet passed.

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

In order to receive an executable application, the SecureUML model is automatically transformed to the PL/SQL code (see illustration in the Appendix of this paper). The transformed PL/SQL code is then compiled to a running application.

In our case study we have selected to analyse the model created using SecureUML, but not its PL/SQL transformation. The reason is that we intend to analyse the model, which is editable by the system developers directly.

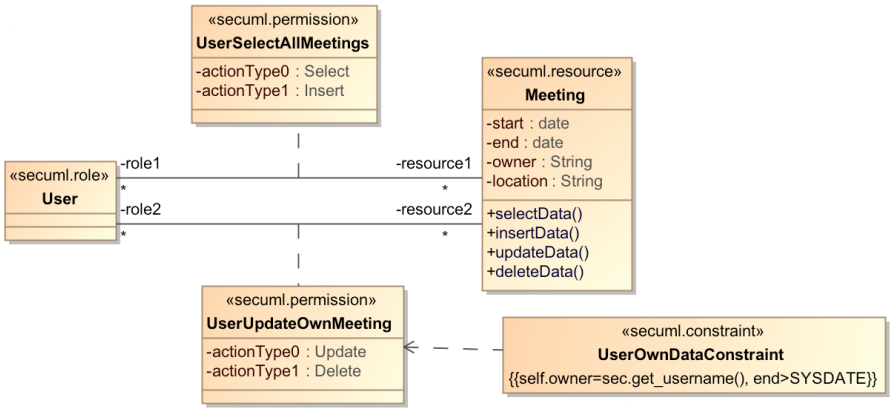

Fig. 7. Excerpt of the SecureUML security model

UMLsec. The UMLsec modelling language [13] is defined as a UML profile extension using stereotypes, tagged values and constraints. Constraints specify security requirements. Threat specifications correspond to actions taken by the adversary. Thus, different threat scenarios can be specified based on adversary strengths.

A subset of UMLsec that is directly relevant to this study is the role-based access control stereotype – <<*rbac*>> – its tagged values and constraints. This stereotype enforces RBAC in the business process specified in the activity diagram. It has three associated tags *{protected}*, *{role}*, and *{right}*. The tag *{protected}* describes the states in the activity diagram where the access to the activities should be protected. The *{role}* tag may have a list of pairs (*actor*, *role*) as its value, where *actor* is an actor in the activity diagram, and *role* is a role. The tag *{right}* has a list of pairs (*role*, *right*) as its value, where *role* is a role and *right* represents the right to access a protected resource. The associated constraint requires that the actors in the activity diagram only perform actions for which they have the appropriate rights.

In Fig. 8 we define an activity diagram, which describes an interaction between *User* and *Meeting*. The diagram specifies that *User* can *Insert data* (e.g., meeting start- and end-dates, meeting owner, and meeting location). Next, *User* is able to *Select data* in order to check if data are correct. If these are not OK *User* is able to *Update* data. After the meeting is over, *User* is able to *Delete data* about this meeting.

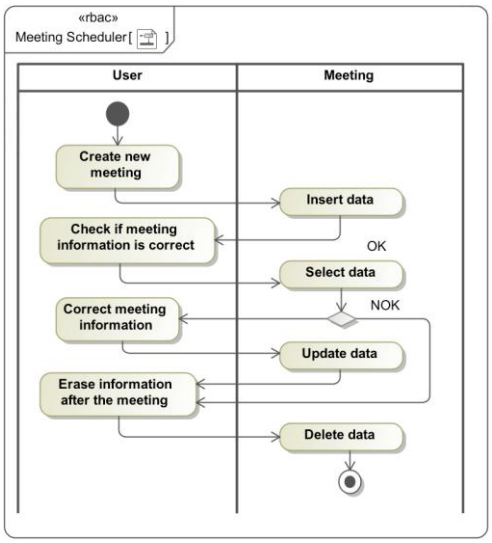

Fig. 8. Meeting Scheduler with UMLsec

This diagram carries an <<*rbac*>> stereotype, meaning that the security policy needs to be applied to the protected actions. For instance, the *User*'s actions lead to the secured actions executed by the *Meeting*. For example, *Insert data* is executed if and only if there exists an associated tag that defines the following: (*i*) *Insert data* is a protected action, (*ii*) there exists a user (e.g., *Bob*) who plays role *User*, and (*iii*) *User* enforces the action *Insert data*. In the activity diagram this associated tag is defined as follows:

```
{protected = Insert data}
{role = (Bob, User)}
{right = (User, Insert data)}
```

Similarly, the sets of associated tags are defined for other three protected actions *Select data*, *Update data*, and *Delete data*. Like in the SecureUML model, using UMLsec we need to define activity diagrams (with the models <<*rbac*>> stereotype) for other two actors – *Admin* and *SuperUser*.

4.4. Performing Evaluation of the Security Models

In this section we will subsequently discuss the results of our assessment of the security models. We will see the results on *semantic*, *syntactic* and *pragmatic* quality types.

4.4.1. Assessment of Semantic Quality

Our analysis of the *semantic quality* for the security models is summarised in Table 1. As defined in Section 4.2 we considered semantic quality according

to *semantic completeness*, *semantic correctness*, *traceability*, *annotation*, and *modifiability*.

Table 1. Semantic quality of the security models

Qualitative property	Measure	PL/SQL security model	SecureUML security model	UMLsec security model
Semantic completeness	Percentage of the RBAC domain coverage	42,86%	71,43% (100%)	85,71%
Semantic correctness	Percentage of security related statements	7,69%	100%	33%
Traceability	Number of traced links	0	0	0
Annotation	Number of annotation elements	0	5	1
Modifiability	Time spent to modify	Not-known	5-10 minutes	5-10 minutes

PL/SQL security model. *Semantic completeness* is assessed through a model correspondence to the RBAC domain (see Section 2.1). In the first condition the PL/SQL model explicitly defines the role (e.g., *User* in Fig. 5) for which security permission is defined. Next the PL/SQL model focuses partially on the presentation of the security *permissions* (e.g., see the second condition expression in Fig. 5), which are defined for the attributes of secured *objects* (e.g., statements like *meeting.start*, *meeting.end*, and others shown in Fig. 5). However it does not define on which *operations* the security permissions are placed. Also the PL/SQL model does not express *users* and *user assignment* relationships. We estimate 42.86% (expresses 3 RBAC concepts out of 7) of the RBAC domain coverage.

The *semantic correctness* of the PL/SQL model is low, because it does not separate the data and programmable concerns from the security concerns. In PL/SQL diagram we found only two statements that are defining security concerns (see two conditions defined in Fig. 5). All other 24 statements are defining different programmable variables or user interface components (e.g., *DO.item_enable('meeting.new_meeting')* is enabling the item of the user interface). We estimate only 7,69% (2 statements out of 26) of the security related statement in the diagram presented in Fig. 5.

The PL/SQL model is not *traced*. This means that origin and rationale for the security decisions are not provided in the model and we did not observe any traceable links in this model. The PL/SQL model is not *annotated*, thus it is difficult to determine which elements are most likely to change.

Modifiability is estimated by the time used to modify different aspects of the model. To estimate this characteristic it was rather difficult because it directly correlates to the *understandability* property (see discussion below). However we acknowledge that, once the model is understood, time spent to modify the model might depend on the scope of the changes and skills of the developer.

SecureUML security model. SecureUML is developed to design the RBAC-based solutions. This means that SecureUML could fully correspond to the semantic domain, thus resulting in high *semantic completeness*. However in our analysed diagram (see Fig. 7) we did not identify RBAC concept of *User* and relationship *User assignment*. Thus we result in 71,43% of the RBAC domain coverage (however we should note that definition of *User* and *User assignment* is not a problem using SecureUML, thus possibly resulting in 100% of semantic completeness).

We identify high *semantic correctness*, because only security solutions are presented in the SecureUML model. We assess percentage of security related statements as 100%.

Like in the PL/SQL security model, in the SecureUML model we did not observe any rationale for security decisions, thus it results in a low *traced* property.

The Secure UML model is partially *annotated*. This annotation is achieved through SecureUML stereotypes (e.g., <<secuml.permission>>, <<secuml.role>>, etc.) and class names given to the *permissions* (e.g., *UserSelectAllMeetings* and *UserUpdateOwnMeeting*) and the *authorisation constraints* (e.g., *UserOwnDataConstraint*). These class names are not directly used in the transformation of the model to code, but they provide additional information to the model reader. They also identify the places in the model where security policy is most likely to be changed. We counted 5 annotation examples in the SecureUML model.

The SecureUML model is *modifiable.* The model implies a certain presentation pattern – *Role-Permission-Resource*, which facilitates the changing of the model. Like for the PL/SQL model we acknowledge that modifiability much depends on the change requirements and on the skills of the developer, but we also observe that the average time of one change might vary from 5 to 10 minutes.

UMLsec security model. The RBAC principles are expressed through the activity diagram using UMLsec. Using UMLsec the majority of the RBAC concepts are defined in the associated tags. For example, *User* and *Roles* are associated in the {*role*} tag, thus, expressing the RBAC *user association* link), *Roles* and *Operations* are combined in the {*right*} tag, thus, defining the RBAC *Permission association* link. The only RBAC *concept* that is not expressed in the UMLsec model is *Permission*, i.e., what the *Roles* are allowed to do with the secure *Objects*. We result in 85,71% (6 concepts out of 7) of the RBAC domain coverage.

Regarding semantic correctness, in the UMLsec diagram we can observe actions related to business/work description (e.g., *Create new meeting*, *Check if meeting information is correct*, *Correct meeting information*, and *Erase information after the meeting*) and actions that needs to support the business/work actions (e.g., ones executed by *Meeting* – *Insert data*, *Select data*, *Update data*, and *Delete data*). The later ones each needs security-related treatment defined through the association tags. Thus we result in 33% of security related statements (actions and association tags) in the UMLsec model.

In the UMLsec model we find only one annotation element, i.e., the <<*rbac*>> (see Fig. 8) stereotype that the modelled security aspect. Similar like in the SecureUML model, we observed no traceability from/to the UMLsec model. In addition, we identify, that depending on the needs for changes, we can modify the UMLsec model in 5-10 minutes.

4.4.2. Assessment of Syntactic Quality

Syntactic quality is expressed through *syntactic validity* and *syntactic completeness*, as defined in Section 4.2. We summarise our analysis of the security models in Table 2.

Table 2. Syntactic quality of the security models

Qualitative property	Measure	PL/SQL security model	SecureUML security model	UMLsec security model
Syntactic validity	Number of syntactically invalid statements	0	1	0
Syntactic completeness	Number of syntactically incomplete statements	0	0	0

PL/SQL security model. The PL/SQL model is of high *syntactic validity* and *syntactic completeness*, because the model is created using the PL/SQL language, a programmable language. We did not observe any syntactically invalid or syntactically incomplete statements. Syntactically this model is also correct because otherwise it would not be possible to compile it to the application.

SecureUML security model. In the current model of the SecureUML we can identify a case of *syntactic invalidity*. For instance the SecureUML documentation [2] [19] identify that *authorisation constraints* need to be written in OCL (Object Constraint Language). However in our model (see Fig. 7) the SQL-based authorisation constraints are used (e.g., see class *UserOwnDataConstraint* constraint *{owner=sec.get_username*(), *end>SYSDATE}*). On the other hand the model is *syntactically complete* – it includes only UML extensions and their relationships proposed by the authors of SecureUML, thus we did not observe any syntactically incomplete statements.

UMLsec security model. We did not observe any syntactically invalid or syntactically incomplete statements in the UMLsec model. However we should note that this model was checked only manually. For the UMLsec model investigated by us, we were not running any transformations to the application code (like we did with the PL/SQL or SecureUML models).

4.4.3. Assessment of Pragmatic Quality

We summarise the analysis of the pragmatic quality for the security models in Table 3. Pragmatic quality is defined in terms of *understandability*, *organisation*, *cross-referencing*, and *executability*, as presented in Section 4.2.

Table 3. Pragmatic quality of the security models

Qualitative property	Measure	PL/SQL security model	SecureUML security model	UMLsec security model
Understand a-bility	Number of explanations	More than 45 minutes	10-15 minutes	10-15 minutes
Organisation	Number of elements for model organisation	2	4	4
Cross referencing	Number of cross-reference links	1	3	3
Executability	Tools to execute the model	Yes	Yes	No

PL/SQL security model. We found the PL/SQL model of low *understandability*. We were not able to understand the PL/SQL model without a proper explanation provided by the model developers. All together it took us more than 45 minutes to grab some security concerns defined in the PL/SQL model. On the one hand the reason might be that we as the evaluators, were not the experts in the PL/SQL language. But, on the other hand, taking into account that the security models should be used to communicate with the users of the software systems (who are not familiar with PL/SQL neither), the time spent to understand security concerns could be even longer.

As presented in Section 4.3, the PL/SQL model is *organised* into the library that accumulates different security-oriented procedures. Thus, this model contains a structure, which could guide finding the relevant security concerns.

Furthermore the PL/SQL model is presented as a plain-text source code, thus it does not contain any hyperlinks that would *cross-reference* related security concerns (but also see Section 4.5.2). On the other hand the library structure could be used to follow from one security procedure to another (in our case between three procedures, defined regarding to the user *role*). However these links could be used only manually; no tool support for them is provided.

Finally, regarding the PL/SQL model *executability*, it is possible to compile this model using the Oracle database management system resulting in a running application.

SecureUML security model. The Secure UML model is well *understood* by those readers familiar with the UML modelling notation. This also opens the way to communicate this model to a larger audience, including various project stakeholders, potential direct users of the system, the systems analysts, and the developers. Our personal experience is that this model is

quite intuitive and did not require a big effort (around 10-15 minutes) to understand it.

As described in Section 4.3, the SecureUML model consists of several diagrams. It is also supported by a modelling tool (in our case – MagicDraw), which simplifies managing the model itself and support the model organisation. The tool provides the containment view and zoom means (see Fig. 9), which developer could use to find the relevant model elements, navigate between and within the model diagrams. As illustrated in Fig. 6 the navigation map diagram helps to navigate from the content diagram to diagrams presenting different security concerns.

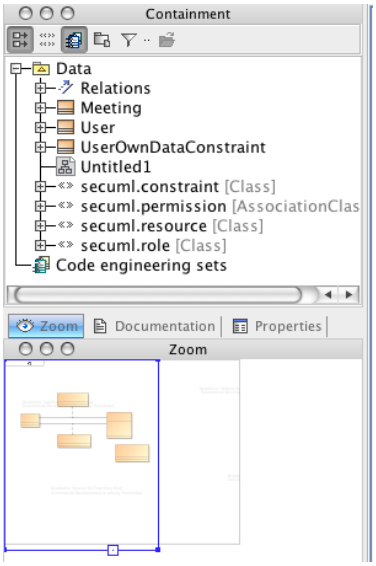

Fig. 9. Means to support SecureUML model organisation provided by the tool

Model *cross-references* includes links between the navigation map and separate diagrams, between the containment views and separate diagrams and model elements. It is also possible to define cross-references between the separate model diagrams (however this possibility was not used in our case).

The SecureUML model is *executable*: there exists a number of the transformation rules defined using the Velocity[3] language (interpretable by the MagicDraw tool).These rules define how to transform the model to PL/SQL code, which could be executed through Oracle database management system.

UMLsec security model. Regarding the social actor interpretation, we result in the same assessment of the UMLsec model as for the SecureUML model. For instance, we found that both models can be understood in 10-15 minutes. The UMLsec contains 4 elements for its organisations (since it is

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

created using MagicDraw, the same modelling tool as the SecureUML security model). Similarly it includes three means to cross reference inter-related parts.

However we were not able to execute the UMLsec model – there are no means to generate the PL/SQL code from this model (at least using the MagicDraw tool). Thus there exist a potential field for improvement regarding the technical interpretation aspect.

4.5. Validating the Evaluation Results

After performing the evaluation of the security models, next step is to validate the received results. In this section we will characterise the potential threats to validity. We will also describe what feedback we received from the models authors regarding our evaluation scores.

4.5.1. Threats to Validity

In our case study *only* two evaluators assessed the security models according to their knowledge and experience. This certainly raises the level of subjectivity and influences the *internal validity* of the case study. To mitigate this threat the evaluation results were communicated to the model developers.

In our case the SEQUAL framework was instantiated with a *certain* set of qualitative properties (and their measures). This certainly affects the *conclusion validity*, because if any other qualitative properties were applied, it might result in different outcome. But this threat is rather limited because these qualitative properties are theoretically sound and the selection is based on the previous experience (i.e., [4], [11], [15], [21], [24]).

In this case study we analysed *only* three different security models and these models were quite limited in their size. This might influence the *external validity* by a fact, that different results might be received if some other security models (created either using PL/SQL, SecureUML, UMLsec or any other language) would be analysed. However our research subject is providing a solution to an industry problem; thus, we believe that our analysis is generalisable in similar situations.

Finally, we try to avoid a use of single type of measuring that might affect the *construct validity*. The evaluation of the security models is followed with the communication of the received results to the models developers (see Section 4.5.2). This certainly reduces a risk of the mono-interpretation.

4.5.2. Communicating Results to Developers

We reviewed our results together with the developers of the security models. Firstly, the developers noted that the overall quality of both models could be improved if these evaluation results were taken into account. For example, the

traceability, *annotation*, and *understandability* of the PL/SQL model could be easily improved using code comments. However, the developers acknowledged that this is not the case in the common practice; or the code comments, even if they are present, are not sufficient.

Secondly, developers provided few remarks regarding some qualitative properties. For instance, *semantic completeness* could be improved by presenting concrete instances in the models (similarly as done in [2] and [19]). This means hard coding in the PL/SQL model and object presentation in the SecureUML model; however, doing so we would neglect the principle of generosity in modelling.

In order to improve *syntactic validity* of the SecureUML model we could write the authorisation constraints in OCL instead of SQL. However the current approach to transform the SecureUML model does not have rules for the OCL interpretation. Further, it is not possible to perform transformation from the UMLsec security model to the executable code. Certainly the targeted transformation templates (as they are provided for the models created in SecureUML) could improve the *executability* of UMLsec.

On the one hand, a tool used to make the PL/SQL model, does not support hyper-linking. Although there exist several PL/SQL editing tools (e.g., *Oracle SQLDeveloper* or *Quest Software Toad for Oracle*, actually used by our industrial partner) that supports cross-references between various model elements, these were not used in this case study. On the other hand, developers also indicated that PL/SQL grammar principles, the ones, which allow expressing procedures (e.g., *PROCEDURE meeting_permissions* in Fig. 5) and referring to them from the main code, could also be seen as textual *cross-referencing*. We took this in mind when scoring for the *Number of cross-reference links*.

4.6. Reporting on the Quality of the Security Models

Table 4 shows the summary of the overall comparison of the security models. We found that three qualitative properties (i.e., *traceability*, *syntactic completeness*, and *executability*) score equally for the PL/SQL and SecureUML models. One qualitative property – *syntactic validity* – is found to be better in the PL/SQL model. The seven remaining qualitative properties (i.e., *semantic completeness*, *semantic correctness*, *annotation*, *modifiability*, *understandability*, *organisation*, and *cross-referencing*) are evaluated to be higher in the SecureUML model.

Regarding models in PL/SQL and UMLsec, we see that PL/SQL was scoring better for *executability* qualitative property. Three qualitative properties – *traceability*, *syntactic validity* and *syntactic completeness* – are assessed equally. The remaining seven qualitative properties (*semantic completeness, semantic correctness, annotation, modifiability, understandability, organisation*, and *cross-referencing*) are evaluated better for the security model created in UMLsec.

Table 4. Summary of quality assessment for the security models

Model A created in	Model B created in	Model A is better in	Two models score equal in	Model B is better in
PL/SQL	**SecureUML**	Syntactic validity	Traceability, syntactic completeness, executability	Semantic completeness, semantic correctness, annotation, modifiability, understandability, organisation, and cross-referencing
		1 qual. property	*3 qual. properties*	*7 qual. properties*
PL/SQL	**UMLsec**	Executability	Traceability, syntactic validity, syntactic completeness	Semantic completeness, semantic correctness, annotation, modifiability, understandability, organisation, cross-referencing
		1 qual. property	*3 qual. properties*	*7 qual. properties*
SecureUML	**UMLsec**	Semantic completeness, semantic correctness, annotation, executability	Traceability, modifiability, syntactic completeness, understandability, organisation, cross-referencing	Syntactic validity
		4 qual. properties	*6 qual. properties*	*1 qual. property*

Six qualitative properties, namely *traceability, modifiability, completeness, understandability, organisation*, and *cross referencing* – are evaluated equally both for the SecureUML and for the UMLsec security models. One qualitative property – syntactic validity – is found better for the UMLsec model. The remaining four qualitative properties (*semantic completeness, semantic correctness, annotation*, and *executability*) are evaluated better for the SecureUML security model.

5. Discussion

In this section we finalise our work. Firstly, we discuss the related work regarding the link between the RBAC, security languages and the model-driven security. Next, we conclude our paper and highlight few future research directions.

5.1. RBAC and Security Languages

In [1] the $BRAC_0$ pattern is applied for comparison of security modelling approaches. The survey shows that, on the one hand, SecureUML does not explicitly model security criteria (such as confidentiality, integrity, and availability) but it focuses on modelling the solutions to security problems guided by the RBAC nature. With SecureUML, a modeller can define assets, however, the language does not allow expressing attacks or harms to the assets. On the other hand, UMLsec is guided by security criteria, however it

does not have means to model them explicitly. The UMLsec application is driven by analysis of system vulnerabilities: (i) once security vulnerabilities have been identified, the system design is progressively refined to eliminate the potential threats; (ii) the refinement of the design might be continued until the system satisfies the security criteria. Although UMLsec was analysed based on the $BRAC_0$ pattern, authors does not specifically indicate how well this approach is suitable for the RBAC modelling.

In [12] 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 and UMLsec. Authors indicate that UMLsec is rather general approach than specific, thus it cannot be used to model access control policies solely. On the other hand SecureUML is suggested as the means to specify access control policies. However SecureUML cannot describe protected resources (system design), thus, it has to be used in conjunction with a base modelling language.

Elsewhere in [22] [23] the SecureUML and UMLsec are compared in order to determine the transformation points between models of these languages. It was noticed the limitation of SecureUML to indicate security criteria, but this language is well suited to engineer security controls after the security decisions are done. It was also observed that the UMLsec application follows the standard security modelling methods [26] and it could provide means for the RBAC modelling: it helps defining the dynamic characteristics of the secure system. The analysis suggests that both SecureUML and UMLsec can complement each other and result in more complete specifications of secure information systems (where both static and dynamic system characteristics are understood).

Although the identified works are useful regarding their timely comparison of the modelling languages against the RBAC model, these studies remain theoretical. It is suggested that such an approach could be used at the initial stage of the languages selection, but for the deeper understanding one needs more fine-grained analysis of the development means. Thus our current proposal – an approach to assess the quality of the security models – suggests the means for the hands-on testing of the modelling and development languages for security. Using our proposal the developers are encouraged to apply the modelling and development languages in order to understand the quality of the resulting security models.

5.2. Model-driven Security

We found none empirical studies that would compare quality of security models prepared using approaches from *different development stages*. The literature reports on a number of case studies [5], [33], [34] 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. Similarly to [3], [20], [34] we observe that security model facilitates automatic code generation, i.e., the SecureUML security model is *executable* through its generation to PL/SQL code. We also argue that the security models should be prepared with the high-quality modelling language [5] that ensures the model *semantic completeness*, and tools [20] 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 [34].

We identified only one case study performed by Clavel *et al* [3], reporting on the SecureUML application in practice. Here authors observe that although the security models are integrated with the data models, the security design remains independent, reusable and evolvable. In our work we also observe that *semantic correctness* of SecureUML and UMLsec models is high, because the representation is oriented to the security aspects. We also observe that SecureUML and UMLsec models are *modifiable*, which means the first step towards model evolvability. Like in [3] we identify that the SecureUML and UMLsec models are understandable at least to readers who are familiar with UML. This might ease communication of requirements and design solutions to project stakeholders [20].

5.3. Conclusion and Future Work

In this paper we have developed a systematic approach to compare quality of security models. Our approach is based on the instantiation of the SEQUAL framework [15] [16]. To illustrate the performance of our proposal we have executed a cases study, where we have compared quality of three security models. One model is prepared at the *implementation stage* using PL/SQL [9]; other two models are developed at the *system design stage* using SecureUML [2] [19] and UMLsec [13]. We resulted in (*i*) a higher quality for the SecureUML security model regarding UMLsec and PL/SQL; and (*ii*) higher quality for the UMLsec security model regarding PL/SQL. Thus, it suggests that practitioners should consider security analysis at the earlier stages (at least design or maybe even requirements engineering) of the software system developing. However we also note that *executability* of the UMLsec model is worse than *executability* of the PL/SQL model. Thus, if one wishes to create executable models he would prefer PL/SQL (or SecureUML) instead of UMLsec.

Our comparison also identifies important directions [33] for improvement of the security analysis at the early stages. For example, a mature *security modelling method* needs to be introduced in order to guide discovery of the early security requirements and to support security quality assurance through overall project planning. This would allow improving the *traceability* qualitative property, also facilitating recording of the rationales for security decisions.

Another concern includes development and improvement of the modelling *tools* (e.g., MagicDraw and Velocity interpreter) that would support the translation of the design models (e.g., SecureUML) to the implementation

code (e.g., PL/SQL). For instance, we need to define guidelines and transformation rules for the OCL-based authorisation constraints. This would also improve the *syntactic validity* of the SecureUML model. On the other hand *executability* of the UMLsec security model is not supported at all – this might result in that practitioners would select the PL/SQL language instead.

For the successful adoption by practitioners, model driven security analysis should be compatible with the working *processes*. We plan to perform another case study where we would investigate quality of processes to develop security models at the system design stage (e.g., using SecureUML, UMLsec or other modelling language) against quality of processes to develop security models at the system implementation stages (e.g., using PL/SQL).

Finally, we need to support a *goal-driven process* [33], where we would define goals to introduce security model-driven development systematically. In this paper we specifically focused 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 application and presentation levels.

Acknowledgment. This research was conducted while the first and third authors were at the Software Technology and Applications Competence Centre (STACC) and the second author was at Logica Estonia. The research is partly funded by the EU Regional Development Funds via Enterprise Estonia. We also thank the anonymous referee for the helpful comments and suggestions.

C

References

1. Bandara, A., Shinpei, H., Jurjens, J., Kaiya, H., Kubo, A., Laney, R., Mouratidis, H., Nhlabatsi, A., Nuseibeh, B., Tahara, Y., Tun, T., Washizaki, H., Yoshioka, N., Yu, Y.: Security Patterns: Comparing Modelling Approaches. Technical Report No 1009/06, Department of Computing Faculty of mathematics, Computing Technology, The Open University (2009)
2. Basin, D., Doser, J., Lodderstedt, T.: Model Driven Security: from UML Models to Access Control Infrastructure. ACM Transactions on Software Engineering and Methodology (TOSEM), 15 (1), 39--91. (2006)
3. Clavel, M., Silva, V., Braga, C., Egea, M.: Model-driven Security in Practice: an Industrial Experience, In Proceedings of the 4th European Conference on Model Driven Architecture: Foundations and Applications, Springer-Verlag, pp. 326--337. (2008)
4. Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Srimani, P., Ta, A., Theofanos, M.: Identifying and Measuring Quality in a Software Requirements Specification. In Proceedings of the 1st International Software Metrics Symposium, pp. 141--152. (1993)
5. de Miguel, M., Jourdan, J., Salicki, S.: Practical Experiences in the Application of MDA. In Proceedings of the 5th International Conference on The Unified Modeling Language, Springer-Verlag, 128--139, (2002)
6. Elahi, G., Yu, E.: A Goal Oriented Approach for Modeling and Analyzing Security Trade-Offs, In: Parent et al. (eds.), Proceedings of the 26th International Conference on Conceptual Modelling (2007)

7. Feather, M.S., Fickas, S., Finkelstein, A., van Lamsweerde A.: Requirements and Specification Exemplars. Automated Software Engineering, 4: 419--438. (1997)
8. Ferraiolo D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST Standard for Role-based Access Bontrol. ACM Transactions on Information and System Security (TISSEC), 4(3), 224--274. (2001)
9. Feuerstein, S., Pribly, B.: Oracle PL/SQL Programming. O'Reilly Media Inc, 4th edition edition (2005)
10. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Modeling Security Requirements Through Ownership, Permision and Delegation. In Proceedings of the 13th IEEE International Conference on Requirements Engineering, IEEE Computer Society (2005)
11. Hakkarainen S., Matulevičius R., Strašunskas D., Su X. and Sindre G.: A Step Towards Context Insensitive Quality Control for Ontology Building Methodologies. In Proceedings of the CAiSE 2004 Open INTEROP-EMOI Workshop, 205--216. (2004)
12. Jayaram, K.R., Mathur, A.P.: Software Engineering for Secure Software – State of the Art: a Survey. Technical report CERIAS TR 2005-67, Department of Computer Sciences & CERIAS, Purdue University (2005)
13. Jurjens, J.: Secure Systems Development with UML. Springer-Verlag Berlin Heidelberg, (2005)
14. Knodel, J., Anastasopolous, M., Forster, T., Muthig, D.: An Efficient Migration to Model-driven Development (MDD). Electronic Notes in Theoretical Com puter Science 137 17--27. (2005)
15. Krogstie, J.: A Semiotic Approach to Quality in Requirements Specifications. In Proceedings of IFIP 8.1 working Conf. on Organisational Semiotics, 231--249. (2001)
16. Krogstie, J.: Using a Semiotic Framework to Evaluate UML for the Development for Models of High Quality. In: Siau, K., Halpin, T. (eds.) Unified Modelling Language: Sys- tem Analysis, Design and Development Issues, IDEA Group Publishing, pp. 89--106. (1998)
17. Lin, L., Nuseibeh, B., Ince, D., Jackson, M.: Using Abuse Frames to Bound the Scope of Security Problems. In Proceedings of the 12th IEEE International Conference on Requirements Engineering, IEEE Computer Society 354--355. (2004)
18. Lindland, O. I., Sindre, G., Sølvberg, A.: Understanding Quality in Conceptual Modelling. IEEE Software, 11(2), pp. 42--49. (1994)
19. Lodderstedt, T., Basin, D., Doser, J.: SecureUML: A UML-based Modeling Language for Model-driven Security. In Proceedings of the 5th International Conference on The Unified Modeling Language, LNCS, vol. 2460 Springer-Verlag, 426--441. (2002)
20. MacDonald, A., Russell, D., Atchison, B.: Model-driven Development within a Legacy System: An Industry Experience Report. In Proceedings of the 2005 Australian Software Engineering Conference (ASWEC'05). IEEE Computer Science. (2005)
21. Matulevičius, R.: Process Support for Requirements Engineering: A Requirements Engineering Tool Evaluation Approach. PhD theses. Norwegian University of Science and Technology. (2005)
22. Matulevičius, R., Dumas, M.: A Comparison of SecureUML and UMLsec for Role-based Access Control, Proceedings of the 9th Conference on Databases and Information Systems, 171--185. (2010)

23. Matulevičius, R., Dumas, M.: "Towards Model Transformation between SecureUML and UMLsec for Role-based Access Control," Databases and Information Systems VI, IOS Press, 339--352. (2011)
24. Matulevičius, R., Heymans, P.: Comparison of Goal Languages: an Experiment. In Proceedings of the Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2007), Trondheim, Norway, Springer-Verlag, 18--32. (2007)
25. Matulevičius, R., Lepmets, M., Lakk, H., Sisask, A.: Comparing Quality of Security Models: a Case Study. In Local Proceedings of the 14th East-European Conference on Advances in Database and Information Systems. University of Novi sad, Serbia, 95 - 109. (2010)
26. Mayer N.: Model-based Management of Information System Security Risk. PhD Thesis, University of Namur (2009)
27. McDermott, J., Fox, C.: Using Abuse Case Models for Security Requirements Analysis. In Proceedings of the 15th Annual Computer Security Applications Conference (1999)
28. Moody, D.L.: Theoretical and Practical Issues in Evaluating the Quality of Conceptual Models: Current State and Future Directions. Data and Knowledge Engineering 55 (3) 243--276. (2005)
29. Mouratidis, H.: Analysing Security Requirements of Information Systems using Tropos. In Proceedings 1st Annual Conference on Advances in Computing and Technology 55--64. (2006)
30. Piattini, M., Genero, M., Poels, G., Nelson, J.: Towards a Framework for Conceptual Modelling Quality. In: Genero, M., Piattini, M., Calero, C. (eds.) Metrics for Software Conceptual Models, Imperial College Press, London 1--18. (2005)
31. Sindre, G.: Mal-activity Diagrams for Capturing Attacks on Business Processes. In Proceedings of the Working Conference on Requirements Engineering: Foundation for Software Quality, Springer-Verlag Berlin Heidelberg 355--366. (2007)
32. Sindre, G., Opdahl, A.L.: Eliciting Security Requirements with Misuse Cases. Requirements Engineering Journal 10 (1) 34--44. (2005)
33. Staron, M.: 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 57--72. (2006)
34. The Middleware Company: Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach: Productivity Analysis, MDA Productivity case study. (2003)
35. van Lamsweerde, A.: Elaborating Security Requirements by Construction of Intentional Anti-models. In Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society 148--157. (2004)

Appendix

In order to get the impression on how the SecureUML security model (e.g., see Fig. 7) is transformed into the PL/SQL code, we included a sample of the transformation outcome with respect to the *Update* security action. Similarly the PL/SQL code is generated for other three security actions – *Select*, *Insert* and *Delete*.

```
-- 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.start, :OLD.start) != 'Y' -- start updated
   THEN
    IF 1 != 1 OR sec.is_role('User') = 'Y' AND
       self.owner = sec.get_username() AND
       self.end > SYSDATE -- Permission from UserUpdateOwnMeeting
     THEN
      self.start := :NEW.start;
    ELSE
      RAISE ex_denied;
    END IF;
  END IF;
  IF util.null_eq(:NEW.end, :OLD.end) != 'Y' -- end updated
   THEN
    IF 1 != 1 OR sec.is_role('User') = 'Y' AND
       self.owner = sec.get_username() AND
       self.end > SYSDATE -- Permission from UserUpdateOwnMeeting
     THEN
      self.end := :NEW.end;
    ELSE
      RAISE ex_denied;
    END IF;
  END IF;
  IF util.null_eq(:NEW.owner, :OLD.owner) != 'Y' -- owner updated
   THEN
    IF 1 != 1 OR
       sec.is_role('User') = 'Y' AND
       self.owner = sec.get_username() AND
       self.end > SYSDATE -- Permission from UserUpdateOwnMeeting
     THEN
      self.owner := :NEW.owner;
    ELSE
      RAISE ex_denied;
    END IF;
```

```
  END IF;
  IF util.null_eq(:NEW.location, :OLD.location) != 'Y' -- location updated
   THEN
    IF 1 != 1 OR
       sec.is_role('User') = 'Y' AND
       self.owner = sec.get_username() AND
       self.end > SYSDATE -- Permission from UserUpdateOwnMeeting
     THEN
      self.location := :NEW.location;
    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;
/
```


Dr. Raimundas Matulevičius received his PhD diploma from the Norwegian University of Science and Technology, Norway in the area of computer and information science. Currently Matulevičius holds an associated professor position at the Institute of Computer Science, University of Tartu, in Estonia. Matulevičius' research interests cover information systems and requirements engineering, system and software development processes, model-driven development, system and software security, and security risk management. Currently, the publication record includes more than 50 articles published in the peer-reviewed international journals, conferences and workshops. Matulevičius was invited for multiple times to co-review papers for the international journals (e.g., REJ, TOSEM, SoSyM, COSE). Few years in a row he is invited to be a program committee member at the international workshops and conferences (e.g., CAiSE, REFSQ, PoEM and other).

Henri Lakk is a master's degree student at University of Tartu, where he is also giving labs and lectures. His study and research interest includes model driven security of information system. Lakk is working also in Webmedia Estonia as a PL/SQL programmer.

Dr. Marion Lepmets is a recognised researcher on software and IT service quality, process improvement and assessment. She is currently a Post-Doctoral fellow in Public Research Centre Henri Tudor conducting research on IT service quality measurement and process improvement impact on IT service quality. She is a technical program committee member at SPICE, EuroSPI and Baltic IT&DB conferences, and Luxembourgish representative to ISO/IEC JTC1 SC7 (software and systems standards subcommittee).

Received: December 31, 2010; Accepted: April 29, 2011.