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*.

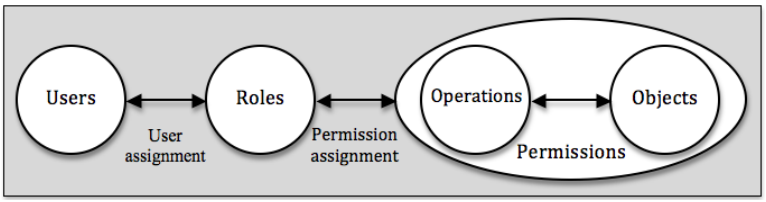

Fig. 1. Role-based Access Control Model (adapted form [8])

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.

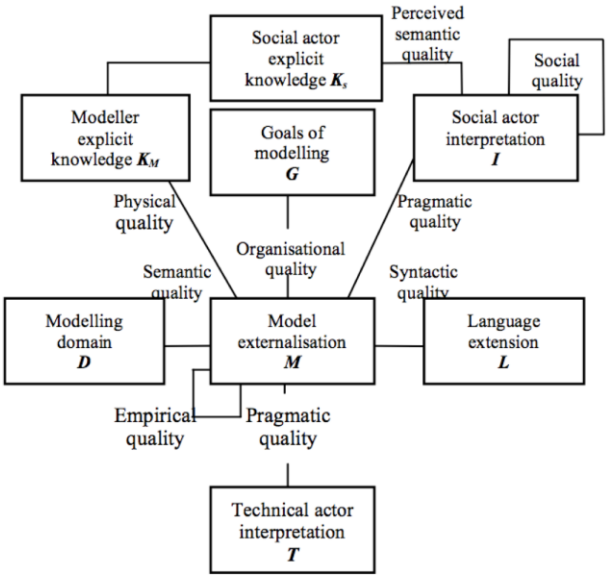

Fig. 2. The SEQUAL framework (adapted from [15], [16])

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.

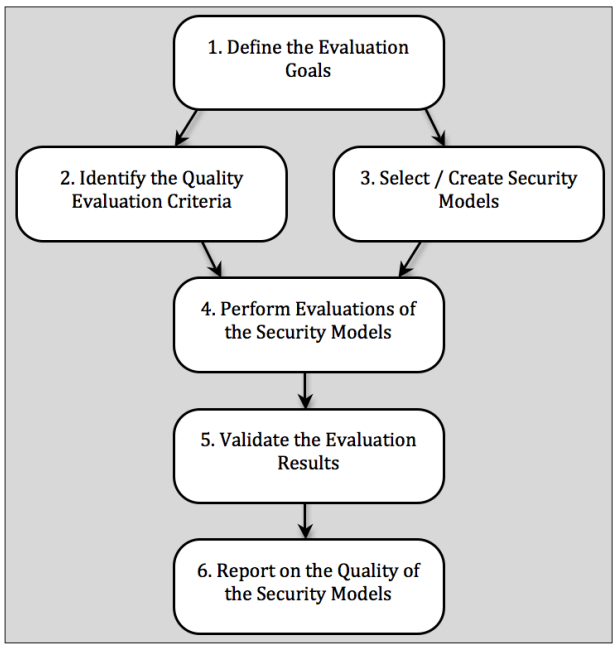

Fig. 3. An Assessment of Quality for Security Models

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.

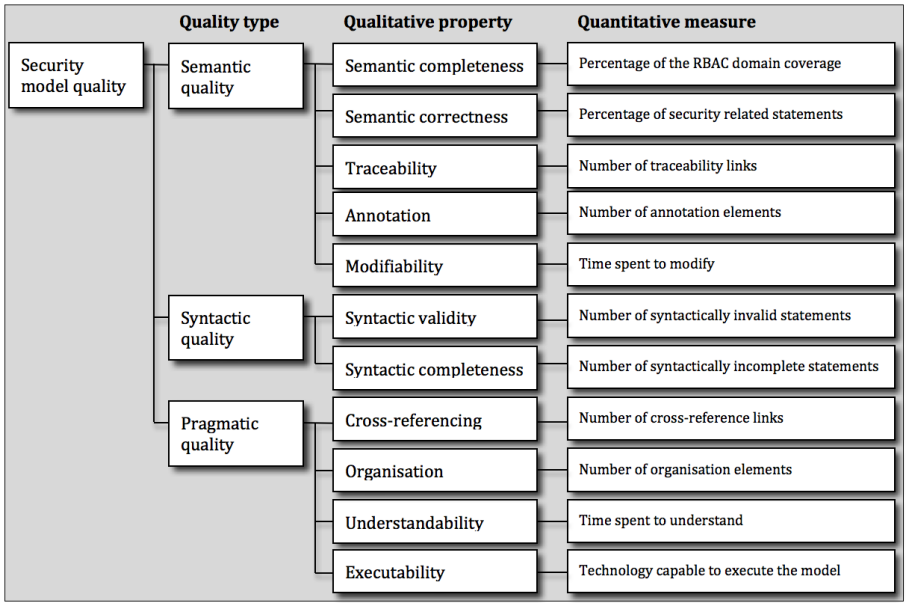

Fig. 4. Instantiation of the SEQUAL framework

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.

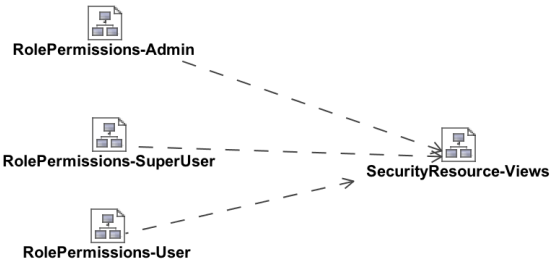

Fig. 6. SecureUML content diagram

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.

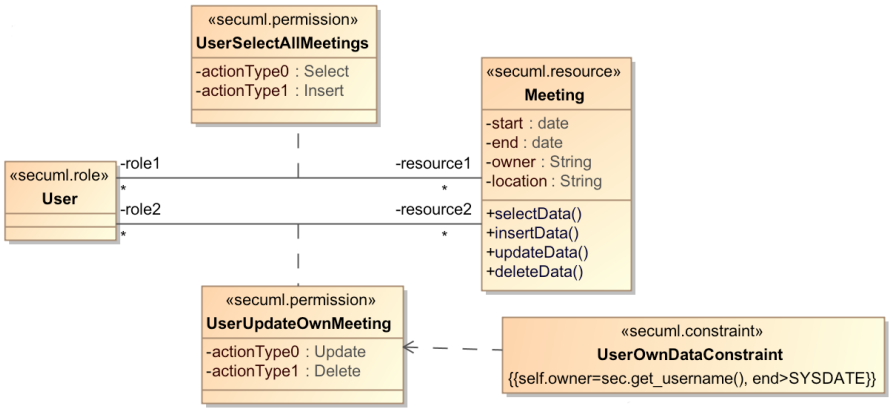

Fig. 7. Excerpt of the SecureUML security model

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.

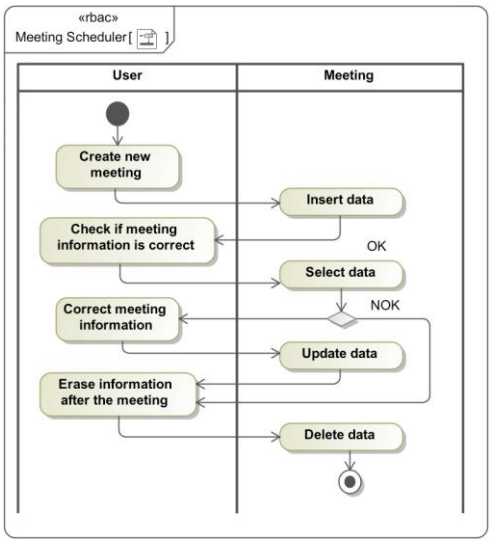

Fig. 8. Meeting Scheduler with UMLsec

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., <<securml.permission>>, <<securml.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.

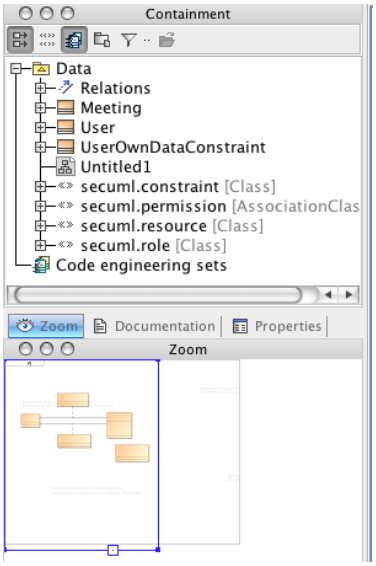

Fig. 9. Means to support SecureUML model organisation provided by the tool

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.

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.