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

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.

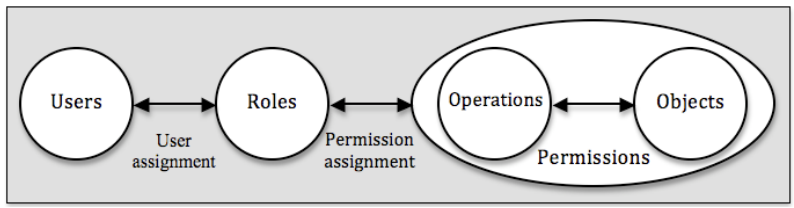

Fig. 1. Role-based access control model (adapted from [5])

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 $\boldsymbol{K}$ of a participant has to be externalised in the model $\boldsymbol{M}$ by the use of a modelling language $\boldsymbol{L}$; and internalisability, meaning that the externalised model $\boldsymbol{M}$ can be made persistent and available, enabling the stakeholders to make sense of it. *Empirical quality* deals with error frequencies when reading or writing $\boldsymbol{M}$, as well as coding and ergonomics when using modelling tools. *Syntactic quality* is the correspondence between $\boldsymbol{M}$ and the language $\boldsymbol{L}$ in which $\boldsymbol{M}$ is written. *Semantic quality* examines the correspondence between $\boldsymbol{M}$ and the domain $\boldsymbol{D}$. *Pragmatic quality* assesses the correspondence between $\boldsymbol{M}$ and its social as well as its technical audiences' interpretations, respectively, $\boldsymbol{I}$ and $\boldsymbol{T}$. *Perceived semantic quality* is the correspondence between the participants' interpretation $\boldsymbol{I}$ of $\boldsymbol{M}$ and the participants' current explicit knowledge $\boldsymbol{K_S}$. *Social quality* seeks agreement among the participants' interpretations $\boldsymbol{I}$. Finally, *organisational quality* looks at how the modelling goals $\boldsymbol{G}$ are fulfilled by $\boldsymbol{M}$. 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.

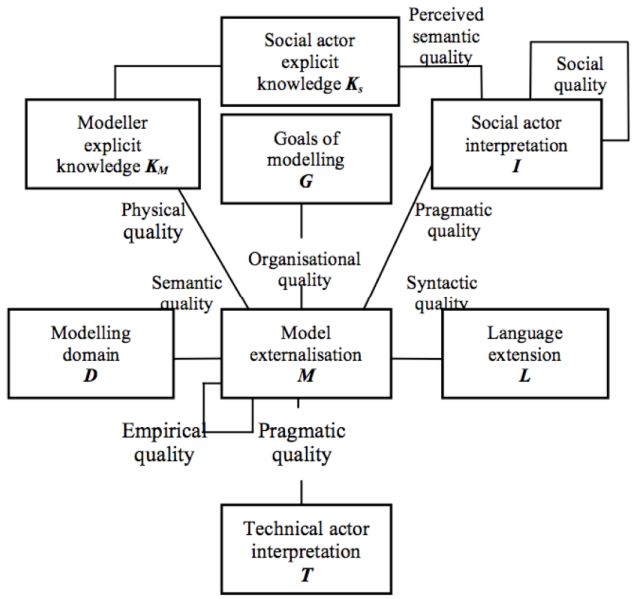

Fig. 2. The SEQUAL framework (adapted from [8][9])

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.

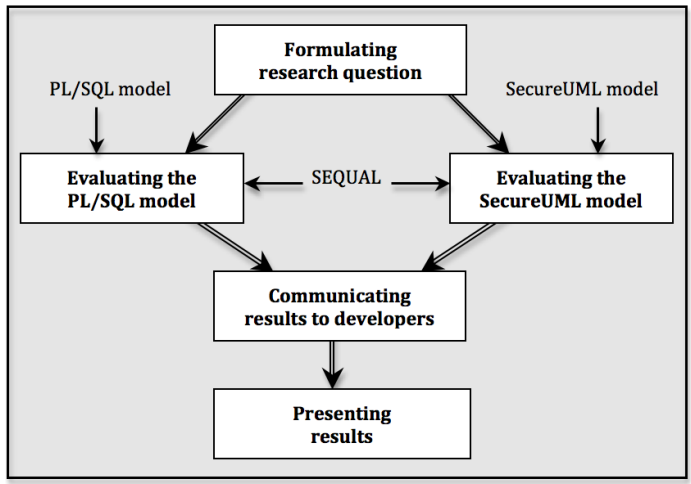

Fig. 3. A case study design

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.

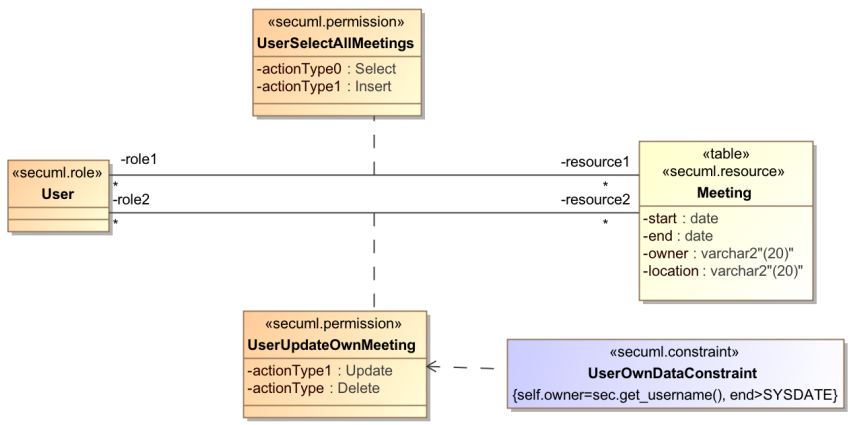

Fig. 5. Excerpt of the SecureUML security model

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.*

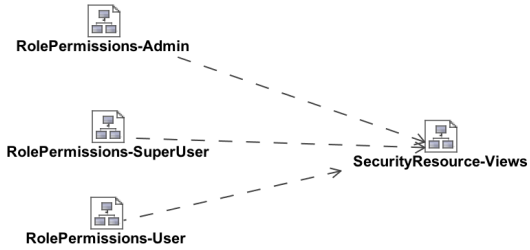

Fig. 7. SecureUML content diagram

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)