Tool Support for Enforcing Security Policies on Databases

Jenny Abramov[2], Omer Anson[2], Arnon Sturm[1,] Peretz Shoval[1]

[1] Department of Information Systems Engineering
[2] Deutsche Telekom Laboratories (T-Labs)
Ben-Gurion University of the Negev
Beer Sheva 84105, Israel
jennyab@bgu.ac.il, oaanson@gmail.com,
sturm@bgu.ac.il, shoval@bgu.ac.il

Abstract. Security in general and database protection from unauthorized access in particular, are crucial for organizations. It has long been accepted that security requirements should be considered from the early stages of the development. However, such requirements tend to be neglected or dealt-with only at the end of the development process. The Security Modeling Tool presented in this study aims at enforcing developers, in particular database designers, to deal with database authorization requirements from the early stages of development. This software demonstration shows how the Security Modeling Tool assists to define organizational security policies and use them during the application development to create a secured database schema.

Keywords: Secure software engineering, database design, authorization.

1 Introduction

Data is the most valuable asset for an organization as its survival depends on the correct management, security, and confidentiality of the data [1]. In order to protect the data, organizations must secure data processing, transmission and storage. Developers of data-oriented systems always face problems related to security. This is the case as security and other non-functional requirements are usually ignored in the early stages of the development process.

To overcome these lacks, we provide a methodology that guides developers in the incorporation of particular organizational security policies, as well as verifying their correct application. In addition, the methodology enables the developer to transform the result into code.

The methodology incorporates ideas from two areas of expertise: in the area of *methodologies for system development*, we adopt the principle of integrating data and functional modeling at the early stages of the development, suggested by the Functional Object-Oriented Methodology (FOOM) [6]. Additionally, in the area of *domain engineering*, we adopt the principles suggested by the Application Based Domain Modeling (ADOM) approach [4]. ADOM supports building reusable assets on the one hand, and representing and managing knowledge in specific domains on

the other hand. This knowledge guides the development of various applications in that domain and serves as a verification template for their correctness and completeness.

The developed methodology is supported by the Security Modeling Tool (SMT), which was tailored for its needs and was equipped with the required facilities. SMT enables the modeling of security patterns and enforces their correct use during application development. The knowledge captured in the security patterns is used to automatically verify that the application models are indeed secure, according to the defined patterns. Having a verified model, secure database schemata can be automatically generated.

The SMT is an Eclipse plug-in. The SMT uses, internally, libraries provided by other Eclipse plug-ins to complete many tasks. For instance, Eclipse Modeling Framework [2] is used to interface with the UML diagrams, and the Standard Widget Toolkit [7] is used to provide additional graphical user interface where needed. It should be noted that the SMT is continuously under development.

The rest of this paper is structured as follows: Section 2 provides an overview on the methodology, Section 3 presents and illustrates the use of the Security Modeling Tool, and Section 4 summarizes and proposes ideas for future work.

2 Methodology Overview

The methodology can be roughly divided into four phases: preparation, analysis, design, and implementation. Fig. 1 presents the methodology scope in terms of the tasks (presented in ellipse) to be performed in each phase and the generated artifacts (presented in rectangle). The preparation phase occurs at the organizational level, whereas the other three occur at the application development level.

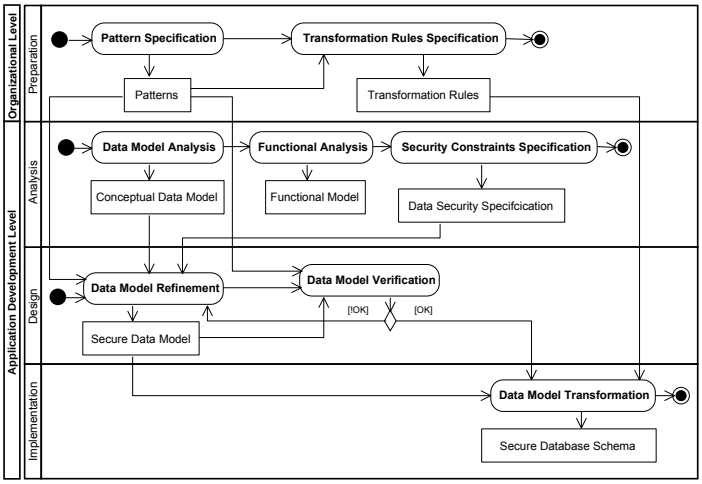

Fig. 1. Methodology overview

At the organizational level, also termed the preparation phase, we define security policies in the form of security patterns. These security patterns provide general

access control policies within the organization. Once the patterns are specified, the transformation rules are defined, depicting how to transform a logical model, based on the pattern, into a database schema. The artifacts created in this phase are reusable and may be applied to various applications.

The application level deals with the development of any application within the organization. In the analysis phase of the development of an application, two models are defined, according to the FOOM methodology [6]: a conceptual data model in the form of an initial class diagram, and a functional model in the form of extended use cases. Then, the security constraints, regarding authorization to access the database, are specified. In the design phase, the artifacts from the preparation and analysis stage are used to refine the data model and create a secure data model. Next, the secure data model is verified. If the verification fails, the data model is refined until it adheres to the rules of the security patterns. In the implementation phase, the secure data model is transformed into a secure database schema. This process is defined in the transformation rules, guided by the knowledge captured in the security patterns.

3 The Security Modeling Tool

3.1 Organizational Level - The Preparation Phase

During the preparation phase, security patterns along with their transformation rules are specified. These patterns will serve as guidelines for application developers as well as a verification template, and they provide the infrastructure for the transformation process. The transformation rules depict how an application model will be transformed into a secure database schema.

Security Pattern Specification. Similarly to the classical pattern approach, security patterns are specified in a structured form. The standard template aids designers, who are not security experts, to identify and understand security problems and solve them efficiently. In order to specify the patterns, we use a common template introduced by Schumacher [5]. The template consists of five main sections: *name, context, problem, solution,* and *consequence*. The *name*, *context*, *problem*, and *consequence* sections are documentation text files. They provide the *name* of the pattern, the *context* in which the security problem occurs, the description of the security *problem*, and the *consequences* of this solution. The *solution* section provides a generic solution to the problem. It is specified with a UML class diagram that provides the static structure of the solution. In addition, OCL constraints are used to provide additional information that is inexpressible in the diagrams. Fig. 2 presents the structure of a simple Role-Based Access Control (RBAC) pattern (upper side) and an example of an OCL rule (lower side). In this example, the OCL rule restricts the number of roles that can have the SYSDBA system privilege to one, and that is the DBA role. In case that a finer grained solution is required, OCL rules in the form of general templates [8] can be defined. These general templates are specified using the specific elements that were already defined by the class diagrams specifying the structure of the pattern. In the RBAC example, the *Role*, *ProtectedObject*, *accessType* are some of those elements. The templates are essentially exemplars of the desired output code with "blanks" that should be filled in with a value of an attribute. These "blanks" contain meta-code and

are delimited between "< >". After the missing values are inserted, a template engine is used to create the output code. Fig. 3 presents the instance level template that is used to specify access constraints on an instance of an object (or a row of a table in terms of relational database). The templates are used to specify fine grained access control policies during the application modeling. The developers need only to fill in the missing parameters that are inside the triangle brackets and do not need to write code in PL/SQL unless they want to express some complex constraint.

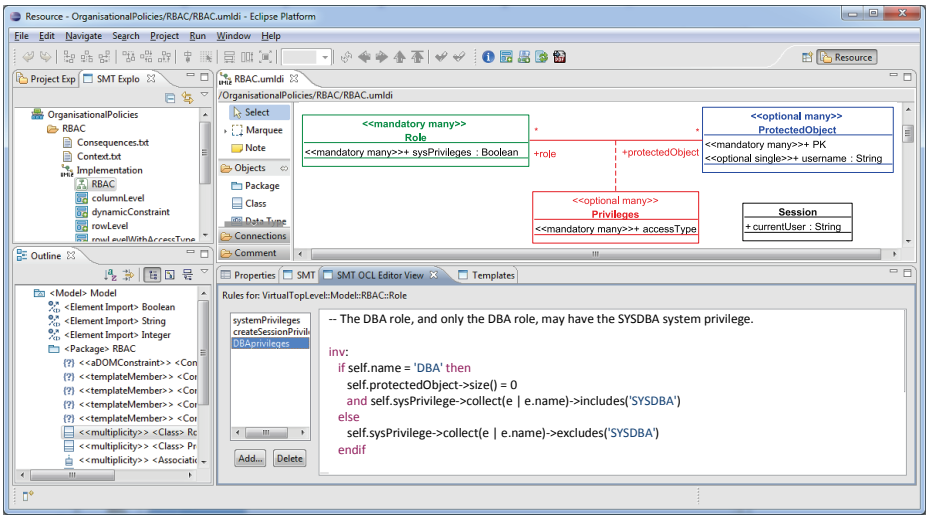

Fig. 2. Pattern specification window

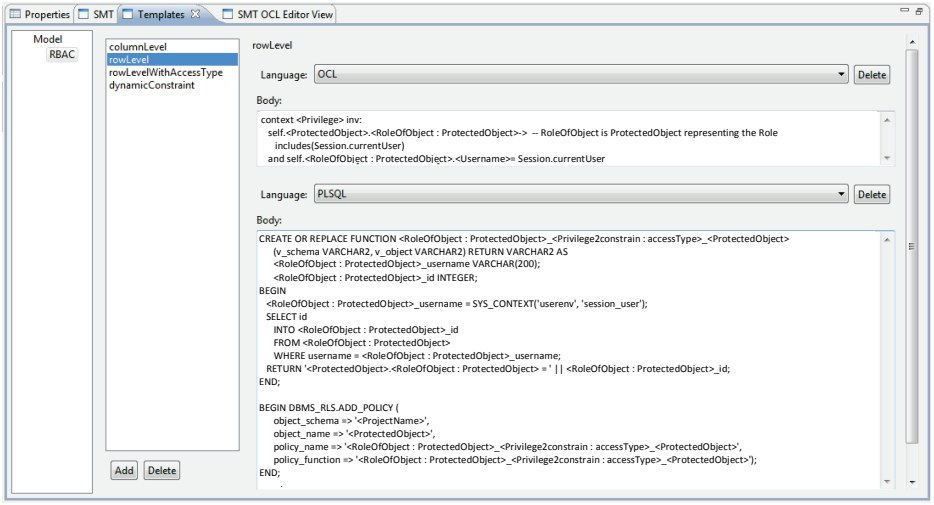

Fig. 3. The Instance (Row) Level Template

Transformation Rules Specification. To transform the UML class diagram into SQL code, we use the ATLAS Transformation Language (ATL) [3]. First, we transform the application model into a SQL application model, and then we translate it to SQL code. Fig. 4 shows the *transformation* rule for *Privilege*.

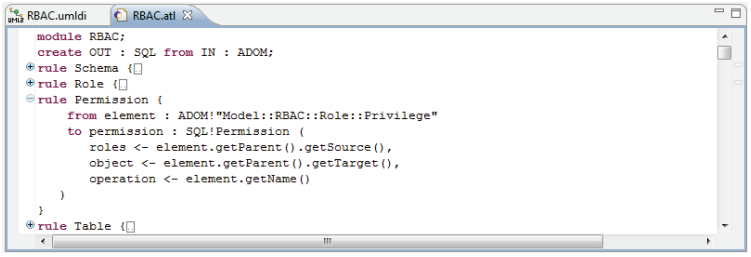

Fig. 4. ATL transformation code for the Privilege association class

3.2 Application Development Level

The Analysis Phase. The first task in the analysis phase is to create a conceptual data model based from the users' requirements. The conceptual data model is an initial class diagram that consists of data classes, their attributes and various types of relationships. Fig. 5 depicts the initial (UML) class diagram of a university registration system.

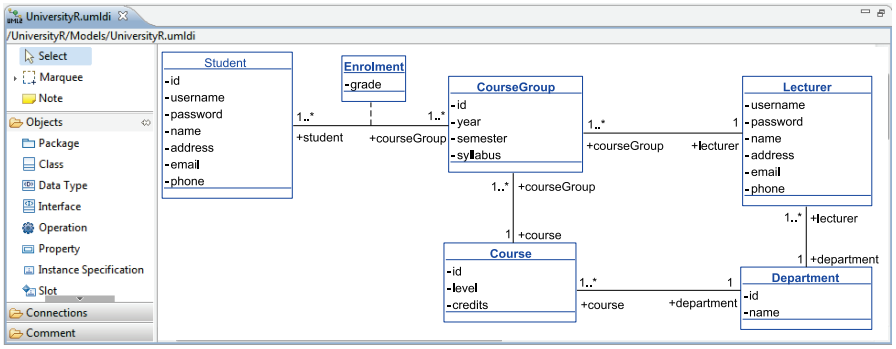

Fig. 5. An example of an initial class diagram

Following that, the functional model of the application is defined using extended use cases (EUC). An EUC is similar to a FOOM transaction [6] as it includes, besides the use case (UC) bubbles (i.e., functions), external/user entities and data classes. An external/user entity provides input data or obtains output information from the system. (It is different from an Actor in ordinary UC, which only signify who operates the use case.) Data classes, which are taken from the initial class diagram, are manipulated (i.e., retrieved or updated) by the functions of the EUC.

The EUCs enable the analyst to define, among other things, the roles of the various users of the EUC and their access privileges. As in ordinary UC, for every EUC diagram we also prepare a description. The template for an EUC description is

extended compared to an ordinary UC description, as it includes definitions of access authorization.

For each class included in an EUC we define: a) the roles, i.e. the authorized operators of the EUC; b) the type of access authorization (e.g., add, read, update or delete); and c) the attributes involved in that operation. Fig. 6 shows an example of an EUC with definition of access authorization. The right side shows a certain EUC diagram; the left side shows part of the EUC description; the bottom shows a part of the security specifications: in a form of a table we show, for each class that participates in the EUC, the security specifications.

Eventually, all the security specifications, defined for all the EUCs are aggregated in one table.

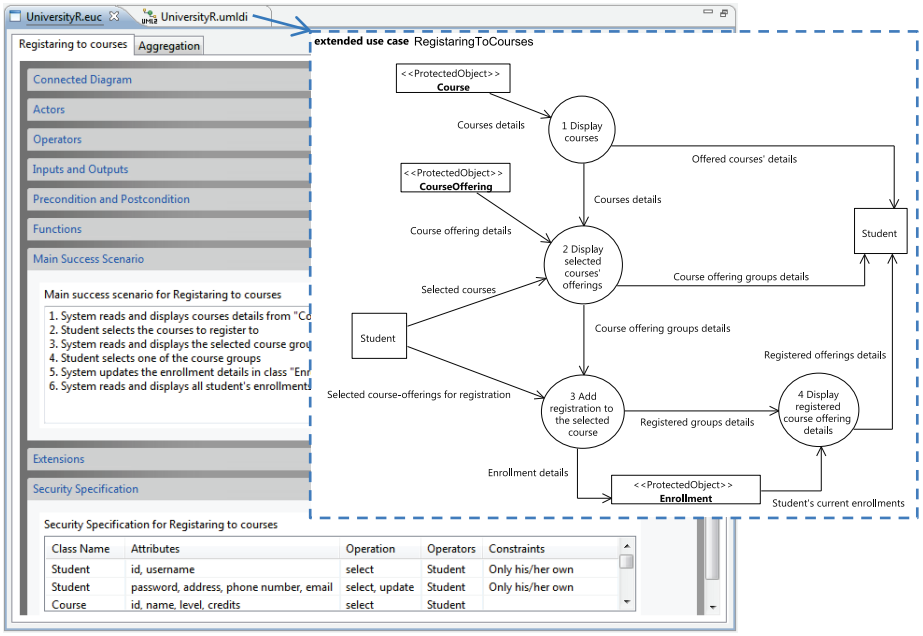

Fig. 6. Example of an EUC with definition of access authorization

The Design Phase. During this phase, the initial class diagram is refined, by the designer, to include the security specification, adhering with the organizational policies. Namely, the authorization rules are added to the initial class diagram as specified by the patterns, and the relevant elements are classified via stereotypes according to the predefined patterns, as presented in Fig. 7. During this task, additional changes to authorization rules may be applied and fine grained restrictions may be specified using the templates that were defined in the patterns. Fig. 8 illustrates the use on the instance (row) level template that was defined in Fig. 3. To use the template, the designer merely instantiates it and provides the missing parameters. In the SMT, these parameters are listed at the bottom of the view. The SMT also provides a preview of the templates after the missing parameters were specified, for instance in OCL and PL/SQL.

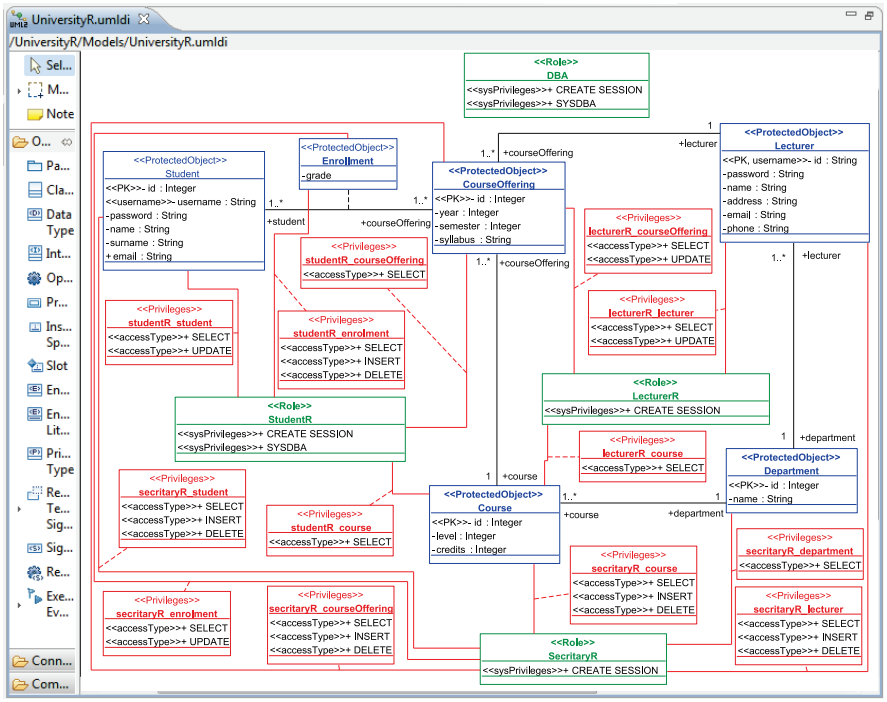

Fig. 7. An example of the RBAC-base refined data model

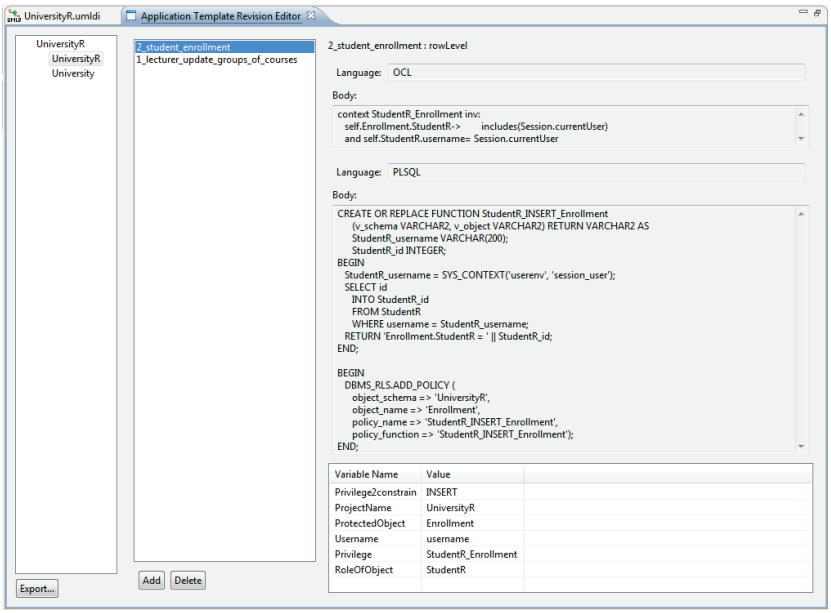

Fig. 8. An example of instance level (row) constraint

After creating a refined data model, we need to check if it adheres to the security policies as defined by the specified security patterns. The SMT provides automatic verification that is based on the ADOM validation algorithm. If the application is invalid, an error message is presented, explaining the verification errors. An example of two such errors are: 1) multiplicity error: access type is not specified to the *Privilege* class *StudentR_CourseOffering*, and 2) OCL error: *StudentR* role has *the SYSDBA* privilege.

The Implementation Phase. During this phase the transformation rules, which were defined during the preparation phase, are used to translate the verified application model into a database schema.

4 Summary

We have presented the Security Modeling Tool, which supports the development of secured database schemata upon the methodology we have developed. This tool utilizes security patterns for enforcing security on database application design. The tool guides developers on how to incorporate security aspects, in particular authorization, within the development process with pre-defined security patterns. It handles the specification and implementation of the authorization aspect from the early stages of the development process, leading to a more secure system design.

Currently, we are in a process of applying the methodology along with its supporting tool within an industrial environment. This will enable us to introduce improvements in the methodology and the tool. In future work, we plan to enrich the methodology and tool to support other non-functional requirements (e.g., in the security era it might include privacy, encryption, and auditing). In addition, we plan to apply the methodology to the code level, similarly to the way we apply it to database schemata; yet, we intend to incorporate behavioral specification as well.

References

1. Dhillon GS. Information Security Management: Global Challenges in the New Millennium. IGI Publishing (2001)
2. Eclipse Modeling Framework (2011). http://www.eclipse.org/modeling/emf/
3. Jouault F, Allilaire F, Bézivin J, Kurtev I. ATL: A model transformation tool. Science of Computer Programming. 72(1-2), 31--39 (2008)
4. Reinhartz-Berger, I., Sturm, A.: Utilizing Domain Models for Application Design and Validation. Information & Software Technology 51 (8), 1275--1289 (2009)
5. Schumacher, M.: Security Engineering with Patterns: Origins, Theoretical Models, and New Applications. Springer-Verlag New York, Inc., Secaucus (2003)
6. Shoval, P.: Functional and Object-Oriented Analysis and Design - An Integrated Methodology. IGI Publishing, Hershey (2007)
7. Standard Widget Toolkit (2011). http://www.eclipse.org/swt/
8. StringTemplate (2011). http://www.stringtemplate.org/