Tool Support for Enforcing Security Policies on Databases

Jenny Abramov[1,2], Omer Anson[2], Arnon Sturm[1], and 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,sturm,shoval}@bgu.ac.il,
oaanson@gmail.com

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 process. 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 paper aims at guiding and enforcing developers, in particular database designers, to deal with database authorization requirements from the early stages of the development process. In this paper we demonstrate how the Security Modeling Tool assists the various stakeholders in designing secure database code and describe the tool architecture.

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. Yet, these types of problems are usually ignored in the early stages of the development process.

In the last decade various methods were suggested to incorporate security aspects within the development process. Several UML security-related extensions were proposed, such as UMLsec [19] and SecureUML [17, 15]. Additionally, a security-oriented extension to the Goal-Driven Requirements Engineering methodology Tropos was proposed - Secure Tropos [18]. Mouratidis and Jurjens combined Secure Tropos and UMLsec [24] to create a structured methodology for secure software development that supports all software development phases. Fernández-Medina and Piattini [23] also proposed a method to design secure databases. Another approach for security specification is security patterns, which is based on the classic idea of design patterns introduced by the Gang of Four [20]. Security patterns were proposed to assist developers to handle security concerns and provide guidelines to be used from the early stages of the development lifecycle [5]. However, to successfully utilize a security pattern, there must be systematic guidelines supporting its application throughout the entire software development lifecycle. Such a methodology to build secure systems using patterns was presented by Schumacher et al. [25] and Fernandez et al. [16]. This methodology integrates security patterns into each one of the software development stages, and each stage can be tested for compliance with the principles presented by the patterns. A catalog of security patterns can help to define the security mechanisms at each architectural level and at each development stage. Hafner and Breu [21] proposed a model driven security methodology for service-oriented architectures. Other methodologies present the use of aspect-oriented software design to model security as separate aspects which would later be weaved within the functional model. For example, in [22] the authors propose to deal with access control requirements while utilizing UML diagrams.

The above studies, and other related studies (which are not referenced here due to space limit), mainly provide guidelines regarding the way security should be handled within certain stages of the software development process, or address specific aspects of security. To the best of our knowledge, no existing method provides a complete framework that both guides and enforces organizational security policies on a system design, and then generates executable code from that design.

To overcome these deficiencies, we have developed a methodology that enables organizations to specify their security policies in the form of security patterns, which will guide developers in the incorporation of these particular organizational security policies, as well as verifies their correct application. In addition, the methodology enables the developer to transform the result into code, based on the organizational policies. In this paper, we explicitly refer to the application of access control in databases.

The methodology incorporates ideas from two areas of expertise: in the area of *system development methodologies*, we adopt the principle of integrating data and functional modeling at the early stages of the development, according to the Functional and Object-Oriented Methodology (FOOM) [6]; 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 enables the modeling of security patterns and enforces their correct usage during application development. The knowledge captured in the security patterns is used to automatically verify that the application models are indeed secure with respect to the defined patterns. Having a verified model, a secure database code can be automatically generated.

SMT is an Eclipse plug-in and is based on existing frameworks such as the Eclipse Modeling Framework [2], which is used to interface with UML diagrams; and the Standard Widget Toolkit [7], which is used to provide additional graphical user interface where needed. 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, Section 4 elaborates on the SMT architecture and design, and Section 5 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 scope of the methodology in terms of the tasks to be performed in each phase (presented in round rectangle) and the generated artifacts (presented in rectangle). The preparation phase occurs at the organizational level, whereas the other three phases occur at the application development level.

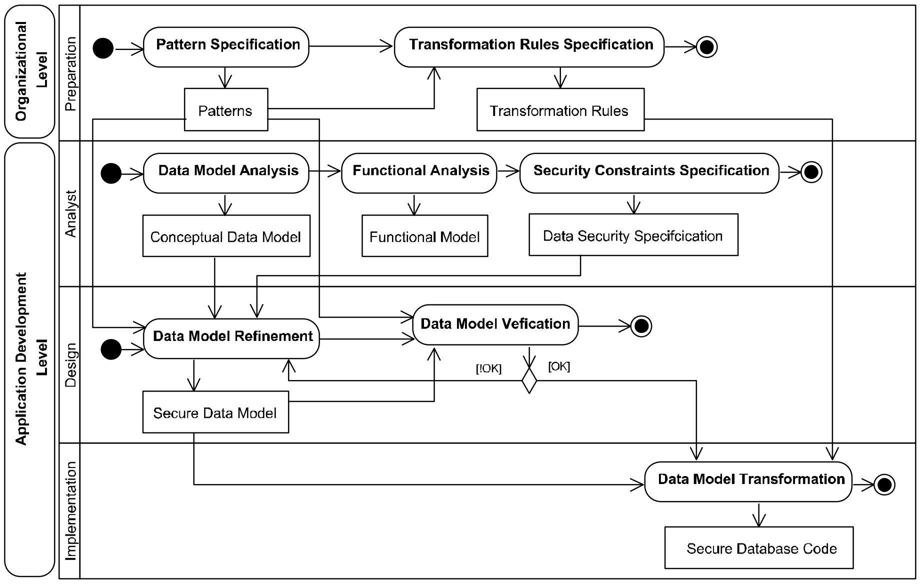


Fig. 1. Methodology overview

At the organizational level, in which the **preparation phase** takes place, we define organizational security policies in the form of security patterns. These security patterns present 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 code. The artifacts created in this phase are reusable and may be applied to various applications.

The application level deals with the development of different applications within the organization. In the **analysis phase** of the application development process, two models are defined, following 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 analyzed and specified in natural language. In the **design phase**, the artifacts from the preparation and analysis stage are used to refine the data model and enhance it with the definitions of the security patterns, in order to 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 with its access control specifications. This process is performed by executing the transformation rules specified in the preparation phase as part of the security patterns.

3 The Security Modeling Tool

3.1 Organizational Level - The Preparation Phase

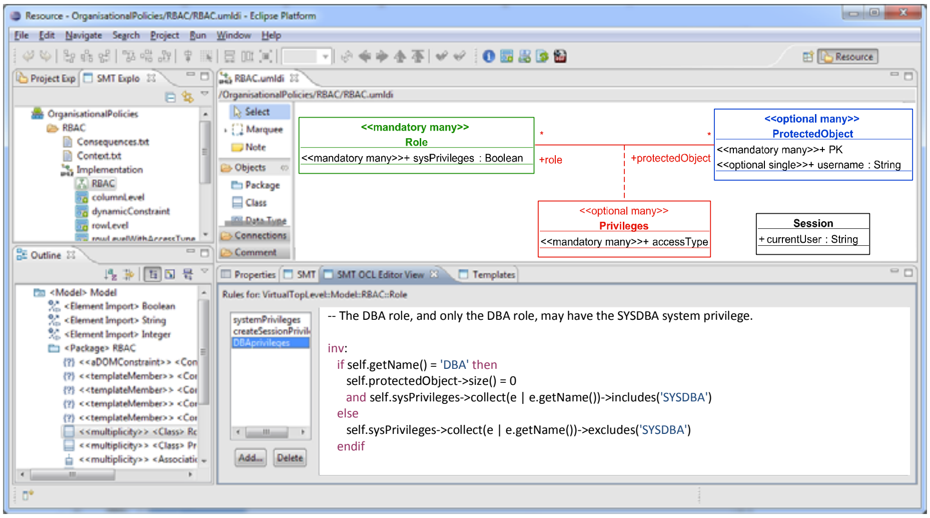
During the preparation phase, security patterns along with their transformation rules are specified. These patterns will serve as guidelines for application developers as well as a verification template. In addition, they provide the infrastructure for the transformation process.

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; the SMT provides a text editor to support the specification of these sections. 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. The SMT uses a UML editor that is based on TOPCASED [9]. Fig. 2 (upper side) presents the structure of a simple Role-Based Access Control (RBAC) pattern. In the described pattern, *Role* is akin to an external group of entities or users playing a specific role that needs to access the database. While applying or implementing this RBAC pattern, it is obligatory to define at least one *Role* as it is defined as a <<*mandatory*>> element. In addition, one can specify the system privileges assigned to some *Role* by using the *sysPrivileges* classification *ProtectedObject* is akin to a database table, where the *PK* classification is used to indicate the primary keys of the table. *Privileges* association class determines the schema object privileges of a *Role* with respect to a specific *ProtectedObject*. A class that is classified as *Privileges* must include at least one object privilege – *accessType*. Both *sysPrivileges* and *accessType* classifications are Boolean properties that should be assigned to TRUE in case a privilege is given.

In addition, OCL constraints are used to specify additional constraints that cannot be expressed via the diagrams. The SMT provides an OCL Editor to add constraints to the pattern. These OCL constraints are evaluated in the application layer during the verification in the design stage, rather than in the domain layer where they are defined. To enable this verification we had to define several operators, such as *getName()* or *getParent()*, that support metadata queries on the elements.

The lower part of Fig. 2 shows an example of an OCL rule. 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. Another example of such OCL rule is the following constraint that limits object privileges to `SELECT`, `INSERT`, `UPDATE` and `DELETE`:

```
context Privileges
inv: Set{'SELECT','INSERT','UPDATE','DELETE'}->
        includesAll(self.accessType->collect(e|e.getName()))
```



Rules for: VirtualTopLevel::Model::RBAC::Role

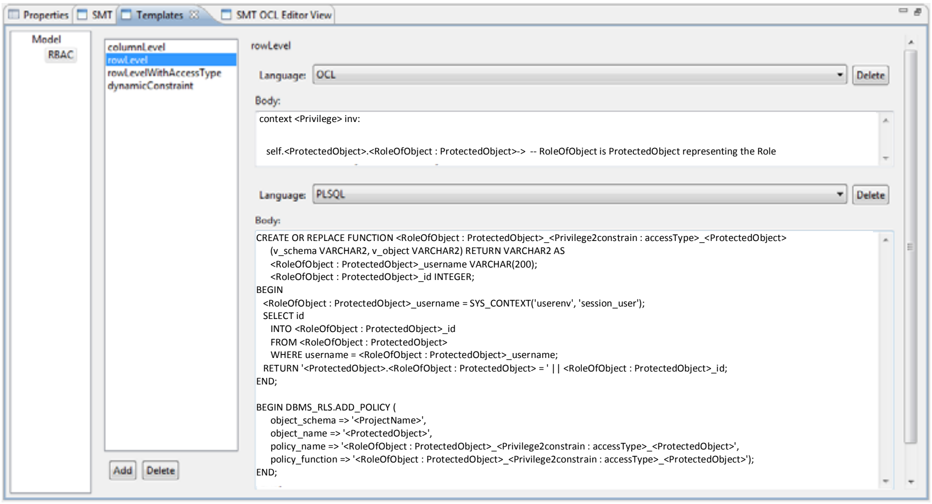
systemPrivileges
createSessionPrivil
DBAprivileges

```
-- The DBA role, and only the DBA role, may have the SYSDBA system privilege.

inv:
  if self.getName() = 'DBA' then
    self.protectedObject->size() = 0
    and self.sysPrivileges->collect(e | e.getName())->includes('SYSDBA')
  else
    self.sysPrivileges->collect(e | e.getName())->excludes('SYSDBA')
  endif
```

Fig. 2. Pattern specification window

In case that a finer grained solution is required, SMT provides a method to define OCL rules in the form of general templates [8]. 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). These 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.

Once the *solution* (i.e., the pattern) is defined, the SMT automatically generates a UML profile, which will be used by the applications to classify security elements. In our case the profile will consist of the following: the stereotypes *Role* and *ProtectedObject* are created and are associated with the class meta element, the *Privilege* stereotype is associated with the association class meta element. The attributes of *sysPrivileges*, *accessType, PK,* and *username,* are created as stereotypes associated with the property meta element. The various OCL constraints are also transformed into the profile. Note that we did not associate any new notations for the profiles; rather we used the standard <<stereotype>> to add semantics to the application model elements.



```
context <Privilege> inv:


  self.<ProtectedObject>.<RoleOfObject : ProtectedObject>->  -- RoleOfObject is ProtectedObject representing the Role
```

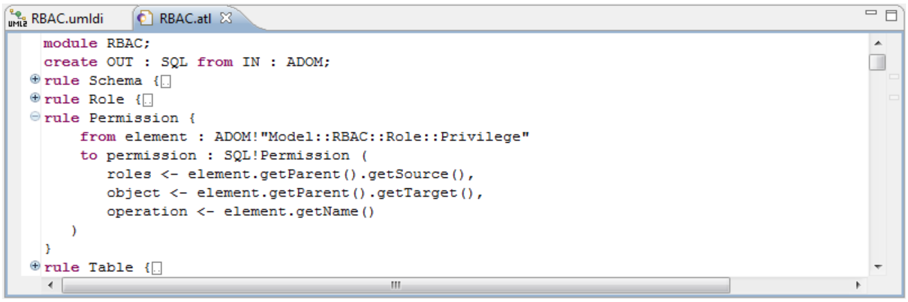
```
CREATE OR REPLACE FUNCTION <RoleOfObject : ProtectedObject>_<Privilege2constrain : accessType>_<ProtectedObject>
    (v_schema VARCHAR2, v_object VARCHAR2) RETURN VARCHAR2 AS
    <RoleOfObject : ProtectedObject>_username VARCHAR(200);
    <RoleOfObject : ProtectedObject>_id INTEGER;
BEGIN
  <RoleOfObject : ProtectedObject>_username = SYS_CONTEXT('userenv', 'session_user');
  SELECT id
    INTO <RoleOfObject : ProtectedObject>_id
    FROM <RoleOfObject : ProtectedObject>
    WHERE username = <RoleOfObject : ProtectedObject>_username;
  RETURN '<ProtectedObject>.<RoleOfObject : ProtectedObject> = ' || <RoleOfObject : ProtectedObject>_id;
END;

BEGIN DBMS_RLS.ADD_POLICY (
    object_schema => '<ProjectName>',
    object_name => '<ProtectedObject>',
    policy_name => '<RoleOfObject : ProtectedObject>_<Privilege2constrain : accessType>_<ProtectedObject>',
    policy_function => '<RoleOfObject : ProtectedObject>_<Privilege2constrain : accessType>_<ProtectedObject>');
END;
```

Fig. 3. The Instance (Row) Level Template

Transformation Rules Specification: To transform an application model (UML class diagram) into SQL code, we use the ATLAS Transformation Language (ATL) [3]. These transformation rules are generic and refer to all applications. The ATL rules specify how the application elements should be transformed into SQL elements. So, applications are transformed to SQL model instantiating an SQL meta-model provided by SMT. Then, the SQL model (created by the ATL transformation) is automatically converted to SQL code by the SMT. Fig. 4 shows the transformation rule for *Privilege*.

3.2 Application Development Level

To demonstrate the use of SMT at the application development level, we use a simple university system, which enables to register students to courses, update student details, assign grades, etc. Naturally, each system operator has different privileges.



```
module RBAC;
create OUT : SQL from IN : ADOM;
rule Schema {
rule Role {
rule Permission {
    from element : ADOM!"Model::RBAC::Role::Privilege"
    to permission : SQL!Permission (
       roles <- element.getParent().getSource(),
       object <- element.getParent().getTarget(),
       operation <- element.getName()
    )
}
rule Table {
```

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

The Analysis Phase: The first task in the analysis phase is to create a conceptual data model 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.

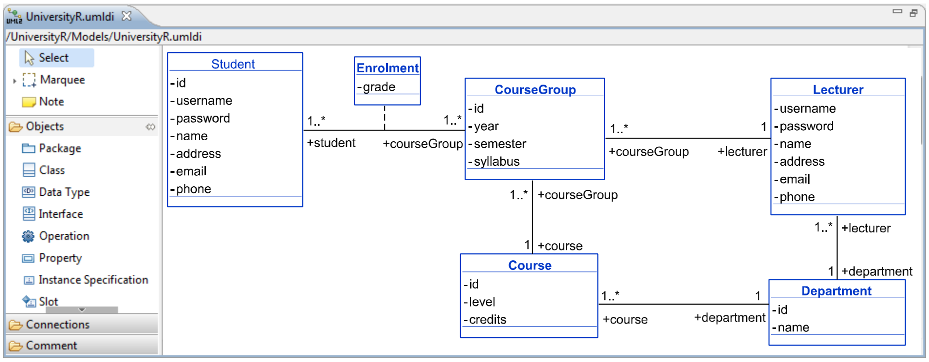


Fig. 5. An example of an initial class diagram

Next, the functional model of the application is defined using **extended use cases** (EUC). A EUC is similar to a FOOM transaction [6]; it includes, besides the functions of the UC, also 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 use cases, 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. As in ordinary use-cases, for every EUC diagram we also prepare a description. The template for a EUC description is extended compared to an ordinary UC description, as it includes definitions of access privileges.

Later on in the development process, for each class included in a EUC the developer defines: a) the authorized operators (i.e., roles) of the EUC; b) the type of access privilege (e.g., add, read, update or delete); and c) the attributes involved in that operation. Fig. 6 shows an example of a EUC diagram that is supported by the EUC editor. A *Student* is an external entity which provides inputs and gets outputs, the *Course*, *Course Offering* and *Enrollment* are classes, and *Display courses*, *Display selected courses' offerings*, and *Add registration to selected course* are functions. The EUC editor extends the TOPCASE use case diagram notations to support the new elements, i.e., classes and the different types of links.

Fig. **7** shows part of the EUC description that is supported by the EUC Analysis Editor. At the bottom of Fig. **7**, the *security specifications* section is presented in a form of a table, where for each class that participates in the EUC the access control privileges are specified. Eventually, all the security specifications, defined for all the EUCs are aggregated in one table.

It should be noted that EUC diagrams also serve as the core functional/behavioral model of the application, and can be used for the generation of the input and output forms and reports, as well as skeleton of the code.

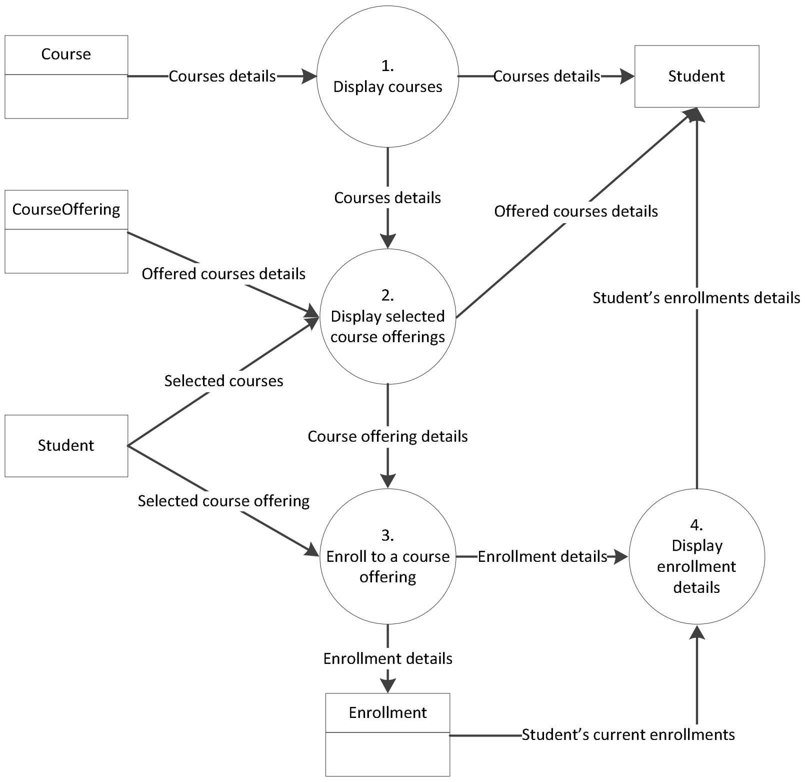
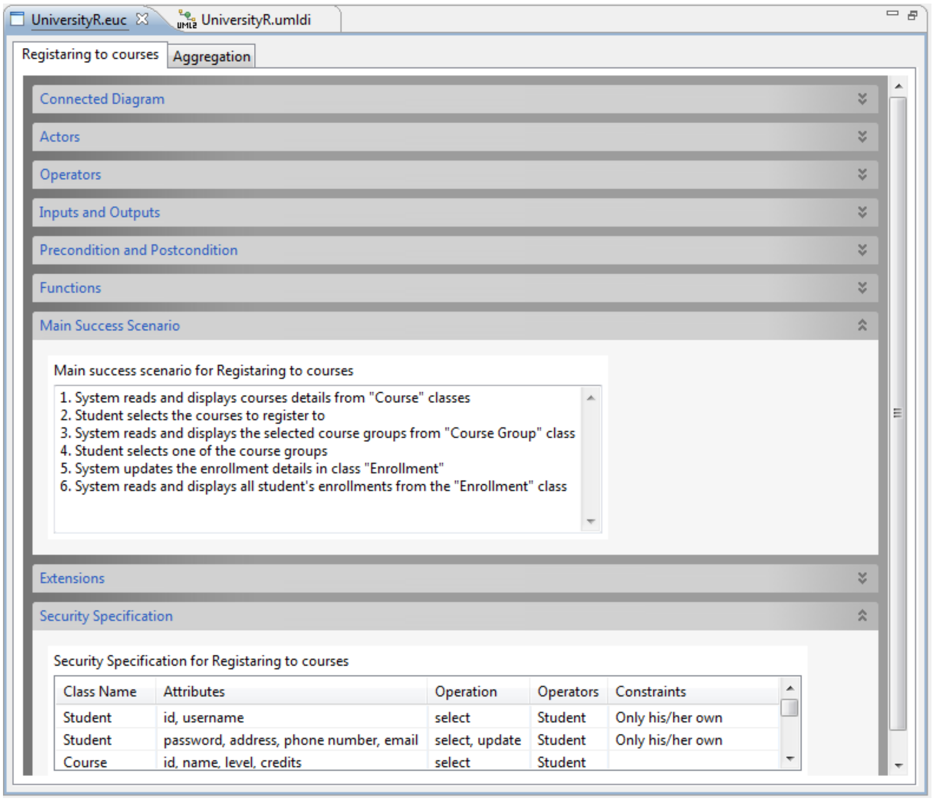


Fig. 6. An example of a EUC diagram

The Design Phase: During this phase, the initial class diagram is refined by the designer, to include the security specification. The SMT allows the designer to specify which security patterns are used in the application. Then, the various elements that appear in the initial class diagram are classified according to the security patterns defined in the preparation stage. Technically, this is done by assigning the stereotypes from the security pattern profile that was created when the security pattern was finalized. Then, the SMT allows the designer to select stereotypes for each element according to the applied patterns, and the element type. **Fig. 8** presents the refined data model of the university application. In that figure, the relevant classes are associated with the *Role* and *ProtectedObject* stereotypes and new *Privileges* classes are introduced. These include the names of the *accessType* attributes as set by the OCL constraint, and the initial values of these attributes (which are not shown visually, yet they are part of the model); in the example their values are True.



Main success scenario for Registaring to courses

1. System reads and displays courses details from "Course" classes
2. Student selects the courses to register to
3. System reads and displays the selected course groups from "Course Group" class
4. Student selects one of the course groups
5. System updates the enrollment details in class "Enrollment"
6. System reads and displays all student's enrollments from the "Enrollment" class

Security Specification for Registaring to courses

Class Name	Attributes	Operation	Operators	Constraints
Student	id, username	select	Student	Only his/her own
Student	password, address, phone number, email	select, update	Student	Only his/her own
Course	id, name, level, credits	select	Student	

Fig. 7. An example of a EUC description

During the design phase, additional changes to authorization rules may be applied and fine grained restrictions may be specified using the templates that were defined in the patterns. The templates are instantiated using the Template Editor. Fig. 9 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. The Template Editor lists the missing parameters at the bottom. The SMT also provides a preview of the templates after the missing parameters were specified.

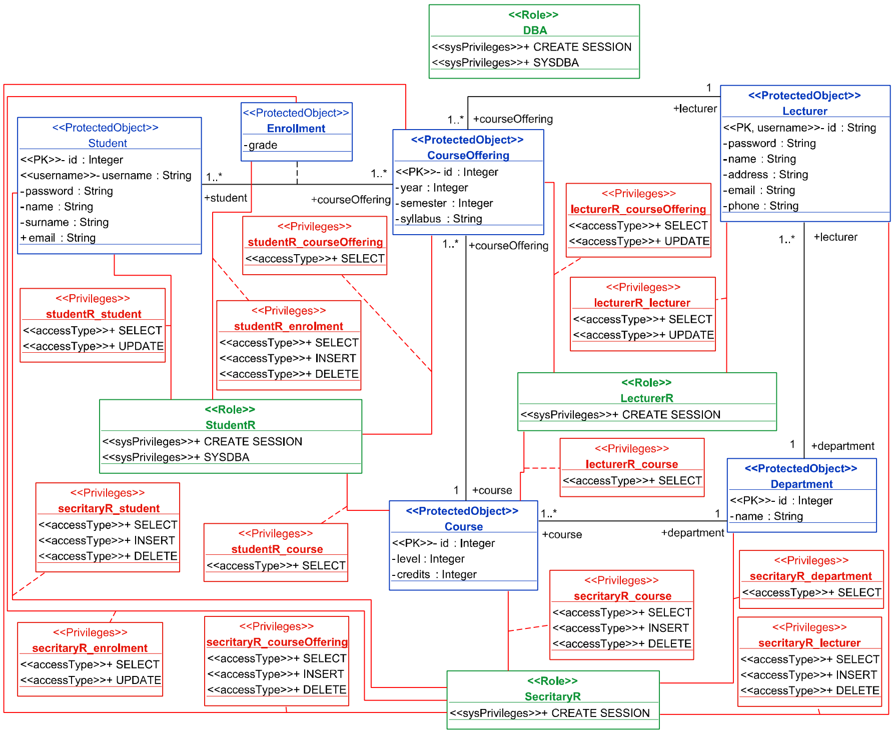
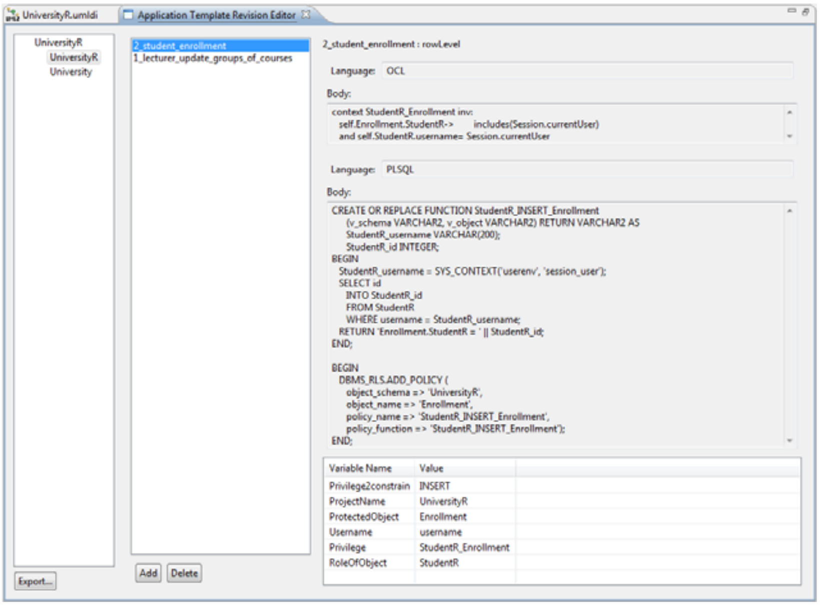


Fig. 8. An example of the RBAC-base refined data model



2_student_enrollment : rowLevel

Language: OCL

Body:

```
context StudentR_Enrollment inv:
  self.Enrollment.StudentR->       includes(Session.currentUser)
  and self.StudentR.username= Session.currentUser
```

Language: PLSQL

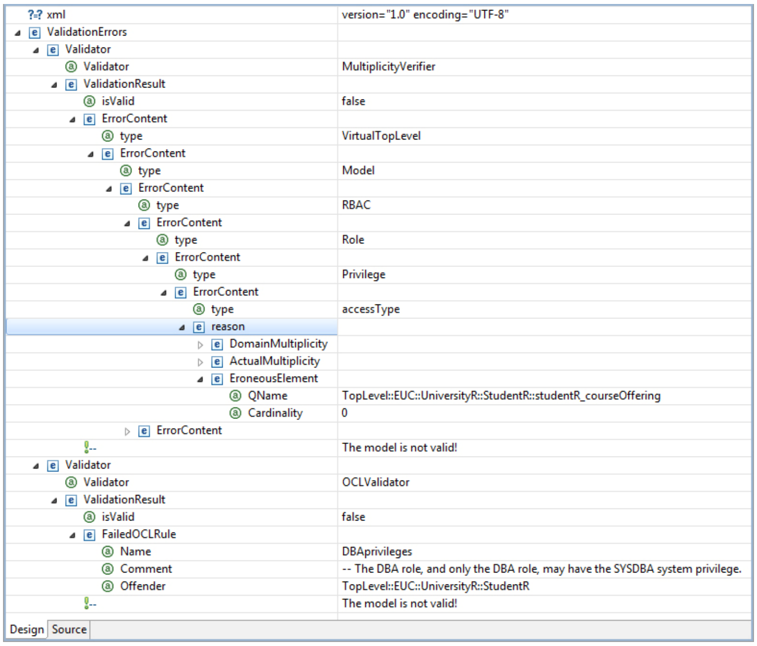
Body:

```
CREATE OR REPLACE FUNCTION StudentR_INSERT_Enrollment
    (v_schema VARCHAR2, v_object VARCHAR2) RETURN VARCHAR2 AS
    StudentR_username VARCHAR(200);
    StudentR_id INTEGER;
BEGIN
  StudentR_username = SYS_CONTEXT('userenv', 'session_user');
  SELECT id
    INTO StudentR_id
    FROM StudentR
    WHERE username = StudentR_username;
  RETURN 'Enrollment.StudentR = ' || StudentR_id;
END;

BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema => 'UniversityR',
    object_name => 'Enrollment',
    policy_name => 'StudentR_INSERT_Enrollment',
    policy_function => 'StudentR_INSERT_Enrollment');
END;
```

Variable Name	Value
Privilege2constrain	INSERT
ProjectName	UniversityR
ProtectedObject	Enrollment
Username	username
Privilege	StudentR_Enrollment
RoleOfObject	StudentR

Fig. 9. An example of instance level (row) constraint in OCL and PL/SQL

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. This verification is essentially a conformance checking with respect to the relevant patterns; it includes checking the number of elements, as depicted in [4], their types, and the available OCL constraints. If the application is invalid, an error message, like the one appears in Fig. 10, is presented, explaining the verification errors. In that example there are two errors: 1) multiplicity error: access type is not specified to the *Privilege* class *StudentR_CourseOffering*; 2) OCL error: *StudentR* role has the *SYSDBA* privilege.



```
?-? xml                                   version="1.0" encoding="UTF-8"
◢ e ValidationErrors
  ◢ e Validator
      a Validator                         MultiplicityVerifier
    ◢ e ValidationResult
        a isValid                         false
      ◢ e ErrorContent
          a type                          VirtualTopLevel
        ◢ e ErrorContent
            a type                        Model
          ◢ e ErrorContent
              a type                      RBAC
            ◢ e ErrorContent
                a type                    Role
              ◢ e ErrorContent
                  a type                  Privilege
                ◢ e ErrorContent
                    a type                accessType
                  ◢ e reason
                    ▷ e DomainMultiplicity
                    ▷ e ActualMultiplicity
                    ◢ e EroneousElement
                        a QName           TopLevel::EUC::UniversityR::StudentR::studentR_courseOffering
                        a Cardinality     0
            ▷ e ErrorContent
      !--                                 The model is not valid!
  ◢ e Validator
      a Validator                         OCLValidator
    ◢ e ValidationResult
        a isValid                         false
      ◢ e FailedOCLRule
          a Name                          DBAprivileges
          a Comment                       -- The DBA role, and only the DBA role, may have the SYSDBA system privilege.
          a Offender                      TopLevel::EUC::UniversityR::StudentR
      !--                                 The model is not valid!
Design | Source
```

Fig. 10. An example of an error massage

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 database code. Fig.11 presents the generated SQL commands for the *Student* role and a sample of the SQL fine-grained code for the university application.

The artifacts produced in the organizational level, and the artifacts leading to the implementation in the application development level, can be exported as documentation in a PDF file.

More details on the pattern-based approach that is applied as part of the methodology can be found in [14].

```
-- Role creation
CREATE ROLE STUDENT;
-- Granting privileges to Student
GRANT CREATE SESSION TO STUDENT;
GRANT SELECT ON COURSE_OFFERING TO STUDENT;
GRANT SELECT ON COURSE TO STUDENT;
GRANT SELECT, INSERT, DELETE ON ENROLLMENT TO STUDENT;
GRANT SELECT, UPDATE ON STUDENT TO STUDENT;
-- Instance level template transformation
-- Students can update only their personal information:
CREATE FUNCTION STUDENT_STUDENT_UPDATE
   (SCHEMAV VARCHAR2, OBJ VARCHAR2) RETURN VARCHAR2 AS
BEGIN
  IF (NOT DBMS_SESSION.IS_ROLE_ENABLED('STUDENT')) THEN
    RETURN NULL;
  END IF;
  RETURN 'username = ' || SYS_CONTEXT('USERENV', 'SESSION_USER');
END;
BEGIN DBMS_RLS.add_policy(
  object_schema   => 'UNIVERSITY',
  object_name     => 'STUDENT',
  policy_name     => 'STUDENT_STUDENT_UPDATE ',
  policy_function => 'STUDENT_STUDENT_UPDATE ',
  statement_types => 'UPDATE',
  update_check    => TRUE);
END;
```

Fig. 11. A sample of the generated SQL commands for the university application

4 SMT Architecture and Design

In this section, we discuss the implementation details of SMT. We first introduce the technologies on which SMT is based, as well as the reasons for choosing them. Then, we elaborate on the specific components developed within SMT, i.e., the different editors and the ADOM library. Finally, we describe how the SMT components interact with each other. Fig. 12 shows the components that SMT uses (marked in a broken line), and the components that were developed internally (marked in a solid line). The figure also shows the dependencies among the various components, which are organized as layer of dependencies. Note that the integration of all these components is done using the plug-ins facilities of the Eclipse framework. In short, at the **organization level**, in order to specify the pattern, SMT uses *text editors* for the pattern description, *UML editor* for the specification of the structure of the pattern along with the ADOM library, *OCL editor* to specify the constraints on the structure of the pattern, and *Templates editor* to specify the fine grain templates. Then, for specifying the pattern transformation we use ATL. At the **application development level**, the *UML and EUC editors* are used to specify the different application diagrams, *EUC analysis editor* to define the textual description of the EUC, the *ADOM library* to refine and verify the application data model by the pattern, the *Dresden OCL* to verify that the OCL constraints hold, and finally, ATL to transform the application model into code.

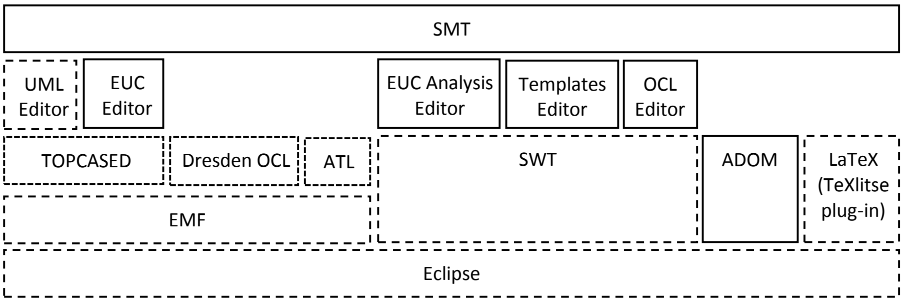


Fig. 12. Overview of the SMT components

4.1 Eclipse

SMT and all its dependencies rely on the framework provided by Eclipse [10]. It was chosen for the following reasons: (a) it is an extensible platform; (b) it is open source, meaning that the source code can be used as documentation and aid in finding errors; and (c) the Eclipse framework has been extended to support many technologies, including UML by TOPCASED [9] and OCL [11].

4.2 EMF

Much of the SMT's functionality relies on modeling capabilities. This functionality is provided by the Eclipse Modeling Framework (EMF) project [2], which is a modeling framework and code generation facility for building tools and other applications based on a structured data model.

EMF is a very mature modeling framework, providing a complete toolset for working in a model-oriented context. Additionally, it is available as an Eclipse extension, making it highly suitable for our needs. Many model-oriented features in Eclipse, and all such features used by SMT, rely on EMF. Note also that Eclipse's UML library too is based on EMF.

4.3 TOPCASED UML Editor

TOPCASED [9] is an Eclipse plug-in that provides a simple, extendible graphical modeling framework. TOPCASED also provides a UML editor as one of its sub-projects. Additionally, this framework may be extended to provide graphical editors to other diagram types. This capability was used within the general UML editors and to create and provide a new editor for the EUC diagrams. TOPCASED relies on EMF for both persistency and representation. Therefore, other modules based on EMF may interact with the artifacts generated by TOPCASED without the need for a new API or adaptor.

4.4 Dresden OCL

Dresden OCL [11] provides a set of tools to parse and evaluate OCL constraints on various models like UML, EMF, and Java. Furthermore, it provides tools for Java/AspectJ and SQL code generation. The tools of Dresden OCL can be either used as a library for other project or as a plug-in project that extends Eclipse with OCL support. In the case of SMT, the Dresden OCL library is used to enforce OCL constraints. As EMF provides only syntactical solutions, complex semantic solutions are beyond the scope of the EMF project. In SMT, we found that OCL may be specified using EMF's implementation of OCL; however EMF was not flexible enough to allow us to interpret and enforce the OCL rules on its own, and so the Dresden OCL library was used.

4.5 ATL

ATLAS Transformation Language (ATL) [12] is a model transformation language and toolkit. In the field of Model-Driven Engineering (MDE), ATL provides ways to produce a set of target models from a set of source models. SMT uses ATL to define a transformation from a pattern-based application design to its equivalent in SQL.

4.6 SWT

The Standard Widget Toolkit (SWT) [13] is an open source, graphical widget toolkit used to develop user interfaces in Java. It is a pre-built part of the Eclipse framework, which allows user interfaces implemented in SWT to be integrated more natively into Eclipse than other Java GUI toolkit. For this reason, user interfaces designed for SMT were implemented using SWT rather than a different widget framework.

4.7 The ADOM Library

The relationship between the organizational security pattern models and the application models within SMT are provided by the ADOM library. As we plan to adopt ADOM in various modeling notations (e.g., class and sequence diagrams), it was designed to be language independent. The ADOM library is separated into two main categories to allow for implementation in various languages and environments: language independent code, and language dependent code. That library also implements the ADOM validation algorithm.

Language independent code makes no assumptions on the used language beyond what is defined by ADOM. That is the multiplicity indicator which is defined as a UML profile. That part of the tool has three sections which are relevant to SMT:

a) The **abstract data-structure,** which provides a tree-like structure of model elements;
b) The **element options**, which allow ADOM elements to be extended to include data and functionality deemed necessary by the developer of the library's extension. For instance, the multiplicity validation algorithm extends each ADOM element to contain required and actual multiplicity. The OCL validation algorithm extends each ADOM element to contain any number of OCL constraints, which must be confirmed in order for the validation to succeed.
c) The **validation algorithms.** This is also a pluggable mechanism, used to allow library extension developers to provide validation algorithms on ADOM applications in reference to their ADOM domains. Usually, validation algorithms also provide element options, providing them with additional data necessary to perform the validation algorithm, such as multiplicity and OCL constraints as stated before. Note that these parts were reused in other ADOM-related projects.

The language dependent code has two sections relevant to SMT. The first section is an implementation of the abstract data-structure described in the language independent part. This implementation may be used by other components to have direct access to the underlying data-structures of the language in use, and retrieve necessary data.

4.8 Document Generation

The document generation facilities of SMT are provided using LaTeX. LaTeX is a high-quality typesetting system that includes features designed for the production of technical and scientific documentation. SMT generates a LaTeX document, which is then processed and generates an output file in PDF. The SMT enables the document generation of both organizational policies and application specification.

5 Summary

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

In this paper we demonstrated the application of an access control policy (RBAC) over a database. We also implemented other policies such as DAC and MAC using the same methodology. In addition, we are in the process of using the same mechanisms to implement patterns other than access control to other software layers besides the database. Currently, we are in a process of applying the methodology along with its supporting tool in 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 security requirements (e.g., privacy, encryption, and auditing). In addition, we plan to further extend the methodology to deal also with the behavioral specification of applications, in addition to its application in structural specification.

References

1. Dhillon, G.S.: 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. 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/`
9. TOPCASED (2011), `http://www.topcased.org/`
10. Eclipse (2011), `http://www.eclipse.org/`
11. Dresden OCL Toolkit (2011), `http://www.dresden-ocl.org/index.php/DresdenOCL`
12. ATL (2011), `http://eclipse.org/atl/`
13. Standard Widget Toolkit (2011), `http://www.eclipse.org/swt/`
14. Abramov, J., Sturm, A., Shoval, P.: A Pattern Based Approach for Secure Database Design. In: Salinesi, C., Pastor, O. (eds.) CAiSE Workshops 2011. LNBIP, vol. 83, pp. 637–651. Springer, Heidelberg (2011)
15. Basin, D., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Transaction on Software Engineering and Methodologies 15(1), 39–91 (2006)
16. Fernandez, E.B., Larrondo-Petrie, M.M., Sorgente, T., VanHilst, M.: A methodology to develop secure systems using patterns. In: Mouratidis, H., Giorgini, P. (eds.) Integrating Security and Software Engineering: Advances and Future Vision. IDEA Press (2006)
17. Lodderstedt, T., Basin, D., Doser, J.: SecureUML: A UML-Based Modeling Language for Model-Driven Security. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 426–441. Springer, Heidelberg (2002)
18. Mouratidis, H., Giorgini, P.: Secure Tropos: a Security-Oriented Extension of the Tropos Methodology. International Journal of Software Engineering and Knowledge Engineering 17, 285–309 (2007)
19. Jurjens, J.: Secure Systems Development with UML. Springer (2005)
20. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Professional (1995)
21. Hafner, M., Breu, R.: Security Engineering for Service oriented Architectures. Springer (2009)
22. Ray, I., France, R.B., Li, N., Georg, G.: An aspect-based approach to modeling access control concerns. Information & Software Technology 46, 575–587 (2004)
23. Fernández-Medina, E., Piattini, M.: Designing secure databases. Information & Software Technology 47(7), 463–477 (2005)
24. Mouratidis, H., Jurjens, J.: From goal-driven security requirements engineering to secure design. International Journal on Intelligent Systems 25(8), 813–840 (2010)
25. Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns: Integrating Security and Systems Engineering. John Wiley & Sons (2006)