Journal of Universal Computer Science, vol. 15, no. 15 (2009), 2957-2980
submitted: 1/2/09, accepted: 29/8/09, appeared: 1/9/09 © J.UCS

ModelSec: A Generative Architecture for Model-Driven Security

Óscar Sánchez, Fernando Molina
Jesús García-Molina, Ambrosio Toval
(University of Murcia, Murcia, Spain
osanchez@um.es, fmolina@um.es, jmolina@um.es, atoval@um.es)

Abstract: Increasingly, the success of software systems depends largely on how their security requirements are satisfied. However, developers are challenged in implementing these requirements, mainly because of the gap between the specification and implementation, and the technical complexities of the current software infrastructures. Recently, Model-Driven Security has emerged as a new software development area aimed at overcoming these difficulties. This new paradigm takes advantage of the benefits of the model driven software development techniques for modeling and implementing security concerns. Following this trend, this paper proposes a model driven security approach named ModelSec that offers a generative architecture for managing security requirements, from the requirement elicitation to the implementation stage. This architecture automatically generates security software artifacts (e.g. security rules) by means of a model transformation chain composed of two-steps. Firstly, a security infrastructure dependent model is derived from three models, which express the security restrictions, the design decisions and the information needed on the target platform. Then, security software artifacts are produced from the previously generated model. A Domain-Specific Language for security requirements management has been built, which is based on a metamodel specifically designed for this purpose. An application example that illustrates the approach and the Eclipse tools implemented to support it are also shown.

Key Words: Requirements Engineering, Requirements Metamodelling, Model Driven Engineering, Model Driven Security

Categories: D.2.1

1 Introduction

Security is a crucial aspect in current software systems. Thus, security requirements must be adequately considered in all the phases of software development, from the consideration of requirements to the system implementation [Fernández-Medina et al., 2009, Haley et al., 2008]. To address the complex issue of developing secure systems which satisfy the desired security requirements, several model-based development approaches have recently appeared [Basin et al., 2006, Jurjens, 2003, Reznik et al., 2007, Lang and Schreiner, 2008]. Models have been traditionally used in Software Engineering as an abstraction mechanism for the analysis and design of software systems. For many years, their usefulness was mainly focused on documenting and considering the system to be built but, in this decade, they have become first-class software components, since source code can be generated from models specified by well-defined modeling languages.

Model-driven Engineering (MDE) [Selic, 2008] has emerged as a new area in software engineering, whose goal is the definition of theories, methods, techniques and tools for applying this model-based development paradigm. The Model Driven Architecture (MDA) initiative [OMG, 2003], promoted by the OMG, is the most popular MDE approach, although there are others such as Domain-Specific Development or Software Factories. Model driven approaches are based on three main principles: i) The creation of modeling languages (also named Domain Specific Languages, DSL) by applying metamodelling concepts, ii) The use of these languages to model different aspects of a software system, and iii) The automatic processing of the models built, by means of model transformations in order to generate software artifacts (e.g. source code) that will be part of the final application. On the other hand, most MDE tools are currently integrated in the Eclipse platform [Eclipse, 2008a] that provides implementations of the OMG specifications, for example, the Ecore metamodelling language. Besides, the definition of new domain specific languages is a more widely used practice than building UML profiles since the profiling mechanism allows for a limited extension form [Kelly and Tolvanen, 2008].

Two seminal works [Jurjens, 2003, Basin et al., 2006] laid the foundations for the application of the model-based development to the building of secure systems. Firstly, Jurjens proposes UMLSec as an UML extension aimed at expressing and evaluating specifications for vulnerabilities using formal semantics. Later, Basin et al. show how the MDE approach can be specialized to deal with security needs, namely Model-Driven Security (MDS). They illustrate the feasibility of the new MDS approach by using models to create software artifacts related to role-based access control infrastructure. A UML profile named SecureUML was built to support visual modeling of access control requirements. It is worth noting that UMLSec has evolved into an MDS approach [Jurjens et al., 2008].

In this paper we present an MDS approach, named ModelSec, which is based on current MDE practices and tools. ModelSec proposes a generative architecture for automatically generating security software artifacts (e.g. rules implementing security policies or security code for a database) from security models. This architecture is based on a model transformation chain composed of two steps. Firstly, a model, dependent on the security infrastructure, is derived from three models, expressing the security requirements, the design decisions made for implementing them, and the information needed on the target platform. Instead of UML profiles, a graphical DSL for expressing security requirements models has been defined, which conforms to a metamodel that includes the more common concepts related to the security aspects of a software system. In the second step, a model-to-code transformation generates software artifacts from the previously generated model.

ModelSec provides a significant advantage compared to the existing approaches: the proposed process enables developers to apply a systematic process. Modeling security requirements are separated from representing the design decisions for their implementation and from specifying the information related to the target platform needed for generating software. To accomplish this, a security model is created at the analysis, design and implementation stages of the development process. A novel feature of ModelSec is the possibility of modeling design choices, so that the developers have to specify in separate models *what* security requirements must be satisfied and *how* they are implemented. On another hand, ModelSec is a general MDS approach, and not centered on any specific aspect of security (e.g. access control). To achieve this, the DSL defined for expressing security requirement models allows the representation of seven different security aspects, and can also be easily extended in order to consider any new kinds of security requirements. Besides, the gap between the security requirements and the implementation code is reduced by introducing an intermediate model, which is the target platform specific model mentioned above.

The approach and the tools implemented on Eclipse to support it will be illustrated through an example of an application for medical information management, which shows how code for Oracle Label Security [ORACLE, 2008] and XACML [OASIS, 2008] policies can be generated from security models. It is important to note that we show how ModelSec enables us to deal with two different security aspects, but that other aspects could also be addressed in a similar way.

The remainder of the paper is organized as follows. Section 2 presents an overview of the proposed MDE. Section 3 shows our proposal for security requirement metamodelling. Sections 4 and 5 explain in detail the DSL implemented and how the architecture works as a whole. In Section 6, an example that further illustrates the use of our approach and its automatic support is shown. Section 7 analyses some related work. Finally, in Section 8, the main conclusions are drawn and further lines of research are outlined.

2 A model driven approach for security requirements

This section presents an overview of our understanding of how model-based development should be specialized in order to deal with security aspects. The proposed generative architecture in ModelSec is intended to support this vision. Figure 1 shows a schema of the designed process with the different models involved. As can be observed, the system security is represented by specific models which are separate from models concerning the rest of the requirements. At each stage of the development process a new security model is created, taking into account the model of the previous stage. These models are used to automatically generate a model dependent on the concrete platform chosen and, finally, code of the final application can be produced. Next, the purpose of each model and the relationships among them is described.

During the requirements analysis stage, the definition of two separate models is proposed. Firstly, the developer creates a model for capturing the requirements of the system. Then, security requirements are represented in a more detailed way using a different model. Both models conform to the metamodel described in Section 3, but while the *requirements model* is focused on the non-specific requirements of the system, the *security requirements model* uses the security concepts of the metamodel (e.g. assets or threats) in order to appropriately represent the security requirements. This metamodel comes with a DSL aimed at creating security requirement models. Commonly, use cases and conceptual models are used to represent functional requirements. The requirement models proposed are not intended to replace the textual descriptions of the use cases, but are complementary. These models are needed to integrate requirements in an MDE process for which the use cases templates, expressed in natural language, are not suitable.

On the other hand, domain or conceptual models defining the domain vocabulary are often represented by class models (e.g. UML class models). Since conceptual models include all the assets, and therefore, the assets that need to be secured, a relationship among *security requirements models* and *domain models* is established: an asset of a security model maps and a concept in a domain model. This mapping will be used in the generative architecture to automatically generate the instances of the assets from classes in the domain model.

When developers consider the system design, they must make design decisions about how to implement every security requirement. These decisions (e.g., concerning security protocols and technologies) are specified in a *security design model*, which is created in two steps. First, a skeleton model is generated from a security requirement model using model transformation, where a security requirement is often expected to be mapped to the security decision that gives a technological solution for it. Then, developers must complete the model with information related to the design of the system security. *Security design models* do not include specific information about concrete protocols or technologies, but refer to abstract platforms (e.g. EJB). A *security implementation model* is another kind of platform dependent model aimed at representing this kind of information on the concrete platform used to implement the security requirements (e.g. an implementation of EJB as BEA WebLogic). It is worth noting that *security design models* and *security implementation models* are platform dependent models, whereas *security requirement models* should be independent

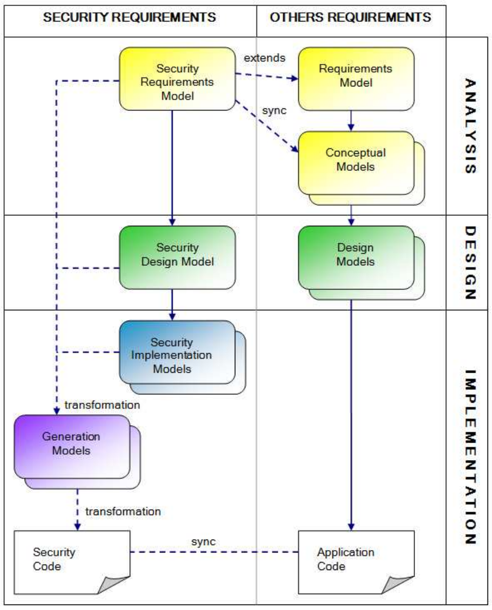

Figure 1: Overview of the models proposed by SecureModel

of any platform. Therefore, in the same way as with the MDA approach, our proposal differentiates between platform independent models (PIM) and platform specific models (PSM). In Section 6, specific examples of these models will be shown through an example of an application.

The security models presented above are the input to a model-to-model transformation whose result is a target platform model that conforms to a metamodel representing a target technology. Security code of the final application is then generated from this intermediate model. There will be a target platform model, and therefore a security design model and a security implementation model, for each security aspect considered for generating software. There are some reasons for the existence of these intermediate models. Firstly, the semantic gap between the business requirements and the implementation code is reduced and a large transformation is replaced by two that are simpler. The existence of different transformations promotes reuse since the target platform metamodels and model-to-code transformations can be used in several contexts. Another benefit is that these models can be used to modify the code that is going to be generated without the need of editing it by hand. Because the *security design model* and the *design models* are not synchronized, in a few cases it cannot be expected that generated code related to security would be completely synchronized with the existing code.

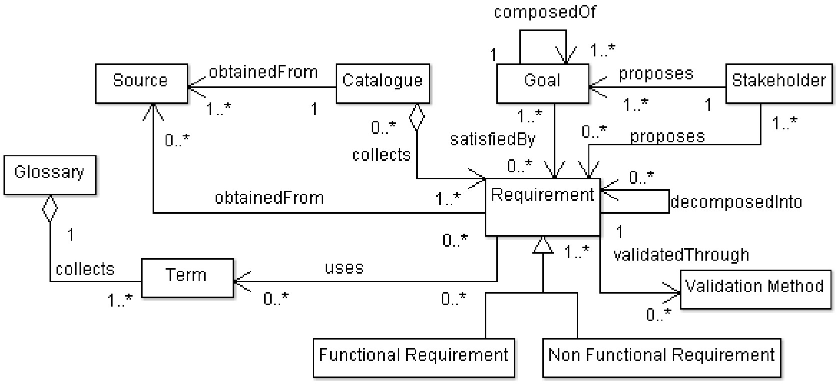

Figure 2: Proposed core requirements metamodel

In Section 5 the above security models in the context of the generative architecture of ModelSec will be explained.

3 A metamodel for security requirements

3.1 Requirements metamodelling

Most traditional approaches to Requirement Engineering (RE) use textual descriptions for the specification of requirements, which are organized in requirement documents that are rarely formally structured. Recently, the emergence of the MDE paradigm, especially the metamodelling technique, has introduced a new perspective for dealing with these requirements. Now metamodels are used to formally define the concepts and relationships involved in the analysis of requirements [Goknil et al., 2008].

Several approaches to metamodelling requirements have recently appeared [Goknil et al., 2008, Vicente et al., 2007, Berre, 2006]. However, a reference model has not yet been established and the existing approaches present a great disparity with regard to the number or semantics of the concerns addressed. Thus, a core requirement metamodel which contains the common concepts of the proposed metamodels and several new concepts and relationships considered relevant in our proposal (e.g. the concept of requirement reuse) has been defined. Moreover, these existing metamodels do not distinguish among different non-functional requirements, so an extension of the core metamodel, aimed at representing the real concepts of security requirements has also been defined. Figure 2 shows the core metamodel while Figures 3 and 4 in Subsection 3.2 show the extension proposed for including security concepts.

The key element in Figure 2 is that of `requirement`, which can be described by using a set of attributes such as an `identifier`, its `type` and its

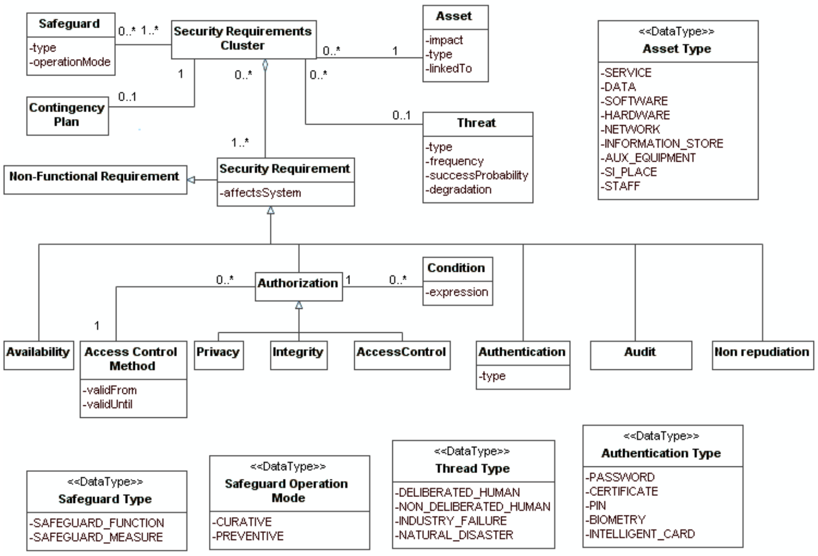

Figure 3: Extending the metamodel of Fig. 2: general security concepts

textual `description`. Requirements can be classified as either `functional` or `non-functional`. Each requirement is connected to the set of `goals` that it attempts to satisfy. Each goal can be composed of other goals, and captures a high-level objective from one or more `stakeholders` that propose it. Another concern is related to the reuse of requirements, which is tackled by including a `catalogue` concept and serves to gather a set of requirements extracted from one or more `sources` (*i.e.*, a law or a particular domain,) and that can be reused in all the projects to which these sources are applicable. A detailed explanation of the concepts on this core metamodel can be found in [Molina and Toval, 2009].

3.2 Extending the requirements metamodel with security concepts

The metamodel of Figure 2 has been extended with specific security concepts in order to define a DSL for security requirements. This extension has been divided into two modules in order to separate the access control mechanisms from the rest of the concepts (see Figures 3 and 4). The reason for this separation is that it is expected it will widen the range of control access mechanisms in the foreseeable future, so changes made to this part of the metamodel should not have side effects upon the rest of the metamodel. To facilitate the explanation, the concepts have been divided into three categories: basic security concepts, security requirements, and access control methods.

The basic security concepts are `Asset`, `Threat`, `Safeguard` and `Contingency Plan`. These terms conform to the standard ISO/IEC 15408 [ISO, 2005]. An `Asset` is a physical or logical object that has value in itself and deserves to have some guarantees with it. Assets can be of different types, for instance, documents, data tables, and so forth, and they have some importance for a business, which is measured by an impact index. Assets refer to elements in the conceptual models specified by the `linkedTo` attribute, which can be implemented in two main ways. An option is to implement it as an explicit reference to a concrete model. Reference to a UML class diagram could be possible, given that an implementation of the UML metamodel exists in Ecore (i.e. the metamodel language on the platform Eclipse used to implement our approach). Another possibility could be to use identifier-based links, i.e. an identifier string that will match an identifier in a conceptual model. The first option allows us straightforward navigation through the conceptual model, while the second one allows us to decouple both models, although consistency between related elements in both models is not ensured. The latter option has been chosen so that any conceptual model can be used, not only concrete models such as UML class diagrams.

Assets can be damaged by a `Threat`, which has properties such as type, frequency (modeled as an annual rate), a probability of concrete success and degradation (that is, the level of damage caused in an asset if a threat achieves its goal). `Safeguards` serve as an impediment to a risk, in order to reduce it. As is shown in the type attribute, `Safeguard Functions` and `Safeguard Measures` are distinguished. The former are actions which reduce the risk whereas the latter are physical or logical devices or processes that reduce the risk. Two operational modes are distinguished for the safeguards: preventive if they act before a threat has taken place and curative if they act on damaged assets. For the sake of reducing a threat that can give rise to damage, a detailed `Contingency Plan` composed of a set of safeguards is recommended.

A standard classification for security requirements does not exist, so based on [Rodríguez et al., 2007], seven categories have been considered, which tackle seven categories of threats. These categories are: privacy, integrity, access control, authentication, availability, non-repudiation and auditing. Privacy consists of ensuring that information only can be read by those who are allowed. Integrity is the guarantee that the information stays complete and correct. Access Control is used to constrain the users that are authorized to access to an asset. These three kinds of requirements are related to `authentication` and can have a `condition`, defined as an expression. Authentication refers to the parties involved in a communication or interaction. Two kinds of authentication can be defined: authentication of the users of the service, also known as target authentication, and authentication of the data source, which is called source authentication. Availability consists of ensuring that authorized users have access to the

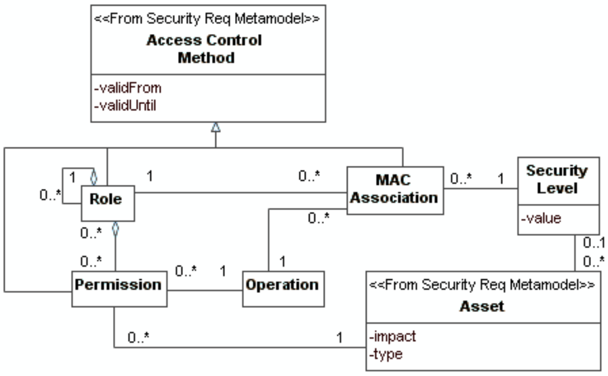

Figure 4: Extending the metamodel of Fig. 2: access control mechanisms

information when needed. Non-repudiation is the ability to prove that parties have been involved in a communication or action, in such a way that parties cannot deny using the system. Finally, audit tries to log the use of services and data. All these sorts of requirements can affect either particular assets or the entire system.

Frequently, a set of requirements exist which are related to the same asset, reduce the effects of the same attack and achieve the same security objective. This concept, extracted from [Mellado et al., 2007], is introduced in our metamodel as a `Security Requirements Cluster`. Regarding privacy, integrity and access control requirements, they are directly associated with an `Access Control Method` which has a period of validity. The different methods considered are `permissions` (Discretionary Access Control, DAC), `security levels` (Mandatory Access Control, MAC) or Roles (Hierarchical Role-Based Access Control, HRBAC) [Samarati and Capitani, 2000]. The concept of `MAC Association` has been introduced for associating a security level and an operation with a role. Note that security levels and roles are related, instead of security levels and users. Users may change from time to time so it is not desirable to model users at such a high level.

Bearing the above security concepts in mind, a security requirement metamodel has been obtained by extending the core metamodel shown in Figure 2 with the specific security concepts shown in Figures 3 and 4. These two figures only show the most important attributes of the concepts, and data types in Figure 3 only include some of the possible values. As can be seen, the extension point of the core metamodel to add new non functional requirements is the `NonFunctional Requirement` metaclass. `Security Requirement` is a specialization of `Non-Functional Requirement` intended to be root for the metaclasses representing security concepts.

4 A DSL for security requirements

In this section the DSL created to express the models which represent the security requirements in a simple and intuitive way is explained. These models are named *security requirement models* and they define the security needs to be satisfied, in such a manner that platform details are not described. While most existing approaches provides UML profiles (e.g. UMLsec and SecureUML), our approach provides a DSL tailored to this domain and called SecML (Security Modeling Language).

Nowadays, the implementation of a DSL is considered more appropriate than defining an UML profile, because their design and implementation is not restricted by the limitations of the profiling mechanism [Kelly and Tolvanen, 2008, Voelter, 2008]. When metamodelling techniques are applied, a DSL consists of three elements: an abstract syntax metamodel, concrete syntax, and semantics. The abstract syntax defines the concepts that take part in the DSL and the relationships between them, and also includes the rules which constrain how the models can be created. The concrete syntax defines a notation for the abstract syntax, and semantics is normally provided by means of model transformations which transform a model expressed in the DSL into models expressed in languages with well-defined semantics [Kelly and Tolvanen, 2008].

In our case, the abstract syntax is given by the metamodels explained in Section 3. The graphical representation of the concepts (i.e. the concrete syntax) included in the DSL is as follows: (see Figure 8(a)). Basic security concepts, security requirements and access control methods in the metamodel are depicted by a rectangle with an icon in the upper-left corner. Each concept has a different icon. For example, the clusters of security requirements have a padlock, security requirements have a star of a different colour, depending on their type, and assets are depicted with three small balls. Different from other elements, clusters have compartments in which the security requirements are included. Some of the relationships between concepts, such as the relationships between cluster and threat or asset, have been explicitly represented by means of arrows. Finally, model-to-model transformations from *security requirements models* to software artifacts give semantics to our DSL.

The choice of a graphical DSL instead of a textual one has been determined by the fact that non-expert users could use it. Graphical DSLs are more readable and meaningful for representing relationships between concepts, such as relationships between requirements and assets or, in our case, threats. In addition, they are often easier to learn than textual DSL.

SecML plays a key role in our generative architecture. SecML models can be created by users who only have business knowledge. The analyst can express the security restrictions by using well-known domain concepts, while the modeling of design decisions and implementation details are left to the experienced developers. A SecML specification has been split into two views: a view for modeling textual requirements and a view specifically designed to model security requirements with SecML. The latter view is related to the former since it details some of the requirements specified in the textual view. SecML has been implemented on Eclipse using the Graphical Modeling Framework (GMF), a powerful, practical and widespread framework for giving a graphical concrete syntax to an abstract syntax expressed with Ecore metamodels. As we indicated, Eclipse is a well known platform that provides, through several projects, the most widely used tools and frameworks for MDE. Moreover, GMF has several interesting features: it is open source like Eclipse, which means that anybody can customize it; a DSL created with GMF takes advantage of the tools and features of the Eclipse IDE; code generation is supported by different template languages; and, tool interoperability is obtained by using XMI, the model serialization format of Eclipse.

5 From security models to security software artefacts

This section explains how the ModelSec generative approach works. Previously described security requirement models are completely platform independent and, thus, other models are needed to represent platform specific security data. Therefore, previous to presenting an overview of the approach, these models will be described.

5.1 Making design decisions: the Security Design Model

Whereas *security requirements models* specify *what* security restrictions the final system must satisfy, *security design models* specify *how* these restrictions will be satisfied. These models conform to the security design metamodel shown in Figure 5, which represents the design decisions related to security requirements specified in SecML. The metamodel is simple, and contains a hierarchy of types of design decisions along with a set of data types. The hierarchy has a metaclass for each security aspect considered by the requirement specification (e.g. login, authorization, and access control policy). The description of the attributes of these metaclasses shows an idea of the type of information provided by means of design security models.

Each requirement in a security requirement model is mapped to an instance of `SecurityDesignDecision`. Currently, authentication requirements are mapped to `AuthenticationDecision`, non-repudiation and audit requirements are mapped to `LoggingDecision`, and authorization requirements are mapped

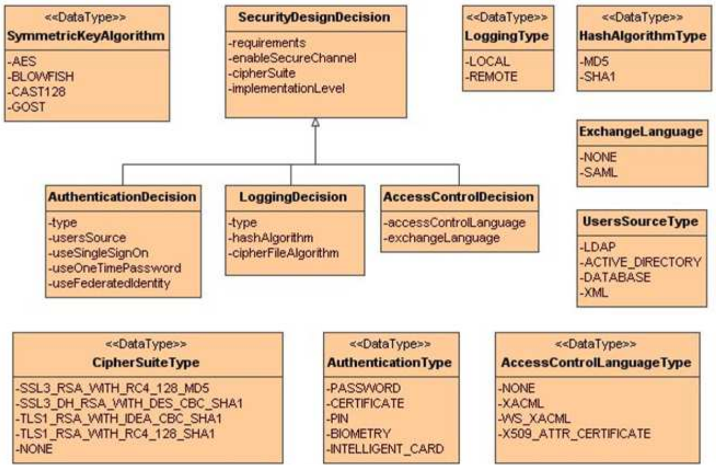

Figure 5: Security Design Metamodel

to `AccessControlDecision`. Other security aspects available are not yet considered due to complexity in implementation in software. They will be considered in further work.

The `SecurityDesignDecision` metaclass has four attributes, representing properties common to any design decision. For each design decision, the list of requirements related to it must be specified. If a requirement implies the use of a network, the `enableSecureChannel` attribute indicates whether the communications must be secured, and `cipherSuite` declares the set of protocols and security algorithms to use. This suite includes the cryptographic protocol, the public-key algorithm, the symmetric-key algorithm and the hash algorithm. The `implementationLevel` attribute lets the developer declare whether the requirement will be implemented by either a pre-defined or a custom solution. Depending on the solution, different artifacts could be generated. For example, authentication could be accomplished by a module server, and a skeleton of the XML server configuration file that includes authentication could be automatically generated. On the other hand, authentication could not rely on a third-party solution, and Java Authentication and Authorization Service (JAAS) [SUN, 2008] could be automatically generated.

Each metaclass inherited from `SecurityDesignDecision` adds its own attributes. An `AuthenticationDecision` includes a type of authentication that must be consistent with the security requirements model, if specified. The source of the users (i.e. `usersSource` attribute) can also be indicated. There are some other options available for authentication, such as `useSingleSignOn`, `useOneTimePassword`, `useFederatedIdentity` that allow us to customize any features of the authentication needed.

A `LoggingDecision` has a `type` of log mechanism, a `cipherFileAlgorithm` property that indicates whether the log file must be ciphered and which is the symmetric-key algorithm involved, and a `hashAlgorithm` property that specifies the algorithm used to perform a hash of the file, if desired.

With regard to the `AccessControlDecision` metaclass, the attribute `accessControlLanguage` is used to specify that we are interested in writing control policies explicitly with a suitable language, and `exchangeLanguage` indicates that we would like to use an appropriate language for considering the access control petitions and decisions.

As can be observed, this metamodel is expected to change in the future, given that new protocols or technologies are constantly evolving, and new solutions (i.e. design decisions) will be considered. For this reason, this metamodel includes only the most common attributes we have detected at the present time.

5.2 Considering the platform: the Security Implementation Models

When design decisions have been taken into account, detailed information about them must be provided in order to accomplish a proper generation of software artifacts. This low-level information is dependent on the target platforms for which these artifacts are generated. This justifies why our approach distinguishes between security `design models` and `security implementation models` and avoids mixing design decisions and platform specific data. Since there are too many specific platform details at this level, metamodels are not shown, but some examples of them can be found in Section 6.

One or more security implementation models are usually created for each design decision. No *implementation model* will be defined if we are not interested in generating some kind of predefined artifact or there is not a current transformation available for the chosen technologies. For example, a policy model should be created if we defined an access control decision enabling the `accessControlLanguage` option. A skeleton of the implementation models can be obtained from the design models through model-to-model transformations, in the same way that a design model can be generated from the requirements model. As indicated in Section 2, *security design models* and *security implementation models* are platform specific models (PSM). Notice that the former is dependent on a platform at a more abstract level that the latter.

As a rule, in addition to the models derived from every design decision, an implementation parameter model is required. The goal of this model is to provide information about the whole target system, such as the type of application, the concrete target platform, the technologies implementing the different layers, etc. Moreover, this model allows us to decouple the declaration of application users from the requirement models. Since users are frequently expected to change

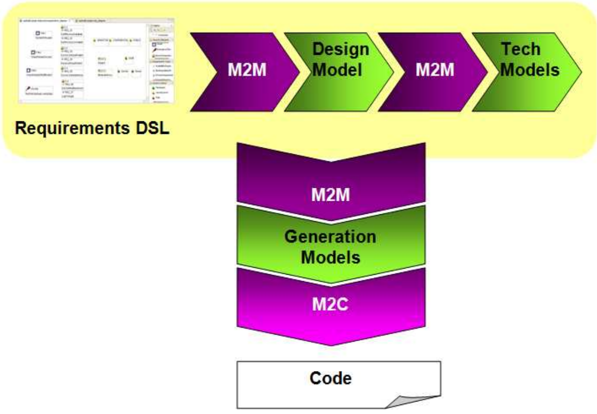

Figure 6: Overview of the generative architecture

throughout the life of the new system, it is desirable to declare them as late as possible so they will not affect the higher level models.

5.3 Generative architecture overview

Figure 6 shows an overview of the presented generative architecture, which has been implemented using Eclipse. As it was previously stated, once a *security requirement model* is available, a *security design model* must be created, which depends on the previous model. In the same way, *security implementation models* must be defined based on the *security design model.* Making use of the generative approach, model-to-model transformations (M2M) have been defined for generating skeletons of these dependences. Thus, a *design model* skeleton is derived from a *security requirement model* that is created by means of the security requirements DSL. Similarly, *security implementation model* skeletons can be derived from the *security design model* by means of M2M transformations. This chain of model transformations can be seen at the top of Figure 6. Notice that design and implementation models are created by developers by adding information to the generated skeletons.

Another chain of model transformations can be observed in Figure 6, which is depicted vertically. This chain aims to automatically generate software artifacts from the previously created security models. A security requirement model, together with the security design and security implementation models are the input of a M2M transformation that generates several target platform models. RubyTL [Sánchez et al., 2006] is the transformation language selected to implement these transformations, since it is compatible with Ecore, which allows us to interoperate with all of the tools implemented. Finally, model-to-code (M2C) transformations are used for generating software artifacts related to the system security. In this case, the MOFScript [Eclipse, 2008b] template language has been chosen due to it being a simple but powerful language based on the use of rules.

Note that the chain of model transformations of the generative architecture works due to the fact that all the tools used are compatible, in the sense that they serialize Ecore models in XMI format. Next section illustrates how this architecture has been used for generating security code for Oracle and XACML policies.

6 Application example

To illustrate how the ModelSec approach works, the example adapted from [Fernández-Medina and Piattini, 2005] of a web application for the management of medical patients will be used. This example, which includes the design of a secure database, serves us to show how a role-based control access policy and database security code can be automatically generated. An in depth explanation of those models which are not related to security has been omitted since they are not needed to understand the example but, when necessary, the dependences among models will be shown.

As indicated in Section 2, the process starts with the definition of the requirements of the system using the SecML. Figure 7 shows the view of the tool provided for managing requirements, which is used by the analyst for editing the textual description of the requirements. Instead of directly creating a model, the analyst could use this view to introduce each requirement (description and type), and then the corresponding model is generated and populated with an instance for each requirement. Then, the analyst uses the graphical view of the tool for the input of the values of the properties of each requirement. In Figure 7 an extract of a catalogue of requirements elicited by a hospital manager related to the data of the hospital patients is shown. The upper part of the catalogue includes a pair of functional requirements (labeled as REQ_F1 and REQ_F2) whereas the lower part involves non-functional requirements (labeled from REQ_S1 to REQ_S7). For reasons of simplicity, only some non-functional requirements have been included. It is important to note that SecML could be used for modeling both functional and non-functional requirements.

An excerpt of the *security requirements model* for this catalogue appears in Figure 8(a), which shows a screenshot of the security view of the tool. The window panel consists of two parts: the editing area and the element palette. A model with different types of elements can be observed in Figure 8(a), while the properties that describe the most relevant types of elements are presented in Figure 8(b). In Figure 8 several elements are visualized: four clusters containing a

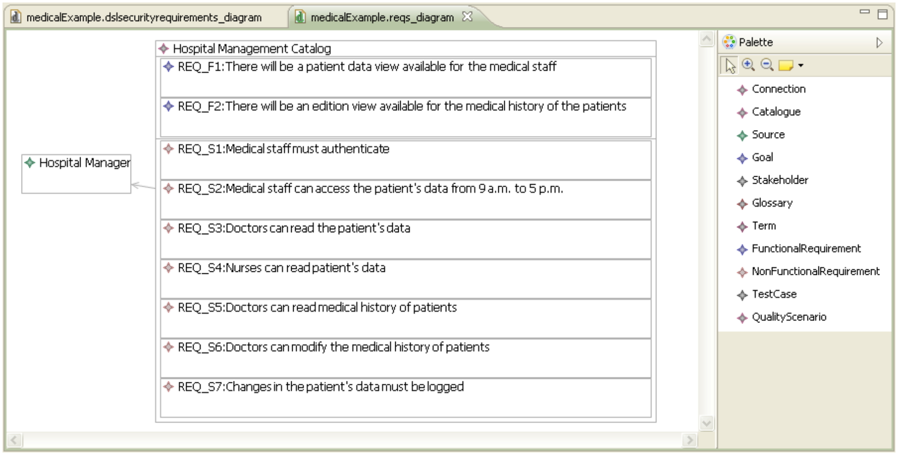

Figure 7: An example of security requirements catalogue

total of seven security requirements, two assets, a hierarchy of roles, three threats, three access control policies, and a contingency plan. *Patient* and *MedicalHistory* are the main assets that need to be secured. The former represents the personal information while the latter is a record of the illnesses and medical treatment of a patient. These two assets reference classes in the conceptual model, which are specified by the `linkedTo` attribute (an explanation of how this attribute can be implemented was indicated in Section 2).

The CL1 cluster includes security requirements that restrict access to the system to the medical staff. In CL1, the requirement REQ_S1 restricts access to previously registered users who must provide Login/Password credentials, and REQ1_S2 states a constraint on access, since only those users with a staff role who attempt to access between 9 a.m. and 5 p.m. are allowed (this is specified in the condition attribute of REQ_S2 in Figure 8(b)). Date, time and IP addresses can be restricted by using the simple expression language defined. These requirements lead the analyst to create a role hierarchy whose root is the *Staff* role, and with two sub-roles named *Doctor* and *Nurse*, which will be used afterwards.

The CL2 cluster is intended to prevent unauthorized staff accessing patient information, and the CL3 cluster has a similar purpose, but related to *MedicalHistory*. The requirements contained in the cluster CL4 attempt to prevent unauthorized modification of the *patient* data.

With regard to the control access policy, the analyst has decided to use MAC for taking control over the users who attempt to query or modify patient data. Thus, *public*, *confidential* and *sensitive* elements have been added to the model. A *confidential* level has been assigned to the asset *Patient*, and *sensitive* to *MedicalHistory*. Requirements of privacy and integrity must specify a security

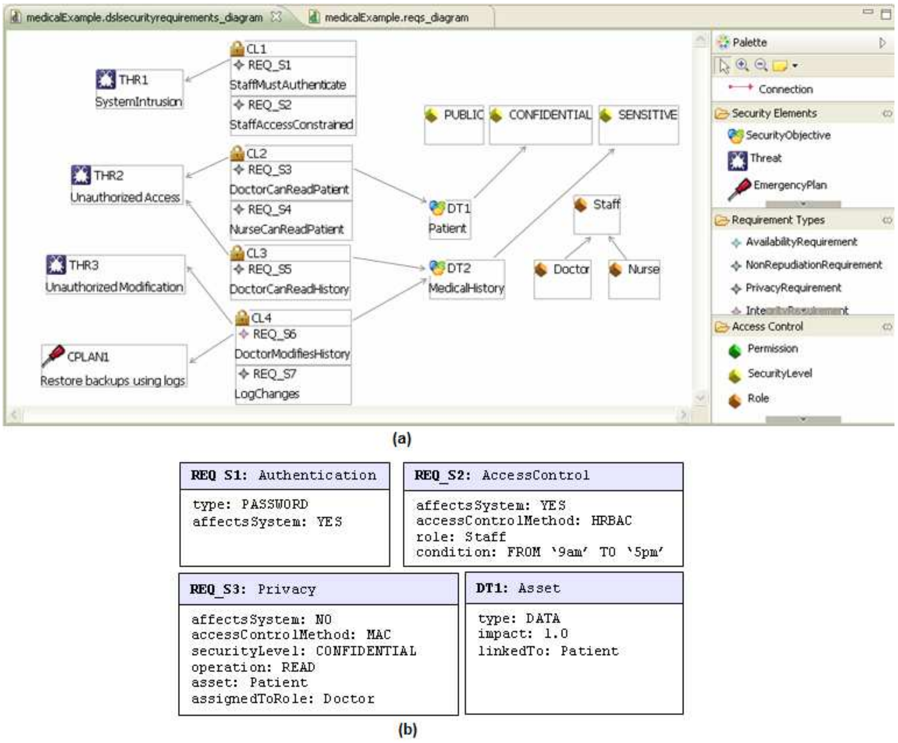

Figure 8: (a) Excerpt of a security requirements model (b) Attribute values

level by considering the labels established for patient data and histories. It is important to note that although security levels are used, we need to specify which group of users will be granted this level, so roles are used for this purpose. For instance, REQ_S3 assigns the confidential read level on the patient data to the role *Doctor*, which means that the doctors can query the information related to the patients because its level is the same as the *Patient* asset.

In the example we can see that the clusters refer to three threats and one contingency plan. These two kinds of elements allow the specifying of information that can be used to generate system documentation, for instance, weaknesses and possible solutions. Although threats and contingency plan elements have been depicted in the requirement model, they will not be considered in the rest of the example because of lack of space.

Once the security requirement model is completed, a *security design model* must be created in order to express the design decisions. As explained previously, due to the existing mapping among elements of both models, a design model skeleton can be automatically generated from the requirement model, and then the developer specifies the information of the model elements. Following with our example, Figure 9(a) shows the four elements of this model: one `AuthenticationDecision`, two `AccessControlDecision` and, finally, one `LogginDecision`. Only those element properties that do not have the default value (i.e., are different from NO or NULL) have been shown. No secure communications have been included since they are not needed in this example.

The REQ_S1 requirement of type authentication is mapped to an instance of `AuthenticationDecision`. In REQ_S1, *password* is specified as the value of the attribute type, which means a login/password authentication. Therefore the value of the type attribute of the `AuthenticationDecision` element also has this value. The `implementationLevel` attribute has been established to *custom*, because the authentication will not be automatically performed for any module, but we will implement this control in our application. We also specify that the users are registered in a LDAP server, but so long as we are going to implement authentication manually, this annotation does not have further implications.

A *custom* implementation level has also been specified for the element `AccessControlDecision`, which is linked to the REQ_S2 requirement of type `AccessControl`. This design decision is motivated by the fact that we intend to implement the `Policy Enforcement Point`, the `Policy Decision Point` and the whole authorization process by ourselves. As the access control policies need to be automatically generated, XACML has been assigned to the `accessControlLanguage` attribute in order to indicate the language for generating the policies. SAML assertions could also be generated, but this possibility has not been considered, in order to simplify the example.

A second `AccessControlDecision` example refers to the requirements from REQ_S3 to REQ_S6, and are those that prevent unauthorized reading or modification of any of the personal data or medical history of the patients (i.e. authorization requirements). In this case, the implementation level of the authorizations is expressed as database, which means that the access control method will be implemented in the database management system and it will be responsible for checking the access rights to the tables related to the assets.

Finally, a `LoggingDecision` element has been included for the REQ_S7 requirement of type auditing. The type attribute takes the value *local*, which indicates that we will use a local log without any kind of protection and, its implementation is the choice of the developer.

At this point, the *security design model* specifies how the security requirements will be implemented, but the information provided only indicates, at an abstract level, which technological solutions will be used, but does not includes details about target platforms. Thus, a *security implementation model* is used to express the implementation details that refer to a concrete target platform which are needed for generating software artifacts. In this example, the *security*

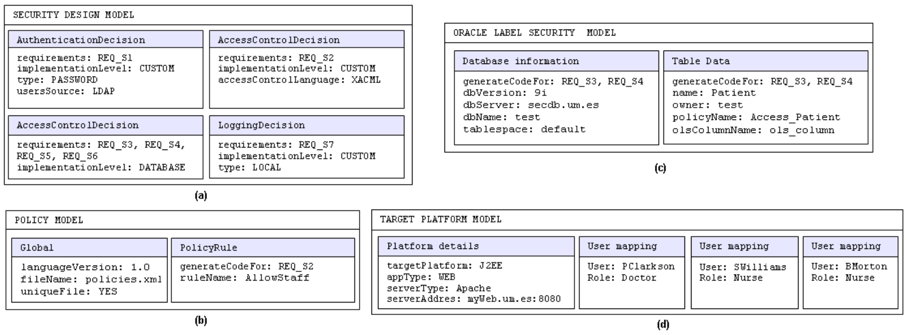

Figure 9: (a) Security design model, (b) Policy model, (c) Oracle model, (d) Target platform model

design model created is detailed in three security implementation models which represent low-level data of the access control policy (Figure 9(b)), the security database (Figure 9(c)) and the target platform (Figure 9(d)). Note that we have a general access control policy model, which could be used with different access control languages, while we have a specific database model for Oracle. The reason for this disparity is that we cannot have a general database model, because each database system has its own security solutions, so specific models are needed.

The number and nature of the security implementation models depends on the technologies chosen, for which software artifacts are generated. In our case, a *policy model* has been defined to generate access control policies, and an *Oracle model* to generate database code. A definition of an implementation model is always required for any generation in order to provide some details about the target platform, such as the application type, and a list of the users initially registered in the system. Regarding the *policy model*, a few global parameters must be specified, such as the target file and some data for each access control requirement. To illustrate this, a permission rule for REQ_S2 will be generated.

Although pair user-roles are defined in this model, it does not necessarily imply that roles will be defined in the database. In the example, we define roles because we need to assign security levels to users and we do not want to define users in the security requirements model. However, no role definition code will be generated. So, roles serve as a mechanism to decouple users and access control methods.

All models created are input to a M2M transformation specified in RubyTL that automatically generates target platform models. A model will be generated for each security implementation model. In this example, a model for access control policies and a model for database code will be generated. As indicated earlier, these models conform to the metamodels representing the target platform; for this example a XACML metamodel and an Oracle Label Security metamodel were defined. Finally, security software artifacts are automatically generated from target platform models by means of M2C transformations specified in MOFScript. In this case, the software artifacts created have been a XACML policy file and an Oracle PL/SQL script. It is remarkable that the target platform models as well as the code are automatically generated, and no human intervention is required.

Figure 10 shows an excerpt of the XACML policy file generated. This is a snippet of code that includes a rule that authorizes users classified as *Staff* to access the resources between 9 a.m. and 5 p.m. The description of the rule is the text of the security requirement itself. The subject of the rule (i.e. the role *Staff*) and the condition that restricts the hours of access are both extracted from the properties stated in the previous models. Figure 11 shows an excerpt of PL/SQL Oracle Label Security code. As we specified in the Oracle model (Figure 9(c)), a policy for controlling the reading of the *Patient* table has been generated. The different security levels are created as well as the labels connected to them. After this, the labels are assigned to the different users defined in the implementation target model, according to the reading and writing restrictions declared in the security requirement model. Finally the policy is applied to the table.

It is worth to remarking that the creation of security artifacts involves a great deal of text even though only simple requirements have been considered.

7 Related work

The approach presented can be related to works both on requirement metamodelling in the field of RE and MDS approaches. As mentioned in Section 3, different requirement metamodels have been proposed [Goknil et al., 2008, Berre, 2006, Vicente et al., 2007, Bolchini and Paolini, 2004]. These metamodels are focused on the basic concepts of the RE and do not include concepts related to concrete non-functional requirements such as, for example, security. We have defined a requirement metamodel which combines a set of core requirement concepts with a set of security concepts. The core concepts have been obtained by identifying the concepts common to the existing metamodels. Moreover, most of the existing approaches for metamodelling requirements have not considered the definition of a concrete syntax (notation) or the automatic generation of software artifacts from requirements.

As indicated above, two of the most relevant approaches in security modeling are UMLSec [Jurjens, 2003] and SecureUML [Basin et al., 2006]. Both of them provide UML extensions to specify security requirements. Whereas UMLSec

```xml
<Rule RuleId="AllowStaff" Effect="Permit">
<Description>Medical staff can access the patient's data from 9 p.m. to 5 p.m.</Description>
<Target>
    <Subjects>
        <Subject>
            <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Staff</AttributeValue>
                <SubjectAttributeDesignator AttributeId="role"
                    DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </SubjectMatch>
        </Subject>
    </Subjects>
    <Resources>...</Resources>
    <Actions>...</Actions>
</Target>
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
            <EnvironmentAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#time"
                AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
            <EnvironmentAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#time"
                AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
    </Apply>
</Condition>
</Rule>
```

Figure 10: Excerpt of the XACML policy file automatically generated

was conceived to verify formal security requirement specifications in the system design, SecureUML was used to illustrate the MDS approach. Although SecureUML is specific to role-based access control infrastructures, it shows how an MDE strategy could be applied to generating code for any aspect of security, and since then a number of MDS approaches have been proposed. For instance, [Reznik et al., 2007] presents an MDS solution to developing secure applications on a middleware platform which integrates an implementation of the Corba Component Model with the OpenPMF security framework. In this approach, another UML profile is created in order to model access control policies. [Lang and Schreiner, 2008] illustrates how the OpenPMF architecture can be used to translate a security-related high-level regulatory requirement into enforceable authorization rules. In this case, the example considered is similar to that presented in this paper: a healthcare security requirement is implemented using XACML.

Our proposal has several characteristics that differentiate it from existing MDS approaches. The generative architecture has been organized in two chains of model transformations, as shown in Figure 6. This organization promotes a systematic process for applying MDS, which separates specifications of security requirements and design decisions of how they are implemented. In contrast to other approaches, models are used to represent design decisions and imple-

```
execute sa_sysdba.create_policy('Access_Patient', 'ols_column', 'read_control,label_default,hide');
execute sa_components.create_level('Access_Patient', 1000, 'PUB', 'PUBLIC');
execute sa_components.create_level('Access_Patient', 2000, 'CONF', 'CONFIDENTIAL');
execute sa_components.create_level('Access_Patient', 3000, 'SENS', 'SENSITIVE');
execute sa_label_admin.create_label('Access_Patient', 1000, 'PUB');
execute sa_label_admin.create_label('Access_Patient', 2000, 'CONF');
execute sa_label_admin.create_label('Access_Patient', 3000, 'SENS');
execute sa_user_admin.set_user_labels('Access_Patient', 'PClarkson', 'CONF', 'CONF', 'PUB', 'CONF', 'CONF');
execute sa_user_admin.set_user_labels('Access_Patient', 'SWilliams', 'CONF', 'CONF', 'PUB', 'CONF', 'CONF');
execute sa_user_admin.set_user_labels('Access_Patient', 'BMorton', 'CONF', 'CONF', 'PUB', 'CONF', 'CONF');
execute sa_policy_admin.apply_table_policy('Access_Patient', 'test', 'Patient');
```

Figure 11: Excerpt of the Oracle automatically generated

mentation details about the target platforms. Model reuse is facilitated by this separation. In order to reduce the semantic gap among security requirements and generated software, ModelSec proposes an intermediate model which conforms to a target platform metamodel. This intermediate step promotes reuse of transformations. Instead of building UML profiles, as with the other approaches, a DSL for expressing security requirements has been defined, which has been implemented by applying metamodelling techniques. Moreover, ModelSec is a generic approach, while most MDS approaches have been focused on the access control policies.

There exist other approaches in the scope of security requirements management that are not aligned to MDS and focus mainly on the elicitation of security requirements more than in the generation of software artefacts from them. That is the case of proposals as Secure Tropos [Bresciani et al., 2007], [van Lamsweerde, 2004], that introduces the concept of antigoal, [Yu et al., 2007] that uses ontologies or [Haley et al., 2008], which presents a framework composed by a set of activities to deal with security requirements.

8 Conclusions and further work

Creating security software artifacts requires handling a large amount of text in formats such as source code or XML text. In a similar way to other MDS approaches, ModelSec offers a high level of automation of the generation of code aimed at dealing with the security requirements of the system (e.g. security policies), and which avoid a tedious, time consuming, costly and error-prone manual process. However, in a different manner to the existing MDS approaches, ModelSec proposes a generative architecture based on a chain of model transformations involving several security models at different levels of abstraction. Throughout this paper how this architecture promotes the reuse and provides a more systematic process than the existing approaches has been shown. Another significant contribution is that the DSL for security requirements is built on our own requirement metamodel.

As for further work, ModelSec will be extended by defining new target platform metamodels that serve as a basis for the generation of other access control and authorization policies. Moreover, ModelSec is being used for the generation of code for real software projects, and it is expected that there will be interesting feedback for the refinement and extension of this proposal.

Acknowledgements

This work has been partially supported by the projects DEDALO (TIN2006-15175-C05-03) and PANGEA (TIN2009-13718-C02-02) from the Spanish Ministry of Science and Technology, MELISA-GREIS (PAC08-0142-335), the project 08797/PI/08 from the Fundación Séneca and the project 129/2009 from the Regional Council of Murcia. Fernando Molina is partially funded by the Fundación Séneca (Murcia).

References

[Basin et al., 2006] Basin, D., Doser, J., and Lodderstedt, T. (2006). Model driven security: From uml models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91.

[Berre, 2006] Berre, A. J. (2006). Comet (component and model based development methodology). http://modelbased.net/comet/.

[Bolchini and Paolini, 2004] Bolchini, D. and Paolini, P. (2004). Goal-driven requirements analysis for hypermedia-intensive web applications. *Requirement Engineering Journal*, 9(2):85–103.

[Bresciani et al., 2007] Bresciani, P., Mouratidis, H., and Zanone, N. (2007). Modelling security and trust with secure tropos. *Integrating Security and Software Engineering: Advances and Future Visions*, pages 160–189.

[Eclipse, 2008a] Eclipse (2008a). Eclipse Modeling Framework Project (EMF). http://www.eclipse.org/modeling/emf/.

[Eclipse, 2008b] Eclipse (2008b). Generative Modeling Technologies (GMT): MOFScript. http://www.eclipse.org/gmt/.

[Fernández-Medina et al., 2009] Fernández-Medina, E., Jurjens, J., Trujillo, J., and Jajodia, S. (2009). Editorial: Model-driven development for secure information systems. *Inf. Softw. Technol.*, 51(5):809–814.

[Fernández-Medina and Piattini, 2005] Fernández-Medina, E. and Piattini, M. (2005). Designing secure databases. *Information & Software Technology*, 47(7):463–477.

[Goknil et al., 2008] Goknil, A., Kurtev, I., and van den Berg, K. (2008). A metamodeling approach for reasoning about requirements. In *ECMDA-FA*, pages 310–325.

[Haley et al., 2008] Haley, C., Laney, R., Moffett, J., and Nuseibeh, B. (2008). Security requirements engineering: A framework for representation and analysis. *IEEE Trans. Software Eng.*, 34(1):133–153.

[ISO, 2005] ISO (2005). ISO/IEC 15408 (Common Criteria v3.0): Information Technology Security Techniques-Evaluation Criteria for IT Security.

[Jurjens, 2003] Jurjens, J. (2003). *Secure Systems Development with UML*. Springer Verlag.

[Jurjens et al., 2008] Jurjens, J., Schreck, J., and Bartmann, P. (2008). Model-based security analysis for mobile communications. In *ICSE*, pages 683–692.

[Kelly and Tolvanen, 2008] Kelly, S. and Tolvanen, J. (2008). *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Press.

[Lang and Schreiner, 2008] Lang, U. and Schreiner, R. (2008). Managing business compliance using model-driven security management. In *Securing Electronic Business Processes*, pages 1–11.

[Mellado et al., 2007] Mellado, D., Fernández-Medina, E., and Piattini, M. (2007). A common criteria based security requirements engineering process for the development of secure information systems. *Comput. Stand. Interfaces*, 29(2):244–253.

[Molina and Toval, 2009] Molina, F. and Toval, A. (2009). Integrating usability requirements that can be evaluated in design time into model driven engineering of web information systems. *Advances in Engineering Software*, 40(12):1306–1317.

[OASIS, 2008] OASIS (2008). XACML: eXtensible Access Control Markup Language. http://www.oasis-open.org/.

[OMG, 2003] OMG (2003). *MDA Guide v1.0.1.* http://www.omg.org/mda.

[ORACLE, 2008] ORACLE (2008). Oracle label security. http://www.oracle.com/technology/deploy/security/database-security/label-security/index.html.

[Reznik et al., 2007] Reznik, J., Ritter, T., Schreiner, R., and Lang, U. (2007). Model driven development of security aspects. *ENCTS*, 163(2):65–79.

[Rodríguez et al., 2007] Rodríguez, A., Fernández-Medina, E., and Piattini, M. (2007). A bpmn extension for the modeling of security requirements in business processes. *IEICE Transactions*, 90-D(4):745–752.

[Samarati and Capitani, 2000] Samarati, P. and Capitani, S. D. (2000). Access control: Policies, models, and mechanisms. In *FOSAD*, pages 137–196.

[Sánchez et al., 2006] Sánchez, J., García-Molina, J., and Menárguez, M. (2006). RubyTL: A Practical, Extensible Transformation Language. In *ECMDA-FA*, pages 158–172.

[Selic, 2008] Selic, B. (2008). Mda manifestations. *The European Journal for the Informatics Professional*, IX(2).

[SUN, 2008] SUN (2008). JAAS: Java Authentication and Authorization Service. http://java.sun.com/javase/technologies/security/.

[van Lamsweerde, 2004] van Lamsweerde, A. (2004). Elaborating security requirements by construction of intentional anti-models. In *ICSE'04: 26th Int. Conf. on Software Engineering*, pages 148–157. IEEE Computer Society.

[Vicente et al., 2007] Vicente, C., Moros, B., and Toval, A. (2007). Remm-studio: an integrated model-driven environment for requirements specification, validation and formatting. *Journal of Object Technology, Special Issue TOOLS EUROPE 2007*, 6(9):437–454.

[Voelter, 2008] Voelter, M. (2008). Md* best practices. http://www.voelter.de/data/articles/dslbestpractices-website.pdf.

[Yu et al., 2007] Yu, E., Liu, L., and Mylopoulos, J. (2007). A social ontology for integrating security and software engineering. *Integrating Security and Software Engineering: Advances and Future Visions*, pages 70–109.