Advances in Model-Driven Security

Levi Lúcio[a], Qin Zhang[b], Phu H. Nguyen[b], Moussa Amrani[b], Jacques Klein[b], Hans Vangheluwe[c,a], Yves Le Traon[b]

[a] *Modeling Simulation and Design Lab, McGill University, Montreal QC, Canada*
[b] *Centre for Security Reliability and Trust, University of Luxembourg, Luxembourg*
[c] *Antwerp Systems and Software Modeling, University of Antwerp, Antwerp, Belgium*

Abstract

Sound methodologies for constructing security-critical systems are extremely important in order to confront the increasingly varied security threats. As a response to this need, *Model-Driven Security* has emerged in the early 2000s as a specialized *Model-Driven Engineering* approach for supporting the development of security-critical systems.

In this chapter we summarize the most important developments of *Model-Driven Security* during the past decade. In order to do so we start by building a taxonomy of the most important concepts of this domain. We then use our taxonomy to describe and evaluate a set of representative and influential *Model-Driven Security* approaches in the literature. In our development of this topic we concentrate on the concepts shared by *Model-Driven Engineering* and *Model-Driven Security*. This allows us to identify and debate the advantages, disadvantages and open issues when applying *Model-Driven Engineering* to the *Information Security* domain.

This chapter provides a broad view of *Model-Driven Security* and is intended as an introduction to *Model-Driven Security* for students, researchers and practitioners.

Keywords: Information Security, Model-Driven Security, Model-Driven Engineering, Separation of Concerns, Survey

Email addresses: levi@cs.mcgill.ca (Levi Lúcio), qin.zhang@uni.lu (Qin Zhang), phuhong.nguyen@uni.lu (Phu H. Nguyen), moussa.amrani@uni.lu (Moussa Amrani), jacques.klein@uni.lu (Jacques Klein), hv@cs.mcgill.ca (Hans Vangheluwe), yves.letraon@uni.lu (Yves Le Traon)

1. Introduction

The world is becoming increasingly digital. On one hand, advances in computers and information technology bring us many benefits. On the other hand, information security is becoming more and more crucial and challenging. Few days pass without new stories in the newspapers about malware, software vulnerabilities, botnet attacks or other digital data related problems. Thus, information security is a significant issue in computer science and keeps attracting the interest of researchers and engineers (Howard and Lipner, 2006).

Security requirements for software are becoming more complex in order to deal with the diverse and constantly changing threats. Given security requirements are often tangled with functional requirements, it is difficult to integrate them properly in the traditional software development process. Also, security requirements are rarely dealt with at the early stages of the development process (Cysneiros and Sampaio do Prado Leite, 2002). Traditional methods for developing security-critical systems are thus becoming increasingly inefficient. Moreover, due to economic pressure, development time is often short and the frequency of required modifications is high. This leads in practice to many security defects that have been exploited and made the headlines in the newspapers. All these issues show the need for more timely, innovative, and sound engineering methods in this area.

Model-Driven Security (MDS) has emerged more than a decade ago as a specialized *Model-Driven Engineering* (MDE) approach for supporting the development of security-critical systems. MDE has been considered by some authors as a solution to the handling of complex and evolving software systems (Bezivin, 2006). The paradigm consists of electing *models* and *transformations* as primary artifacts for each software development stage. By manipulating *models*, engineers work at higher-level of abstraction than code. MDS applies this paradigm to information security engineering, bringing several benefits to the domain. *First*, MDS models security concerns explicitly from the very beginning and throughout the development lifecycle. An MDS approach is expected to deliver a complete secure system implementation, not only the security infrastructure or the secure specification. *Second*, using models at a higher-level than the final target platform and independently from business functionality enables platform independence as well as cross-platform interoperability. Security experts can therefore focus on security-related issues, instead of dealing with the technical problems of integrating

the solutions to those issues in the system's infrastructure. *Third*, MDS leverages on MDE automation provided by *model transformations* such that human interference, which is naturally error-prone, is reduced.

In this chapter we start by summarizing the background theory of MDS in the light of MDE. We then propose a taxonomy for MDS, based on which we evaluate and discuss in depth five representative technical approaches from the MDS research community. The main contributions of this work are: 1) a comprehensive taxonomy for MDS; and 2) an thorough evaluation and discussion of some of today's most relevant MDS approaches. The goal of this chapter is to help readers better understand MDS and, if needed, point a potential MDS researcher or engineer towards an appropriate MDS approach among the existing ones in the literature. As mentioned previously, we provide a broad picture of research activities in MDS for the last decade.

The remainder of this chapter is organized as follows. In Section 2, we introduce some background information on MDE. Section 3 recalls several MDS definitions in the literature which help us to identify MDS approaches among a mass of security-related research studies. Then, a set of characteristics of MDS is identified and described in Section 3.2 in order to form a taxonomy for further evaluation of MDS approaches. In Section 4 we evaluate a few selected MDS approaches against our taxonomy. Section 5 provides a table comparing the evaluated approaches, and discusses some open issues and validity threats for MDS. Section 6 addresses the related work and finally Section 7 concludes with potential challenges for MDS.

2. Model-Driven Engineering

Model-Driven Engineering (MDE) encompasses both a set of tools and a loose methodological approach to the development of software. The claim behind Model-Driven Engineering is that by building and using abstractions of the processes the software engineer is trying to automate, the produced software will be of better quality than by using a general purpose programming language (GPL). The reasoning behind this claim is that abstractions of concepts and of processes manipulating those concepts are easier to understand, verify and simulate than computer programs. The reason for that is that those abstractions are close to the domain being addressed by the engineers, whereas general GPLs are built essentially to manipulate computer architecture concepts.

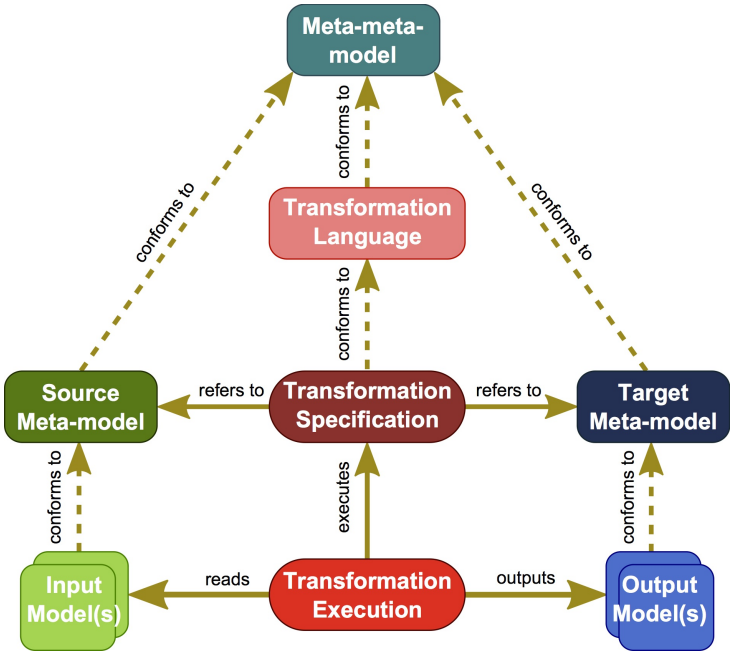

Figure 1: The Model Transformation Process (adapted from Syriani, 2011)

2.1. Models, Metamodels and Model Transformations

The central artifact in MDE is the *model*. A model in the computing world is a simplification of a process one wishes to capture or automate. The simplification is such that it does not take into account details that can be overseen at a given stage of the engineering cycle. The purpose is to focus on the relevant concepts at hand – much as for example a plaster model of a car for studying body aerodynamics will not take into account the real materials a car is made of.

In the computing world a *model* is defined by using a given language. Coming back to the car analogy, if an engineer wishes to have a computational model of a car for 3D visualization, a language such as the one defined by a Computer Assisted Design (CAD) tool will be necessary to express a particular car design. In the computing world several such languages – called *metamodels* – are used to describe families of models of computational artifacts that share the same abstraction concerns. Each *metamodel* is a language (also called *formalism*) that may have many *model* instantiations, much as in a CAD tool many different car designs can be described.

The missing piece in this set of concepts are *Model Transformations*.

Model transformations allow passing relevant information from one modeling formalism to another and are, according to Sendall and Kozaczynski (Sendall and Kozaczynski, 2003) the "*heart and soul of model-driven software development*". Model transformations have been under study from a theoretical point a view for a number of years (see e.g. the work of Ehrig et al. (2006)), but only recently have become a first class citizen in the the software development world. Their need came naturally with the fact that MDE started to be used professionally in certain areas such as mobile telephony or the automotive industry. Implementations for transformation languages such as ATL (ATLAS, 2008), Kermeta (Muller et al., 2005) or QVT (Dupe et al.) have been developed in the last few years and provide stable platforms for writing and executing model transformations.

Model transformations can have multiple uses in MDE: for example, if it becomes necessary to transform a UML statechart into code a model transformation can be seen as a *compiler*; also, a transformation to translate the statechart into a formalism amenable to verification by some existing tool may be seen as a *translator*. These transformations clearly exist in traditional software development, although in an implicit fashion. Being that in an MDE setting model transformations are responsible for translating models from one formalism into another, it becomes important for the quality of the whole software development process that those transformations are correct. The validation, verification and testing of model transformations is currently an active research topic as witnessed by the amount of recent publications on the topic such as (Amrani et al., 2012; Büttner et al., 2012; Guerra et al., 2013; Lúcio et al., 2010; Selim et al., 2012a,b; Vallecillo and Gogolla, 2012).

2.2. Model-Driven Engineering Approaches

In this Section, we briefly detail the main approaches following the MDE paradigm, namely *Model-Driven Architecture*, an early OMG standard generative approach; *Domain-Specific Modeling*, an approach aiming at defining a language for each different domain that contributes to an application; *Multi-Paradigm Modeling*, a generalization of Domain-Specific Modeling Languages where different models of computation interact; and *Aspect-Oriented Modeling*, studying more precisely how different models can be combined or composed.

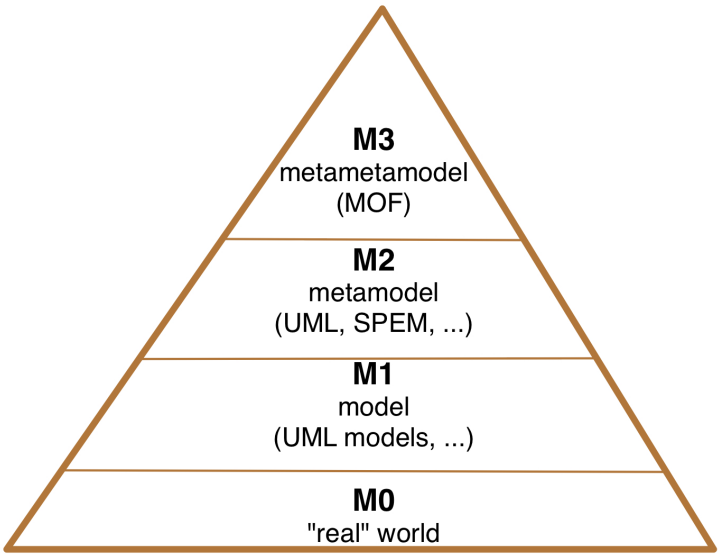

Figure 2: The MDA Pyramid (adapted from Thirioux et al., 2007)

2.2.1. Model-Driven Architecture

Model-Driven Architecture (MDA) is an OMG proposal launched in 2001 to help standardize model definitions, and favor model exchanges and compatibility. The MDA consists of the following points (Kleppe et al., 2003):

- It builds on the UML, an already standardized and well-accepted notation, already widely used in Object-Oriented systems. In an effort to harmonize notations and clean the UML's internal structure, Meta-Object Facility (MOF) was proposed for coping with the plethora of model definitions and languages;

- It proposes a pyramidal construction of models as can be observed in Fig. 2: artifacts populating the level $M0$ represents the actual system; those in the $M1$ level model the $M0$ ones; artifacts belonging to the $M2$ level are metamodels, allowing the definition of $M1$ models; and finally, the unique artifact at the $M3$ level is MOF itself, considered as meta-circularly defined as a model itself;

- Along with this pyramid, MDA enforces a particular vision of software systems development seen as a process with the following steps (Fig. 3): requirements are collected in a Computation Independent Model (CIM),

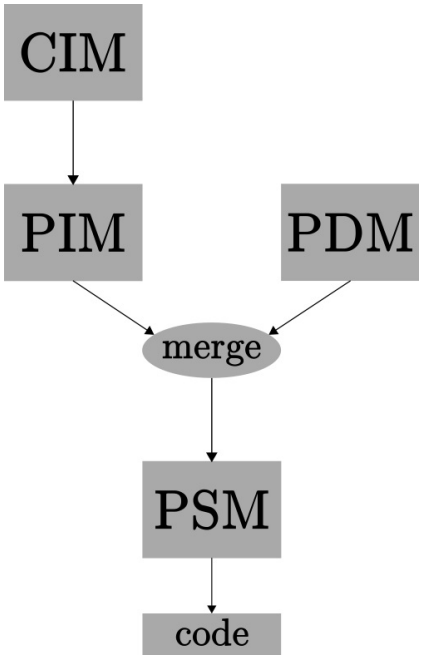

Figure 3: Model-Driven Architecture Overview

> independently of how the system will be ultimately implemented; then a Platform Independent Model (PIM) describes the design and analysis of all system parts, independently of any technical considerations about the final execution platforms and their embedded technologies; a PIM is then refined into a Platform Specific Model (PSM) and combined with a Platform Description Model (PDM) to finally reach code that will run on a specific platform.

MDA promotes a *vertical* separation of concerns: the system is designed at a high level, without any considerations about the target platform specificities; these specificities are then integrated by automated generators to produce code compliant with each platform. This methodology directly inspired several MDS proposals for enforcing security concerns within software applications.

2.2.2. Domain Specific Modeling

A common way to tackle the increasing complexity of current software systems consists in applying the "divide-and-conquer" approach: by dividing

the design activity into several areas of concern and focusing on specific aspects of the system, it becomes possible to raise the abstraction level of the produced specifications with the immediate benefit of increasing the level of confidence attached to them. This also allows making those specifications closer to the domain's experts, which facilitates controlling the quality of the produced artifacts and sometimes even delegating their creations to those experts. Within MDE, Domain-Specific Modeling (DSM) is a key methodology for the effective and successful specification of such systems. This methodology makes systematic use of Domain-Specific Modeling Languages (DSMLs, or DSLs for short) to represent the various artifacts of a system as models. The idea is simple: focusing designers' efforts on the variable parts of the design (e.g., capturing the intricacies of a new insurance product), while the underlying machinery takes care of the repetitive, error-prone, and well-known processes that make things work properly within the whole system.

A well-known white paper on the subject from Metacase (2009) presents anecdotal evidence that DSLs can boost productivity up to 10 times, based on experiences with developing operating systems for cell phones for Nokia^{TM} and Lucent^{TM}. These encouraging results pushed the scientific community to invest further in this topic and build environments to facilitate the construction, management and maintenance of DSLs. This effort has been materialized with concrete frameworks: EMF and GMF (Moore et al., 2004), AToM^3 (de Lara and Vangheluwe, 2002) or Microsoft's^{TM} DSL Tools (Cook et al., 2007), among others.

2.2.3. Multi-Paradigm Modeling

Multi-Paradigm Modeling (MPM), as introduced by Mosterman and Vangheluwe (2004), is a perspective on software development that advocates models should be built at the right level of abstraction regarding their purpose. Automatic *model transformations* should be used to pass information from one representation to another during the development of a system. In this case it is thus desirable to consider modeling as an activity that spans different models, expressed in different paradigms. The main advantage that is claimed of such an approach is that the software engineer can benefit from the already existing multitude of languages and associated tools for describing and automating software development activities – while pushing the task of transforming data in between formalisms to specialized machinery.

To make this idea more concrete, one may think of a UML statechart

model representing the abstract behavior of a software system being converted into a Java model for execution on a given platform; or of the same statechart being transformed into a formalism that is amenable for verification. Another possible advantage of this perspective on software development is the fact that toolsets for implementing a particular software development methodology become flexible. This is due to the fact that formalisms and transformations may be potentially plugged in and out of a development toolset given their explicit representation.

The idea of Multi-Paradigm Modeling is close to the idea of Model-Driven Architecture (MDA): in MPM the emphasis is mainly on the fact that several modeling paradigms are employed at the right level of abstraction during software development; MDA is rather focused on proposing a systematic methodology where a set of model transformations are chained in order to pass from a set of requirements for a system to software to be run on a given platform. MDA can thus be seen as an instance of MPM.

2.2.4. Aspect-Oriented Modeling

In MDS, when specified in isolation, security models can be composed into a business model (or target model) using Aspect-Oriented Modeling (AOM) techniques.

Modularization of crosscutting concerns has been popularized by the AspectJ programming language (Kiczales et al., 1997), but there is a growing interest in also handling them earlier in the software life-cycle, for instance at design time (Clarke, 2001), or during requirements analysis (Jacobson and Ng, 2004). Aspect-Oriented Software Development (AOSD) follows the well-known Roman principle of divide and conquer. Put in another way, separation of concerns is a long-standing idea that simply means a large problem is easier to manage if it can be broken down into pieces; particularly so if the solutions to the sub-problems can be combined to form a solution to the large problem. More specifically, AOSD aims at addressing crosscutting concerns (such as security, synchronization, concurrency, persistence, response time, among others) by providing means for their systematic identification, separation, representation and composition. Crosscutting concerns are encapsulated in separate modules, known as *aspects*. Once the different aspects are specified, they can be assembled to build the whole application. This mechanism of integration is called *weaving*. Generally, the weaving process is decomposed into two phases: a phase of detection, where a part of an aspect (called pointcut) is used as a predicate to find all the areas in a model (called

base model) where the aspects have to be woven; and a composition phase, where a second part of the aspect (called advice) is composed or merged with the base model at the previously detected areas (called join points).

Currently several techniques exist to represent, compose or weave aspects at a modeling level. Clarke and Baniassad (2005) define an approach called Theme/Uml. It introduces a theme module that can be used to represent a concern at the modeling level. Themes are declaratively complete units of modularization, in which any of the diagrams available in the UML can be used to model one view of the structure and behaviour the concern requires to execute.

France et al. (2004) and Reddy et al. (2006) propose a symmetric model composition technique that supports composition of model elements that present different views of the same concept. This composition technique has been implemented in a tool called Kompose (Fleurey et al., 2007). The model elements to be composed must be of the same syntactic type, that is, they must be instances of the same metamodel class. An aspect view may also describe a concept that is not present in a target model, and vice versa. In these cases, the model elements are included in the composed model. The process of identifying model elements to compose is called element matching. To support automated element matching, each element type (i.e., the elements meta-model class) is associated with a signature type that determines the uniqueness of elements in the type space: two elements with equivalent signatures represent the same concept and thus are composed.

Similar contributions follow the same lines and develop specific weaving techniques: either based on *behavioral aspects* (Whittle and Araújo, 2004; Cottenier et al., 2007; Klein et al., 2007, 2006), or on *generic weavers* that can be applied to any modeling language with a well-defined meta model: e.g., Mata (Whittle et al., 2009), SmartAdapter (Morin et al., 2009) or Geko (Kramer et al., 2013; Morin et al., 2008). Finally, advanced mechanisms have been proposed to unweave an aspect previously woven (Klein et al., 2009), to finely tune the weaving (Morin et al., 2010), or to weave aspect (or view) instances of different metamodels (Atkinson et al., 2011).

3. Model-Driven Security

Model-Driven Security (Mds) can be seen as a specialization of Mde for supporting the development of security-critical applications. Mds makes use of the conceptual approach of Mde as well as its associated techniques and

tools to propose methods for the engineering of security-critical applications. More specifically, *models* are the central artifacts in every MDS approach. Besides being used to describe the system's business logic, they are used extensively to capture security concerns. Models allow the introduction and enforcement of security in the application being built.

In this section, we start by going back to the roots of MDS to see how it was perceived by the authors who introduced it in the first place. We then then propose a taxonomy for MDS which we later use in this text to evaluate different MDS approaches in the literature.

3.1. A brief history of MDS

Several contributions provided early tentative definitions for MDS. We review them to extract their common features.

Jürjens (2001) started a pioneering work in the domain of MDS with his approach UMLSEC. Based on UML extensions, UMLSEC combines several diagrams for modeling and analyzing systems: class diagrams for the static structure, statecharts for the dynamic behavior, interaction diagrams for object interactions within distributed systems and deployment diagrams to enforce security in the target platform. Although the approach does not explicitly mention the idea of MDE, it had from the beginning the vision that such systems need some kind of analyzing process, early in the development process: the author proposed a preliminary formal semantics (Jürjens, 2002a,b), sufficient for addressing the verification of the targeted security properties.

In 2002, Basin noticed, together with other authors (Lodderstedt et al., 2002), that the MDA approach already has partial answers for the problem of security enforcement: models allow the direct manipulation of business domain's concepts (in this case, business processes); and model transformations enable the automatic generation of executable systems with fully configured security infrastructures. With SECUREUML (Basin et al., 2003), the authors demonstrated the feasibility and efficiency of an MDA-based approach: in the course of the following decade (Basin et al., 2011), the authors applied SECUREUML to various application domains, showing that models are powerful enough to precisely document security and design requirements and to allow their analysis, and that model transformations can successfully generate security-critical systems on different platforms.

Those two seminal works opened the way to an extensive use of MDS. MacDonald (2007) promoted the use of DSLs for each concern, business and

security, and introduced the key idea of Separating of Concerns (SoC): "*the use of visual models or domain specific modeling languages during application design, development and composition to represent and assign security primitives – such as confidentiality, integrity, authentication, authorization and auditing – to application, process and information flows independent of the specific security enforcement mechanisms used at runtime.* (MacDonald, 2007)".

Not longer after, Lang and Schreiner (2008) introduced the necessity of using Domain-Specific Languages (DSLs) for capturing requirements at higher levels of abstraction, and generating code automatically: they view MDS as "*the tool-supported process of modeling security requirements at a high level of abstraction, and using other information sources available about the system (produced by other stakeholders). These inputs, which are expressed in Domain Specific Languages, are then transformed into enforceable security rules with as little human intervention as possible.* MDS *explicitly also includes the run-time security management (e.g. entitlements / authorizations), i.e. runtime enforcement of the policy on the protected IT systems, dynamic policy updates and the monitoring of policy violations.* (Lang and Schreiner, 2008)" They also highlighted the necessity of supporting these security engineering phases by appropriate tools.

From the early beginning of MDS, we see three challenges that are still crucial in secure systems: abstracting away from implementation details to focus on the domain and the security properties (through the use of models); separating, when possible, both specifications (business and security); and using advanced engineering techniques to ensure the correctness of such systems (by automating the generation of the final code, and by applying analysis techniques on the resulting system).

3.2. Evaluation Taxonomy

We now identify and describe a set of concepts pertaining to MDS approaches. These concepts form a taxonomy that complements and extends existing ones (see Khwaja and Urban, 2002; Kasal et al., 2011; Nguyen et al., 2013, further discussed in Section 6). We describe each taxonomy entry, and summarize them in Table 1 for quicker reference.

Application Domains. This entry focuses on the applicability domain of MDS approaches: web applications, e-commerce systems, embedded systems, distributed systems, among others. Often, MDS approaches are very specific

to a domain, using the domain's knowledge to further improve some aspects (like code generated for dedicated distributed frameworks used for web applications), but some are more general. This entry helps comparing our approaches with respect to the application range.

Security Concerns. The European Network and Information Security Agency defines security as "*the capacity of networks or information systems to resist unlawful or malicious accidents or actions that compromise the* availability*,* authorization*,* authenticity*,* integrity *and* confidentiality *of stored or transferred data with a certain level of confidence, as well as the services that are offered and made accessible by these networks*" (Muñoz, 2009). Although this Agency promotes an unified way to handle security, the diversity, scope and range of security concerns and properties make it difficult to handle each of them uniformly. Moreover, some concerns can cover different properties that need to be explicited for better comparing MDS approaches: for example, *authorization* encompasses *access control*, *delegation*, and *obligation* among others, but MDS approaches usually focus on only one of those. In contrast, other MDS approaches might handle more than one "large" security concern simultaneously. This entry helps comparing which security properties can be enforced in a system.

Modeling Approach. As already stated, MDS is a specialisation of MDE: it becomes natural to classify MDS approaches according to their *modeling paradigm* and *modeling languages*. The modeling paradigm addresses the general principle for representing, managing and combining models. For example for standard UML modeling, cross-cutting concerns are scattered across several related models, and are then combined at different levels, such as is typically the case in MDA. Aspect-Oriented Modeling paradigm (AOM) advocates a clear separation of concerns that are subsequently woven into a primary model using a set of model weaving rules, thus obtaining a full model before full code generation. Domain Specific Modeling (DSM) is another slightly different approach, where each model captures specifically the knowledge of a subject domain, that can eventually be cross-cutting. Multi-Paradigm Modeling (MPM) can be seen as an extension of DSM, where different models of computation coexist in a controlled, modeled way. Those paradigms are not completely disjoint, but rather share many features and it is sometimes difficult to distinguish between them, and we often rely on the authors' statements to tag an approach with an underlying modeling paradigm.

There exist in MDE a plethora of modeling languages, and this richness is directly reflected in MDS: standard UML diagrams; UML profiles; tailored DSLs; or formal specification languages (e.g. Petri nets). According to Nguyen et al. (2013), standard UML and UML profiles are the most commonly used in the literature, undoubtedly due to UML's reach within the modeling and engineering communities. Nonetheless, several researchers make use of DSLs with the goal of achieving more customized modeling capabilities. Formal specification languages are not so much represented, most probably because they are perceived as difficult to integrate with standard engineering tools, and because they often require further expertise. However, these languages are often used as back-ends for their analyzing capabilities (see e.g. Armando et al., 2005; Shafiq et al., 2005).

Separation of Security from Business. Separation of Concerns (SOC), a well-accepted design principle in computer science, consists of dividing a system into distinct features (aspects) that are ideally loosely coupled, i.e. their functionalities overlap minimally (Kienzle et al., 2010). Security is a good example: ideally, security concerns should not interfere with other functionalities, but rather complete them (Shin and Gomaa, 2009). Most of the MDS approaches follow this principle (Nguyen et al., 2013), especially when a generative approach like MDA is used: platform-independent models are designed for business and security separately, then transformed into platform-specific models from which the security infrastructure is integrated directly in the system application logic. This entry covers in fact two points: whether each concern is separated from the other and until when, and how the integration is performed.

Model Transformations. Model Transformation is the key ingredient for bridging models with the final code that will ultimately be executed. Its main advantage resides in the fact that it automatically processes tedious and error-prone manipulations on models or code, and can be easily repeated if modifications of models are necessary. It is specially important when different models need to be integrated (in the context of SOC), but also for ensuring consistency between models, and correct refinement of code.

This entry relates to the way transformations are used in the engineering process: Model-To-Model Transformation (MMT) normally serve the purpose (but are not restricted to) of refining models between abstraction levels; and Model-To-Text Transformation (MTT) are traditionally used for generating textual artifacts from models, in particular code, test cases, etc.

Verification. After deriving a number of security properties from security requirements, a key issue is how to make sure those properties hold on the artifacts generated by the MDS approach.

Ideally, a model at a given level of abstraction during an MDS process is *executable*. By executable we mean that the model has well understood operational semantics and can be interpreted by a model checker or a theorem prover such that properties about it can be formally shown. At the lowest level of abstraction the generated code and system infrastructure are by definition executable since they are built to run on a specific platform.

In classical software engineering methodologies, various manual/automatic testing techniques can be applied to the final artifacts (source code or runnable system infrastructure) to check for their correctness: e.g., black-box, white-box or mutation testing, among others (Papadakis and Malevris, 2012). However, despite their success within the developed community, these testing techniques are often error-prone themselves and require a large investment time. Furthermore, testing techniques are often only applied at the final stage of the development lifecycle.

An important advantage of MDS approaches is the possibility of performing security property checking on abstract MDS models. In terms of verification approaches, while *model checking* verifies the conformance of the semantics a model to a specific security requirement expressed as a temporal logic formula, *theorem proving* involves verifying if the system's specification expressed as a theory entails the requirement expressed as a logic formula. Automatic test case generation from abstract models is also a possible verification method.

Traceability. *Traceability* allows to keep track of the history of how models are generated throughout the software lifecycle, and how they relate to each others. It helps in tracing design flaws back to a model when a counterexample is detected during the verification of less abstract model, or errors are found during the testing of the produced system's infrastructure.

Tool Support. Tool support naturally plays a very important role regarding the usability of an MDS approach. By automating error-prone manual tasks the overall quality of the code resulting from the MDS process also improves. Tool support can cover many of the phases of an MDS development process. This includes modeling editors, model transformations editors and engines, checking the syntax and consistency of the system specification, consistency

checks between different levels of abstraction, validation of the specification against user requirements, traceability of errors between layers of abstraction, automatic code generation, automatic testing, among others.

Validation. The validity of a particular MDS approach is evaluated by analyzing the number and the scale of the case studies an MDS approach is applied to. For each approach we will evaluate how many proof-of-concept and/or industrial case studies exist, what are the conclusions of those studies.

Table 1 summarizes the taxonomy entries (together with a brief description) we identified in this Section. Although described separately, these entries are largely related and depend on each others. For example, an MDS approach using *separation of concerns* to distinguish business logic from security concerns would require further in the development process to combine models of each side, which related to the *modeling paradigm* underlying it. As another example, the *verification* methods used depend on the *modeling approach* that has been taken, and in particular on the modeling languages used. If languages with formally defined operational semantics are used in the MDS lifecycle, then model checking can be used as a *verification* technique on models of those languages.

4. Evaluation of Current Model-Driven Security Approaches

In Section 3.1 we have explored various MDS definitions in the literature. These definitions have helped helps us to identify MDS studies among a mass of security-related work in the literature.

In this section, we evaluate five MDS approaches selected from the literature against the taxonomy defined in Section 3.2. In order to select which approaches are part of our set we have based ourselves on the *popularity* of the approach. We have measured *popularity* using the number of citations of the major publications for the approach and how that approach stands in recent surveys (Jensen and Jaatun, 2011; Kasal et al., 2011; Nguyen et al., 2013).

The remainder of this section presents the result of our evaluation of the five selected approaches: UMLSEC, SECUREUML, SECTET, MODELSEC and SECUREMDD.

Table 1: MDS Taxonomy Entries

Taxonomy Entry	**Description**
Application Domains	Is the MDS approach domain specific or generalistic?
Security Concerns	Which concerns does the MDS approach focus on? Are they expressible in a metamodel?
Modeling Approach	Which Modeling paradigm(s) (MDA, AOM, DSM)? Which Modeling language(s) (standard UML; UML profiles; DSLs; Formal Languages)?
Separation of Concerns	Is it used? If it is, how is it implemented?
Model Transformations	Are MMT/MTT used? Which model transformation engine is used?
Verification	Which verification techniques are used: model checking; theorem proving; testing
Traceability	Are *backward* and/or *forward* traceability implemented? Is traceability automatic or manual?
Tool Support	What is the automation level of the MDS approach? Which features does it provide?
Validation	Is the approach validated on large, meaningful cases? Has it been industrially validated?

4.1. UMLsec

Application Domains. UMLsec (Jürjens, 2001; Jürjens, 2002c, 2004, 2005b; Best et al., 2007; Hatebur et al., 2011) is a Uml profile extension for the analysis of security-critical systems. Uml *stereotypes* are used with annotations called *tags* in order to specify security requirements and assumptions. Additional *constraints* attached to the *stereotypes* provide the means to understand when security requirements are violated. UMLsec takes advantage of the wide spread use of the Uml as a general purpose modeling approach and can be applied to model and analyze security concerns from a wide range of domains, including *web applications* (Houmb and Jürjens, 2003), *embedded systems* (Jürjens, 2007) and *distributed systems*.

Security Concerns. UMLsec deals with a relatively large number of security requirements: *confidentiality*, *integrity*, *authenticity*, *authorization*, *freshness*, *secure information flow*, *non-repudiation* and *fair exchange*.

UMLsec concentrates on providing the means to analyze the enforcement of security concerns in system models specified in the UMLsec profile. As such, the approach requires building attacker models, also called *Adversary Machines*, to execute adversary behaviors during system analysis. For example, in order to analyze the *integrity* security property the system is jointly executed with a particular adversary machine that attempts to assign an erroneous value to a system variable. The *integrity* property holds if no *constraints* in the UMLsec model associated to the variable are violated during the attack.

UMLsec is relatively different from other Mds approaches in the literature given that it concentrates on providing analysis capabilities for security models, rather than insisting on explicit languages for modeling security concerns. Although clear formal semantics have been defined in for the Uml fragment used in the UMLsec profile (Jürjens, 2001), no explicit metamodel has been provided.

Modeling Approach and Separation of Security from Business. UMLsec does implement the principle of separation of security concerns from business logic, but in a manner that differs from the definition of *Separation of Security from Business* we have provided in section 4. In Fig. 4 we provide a graphical depiction of the UMLsec methodology, where:

- *Stereotypes* and *tags* are used to model and formulate security requirements in the *system models*;

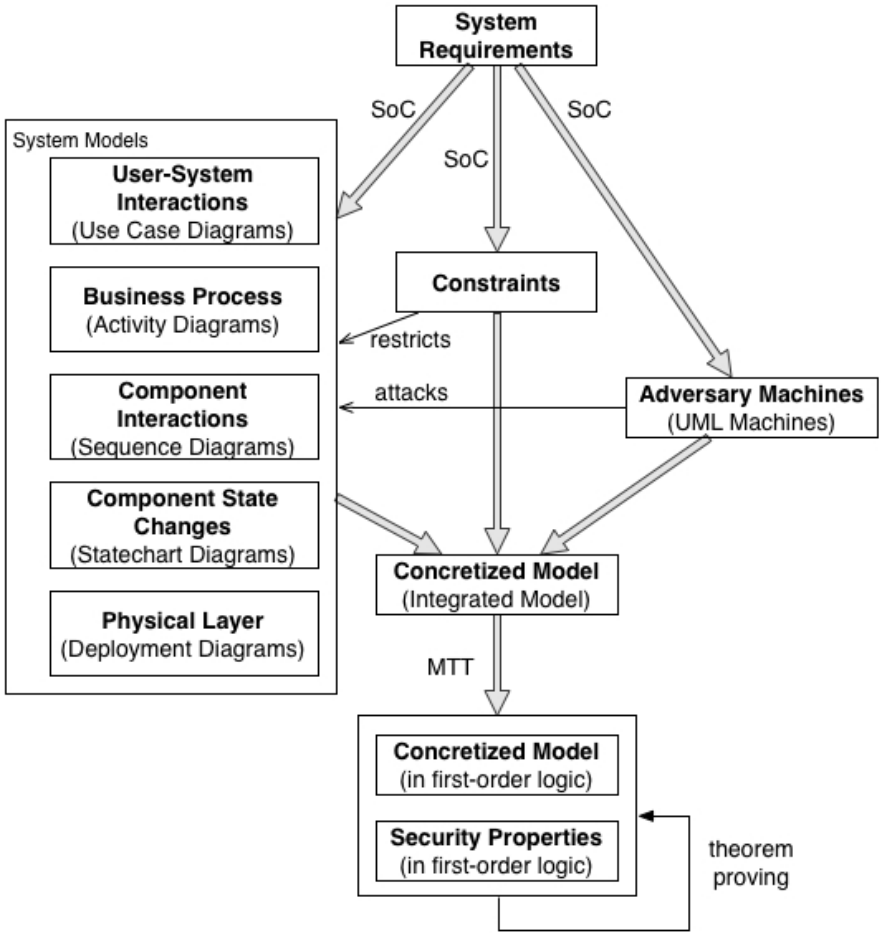

Figure 4: Overview of the UMLsec approach

- Potential attacks are modeled by *adversary machines*, independently from system models;
- *Constraints* attached to the *stereotypes* contain criteria for checking whether a security requirement violation is caused by *adversary machines*;
- *Security* and *recovery* mechanisms are defined together with the business logic in the system models.

Due to the fact that the security mechanisms are deeply coupled with system models, the separation of concerns in UMLsec is relatively weak

when compared to other MDS approaches we survey in this chapter. For example, in order to prevent a certain attack, it may be necessary to add a new condition guard on a particular transition in an activity diagram which is a subsystem specification covering non-security related functions. This being, such specification cannot be dedicated to security specialists only.

Regarding system modeling, UML diagrams are employed to model the different perspectives of the system, where each diagram kind is used to define a subsystem specification (see Fig. 4):

- **Use Case Diagrams** are used to capture user-system interactions;
- **Activity Diagrams** specify the business workflow of the system;
- **Sequence Diagrams** concentrate on component interactions, such as inter-component data flow and protocol;
- **Statechart Diagrams** model the dynamic behaviors of components;
- **Deployment Diagrams** are used to model communication links among components such as the client, server, database, etc.

Note that the fragment of the UML used by UMLSEC as well as the *adversary machines* are provided a precise semantics using UML *Machines* (Jürjens, 2005b). These machines are a type of state machine with input/output interfaces for which behavior can be specified in a notation similar to that of the well known Abstract State Machines. A composed model (called *concretized model* in UMLSEC) can be created by weaving the *constraints* and *adversary machines* with the *system models*. The concretized model is the system design model that can be used for further security analysis.

Model Transformations and Tool Support. UMLSEC does not explicitly use model transformations during the development process.

Renaming is used for model composition of the *system models*, *constraints* and *adversary machines* (see Fig. 4). Renaming is a form of mapping used by UMLSEC that associates *stereotypes*, *tags* and *constraints* such that the three separate models can be composed in a *concretized model*. From the *concretized model*, a first-order logic theory is generated and security properties can be proved about it by using a dedicated PROLOG-based tool called AICALL.

UMLsec's toolset does not generate code from the models, given the purpose of the approach is to provide analysis capabilities. It is possible to apply the approach to existing systems by reverse-engineering the code into UMLsec models (Best et al., 2007).

Verification. As previously mentioned, the UMLsec approach focuses on the security analysis of the system's design. For this purpose, UMLsec composes the behavioral model of the system with that of the potential attackers of the system. The integrated model is compiled into first-order logic axioms which can be verified by aiCall. A Prolog-based tool is then used to verify the system's security requirements. When an attack is feasible, i.e. a security concern can be violated, the tool automatically generates an *attack sequence* showing how the attack may occur.

Traceability. In UMLsec, the security analysis of the system reveals the security concerns that may be violated by potential attack sequences. This information may be used to trace security-related design flaws back to the models. However, no tool-supported automatic traceability is provided.

Validation. UMLsec has been applied for analyzing the security of several industrial applications, such as a biometric authentication system (Jürjens, 2005a; Lloyd and Jürjens, 2009), the Common Electronic Purse Specifications (CEPS) project (Jürjens, 2004), a web-based banking application (Jürjens, 2005c; Houmb and Jürjens, 2003), the embedded system described in (Jürjens, 2007) and the mobile system described in (Jürjens et al., 2008). Most case study results are positive regarding the benefits of the UMLsec approach.

4.2. SecureUML

Application Domains. In 2002 Basin et al. have proposed SecureUML (Lodderstedt et al., 2002; Basin et al., 2003, 2006, 2011) to allow knitting system models with security concerns. This is achieved using Uml-like modeling languages for system modeling and a *Role-Based Access Control* (Rbac) language (Sandhu et al., 1996) to describe security policies. All languages are defined as Uml profiles. The framework is sufficiently general to be applied to other system model or security languages, but for that new system modeling and security description languages need to be defined as Uml *profiles*. Given its generality, SecureUML can be applied to many domains.

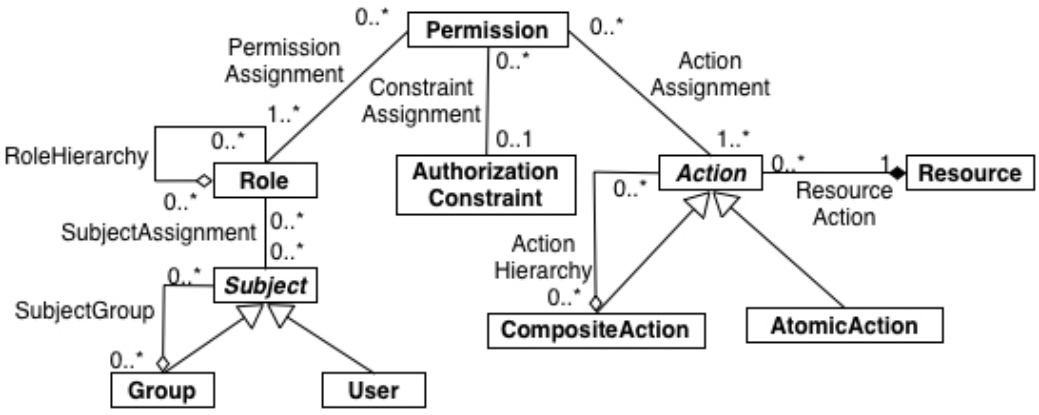

Figure 5: SecureUML metamodel (adapted from Basin et al., 2006)

Nonetheless, the several concrete examples presented in the literature pertaining to SecureUML refer to web applications.

Security Concerns. The SecureUML approach is focused on access control, in particular Rbac. The metamodel for SecureUML is depicted in Fig. 5 and consists of an extension of Rbac's metamodel. While Rbac's central concepts are Subject, Role and Permission, in the metamodel in Fig. 5 the additional notions of Resource and Action are introduced. A Resource is, as the name indicates, an abstraction of a system's resource where security should be enforced. Actions corresponds to activities that can be performed on a resource. By enforcing Rbac Permissions on Actions, Basin et al. are thus capable of attaching security policies to the resources described in a separate *system model*.

All examples found in the SecureUML literature use an Rbac security language. To point out the approach is not restricted to access control, the authors of (Basin et al., 2006) sketch the applicability of SecureUML to other access control methods such as *Chinese Wall* policies or the *Bell-LaPadula* model. Later, in (Basin et al., 2011) the authors mention the potential modeling of *usage control policies* within SecureUML.

Modeling Approach and Separation of Security from Business. We present in Fig. 6 a graphical depiction of the SecureUML approach. *Separation of Concerns* is naturally embedded in the framework as can be seen by the explicit separate modeling of the business and the security Rbac

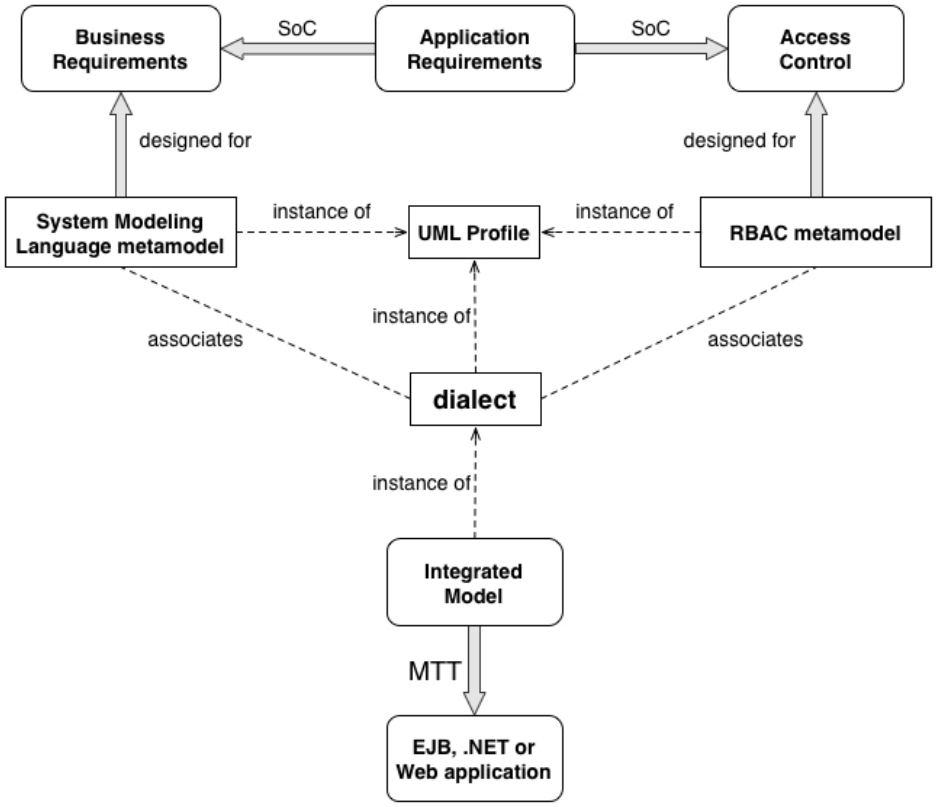

Figure 6: Overview of the SecureUML approach

concerns. Basin et al. use in their work two main Uml-based modeling languages for system modeling: simplified Uml *Class Diagrams* and simplified Uml *Statecharts*.

Composing the business and security models is achieved by a *dialect* language, linking the system modeling language to the access control Rbac language. This dialect language is also defined as a Uml profile and allows specializing the Resource and Action classes in the SecureUML metamodel in Fig. 5 into the several concepts of the system modeling language that require security.

For example, Basin et al. defined in (Basin et al., 2003) a statechart-like system modeling language. Its protected resources naturally include State and StateMachineAction and the actions on those resources are e.g. Execution or Activation. The dialect language specializes the Resource and Action classes in the SecureUML metamodel in Fig. 5 into the State and StateMa-

chineAction concepts of the statechart-like system modeling language. By specializing also the Action class into actions that can be specifically performed on the statechart's resources a composed *dialect* language is created. Naturally, if either the system modeling language or the security language change, a new dialect language will need to be created. *Integrated models* can then be built as instances of the dialect language.

Model Transformations and Tool Support. From the integrated model in Fig. 5, code can be automatically generated. In (Basin et al., 2006) the authors describe both Enterprise Java Beans (EJB) and .NET systems can be generated from a dialect model of a composed SECUREUML and a UML class diagram-like language called COMPONENTUML. The instances of the code generation technique map the security language's concepts into built-in security mechanisms in EJB or .NET. For example, for the EJB platform roles and permissions are mapped into an EJB security infrastructure based on RBAC. Additionally, Java assertions are used to enforce authorization constraints.

Despite the fact that code generation can be seen as a model transformation, the generation of EJB and .NET code is not accomplished using a model transformation language. Basin et al. refer nonetheless in (Basin et al., 2011) to the usage of QVT to automate the composition of several separate parts of a system model with a single security model. The goal is to avoid redundant information scattered in several models. An example of this kind of composition for a web application can be found in (Basin et al., 2010).

Verification. In (Basin et al., 2006) a proof sketch of the code generation procedure for EJB from a model of secure *ComponentUML* is presented. The proof is a correct-by-construction argument for the soundness of the EJB code generated from the secure model. If other system modeling languages are considered then similar proofs will need to be provided, taking into consideration the new target platform for code generation.

The verification of secure models is the subject of (Basin et al., 2009), where the authors describe the SECUREMOVA tool. The tool allows verifying security related properties about integrated models (see Fig. 6). Security properties are expressed as OCL constraints and regard relations between users, roles, permissions, actions and systems states. Coming back to our statechart-like system modeling language, it is for example possible to show using SECUREMOVA that a certain user will be able to activate at least once a certain state in the model of the system.

Traceability. No explicit traceability exists in SecureUML.

Validation. Several simple examples of using SecureUML can be found in (Basin et al., 2006). A large E-Commerce J2EE "Pet Store" application has been described by Lodderstedt (2003) in his Ph.D. thesis. Finally, Clavel et al. (2008) reports about the applicability of SecureUML to an industrial case study. The conclusion is that the SecureUML approach is useful for understanding the manipulated concepts, in performing early analysis and fault detection, and improves reusability and evolvability. The study also mentions that while access control proved relevant in the case study, it is merely one of the several security concerns that the industrial partner demonstrated interest in.

4.3. Sectet

Application Domains. Sectet (Alam et al., 2007a; Hafner et al., 2006b; Alam et al., 2006c,a; Hafner et al., 2008; Breu et al., 2007) is an extensible Mds framework for supporting the design, implementation and management of secure workflows for social structures such as *government*, *health* or *education* (Breu et al., 2005; Hafner et al., 2005). The framework assumes a distributed peer-to-peer technological space based on the concept of Service-Oriented Architecture (Soa) and implemented as web services.

Security Concerns. Sectet handles two categories of security policies:

- *Basic Security Policies*: *integrity*, *confidentiality* and *non-repudiation* for messages passed among components of the distributed system. By *non-repudiation* we mean that correct messages are not discarded;
- *Advanced Security Policies*: *Static* and *dynamic* Role-based access control (Rbac).

Regarding the three basic security policies, their implementation is achieved in Sectet by using known and proven mechanisms for message passing in peer-to-peer systems. Existing component communication protocols at the level of the web-services are used to enforce *confidentiality* and *integrity*. Cryptography is used to implement *non-repudiation*. The authors of Sectet have thus concentrated their efforts on the modeling, analysis and enforcement of *access control* policies, especially dynamic access control constraints, and on how to integrate those policies with system models expressed in the

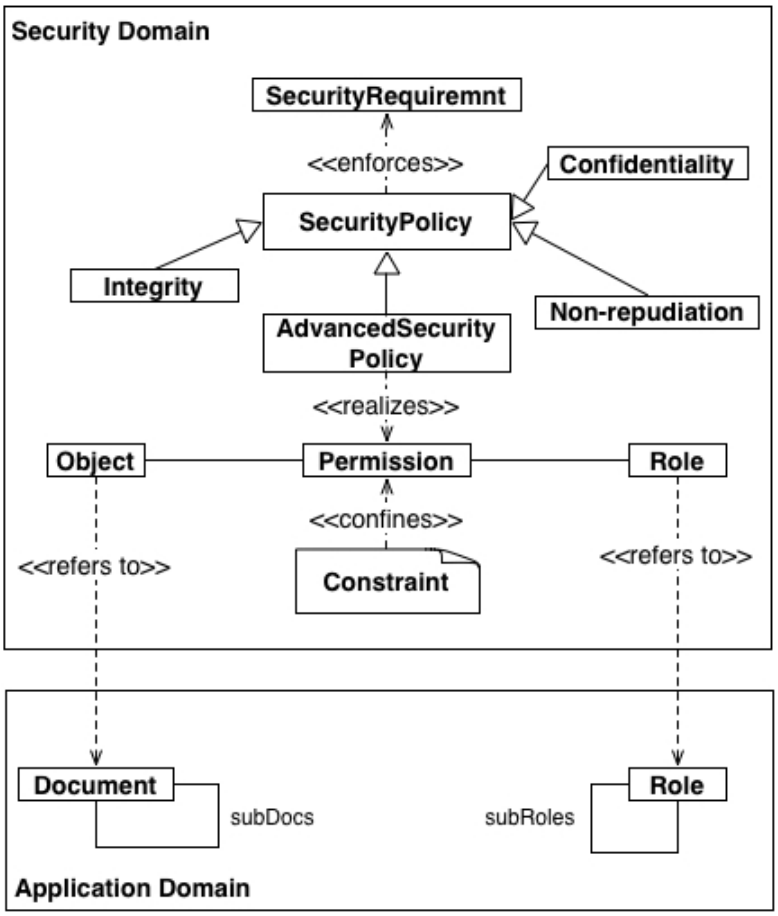

Figure 7: The metamodel of RBAC in the SECTET framework (adapted from Hafner and Breu, 2009)

UML (Breu et al., 2007; Agreiter and Breu, 2009; Hafner et al., 2008; Alam et al., 2006a,b).

We illustrate in Fig. 7 the metamodel of RBAC policies in the SECTET framework. It consists of a standard RBAC metamodel including the notions of Object, Role and Permission, extended by dynamic constraints defined on the permission. The Role and the Object (resource) are mapped to the corresponding entities in the application domain metamodel. Other than *access control*, SECTET can be extended to deal with other advanced security policies, such as *availability* (Hafner and Breu, 2009), *delegation of rights* (Alam et al., 2006c) or *trust management* (Alam et al., 2007a,b). In order to do

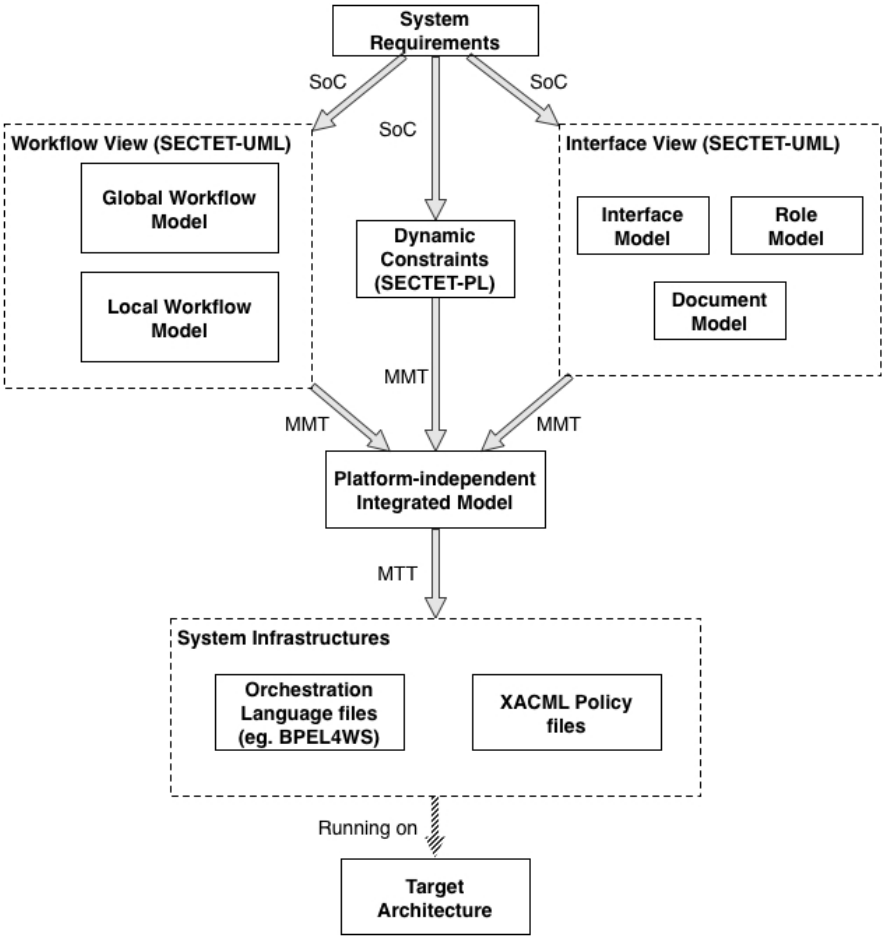

Figure 8: Overview of the SECTET methodology

this, the lower part of the metamodel in Fig. 7 needs to be extended in order to *realize* other instances of the AdvancedSecurityPolicy class.

Modeling Approach and Separation of Security from Business. The SECTET framework makes use of a methodological standard (Model-Driven Architecture), an architectural paradigm (Service-Oriented Architecture) and a technical standard (Web Services). It uses UML profiles to create two languages: 1) a system modeling language SECTET-UML and 2) an Object Constraint Language (OCL)-style predicative language SECTET-PL.

SECTET-UML is used to model business requirements and static security requirements. These are expressed in three kinds of workflow views (as shown

in Fig. 8):

- *Global Workflow*: The global workflow represents a virtual and distributed inter-organizational workflow, which models an abstract view of interactions among partners (organizations);

- *Local Workflow*: The local workflow represents an intra-organizational workflow which represents the execution of a business process by a particular service component. A local workflow model can be used as an input to a workflow management system;

- *Interface View*: The interface view presents the properties and permissions of each service component in the system. It incorporates three sub-models, i.e. the *interface model*, the *document model* and the *role model*, corresponding to the publicly visible part of the *local workflow* which is accessible to the inter-organizational *global workflow*.

SECTET uses a subset of the metamodel of UML 2.0 Activity Diagrams to model both *global workflows* and *local workflows*. Stereotyped UML Class Diagrams are used to model the *interface view*.

Dynamic security requirements such as access control constraints are expressed in the SECTET-PL language, as shown in Fig. 8. In SECTET security policies are separated from the business logic at the language level as can be seen from the fact that the elements in the *Security Domain* part of the SECTET metamodel in Fig. 7 *refer to* elements in the *Application Domain* metamodel. As such, the SECTET framework implements the principle of separation security concerns from business logic.

As can also be seen in Fig. 8, an integrated *platform independent application model* (PIM) is built by composing SECTET-UML models with the dynamic security requirement expressions in SECTET-PL. Verification activities can then be performed on this PIM.

Model Transformations and Tool Support. As shown in Fig. 8, in the SECTET methodology the system requirements are first concretized into three separate modeling views: the *workflow view*, the *dynamic constraints* and the *interface view*. Model construction at this level is supported by *MagicDraw*, which allows exporting the UML models to XMI files.

Model-to-Model Transformations (MMT) are then applied to integrate these models. Model composition based on annotated models is used to

integrate SECTET-UML models with the dynamic security requirement expressions SECTET-PL in order to form a platform-independent application model (PIM). These MMT transformations rely on the transformation language QVT. From the composed model, Model-to-Text transformations are used to generate two kinds of system infrastructure: 1) the orchestration files generated from the workflow models, and 2) the XACML policy files for security configuration. Regarding the XACML artifacts generation, the XPAND model transformation language is used. The produced infrastructure can then be executed by a workflow engine (the *Target Architecture* in Fig. 8).

Verification. No verification method has been used in SECTET.

Traceability. Traceability has not been defined or implemented in SECTET.

Validation. Various case studies from the healthcare and e-government domains can be used to validate the SECTET framework in real life scenarios. In (Breu et al., 2007; Agreiter and Breu, 2009; Hafner et al., 2008; Alam et al., 2004; Breu et al., 2005; Hafner et al., 2005; Alam et al., 2006a,b), SECTET is mainly used to deal with RBAC policies in web applications. Also, Alam et al. (2006c) depict how to handle delegation of rights in SECTET and case studies in (Alam et al., 2007a,b) show SECTET can handle *trust management*. Additionally, in (Hafner et al., 2006b,a) the authors also illustrates the extensibility of the SECTET framework.

Regarding the complexity reach of the case studies, (Hafner et al., 2008; Breu et al., 2008) conducted a large research project called *health@net* involving many of academic and industrial partners. The project's goal was to handle complex healthcare scenarios based on *Usage Control* and dealing with multiple advanced access control policies such as dynamic *access control* or *delegation of rights*.

4.4. MODELSEC

Application Domains. In Sánchez et al. (2009) the authors illustrate the MODELSEC approach using an example taken from of a web application for the management of medical patients (Fernández-Medina and Piattini, 2005). The core of the example is the design of a secure database where the authors show how MODELSEC deals with access control and database security code. Even though the authors demonstrate their approach via the secure database example, to the best of our knowledge the MODELSEC approach is not restricted to a particular application domain.

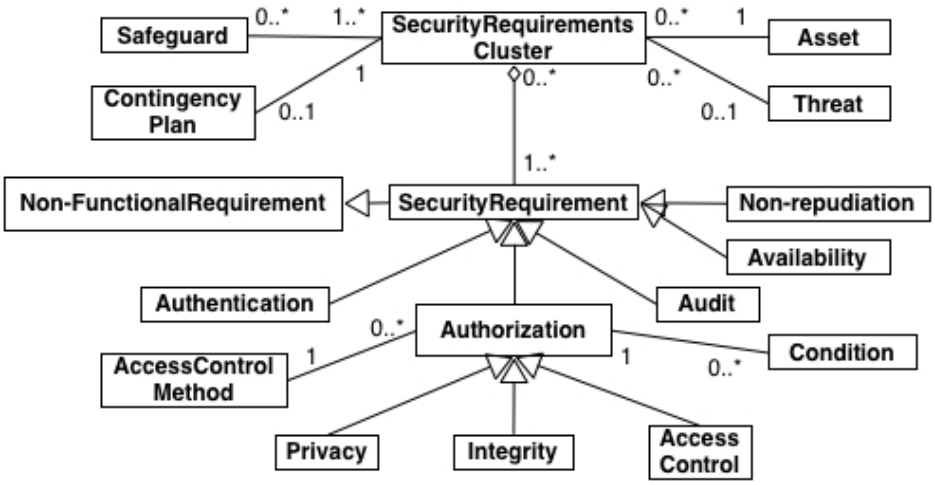

Figure 9: Security Requirements metamodel, part of the SecML Dsl of the ModelSec approach (adapted from Sánchez et al., 2009).

Security Concerns. ModelSec supports defining and managing security requirements by building security requirements models for an application from which operational security models can then be generated. The security requirement models encompass multiple security concerns in an integrated fashion, including *privacy, integrity, access control, authentication, availability, non-repudiation* and *auditing*.

Figure 9 depicts the ModelSec Dsl for designing security requirement models. A complete SecurityRequirementsCluster consists of the following components:

- An Asset is a physical or logical object that may be exposed to threats;
- A Threat is an asset that can be damaged by a *threat*;
- A Safeguard is a measurable risk impediment, or an action against a risk;
- A Contingency Plan consists of a set of safeguards recommended for reducing risks;
- A Security Requirement expresses the security concerns the final system should implement or guarantee.

Modeling Approach & Separation of Concerns. ModelSec proposes a general Requirement metamodel (not presented here because it is not specifically targeted to security requirements, but visible in Sánchez et al. (2009)) that is extended with the Security Requirement metamodel of Fig. 9. The approach is based on Uml and Mda: for example, business concerns are represented using use case and class diagrams, and models are kept separated from platform details.

Figure 10 depicts the ModelSec approach. Business and security models are clearly separated from the very beginning, and become synchronised at several steps of the process. Model transformations produce two types of models from top-level requirements models: security and business design models (Sánchez et al., 2009), which can capture design decisions. Therefore, requirement models specify *what* restrictions the system must satisfy, whereas the design models specify *how* these restrictions will be satisfied. Up until the design models the process is platform-independent. The application code is directly generated from the business design model, by including platform-specific information, producing a security implementation model.

Sánchez et al. (2009) describe the entire ModelSec approach. However, the authors do not provide sufficient details regarding the way things are integrated together, and the way security policies can be synchronized with the application code.

Model Transformations and Tool Support. ModelSec leverages model transformations extensively to generate the necessary artifacts from models.

Model-To-Model Transformations (Mmt) plays a key role in ModelSec as shown in our synthesis in Fig. 10. Mmts are used to transform the analysis models (security requirement model and conceptual model) to design models (security and business design models), as well as to transform the security design model to the security implementation models. Mmt transformations in ModelSec are written in RubyTL, a transformation language embedded in the Ruby programming language.

Model-To-Text Transformations (Mtt) are used for transforming the implementation and design models to security infrastructure and application code respectively. The Eclipse MOFScript template language was chosen to implement Mmts in ModelSec.

Verification, Traceability & Validation. Verification has not been addressed in the ModelSec approach. The ModelSec approach does not

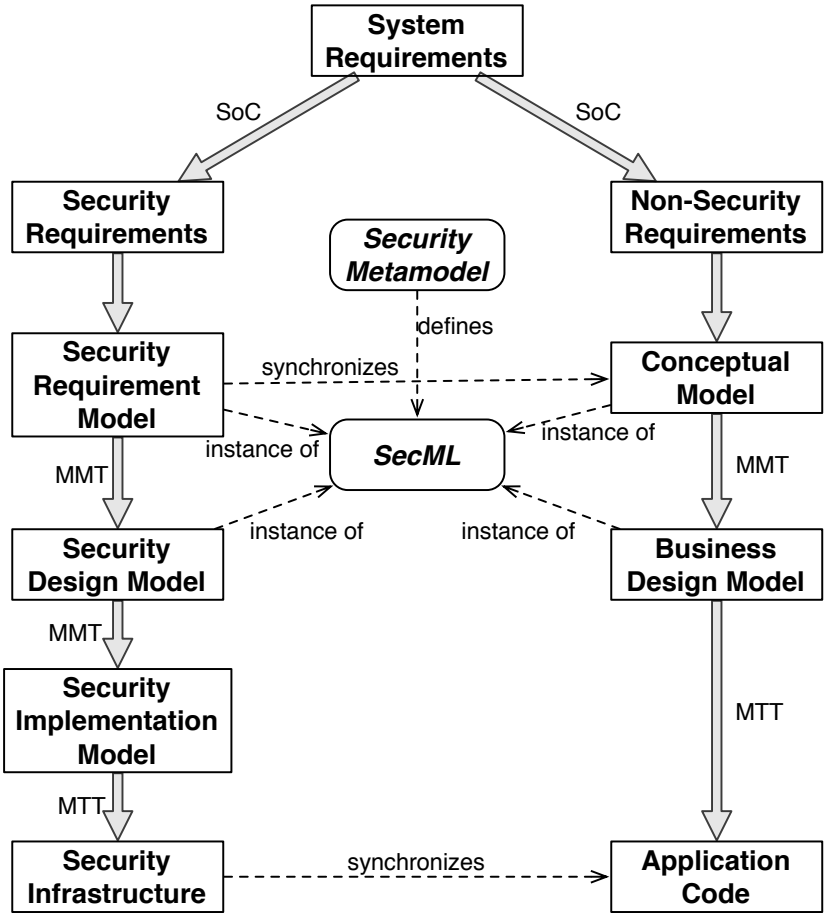

Figure 10: Overview of the ModelSec approach

mention anything about traceability support, and is only illustrated on an academic case study (Sánchez et al., 2009).

4.5. SecureMDD

Application Domains. SecureMDD (Moebius et al., 2009a,b,c, 2010, 2012) is another Uml-based approach aiming at facilitating the development of security-critical embedded applications based on, or built upon cryptographic protocols. In this application domain the protocol is often subject to attacks known as *third-party intruder*. In this scenario an attacker that can intercept and read messages, breaching confidentiality. It then proceeds

deletes those messages or forges new fake messages, thus compromising authenticity, privacy and so on between an user and a third-party server.

Although the authors claim that their approach can easily be generalized to other context, SecureMDD targets applications with the following characteristics. A *Terminal* is a device that has the ability to read and communicate with *Smart Cards*, which are similar to credit cards, but can store private data about its owner (e.g. medical data, or payment facilities for online transactions). When accessing data on the Smart Card, or contacting third-party institutions such as a health center to access data for a patient, the *Terminal* makes use of identified protocols that need to be trusted to ensure the integrity of the whole system.

The authors have worked on applications of different sizes: starting from small, simple ones, they improved their approach and proposed a development methodology for handling industrial size applications, while ensuring the formal verification of the resulting code.

Security Concerns. SecureMDD handles general-purpose security properties, like *secrecy*, *data integrity*, *confidentiality*, among others, that are common to all applications in this domain. Strictly speaking, SecureMDD does not use a dedicated language for specifying security properties. Instead, properties are specified directly using a logic language, called the *Dynamic Logic*.

Modeling Approach and Separation of Security from Business. SecureMDD combines Uml profiles for the structural part of the system and Uml sequence and activity diagrams for capturing behavior. Mel, the Dsl created by the authors, allows defining fine-grained behaviors over Uml diagrams and interacting with cryptographic protocols. It has a textual representation, possibly more suitable for early experimentation when designing the application; it also has a graphical representation, aimed at being integrated with Uml diagrams.

Since SecureMDD enables formal verification by means of theorem-proving, all languages have a formal semantics: the semantics for the various Uml diagrams used for designing the application has been defined in Kiv[1], together with a semantics for Java, the target language chosen for generat-

[1] Web page for the Kiv Theorem-Prover: http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/

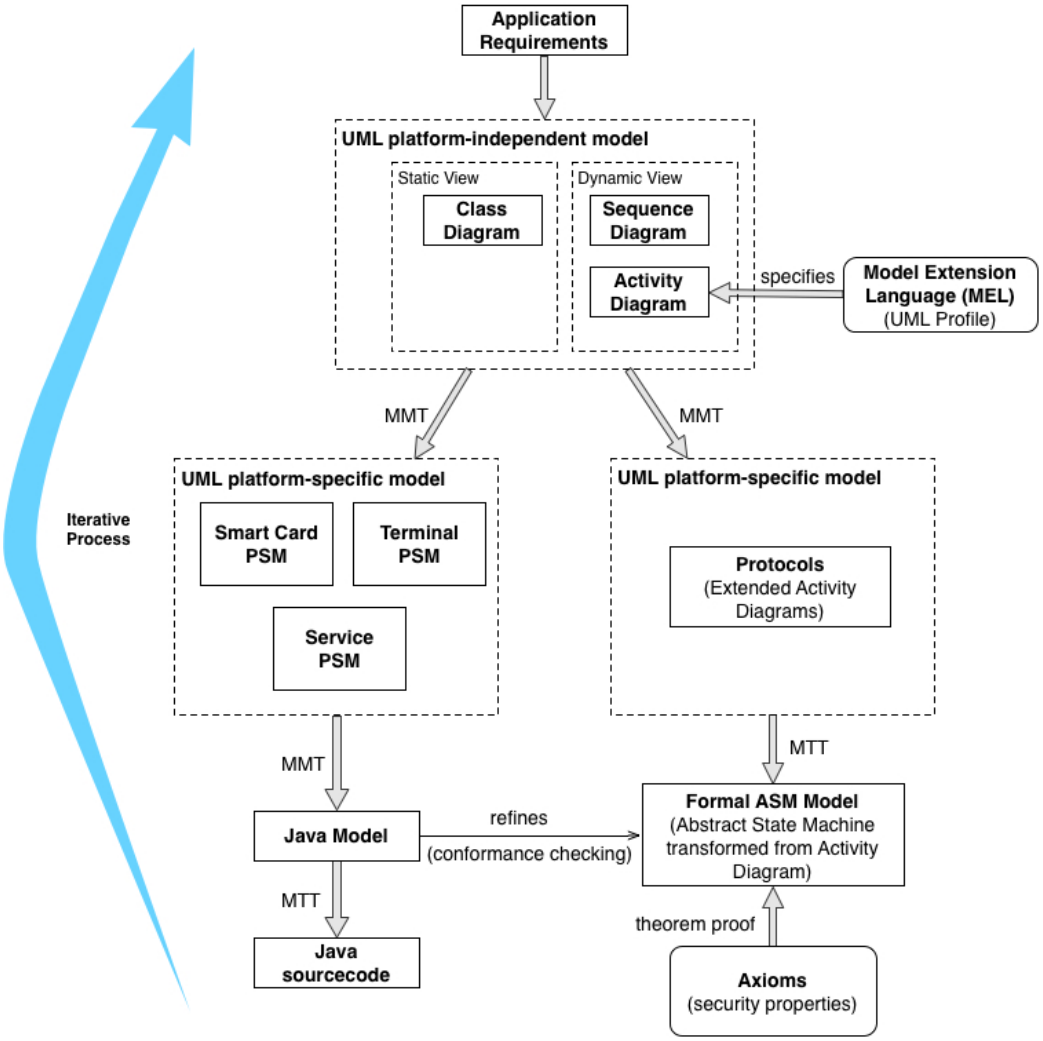

Figure 11: Overview of the SecureMDD approach (inspired from Moebius et al., 2009b, 2012).

ing code from Uml. These semantics are expressed in terms of Asms, the formalism the authors have chosen for performing verification.

Figure 11 depicts the general approach for SecureMDD. The design process starts with the creation of a Uml model for the functional requirements of the application. Class diagrams describe the different application entities (terminals and cards, as well as the users and attackers, and the protocol communication infrastructure), whereas sequence and activity diagrams

model the system's behavior and the interactions between those entities. The authors provide concrete examples of such modeling artifacts: Moebius et al. (2009b) shows such diagrams for a copy card application; Moebius et al. (2012) provides several diagrams for another SecureMDD project implementing security for the German electronic health card.

The language Mel allows engineers to express the processing of messages: for example for the copy card application, how the state of a component changes, how encryption/decryption operations interact with predefined cryptographic operations, etc.

Moebius et al. (2012) also proposes a methodology for designing applications in an interactive way, which is more suitable for industrial sized applications. Since new functionalities are added progressively, after having checked that the application resulting from an iteration works as expected, the way Uml diagrams and security property specifications are modeled and handled can quickly become a challenge in itself. Iterative development can also have an impact on the security properties, since new functionalities can break previously established proofs. The authors describe how they developed an important case study, the German electronic health card system, after three iterations, while managing to minimally interfere with the previous models and reusing the previous proofs.

Separation of Concerns does not strictly apply to SecureMDD, since security properties are defined on a platform-specific model generated from Uml diagrams.

Model Transformations and Tool Support. SecureMDD makes extensive use of transformations to reach executable code from platform-independent models that capture early requirements. Mmt techniques are used to transform the platform-independent models into two different models: on the left of Fig. 11, a Java model for execution, while on the right, an Abstract State Machine (Asms) model to be used for verification purposes. The transformations integrate specific information to produce adequate code, in particular by following the target programming languages paradigms (object-orientation for JavaCard, algebraic specifications for Asms). In particular the specific built-in data types and values, the configuration settings, the serialization mechanism for protocol exchange messages, the encryption/decryption specific algorithms, among other implementation details, are taken into consideration.

From these models, Mmt techniques are used for generating the actual

code necessary for execution and verification. These transformations are implemented with XPAND, a statically typed template language focusing on the generation of textual artifacts. Note that only the security-critical part of the application is generated; other components that are less critical, and more subject to evolution and modification, need to be programmed and integrated with the trusted code: for example, database access and the user interfaces construction are left to programmers (Moebius et al., 2009c,a).

The authors of SECUREMDD do not specifically describe tool support for their approach; however, several papers provide insights on the modeling artifacts, the transformation chains and the verification procedure, in particular Moebius et al. (2009c, 2012).

Verification. In SECUREMDD, verification is performed regarding two aspects: verifying that security properties effectively hold within the models, and verifying that the generated Java model is a correct refinement of the ASM model. The refinement correctness proof is handled at the level of the specific models for Java and ASM, instead of the original UML diagrams. This way, all the information about the security concerns and the functionalities of the application are available, and fully executable in each respective tool. SECUREMDD reuses existing work by Stark et al. (2001) that provides a formal semantics to Java and its Virtual Machine in terms of ASM. Since KIV is built on top of ASM and integrates Java constructions, the proof is facilitated, although it remains interactive as theorem-proving always is (Moebius et al. (2010) reports a period of several weeks to build the proof for the copy card application, which can be considered, according to the authors, as a small application). The authors are currently working on extending this approach: they plan to define a calculus for QVT, the language they use for MMT, that will allow them to formally prove in KIV the correctness of such transformations.

The verification of security properties is handled by KIV, a theorem-prover based on ASMs. Security properties are expressed using a dedicated logic, the Dynamic Logic, tailored to algebraic specifications which constituting the mathematical background of ASMs. They noticed an interesting fact: application-specific security properties generally give better guarantees than standard properties (like secrecy or integrity), but these general-purpose security properties still need to be proved, and are in fact often required as a background for specific ones. For example for the German health card, ensuring that only a qualified doctor can issue a prescription, implies that

the prescription becomes a secrecy for any intruder (except of course, the patient). Moebius et al. (2012) also provides a methodology for facilitating iterative verification: by following simple guidelines (e.g., specifying system invariants into several pieces to be able to re-prove most of them, or modifying protocols' behavior only when necessary), they were capable of re-running most of their proofs in further incremental steps. It remains to see if this methodology is applicable beyond their application domain.

At a later stage, SecureMDD introduced the possibility to define test cases directly on Uml (Moebius et al., 2012). Using Model-To-Test transformations, these test cases can be executed on the generated Java code to validate scenarios not handled by the verification process, such as interactions with the user interface for which code is not automatically generated.

Traceability. SecureMDD uses a generative approach: once the composed model is verified for security properties, the relevant part of the code corresponding to the security-related part of the system is generated. Therefore, all security-related flaws are caught early, and the approach does not require *backward* traceability.

Validation. SecureMDD seems to be a promising approach: when it was first proposed in 2009, only small-sized proof-of-concept case studies were addressed (e.g. the Mondex protocol for electronic payment (Moebius et al., 2009a)). It then evolved towards a medium-sized case study, the copy card application (Moebius et al., 2010), where more advanced application-specific security properties were handled. As a last step, the authors were capable of handling an industrial scale project, the German electronic health card application (Moebius et al., 2012). These results were obtained in a relatively short time (from 2009 to 2012)[2].

5. Discussion

This section synthesizes our evaluation in a comparison table, and discusses some open issues and current challenges in Mds.

[2]The webpage of the project provides details about other projects: https://www.informatik.uni-augsburg.de/de/lehrstuehle/swt/se/projects/secureMDD/

5.1. Evaluation Synthesis

We revisit now the MDS approaches we evaluated in Section 4 to extract a synthesis regarding our taxonomy of Section 3.2. These considerations are summarised in Table 2, where column corresponds to a taxonomy entry and each row to one of the MDS approaches we selected.

Application Domains Most of the evaluated MDS approaches are designed to be "general purpose", i.e., their objective is to address a large range of application domains. One exception is SECUREMDD which only focuses on supporting the development of smart card and service applications.

Security Concerns Only SECUREUML concentrates on one specific concern, i.e. access control. The SECUREMDD approach mainly deals with cryptographic protocols but also targets some application-specific security properties. The other approaches, i.e. UMLSEC, SECTET, and MODELSEC are open to deal with multiple security concerns.

Modeling Paradigm All approaches combine the model-driven architecture paradigm with domain-specific modeling paradigm. One exception is UMLSEC, which applies the multi-paradigm modeling paradigm (MPM). This is do since different views of a UMLSEC system are modeled by different UML diagrams and finally all these views are composed to form the complete specification of the system. However, no platform specificity or independence is explicitly modeled. MPM can be regarded as encompassing MDA or DSM, while not restricted to any of those paradigms.

Modeling Language Only MODELSEC uses a DSL based on a non-UML-based language. The other approaches use DSLs defined as UML profiles.

Separation of Concerns Only UMLSEC does not fully follow it. Indeed, in UMLSEC security-related information is partially contained in the business models. The other MDS approaches we have analyzed either encapsulate security concerns in one specific kind of model artifact (e.g. SECUREMDD), or clearly separate security concerns from business logic at the metamodel level (e.g. SECTET).

Model Transformations ModelSec defines its own Mmt tool by extending the Ruby programming language and uses Mof-Script for Mtt. Sectet and SecureMDD leverage the well-known existing model transformation tools in the Eclipse platform: Qvt for Mmt and Xpand for Mtt. In contrast, UMLsec and SecureUML do not use Mmts in their methodologies. For Mtt, each of them uses a specific tool as compiler: to generate source code in the case of SecureUML; to generate first-order logic formulas in the case of UMLsec.

Security Verification Sectet and ModelSec do not provide explicit information about how to verify security properties either on the models or on the system infrastructure. Other Mds approaches make use of either theorem provers or model-checkers to verify security properties on models of the system.

Traceability It seems that current existing Mds approaches in the literature lack this important functionality. In the Mds approaches we have evaluated only UMLsec implements an incomplete error-tracking mechanism using attack sequences generated by the theorem prover. Human effort is still necessary to interpret such an attack sequence at the level of the abstract models.

Tool Support Most of the selected approaches directly benefit from recent progress of Mde in terms of support for editing models and automatically performing transformations. However, tool support in current Mds approaches is only partial and limited to certain steps of the development process.

Validation UMLsec, Sectet and SecureMDD are ranked *high* level for validation because each of them was experimented using a series of case studies, involving large-size industrial experiments. SecureUML has been validated using several case studies, but only one mid-size industrial case study. We thus rank its validation level as *medium*. ModelSec seems immature at the moment because it has only been applied to an academic case study.

5.2. Threats to the Validity of Mds

In this section we summarize a few of the disadvantages of current Mds approaches, as mentioned in the literature.

Table 2: Comparison of the Evaluated MDS Approaches

	Application Domains	Security Concerns	Modeling Approach		(*)SoC	Model Transformation		Verification	Traceability	Tool Support	Validation
			Paradigm	Modeling Language		MMT	MTT				
UMLsec	web applications, embedded systems, distributed systems	confidentiality, integrity, authenticity, authorization, freshness, information flow, non-repudiation, fair exchange	MPM	UML profiles	o	✗	compiler	aiCall theorem prover	Attack sequence	✓	high
SecureUML	web applications	access control	MDA + DSM	UML profiles	✓	✗	compiler	SecureMova model-checker	✗	✓	medium
SECTET	e-government, e-health, e-education	integrity, confidentiality, non-repudiation, access control	MDA + DSM	UML profiles	✓	QVT	XPAND	✗	✗	✓	high
ModelSec	web applications, databases	privacy, integrity, authentication, availability, non-repudiation, auditing, access control	MDA + DSM	SecML (tailored DSL)	✓	RubyTL	MOF-Script	✗	✗	✓	low
SecureMDD	smart card and service applications	cryptography (secrecy, integrity, confidentiality), application-specific security properties	MDA + DSM	UML profiles	✓	QVT	XPAND	KIV theorem prover, test cases from UML specifications	✗	✓	high

Note: Supported (✓); Partially supported (o); Not supported (✗)

Employing UML profiles is the subject of debate within the community. Sánchez et al. (2009) mention that the usability of UML profiles for modeling and analyzing security-critical systems is limited. They argue that adapting a UML profile to model new security concerns beyond its original capabilities is difficult, if not impossible. Second, Ma et al. (2013) show that using general-purpose modeling languages, such as UML profiles, hinders reusability, although it does favor communication between models. Moreover, the same authors mention that adapting UML profiles to new systems requires a large effort when security is not already integrated in the modeling effort.

Breu et al. (2008) emphasize the importance of traceability in MDS approaches. However, traceability is rarely presented in the MDS methodologies we have analyzed.

As a last and general remark, Ma et al. (2013) state that most MDS approaches (some of which are beyond our selection) are still merely academic. Some of those MDS approaches are prototypes illustrating theoretical concepts as part of a research project. Most of them are implementations designed for a specific business domain, and/or a specific security concern. Moreover, the results of the systematic reviews on MDS, conducted by Nguyen et al. (2013) and by Jensen and Jaatun (2011) both show that there is a lack of empirical studies on MDS research.

5.3. Relevant Open Issues

Based on the evaluation result, the following issues seems to be open for discussion.

5.3.1. Choice of Modeling Paradigms

Table 2 shows that the separation of security concerns from business logic (SoC) is present in almost all MDS approaches. However, even if the notion of *Separation of Concerns* is at the heart of AOM (Aspect-Oriented Modeling), none of the evaluated MDS approaches explicitly uses the AOM paradigm. Indeed, in the literature we have analyzed the term *weaving* is not used, and no approach uses a *model weaver*. This result conforms to the survey statistics in (Nguyen et al., 2013) that 87% of their selected primary studies are not based on AOM paradigm.

It is difficult to explain why no evaluated MDS approach explicitly applies the AOM paradigm, but we will attempt to propose a few possible reasons. One reason could be that the AOM tools were not mature enough at the time the MDS approaches have been proposed. Another reason could be

that the AOM tools do not exactly offer what the MDS approaches require. This points to the fact that the explicit use of AOM paradigm and related tools in MDS context should be further investigated. Such an investigation could allow to determine whether AOM paradigm could actually help, or not, MDS.

Other examples of the usage of the MPM paradigm in MDS, other than in the UMLSEC, approach have been published in (Zia et al., 2007; Decker et al., 2008; Layouni et al., 2009).

5.3.2. Security Concerns and Corresponding Metamodel

According to the security concern metamodels for the evaluated MDS approaches in Section 4 (Figs. 5, 7 and 9), only access control is metamodeled in detail, while other security concerns such as e.g. integrity and confidentiality, are modeled as simple entities in those metamodels.

The reason for this is that security concerns such as *confidentiality* and *integrity* are dynamic security properties which involve the *state* of the business part of the system. Such security concern entities in the metamodel can for example be associated with elements of dynamic models of the system such as UML sequence or activity diagrams. Another possibility is to implement such security concerns directly at the level of the infrastructure, therefore bypassing additional modeling. For example, in SECTET *confidentiality* and *integrity* are enforced by existing communication protocols at the level of the web-services. Being that the treatment of dynamic security properties either by models or by the system's infrastructure is heavily dependent on the application scenario, such requirements cannot easily be abstracted by a metamodel.

This observation matches the results of the systematic review (Nguyen et al., 2013) which states that access control is dealt with in the majority (around 42%) of the selected studies on MDS.

5.3.3. Choice of Modeling Languages

Four out of the five MDS approaches that we evaluate in this chapter employ UML profiles as modeling languages. Only the MODELSEC approach proposes using SECML, a tailored DSL. It would thus seem like UML profiles are widely employed in MDS. This is understandable given the popularity of the UML. The results shown in the systematic review (Nguyen et al., 2013) also reveal that UML standard models and UML profiles are used 79% of the studies used for their review. In contrast, only 21% of the reviewed studies

use tailored DSLs. However, even though those MDS approaches do use UML profiles as their modeling languages, UML profiles can be considered as DSLs. In other words, DSL seems to be a key concept of MDS.

6. Related Work

Despite more than a decade of existence, there is only one survey (Kasal et al., 2011) and two systematic reviews (Jensen and Jaatun, 2011; Nguyen et al., 2013) on the subject of MDS.

Kasal et al. (2011) share with our work a taxonomy for evaluating model-based security engineering which directly inspires ours. Several taxonomy entries are common, for example their *paradigm*, *verification* and *security mechanisms* correspond respectively to our *modeling approach*, *verification* and *security concerns*. Our taxonomy however clearly distinguishes characteristics coming from the MDE orientation. For example, the fact that we consider *modeling approach* with the possible use of DSLs already covers some of their entries (namely, *formality*, *granularity* and *executability*). Their work only reviews four approaches: UMLSEC (Jürjens, 2004), Secure Software Architecture (Yu et al., 2005), a Model-Based Aspect-Oriented Framework (Zhu and Zulkernine, 2009) and AVISPA (Armando et al., 2005) – a push-button tool for validating Internet security protocols. Surprisingly, the authors also include, at the same level, tools aimed at general-purpose analysis: SMV (Symbolic Model Verifier)[3], – a LTL/CTL general-purpose model-checker, and Alloy – a SAT-based solver for relational specifications (Jackson, 2012). Neither can be considered as an MDS approach according to our definition.

The systematic review conducted by Jensen and Jaatun (2011) is clearly oriented towards code generation. We also cover three out of five of the approaches they consider. However, the authors have not provided the MDS approach selection criteria. Furthermore, as an inherent limitation of a systematic review, their work does not use a systematic evaluation schema (such as the taxonomy defined in our work) such that the evaluation result is less detailed when compared with ours.

Nguyen et al. (2013) proposed a systematic literature review targeting specifically MDS approaches. Their selection process retained 80 papers out of more than 10 thousands papers. This review also succinctly describes five

[3]The SMV model-checker is freely accessible at `http://www.cs.cmu.edu/~modelcheck/smv.html`, although Kasal et al. (2011) do not provide any reference.

approaches, all evaluated in this chapter. This paper describes statistical results on several dimensions (the application domains, the level of automation when reaching executable code for security systems, and the security concerns) that provides an interesting snapshot of the current practice in MDS. Our taxonomy fully encompasses their evaluation criteria, and looks at dimensions that would become challenging for MDS (such as verification and traceability). Furthermore, we evaluate and compare in details several well-known MDS approaches, instead of focusing on statistical results.

Basin et al. (2011) go through a decade of MDS with the work on SECUREUML. The authors concentrate on the following topics: the *modeling* issues, both for business and security concerns; the transformations for obtaining composed models, both for generating executable code and for deriving test cases; the analysis capabilities of their approach; and finally, the tool support for SECUREUML. This survey, although very specific to the authors' tool, provides an interesting vision of a viable approach to MDS.

7. Conclusion

In this chapter we have presented the main advances in *Model-Driven Security* for the past decade. In order to propose a clear vision of what is MDS, we first introduced the main concepts on which MDS is based on. In particular, we focused on the notions of MDE, such as metamodels, model transformations, etc, but also on the notion of separation of concerns or separation of views. We have then proposed a detailed taxonomy consisting of the main concepts and elements of MDS. Based on our taxonomy we have described, summarized, evaluated and discussed five well-known MDS approaches: UMLSEC, SECUREUML, SECTET, MODELSEC and SECUREMDD. We pointed out some current limitations of MDS approaches, and sketched some relevant open issues. Overall, this chapter provides a broad view of the field and is intended as an introduction to MDS for students, researchers or practitioners.

This work allows us to provide insights about future directions for MDS research and industrial practice. A primary challenge is to reach a better level of maturity: this of course requires building tools, but also conducting more systematic industrial experimentation. The latter is obviously difficult in such a critical software application domain. However, recent progress in MDE in theories and tooling, as well as the continuous interest from industry in modeling, may directly be of benefit to the development of MDS.

We noticed that most of the existing MDS approaches implement separation of security concerns from the business logic (SOC), even if for the approaches we surveyed this principle cannot be considered as following the AOM paradigm. By leveraging AOM, security concerns can be specified as *aspect* models that can be *woven* into the primary (business logic) models. The AOM paradigm could thus be used to enhance the modularity of the security-critical systems and the reusability of the security models.

MDS has to deal with the business complexity, but also with the additional complexity of security concerns which are multiple in nature. An intuitive solution to this variety is for the software development methodology to reflect the heterogeneous nature of such systems with the goal of making their design simpler. From our evaluation in Section 4 we can deduce that it is difficult to develop a general-purpose DSL intended to model all security concerns simultaneously. This is because different security properties may require different interactions with the business models that are too complex to be modeled by a human using a single DSL. For easing the modeling task and allowing better verification possibilities, an interesting possibility is that each security concern is modeled using a specifically tailored DSL.

However, spreading security concerns over several models raises several crucial challenges in our opinion: *First*, it hinders the understanding of the overall system's security by the experts since they need to deal with several models simultaneously. This drawback can however be balanced by the narrowed focus of those models. *Second*, it requires powerful composition operators for creating models that amalgamate all security aspects. This is crucial for later phases: whereas separate security models make it possible to analyze security properties independently, the enforcement of those policies in the business part of the systems and code generation requires merging all security aspects before reaching platform code. *Third*, it complicates keeping all security models synchronized over common information and poses additional challenges when tracking integrated verification results back to security information that is distributed over several models.

Acknowledgements. This work was partially supported by the the NECSIS project, funded by the Automotive Partnership Canada; and also partially supported by the *Fonds National de la Recherche* (FNR), Luxembourg (partly under the MITER project C10/IS/783852).

References

Agreiter, B., Breu, R., 2009. Model-Driven Configuration of SELinux Policies, in: On the Move to Meaningful Internet Systems: OTM 2009. Springer-Verlag. volume 5871 of *Lecture Notes in Computer Science*, pp. 887–904.

Alam, M., Breu, R., Breu, M., 2004. Model Driven Security for Web Services (MDS4WS), in: Proceedings of the 8th IEEE International Multitopic Conference, INMIC 2004, IEEE Computer Society, Lahore, Pakistan. pp. 498–505.

Alam, M., Breu, R., Hafner, M., 2007a. Model-Driven Security Engineering for Trust Management in SECTET, in: Journal of Software. Academy Publisher. volume 2, pp. 47–59.

Alam, M., Hafner, M., Breu, R., 2006a. A Constraint Based Role Based Access Control in the SECTET a Model-Driven Approach, in: Proceedings of the International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services, PST 2006, ACM, Markham, Ontario, Canada. pp. 13:1–13:13.

Alam, M., Hafner, M., Breu, R., 2006b. Constraint Based Role Based Access Control (CRBAC) for Restricted Administrative Delegation Constraints in the SECTET, in: Proceedings of the International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services, PST 2006, ACM, Markham, Ontario, Canada. pp. 44:1–44:5.

Alam, M., Hafner, M., Breu, R., Unterthiner, S., 2006c. A Framework for Modeling Restricted Delegation in Service Oriented Architecture, in: Proceedings of the 3rd International Conference on Trust, Privacy, and Security in Digital Business, TrustBus 2006, Springer-Verlag, Krakow, Poland. pp. 142–151.

Alam, M., Seifert, J.P., Zhang, X., 2007b. A Model-Driven Framework for Trusted Computing Based Systems, in: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2007, IEEE Computer Society, Annapolis, Maryland, USA. pp. 75–86.

Amrani, M., Lúcio, L., Selim, G.M., Combemale, B., Dingel, J., Vangheluwe, H., Le Traon, Y., Cordy, J.R., 2012. A Tridimensional Approach for Studying the Formal Verification of Model Transformations, in: Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, IEEE Computer Society, Montreal, Canada. pp. 921–928.

Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P.H., Heám, P.C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L., 2005. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications, in: Proceedings of the 17th International Conference on Computer Aided Verification, CAV 2005, Springer-Verlag, The University of Edinburgh, Scotland, UK. pp. 281–285.

Atkinson, C., Stoll, D., Tunjic, C., 2011. Orthographic Service Modeling, in: Workshops Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOCW 2011, IEEE Computer Society, Helsinki, Finland. pp. 67–70.

ATLAS, 2008. ATLAS Transformation Language. http://www.eclipse.org/m2m/atl/.

Basin, D., Clavel, M., Doser, J., Egea, M., 2009. Automated Analysis of Security-Design Models, in: Information and Software Technology. Elsevier. volume 51, pp. 815–831.

Basin, D., Clavel, M., Egea, M., 2011. A Decade of Model-Driven Security, in: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, SACMAT 2011, ACM, Innsbruck, Austria. pp. 1–10.

Basin, D., Clavel, M., Egea, M., Schläpfer, M., 2010. Automatic Generation of Smart, Security-Aware GUI Models, in: Proceedings of the International Symposium on Engineering Secure Software and Systems, ESSoS 2010, Springer-Verlag, Pisa, Italy. pp. 201–217.

Basin, D., Doser, J., Lodderstedt, T., 2003. Model Driven Security for Process-Oriented Systems, in: Proceedings of the 8th ACM Symposium

on Access Control Models and Technologies, SACMAT 2003, ACM, Como, Italy. pp. 100–109.

Basin, D., Doser, J., Lodderstedt, T., 2006. Model Driven Security: from UML Models to Access Control Infrastructures, in: Transactions on Software Engineering and Methodology. ACM. volume 15, pp. 39–91.

Best, B., Jürjens, J., Nuseibeh, B., 2007. Model-Based Security Engineering of Distributed Information Systems Using UMLsec, in: Proceedings of the 29th International Conference on Software Engineering, ICSE 2007, IEEE Computer Society, Minneapolis, MN, USA. pp. 581–590.

Bezivin, J., 2006. Model Driven Engineering: An Emerging Technical Space, in: Generative and Transformational Techniques in Software Engineering. Springer-Verlag. volume 4143 of *Lecture Notes in Computer Science*, pp. 36–64.

Breu, R., Hafner, M., Innerhofer-Oberperfler, F., Wozak, F., 2008. Model-Driven Security Engineering of Service Oriented Systems, in: Information Systems and e-Business Technologies. Springer-Verlag. volume 5 of *Lecture Notes in Business Information Processing*, pp. 59–71.

Breu, R., Hafner, M., Weber, B., Novak, A., 2005. Model Driven Security for Inter-Organizational Workflows in e-Government, in: E-Government: Towards Electronic Democracy. Springer-Verlag. volume 3416 of *Lecture Notes in Computer Science*, pp. 122–133.

Breu, R., Popp, G., Alam, M., 2007. Model Based Development of Access Policies, in: International Journal on Software Tools for Technology Transfer. Springer-Verlag. volume 9, pp. 457–470.

Büttner, F., Egea, M., Cabot, J., Gogolla, M., 2012. Verification of ATL Transformations Using Transformation Models and Model Finders, in: Proceedings of the 14th International Conference on Formal Engineering Methods, ICFEM 2012, Springer-Verlag, Kyoto, Japan. pp. 198–213.

Clarke, S., 2001. Composition of Object-Oriented Software Design Models. Ph.D. thesis. Dublin City University, Ireland.

Clarke, S., Baniassad, E., 2005. Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley.

Clavel, M., Torres da Silva, V., Braga, C., Egea, M., 2008. Model-Driven Security in Practice: An Industrial Experience, in: Proceedings of 4th European Conference on Model Driven Architecture - Foundations and Applications, ECMDA-FA 2008, Springer-Verlag, Berlin, Germany. pp. 326–337.

Cook, S., Jones, G., Kent, S., Wils, A.C., 2007. Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley.

Cottenier, T., Van Den Berg, A., Elrad, T., 2007. The Motorola WEAVR: Model Weaving in a Large Industrial Context, in: Proceedings of the 6th International Conference on Aspect-Oriented Software Development (Industry Track), AOSD 2007, Vancouver, Canada. URL: http://aosd.net/2007/program/industry/I3-MotorolaWEAVR.pdf.

Cysneiros, L., Sampaio do Prado Leite, J., 2002. Non-Functional Requirements: from Elicitation to Modelling Languages, in: Proceedings of the 24th International Conference on Software Engineering, ICSE 2002, IEEE Computer Society, Orlando, Florida, USA. pp. 699–700.

Decker, B.D., Layouni, M., Vangheluwe, H., Verslype, K., 2008. A Privacy-Preserving eHealth Protocol Compliant with the Belgian Healthcare System, in: Proceedings of the 5th European Public Key Infrastructure Workshop: Theory and Practice, EuroPKI 2008. Springer. volume 5057 of *Lecture Notes in Computer Science*, pp. 118–133.

Dupe, G., Belaunde, M., Perruchon, R., Besnard, H., Guillard, F., Oliveres, V., . SmartQVT. http://smartqvt.elibel.tm.fr/.

Ehrig, H., Ehrig, K., Prange, U., Taentzer, G., 2006. Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series). Springer-Verlag.

Fernández-Medina, E., Piattini, M., 2005. Designing Secure Databases, in: Information and Software Technology. Elsevier. volume 47, pp. 463–477.

Fleurey, F., Baudry, B., France, R., Ghosh, S., 2007. A Generic Approach For Automatic Model Composition, in: Proceedings of the 11th International Workshop on Aspect-Oriented Modeling, AOM at MoDELS 2007, Springer-Verlag, Nashville, TN, USA. pp. 7–15.

France, R., Ray, I., Georg, G., Ghosh, S., 2004. Aspect-Oriented Approach to Early Design Modelling, in: IEE Proceedings - Software. IEEE Computer Society. volume 151, pp. 173–185.

Guerra, E., de Lara, J., Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., Schwinger, W., 2013. Automated Verification of Model Transformations Based on Visual Contracts, in: Automated Software Engineering. Springer-Verlag. volume 20, pp. 5–46.

Hafner, M., Alam, M., Breu, R., 2006a. Towards a MOF/QVT-Based Domain Architecture for Model Driven Security, in: Model Driven Engineering Languages and Systems. Springer-Verlag. volume 4199 of *Lecture Notes in Computer Science*, pp. 275–290.

Hafner, M., Breu, M., Breu, R., Nowak, A., 2005. Modeling Inter-Organizational Workflow Security in a Peer-To-Peer Environment, in: Proceedings of the IEEE International Conference on Web Services, ICWS 2005, IEEE Computer Society, Orlando, Florida, USA. pp. 533–540.

Hafner, M., Breu, R., 2009. Security Engineering for Service-Oriented Architectures. Springer-Verlag.

Hafner, M., Breu, R., Agreiter, B., Nowak, A., 2006b. SECTET: An Extensible Framework for the Realization of Secure Inter-Organizational Workflows, in: Internet Research. Emerald Group Publishing Limited. volume 16, pp. 491–506.

Hafner, M., Memon, M., Alam, M., 2008. Modeling and Enforcing Advanced Access Control Policies in Healthcare Systems with SECTET, in: Models in Software Engineering. Springer-Verlag. volume 5002 of *Lecture Notes in Computer Science*, pp. 132–144.

Hatebur, D., Heisel, M., Jürjens, J., Schmidt, H., 2011. Systematic Development of UMLsec Design Models Based on Security Requirements, in: Fundamental Approaches to Software Engineering. Springer-Verlag. volume 6603 of *Lecture Notes in Computer Science*, pp. 232–246.

Houmb, S., Jürjens, J., 2003. Developing Secure Networked Web-Based Systems Using Model-Based Risk Assessment and UMLsec, in: Proceedings of the 10th Asia-Pacific Software Engineering Conference, APSEC 2003, IEEE Computer Society, Chiang Mai, Thailand. pp. 488–497.

Howard, M., Lipner, S., 2006. The Security Development Lifecycle. Microsoft Press.

Jackson, D., 2012. Software Abstractions: Logic, Language and Analysis. MIT Press.

Jacobson, I., Ng, P.W., 2004. Aspect-Oriented Software Development with Use Cases. Addison-Wesley.

Jensen, J., Jaatun, M.G., 2011. Security in Model Driven Development: A Survey, in: Proceedings of the 6th International Conference on Availability, Reliability and Security, ARES 2011, IEEE Computer Society, Vienna, Austria. pp. 704–709.

Jürjens, J., 2001. Towards Development of Secure Systems Using UMLsec, in: Fundamental Approaches to Software Engineering. Springer-Verlag. volume 2029 of *Lecture Notes in Computer Science*, pp. 187–200.

Jürjens, J., 2002a. Formal Semantics for Interacting UML Subsystems, in: Proceedings of the IFIP TC6/WG6.1 Fifth International Conference on Formal Methods for Open Object-Based Distributed Systems, FMOODS 2002, Wolters Kluwer, University of Twente, Netherlands. pp. 29–43.

Jürjens, J., 2002b. Principles for Secure Systems Design. Ph.D. thesis. Oxford University, United Kingdom.

Jürjens, J., 2002c. UMLsec: Extending UML for Secure Systems Development, in: UML 2002 - The Unified Modeling Language. Springer-Verlag. volume 2460 of *Lecture Notes in Computer Science*, pp. 412–425.

Jürjens, J., 2004. Model-Based Security Engineering with UML, in: Proceedings of the 4th International School on Foundations of Security Analysis and Design, FOSAD 2004, Springer-Verlag, Bertinoro University Residential Center, Italy. pp. 42–77.

Jürjens, J., 2005a. Code Security Analysis of a Biometric Authentication System Using Automated Theorem Provers, in: Proceedings of the 21st Annual Computer Security Applications Conference, ACSAC 2005, IEEE Computer Society, Tucson, Arizona, USA. pp. 138–149.

Jürjens, J., 2005b. Secure Systems Development with UML. Springer-Verlag.

Jürjens, J., 2005c. Sound Methods and Effective Tools for Model-Based Security Engineering with UML, in: Proceedings of the 27th International Conference on Software Engineering, ICSE 2005, IEEE Computer Society, St. Louis, Missouri, USA. pp. 322–331.

Jürjens, J., 2007. Developing Secure Embedded Systems: Pitfalls and How to Avoid Them, in: Proceedings of the 29th International Conference on Software Engineering, ICSE 2007, IEEE Computer Society, Minneapolis, MN, USA. pp. 182–183.

Jürjens, J., Schreck, J., Bartmann, P., 2008. Model-Based Security Analysis for Mobile Communications, in: Proceedings of the 30th International Conference on Software Engineering, ICSE 2008, IEEE Computer Society, Leipzig, Germany. pp. 683–692.

Kasal, K., Heurix, J., Neubauer, T., 2011. Model-Driven Development Meets Security: An Evaluation of Current Approaches, in: Proceedings of the 44th Hawaii International Conference on System Sciences, HICSS-44 2011, IEEE Computer Society, Kauai, Hawaii, USA. pp. 1–9.

Khwaja, A., Urban, J., 2002. A Synthesis of Evaluation Criteria for Software Specifications and Specification Techniques, in: International Journal of Software Engineering and Knowledge Engineering. World Scientific. volume 12, pp. 581–599.

Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J., 1997. Aspect-Oriented Programming, in: Proceedings of the European Conference on Object-Oriented Programming. Springer-Verlag. volume 1241, pp. 220–242.

Kienzle, J., Al Abed, W., Fleurey, F., Jézéquel, J., Klein, J., 2010. Aspect-Oriented Design with Reusable Aspect Models, in: Transactions on Aspect-Oriented Software Development VII. Springer-Verlag. volume 6210 of *Lecture Notes in Computer Science*, pp. 272–320.

Klein, J., Fleurey, F., Jézéquel, J., 2007. Weaving Multiple Aspects in Sequence Diagrams, in: Transactions on Aspect-Oriented Software Development (TAOSD). Springer-Verlag. volume 4620 of *Lecture Notes in Computer Science*, pp. 167–199.

Klein, J., Hélouët, L., Jézéquel, J., 2006. Semantic-Based Weaving of Scenarios, in: Proceedings of the 5th International Conference on Aspect-Oriented Software Development, AOSD 2006, ACM, Bonn, Germany. pp. 27–38.

Klein, J., Kienzle, J., Morin, B., Jézéquel, J.M., 2009. Aspect Model Unweaving, in: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2009, Springer-Verlag, Denver, Colorado, USA. pp. 514–530.

Kleppe, A.G., Warmer, J., Bast, W., 2003. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley.

Kramer, M.E., Klein, J., Steel, J.R., Morin, B., Kienzle, J., Barais, O., Jézéquel, J., 2013. Achieving Practical Genericity in Model Weaving through Extensibility, in: Proceedings of the 6th International Conference on the Theory and Practice of Model Transformations, ICMT 2013, Springer-Verlag, Budapest, Hungary. pp. 108–124.

Lang, U., Schreiner, R., 2008. Model Driven Security Management: Making Security Management Manageable in Complex Distributed Systems, in: Proceedings of the 1st Workshop on Modeling Security, MODSEC 2008, CEUR Workshop Proceedings (CEUR-WS.org), Toulouse, France. URL: http://ceur-ws.org/Vol-413/paper10.pdf.

de Lara, J., Vangheluwe, H., 2002. AToM^3: A Tool for Multi-Formalism and Meta-Modelling, in: Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering, FASE 2002, Springer-Verlag, Grenoble, France. pp. 174–188.

Layouni, M., Verslype, K., Sandikkaya, M.T., Decker, B.D., Vangheluwe, H., 2009. Privacy-Preserving Telemonitoring for ehealth, in: Proceedings of the 23rd IFIP WG Working Conference on Data and Applications Security. Springer. volume 5645 of *Lecture Notes in Computer Science*, pp. 95–110.

Lloyd, J., Jürjens, J., 2009. Security Analysis of a Biometric Authentication System Using UMLsec and JML, in: Model Driven Engineering Languages and Systems. Springer-Verlag. volume 5795 of *Lecture Notes in Computer Science*, pp. 77–91.

Lodderstedt, T., 2003. Model Driven Security, from UML Models to Access Control Architectures. Ph.D. thesis. University of Freiburg, Germany.

Lodderstedt, T., Basin, D., Doser, J., 2002. SecureUML: A UML-Based Modeling Language for Model-Driven Security, in: <<UML>> 2002 - The Unified Modeling Language. Springer-Verlag. volume 2460 of *Lecture Notes in Computer Science*, pp. 426–441.

Lúcio, L., Barroca, B., Amaral, V., 2010. A Technique for Automatic Validation of Model Transformations, in: Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2010, Springer-Verlag, Oslo, Norway. pp. 136–150.

Ma, Z., Wagner, C., Woitsch, R., Skopik, F., Bleier, T., 2013. Model-Driven Security: from Theory to Application, in: International Journal of Computer Information Systems and Industrial Management Applications. MIR Labs. volume 5, pp. 151–158.

MacDonald, N., 2007. Model-Driven Security: Enabling a Real-Time, Adaptive Security Infrastructure. Technical Report. Gartner Inc. URL: http://www.gartner.com/id=525109.

Metacase, 2009. Domain-Specific Modeling with MetaEdit+: 10 times faster than UML. White Paper.

Moebius, N., Stenzel, K., Borek, M., Reif, W., 2012. Incremental Development of Large, Secure Smart Card Applications, in: Proceedings of the 1st International Workshop on Model-Driven Security, ACM, Innsbruck, Austria. pp. 1–6.

Moebius, N., Stenzel, K., Grandy, H., Reif, W., 2009a. Model-Driven Code Generation for Secure Smart Card Applications, in: Proceedings of the 20th Australian Software Engineering Conference, ASWEC 2009, IEEE Computer Society, Gold Coast, Australia. pp. 44–53.

Moebius, N., Stenzel, K., Grandy, H., Reif, W., 2009b. SecureMDD: A Model-Driven Development Method for Secure Smart Card Applications, in: Proceedings of the International Conference on Availability, Reliability and Security, ARES 2009, IEEE Computer Society, Fukuoka, Japan. pp. 841–846.

Moebius, N., Stenzel, K., Reif, W., 2009c. Generating Formal Specifications for Security-Critical Applications - A Model-Driven Approach, in: Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems, IWSESS 2009, IEEE Computer Society, Washington, DC, USA. pp. 68–74.

Moebius, N., Stenzel, K., Reif, W., 2010. Formal Verification of Application-Specific Security Properties in a Model-Driven Approach, in: Engineering Secure Software and Systems. Springer-Verlag. volume 5965 of *Lecture Notes in Computer Science*, pp. 166–181.

Moore, W., Dean, D., Gerber, A., Wagenknecht, G., Vanderheyden, P., 2004. Eclipse Development Using the Graphical Editing Framework and the Eclipse Modeling Framework. IBM Redbooks.

Morin, B., Barais, O., Nain, G., Jézéquel, J., 2009. Taming Dynamically Adaptive Systems with Models and Aspects, in: Proceedings of the 31st International Conference on Software Engineering, ICSE 2009, IEEE Computer Society, Vancouver, Canada. pp. 122–132.

Morin, B., Klein, J., Barais, O., Jézéquel, J., 2008. A Generic Weaver for Supporting Product Lines, in: Proceedings of the Early Aspects Workshop at ICSE 2008, ACM, Leipzig, Germany. pp. 11–18.

Morin, B., Klein, J., Kienzle, J., Jézéquel, J., 2010. Flexible Model Element Introduction Policies for Aspect-Oriented Modeling, in: Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2010, Springer-Verlag, Oslo, Norway. pp. 63–77.

Mosterman, P.J., Vangheluwe, H., 2004. Computer Automated Multi-Paradigm Modeling: An Introduction, in: Simulation. SAGE Publications. volume 80, pp. 433–450.

Muller, P., Fleurey, F., Jézéquel, J., 2005. Weaving Executability into Object-Oriented Meta-Languages, in: Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2005, Springer-Verlag, Half Moon Resort, Montego Bay, Jamaica. pp. 264–278.

Muñoz, J., 2009. Information Security Industry: State of the Art, in: Proceedings of the Security Electronic Business Processes, Highlights of

the Information Security Solutions Europe 2008 Conference, ISSE 2008. Vieweg+Teubner, Madrid, Spain, pp. 84–89.

Nguyen, P.H., Klein, J., Kramer, M., Le Traon, Y., 2013. A Systematic Review of Model Driven Security, in: Proceedings of the 20th Asia-Pacific Software Engineering Conference, APSEC 2013, IEEE Computer Society, Bangkok, Thailand. p. (to appear).

Papadakis, M., Malevris, N., 2012. Mutation-Based Test Case Generation via a Path Selection Strategy, in: Information and Software Technology. Elsevier. volume 54, pp. 915–932.

Reddy, R., Ghosh, S., France, R.B., Straw, G., Bieman, J.M., Song, E., Georg, G., 2006. Directives for Composing Aspect-Oriented Design Class Models, in: Transactions on Aspect-Oriented Software Development (TAOSD). Springer-Verlag. volume 3880 of *Lecture Notes in Computer Science*, pp. 75–105.

Sánchez, O., Molina, F., García-Molina, J., Toval, A., 2009. ModelSec: A Generative Architecture for Model-Driven Security, in: Journal of Universal Computer Science. Verlag der Technischen Universität Graz. volume 15, pp. 2957–2980.

Sandhu, R., Coyne, E., Feinstein, H., Youman, C., 1996. Role-Based Access Control Models, in: IEEE Computer. IEEE Computer Society. volume 29, pp. 38–47.

Selim, G.M., Cordy, J.R., Dingel, J., 2012a. Analysis of Model Transformations. Technical Report 2012-592. Queen's University.

Selim, G.M., Cordy, J.R., Dingel, J., 2012b. Model Transformation Testing: The State of the Art, in: Proceedings of the 1st International Workshop on the Analysis of Model Transformations, AMT 2012, Springer-Verlag, Innsbruck, Austria. pp. 21–26.

Sendall, S., Kozaczynski, W., 2003. Model Transformation: The Heart and Soul of Model-Driven Software Development, in: IEEE Software. IEEE Computer Society. volume 20, pp. 42–45.

Shafiq, B., Masood, A., Joshi, J., Ghafoor, A., 2005. A Role-Based Access Control Policy Verification Framework for Real-Time Systems, in: Proceedings of the IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, IEEE Computer Society, Los Alamitos, CA, USA. pp. 13–20.

Shin, M., Gomaa, H., 2009. Separating Application and Security Concerns in Modeling Software Product Lines, in: Applied Software Product Line Engineering. 1st ed.. Auerbach Publications Boston, MA, USA, pp. 337–366.

Stark, R.F., Borger, E., Schmid, J., 2001. Java and the Java Virtual Machine: Definition, Verification, Validation. Springer-Verlag.

Syriani, E., 2011. A Multi-Paradigm Foundation for Model Transformation Language Engineering. Ph.D. thesis. McGill University, Canada.

Thirioux, X., Combemale, B., Crégut, X., Garoche, P.L., 2007. A Framework to Formalise the MDE Foundations, in: Proceedings of the 1st International Workshop on Towers of Models, TOWERS 2007, Springer-Verlag, Zürich, Switzerland. pp. 14–30.

Vallecillo, A., Gogolla, M., 2012. Typing Model Transformations Using Tracts, in: Proceedings of the 5th International Conference on the Theory and Practice of Model Transformations, ICMT 2012, Springer-Verlag, Prague, Czech Republic. pp. 56–71.

Whittle, J., Araújo, J., 2004. Scenario Modelling with Aspects., in: IEE Proceedings - Software. IEEE Computer Society. volume 151, pp. 157–172.

Whittle, J., Jayaraman, P.K., Elkhodary, A.M., Moreira, A., Araújo, J., 2009. MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation, in: Transactions on Aspect-Oriented Software Development VI. Springer-Verlag. volume 5560 of *Lecture Notes in Computer Science*, pp. 191–237.

Yu, H., Liu, D., He, X., Yang, L., Gao, S., 2005. Secure Software Architectures Design by Aspect Orientation, in: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2005, IEEE Computer Society, Shanghai, China. pp. 47–55.

Zhu, Z.J., Zulkernine, M., 2009. A Model-Based Aspect-Oriented Framework for Building Intrusion-Aware Software Systems, in: Information and Software Technology. Elsevier. volume 51, pp. 865–875.

Zia, M., Posse, E., Vangheluwe, H., 2007. Addressing Security Requirements Through Multi-Formalism Modelling and Model Transformation, in: Proceedings of the 2nd International Conference on Software and Data Technologies, ICSOFT (SE), INSTICC Press. pp. 129–137.

Appendix A. List of Acronyms

AOM	Aspect-Oriented Modeling
AOSD	Aspect-Oriented Software Development
ASM	Abstract State Machine
ATL	ATLAS Transformation Language
CIM	Computation Independent Model
CTL	Computation Tree Logic
DSM	Domain-Specific Modeling
DSML/DSL	Domain-Specific (Modeling) Language
EJB	Enterprise JavaBeans
EMF	Eclipse Modeling Framework
GMF	Graphical Modeling Framework
GPL	General-purpose Programming Language
J2EE	Java 2 Enterprise Edition
LTL	Linear Temporal Logic
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
MDS	Model-Driven Security
MMT	Model-To-Model Transformation
MOF	Meta-Object Facility
MPM	Multi-Paradigm Modeling
MTT	Model-To-Text Transformation
OCL	Object Constraint Language
OMG	Object Management Group
PDM	Platform Description Model
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query/View/Transformation, a set of model transformation languages defined by Object Management Group
RBAC	Role-Based Access Control
SOA	Service-Oriented Architecture
SOC	Separation of Concerns
UML	Unified Modeling Language
XACML	Extensible Access Control Markup Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XPAND	A statically-typed template language under Eclipse platform