Advances in Model-Driven Security

Levi Lúcio\textsuperscript{a}, Qin Zhang\textsuperscript{b}, Phu H. Nguyen\textsuperscript{b}, Moussa Amrani\textsuperscript{b}, Jacques Klein\textsuperscript{b}, Hans Vangheluwe\textsuperscript{c,a}, Yves Le Traon\textsuperscript{b}

\textsuperscript{a}Modeling Simulation and Design Lab, McGill University, Montreal QC, Canada
\textsuperscript{b}Centre for Security Reliability and Trust, University of Luxembourg, Luxembourg
\textsuperscript{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

Preprint submitted to Advances in Computers October 30, 2013
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.
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 (Müller 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.
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 $M_0$ represents the actual system; those in the $M_1$ level model the $M_0$ ones; artifacts belonging to the $M_2$ level are metamodels, allowing the definition of $M_1$ models; and finally, the unique artifact at the $M_3$ 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),
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™ and Lucent™. 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), AToM³ (de Lara and Vangheluwe, 2002) or Microsoft’s™ 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 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. run-time 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>
<thead>
<tr>
<th>Taxonomy Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Domains</td>
<td>Is the MDS approach domain specific or generalistic?</td>
</tr>
<tr>
<td>Security Concerns</td>
<td>Which concerns does the MDS approach focus on? Are they expressible in a metamodel?</td>
</tr>
<tr>
<td>Modeling Approach</td>
<td>Which Modeling paradigm(s) (MDA, AOM, DSM)? Which Modeling language(s) (standard UML; UML profiles; DSLs; Formal Languages)?</td>
</tr>
<tr>
<td>Separation of Concerns</td>
<td>Is it used? If it is, how is it implemented?</td>
</tr>
<tr>
<td>Model Transformations</td>
<td>Are MMT/MTT used? Which model transformation engine is used?</td>
</tr>
<tr>
<td>Verification</td>
<td>Which verification techniques are used: model checking; theorem proving; testing</td>
</tr>
<tr>
<td>Traceability</td>
<td>Are backward and/or forward traceability implemented? Is traceability automatic or manual?</td>
</tr>
<tr>
<td>Tool Support</td>
<td>What is the automation level of the MDS approach? Which features does it provide?</td>
</tr>
<tr>
<td>Validation</td>
<td>Is the approach validated on large, meaningful cases? Has it been industrially validated?</td>
</tr>
</tbody>
</table>
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**;
Potential attacks are modeled by \textit{adversary machines}, independently from system models;

Constraints attached to the \textit{stereotypes} contain criteria for checking whether a security requirement violation is caused by \textit{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 (Jürgens, 2005a; Lloyd and Jürgens, 2009), the Common Electronic Purse Specifications (CEPS) project (Jürgens, 2004), a web-based banking application (Jürgens, 2005c; Houmb and Jürgens, 2003), the embedded system described in (Jürgens, 2007) and the mobile system described in (Jürgens 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.
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.
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
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 7: The metamodel of RBAC in the SECTET framework (adapted from Hafner and Breu, 2009)
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 Figure 8: Overview of the SECTET methodology)
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.

29
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
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/](http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/)
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, SECET, 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 done 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. SECET).
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>
<thead>
<tr>
<th>Application Domains</th>
<th>Security Concerns</th>
<th>Modeling Approach</th>
<th>(*)SoC Model</th>
<th>Tool Support</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMLsec</td>
<td>confidentiality, integrity, authenticity, authorization, freshness, information flow, non-repudiation, fair exchange</td>
<td>Mpm</td>
<td>o</td>
<td>X</td>
<td>compiler</td>
</tr>
<tr>
<td>SecureUML</td>
<td>web applications</td>
<td>MDA + DSM</td>
<td>✓</td>
<td>X</td>
<td>compiler</td>
</tr>
<tr>
<td>SECTET</td>
<td>access control</td>
<td>MDA + DSM</td>
<td>✓</td>
<td>QVT</td>
<td>Xpand</td>
</tr>
<tr>
<td>MODELSEC</td>
<td>web applications, databases</td>
<td>MDA + DSM</td>
<td>✓</td>
<td>RubyTL</td>
<td>MOF-Script</td>
</tr>
<tr>
<td>SECUREMDD</td>
<td>cryptography (secrecy, integrity, confidentiality), application-specific security properties</td>
<td>MDA + DSM</td>
<td>✓</td>
<td>QVT</td>
<td>Xpand</td>
</tr>
</tbody>
</table>

Note: Supported (✓); Partially supported (o); Not supported (X)
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), 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

---

3The 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


54


## Appendix A. List of Acronyms

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