Advances in Model-Driven Security

Levi Lúcio\textsuperscript{1}, Qin Zhang\textsuperscript{1}, Phu H. Nguyen\textsuperscript{1}, Moussa Amrani\textsuperscript{1}, Jacques Klein\textsuperscript{1}, Hans Vangheluwe\textsuperscript{c}, Yves Le Traon\textsuperscript{1}

\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

In this paper 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 for 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.

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

1. Introduction

The world is becoming more and more digital. On the 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, etc. 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 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 a higher-level of abstraction than code. MDS applies this paradigm to security engineering. There are several benefits to this. 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 those issues in the system infrastructure. Third, MDS leverages on MDE automation provided by model transformations such that human interference, which is naturally error-prone, is reduced.

In this paper 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 paper 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. Overall, we provide a broad picture of research activities in MDS for the last decade.

The remainder of this paper is organized as follows. In Section 2, we introduce some background information on MDE. Section 3 recall 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 describe a few selected MDS approaches against our taxonomy. Section 5 summarizes the result of our evaluation of the MDS approaches in a comparison table and provides a discussion of our findings during this study. Finally, the paper ends with some conclusive remarks in Section 7.

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 languages (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 studying aerodynamicity 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 visualisation, 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 (Ehrig et al., 2006)), but only recently have become a first class citizen in the software development world. Their need came naturally with the fact that MDE started to be used professionally in some software development environments, e.g. software
for mobile phones or software for the automotive industry. Implementations for transformation languages such as ATL (ATLAS, 2008), Kermeta (Muller et al., 2005) or QVT (Grégoire 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 (Fabian Büttner et al., 2012; Esther Guerra et al., 2013; Levi Lúcio et al., 2010; Gehan M.K. Selim et al., 2012a,b; Antonio Vallecillo and Martin 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 Modelling, an approach aiming at defining a language for each different domain that contributes to an application; Multi-Paradigm Modeling, a generalisation of Domain-Specific Modelling Languages where different models of computation interact; and Aspect-Oriented Modelling, studying more precisely how different models can be combined, or composed, together.

2.2.1. Model-Driven Architecture

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

- It builds on UML, an already standardised and well-accepted notation, already widely used in object-oriented systems. In an effort to harmonise notations and clean the uml internal structure, they proposed MOF for coping with the plethora of model definitions and languages;
• It proposes a pyramidal construction of models 3: 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, it enforces a particular vision of software systems development seen as a process with the following step: 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; these are then refined into Platform Specific Models (PSM) and combined with Platform Description Models (PDM) to finally use model transformations to reach the specific code running on the 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 within automated generators to produce code compliant with each platform. This methodology directly
inspired several MDS proposals for enforcing security concerns within applications.

2.2.2. Domain Specific Modelling

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 concerns, and focusing each one on a specific aspect of the system, it becomes possible not only to raise the abstraction level of the produced specifications, with the immediate benefit of raising the confidence attached to them, but also to make them closer to each domains experts, which facilitate the control of the produced artifacts, and sometimes even delegating their creations to these experts. Within MDE, Domain-Specific Modelling (DSM) becomes a key methodology for the effective and successful specification of such systems. This methodology makes systematic use of Domain-Specific Modelling Languages (DSMLs, or DSLs for short) to represent the various artifacts of a system, in terms of 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\textsuperscript{TM} and Lucent\textsuperscript{TM}. These encouraging results pushed the scientific community to investing further this topic, and build environments to facilitate the construction, management and maintenance of DSLs. This effort has been materialised with concrete frameworks: EMF and GMF (Moore et al., 2004), AToM\textsuperscript{3} (de Lara and Vangheluwe, 2002) or Microsoft’s\textsuperscript{TM} DSL Tools (Cook et al., 2007), among others.

2.2.3. Multi-Paradigm Modelling

Multi-Paradigm Modeling (MPM), as introduced in (Mosterman and Vangheluwe, 2004), is a perspective on software development that advocates not only that models should be built at the right level of abstraction regarding their purpose, but also that automatic model transformations should be used to pass information from one representation to another during development. In this case it is thus desirable to consider modeling as an activity that spans different models, or 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 Modelling

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

Modularisation of crosscutting concerns has been popularised 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). AOSD follows the well-known Roman principle of divide and conquer. Put 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, synchronisation, concurrency, persistence, response time, . . . ) 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 phase of composition, 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 modelling 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 modelling level. Themes are declaratively complete units of modularisation, 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. (August 2004); Raghu 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 modelling language with a well-defined meta model: e.g., Mata (Whittle et al., 2009), SmartAdapter (Morin et al., 2009), Geko (Kramer et al., 2013; Morin et al., 2008). Finally, advanced mechanisms have been proposed to unweave an aspect previously woven (Jacques Klein et al., 2009), to finely tune the weaving (Morin et al., 2010), or to weave aspects (or views) instance 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 leverages the conceptual approach of MDE as well as the associated techniques and tools to propose sound methods for engineering security-critical applications. To be more specific, models have to be the central artifacts in every MDS approach. More importantly, models are used extensively to capture security concern(s) and are the main materials throughout the development process in order to introduce/enforce security into the application. In this section, first we briefly mention several approaches that provide early the general concept of MDS. Then, we propose a taxonomy for MDS which we use later to evaluate different MDS approaches.

3.1. A brief history of MDS

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

The pioneering work of Jan Jürjens (2001) proposed in 2001 UMLSec: it is based on the UML extension mechanism for modeling and analyzing systems. Several UML diagrams are combined to ensure at different levels:
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. Jan Jürjens already considered a preliminary formal semantics of UMLSec sufficient for the set of addressed security properties. This approach does not explicitly mention MDE.

In 2002, Basin noticed, together with other authors (Torsten 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 the 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 secure 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 (DSL) 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.

### 3.2. Evaluation Taxonomy

This section identifies and describes a set of concepts pertaining to Model-Driven Security approaches which we use to build a taxonomy for MDS. We have based our taxonomy on the work of Khwaja and Urban (2002); Kasal et al. (2011); Nguyen et al. (2013). The taxonomy entries we have identified are as follows:

**Application Domains.** In order to develop secure systems targeting diverse application domains such as web applications, e-commerce systems, embedded systems, distributed systems, or others, an MDS approach may be more or less specifically developed with a domain in mind. The application domains entry of our taxonomy is devoted to evaluating each selected MDS approach in terms of its domains of applicability.

**Security Concerns.** There is a broad range of security concerns when information systems are considered. The European Network and Information Security Agency stipulates that systems need to address the various security aspects in a unified manner. According to that entity, security is “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).

An MDS approach may be so specific that it deals only with a specialized security concern. For example, the authorization concern encompasses several sub-concerns such as access control, delegation, obligation among others. In contrast, other MDS approaches might handle more than one “large” security concern simultaneously.

Regarding this taxonomy entry we will identify in Sect. 4 which security concern(s) the selected MDS approach deals with. If specific languages have been associated with the identified security concerns, we will also describe the metamodels of those languages.

**Modeling Approach.** Given Model-Driven Security specializes Model-Driven Engineering, it is natural that we classify MDS approaches according to their modeling paradigm. An MDS approach can for example rely on standard
UML modelling, where cross-cutting concerns are scattered across several related models such as is typically used in MDA. On the other hand, another MDS approach may make use of the Aspect-Oriented Modeling paradigm (AOM) in such a manner that crosscutting concerns are modeled in separately and subsequently woven into a primary model using a set of model weaving rules. Domain Specific Modelling (DSM) can also be used where a customized approach is required.

Together with the modeling paradigm, another important issue regarding modelling is the modeling language used: standard UML diagrams; UML profiles; tailored DSlS; or formal specification languages (e.g., Petri nets). According to the systematic review (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 modelling and engineering communities. Nonetheless, several researchers make use of DSlS with the goal of achieving more customized modeling capabilities. Formal specification languages are nevertheless well-accepted for handling specialized security/safety related issues at certain moments of the software development lifecycle, in particular for the verification of security protocols (Armando et al., 2005) or extending access control frameworks (Shafiq et al., 2005).

Separation of Security from Business. Separation of Concerns (SOC) is nowadays becoming a well-accepted design principle in computer science: it 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 one of these aspects: security concerns are clearly orthogonal to other application functionalities (Shin and Gomaa, 2009). According to the systematic review (Nguyen et al., 2013), most of the selected primary studies follow the principle of SOC, meaning that the security concerns are specified separately from the business logic in platform independent models (PIMs) and are later transformed into platform specific models (PSMs). The PSMs which are then are then refined into the security infrastructure and integrated with the system application logic.

In Sect. 4, we will evaluate for each selected MDS approach whether it deals with the security concerns separately from the business logic and, if so,
how that separation is achieved.

**Model Transformations.** Model transformations play a key role in the MDE development process. Model transformations allow preserving the consistency and preciseness of a family of MDE artifacts from the abstract models to the final running system’s infrastructure. The goal of employing model during MDE transformations is to automate as much as possible of the development process and as such to reduce error-prone manual activities.

In our MDSD context there are two main kinds of model transformations: 1) Model-To-Model Transformation (MMT), which normally serves the purpose of (but is not restricted to) refining between levels of abstraction during the development process; and 2) Model-To-Text Transformation (MTT) which allows producing textual artifact from models. Such textual artifacts may include source code, test cases, etc.

Another classification criterion described in (Nguyen et al., 2013) includes the notions of endogenous and exogenous transformations. This criterion distinguishes if the models involved in the transformation are expressed in the same language (endogenous) or in different languages (exogenous).

**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 MDSD approach.

Ideally, a model at a given level of abstraction during an MDSD 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 classic 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. Examples of such techniques are black-box, white-box or mutation testing (Mike Papadakis and Nicos Malevris, 2012), among others. However, despite their success within the developed community, these testing techniques are often error-prone themselves and require a large time investment. Furthermore, testing techniques are often only applied at the final stage of the development lifecycle.

An important advantage of MDSD approaches is the possibility of performing security property checking on abstract MDSD 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* is a very important feature of MDE as it allows keeping a *history* of how the models generated throughout the software lifecycle relate to each other. We distinguish between *backward* and *forward* traceability. *Backward traceability* 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. On the other hand, *forward traceability* implies that when a design flaw is corrected in a model at a higher level of abstraction, the corresponding modifications are propagated, when such propagation is meaningful, throughout the lower-level models onto the produced system’s infrastructure.

From our experience and after analyzing the literature, MDS traceability is usually done manually. In fact, automating traceability is hard due to the semantic gaps between layers of abstraction during the software generation lifecycle. Because of the fact that models in the MDS lifecycle are often refined, composed and recomposed, it becomes difficult to trace mistakes among different, or even within the same, layer of abstraction. For example errors in the generated system’s infrastructure are particularly hard to trace back given that security concerns can be distributed all over the generated code due to its low-level nature.

**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 modelling 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.
<table>
<thead>
<tr>
<th><strong>Taxonomy Entry</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Domains</td>
<td>Is the MDS approach domain specific or can it be used for multiple domains?</td>
</tr>
<tr>
<td>Security Concerns</td>
<td>What security concern(s) does the MDS approach focus on?</td>
</tr>
<tr>
<td>Modeling Approach</td>
<td>Modelling paradigm(s) used? (MDA, AOM, DSM)</td>
</tr>
<tr>
<td></td>
<td>Modeling language(s) used? (standard UML; UML profiles; Domain Specific Languages; Formal Languages)</td>
</tr>
<tr>
<td>Separation of Security from Business</td>
<td>Is separation of concerns (SOC) used?</td>
</tr>
<tr>
<td></td>
<td>If yes how is it implemented?</td>
</tr>
<tr>
<td>Model Transformations</td>
<td>Are Model-To-Model Transformations (MMT) used?</td>
</tr>
<tr>
<td></td>
<td>Are Model-To-Text (MTT) transformations used?</td>
</tr>
<tr>
<td></td>
<td>What model transformation engine is used?</td>
</tr>
<tr>
<td>Verification</td>
<td>What verification techniques are used? (model checking; theorem proving; testing)</td>
</tr>
<tr>
<td>Traceability</td>
<td>Are backward and/or forward traceability implemented?</td>
</tr>
<tr>
<td></td>
<td>Is traceability automatic or manual?</td>
</tr>
<tr>
<td>Tool Support</td>
<td>What is the automation level of the MDS approach and what features does it provide?</td>
</tr>
<tr>
<td>Validation</td>
<td>What studies exist and how large and meaningful are they? Has the approach been industrially validated?</td>
</tr>
</tbody>
</table>

Table 1: MDS Taxonomy Entries
Validation. The validity of a particular MDS approach is evaluated by analyzing the case studies available in the literature. For each approach we will evaluate how many proof-of-concept and/or industrial case studies exist, what are the conclusions of those studies.

In table 3.2 we provide a summary of the taxonomy entries identified in this chapter including a brief description of each of them. Despite the fact that we describe them separately, the MDS concepts that compose our taxonomy naturally relate and depend on to each other. As an example, if an MDS approach uses the taxonomy concept separation of security from business to distinguish security concerns from business logic at the modelling level, further in the MDS chain the taxonomy concept model transformations is required to allow integration of the security and business models at the level of code and overall infrastructure generation. As another example, the verification methods used depend on the modelling approach that has been taken, and in particular on the modelling 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 for 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 us to identify MDS studies among a mass of security-related studies in the literature.

In this section, we evaluate five MDS approaches selected from the literature against the taxonomy defined in Sect. 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 the 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.
4.1. UMLsec

**Application Domains.** UMLsec (Jan Jürjens, 2001; Jürjens, 2002; Jan Jürjens, 2004, 2005b; Best et al., 2007; Hatebur et al., 2011) is a UML profile extension for the analysis of secure 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 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* (Jan Jürjens, 2007), and *distributed systems*.

**Security Concerns.** UMLsec deals with 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 of 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 attempts to assign to a system variable a erroneous value. The *integrity* property holds if no *constraints* associated to the variable are violated during the attack.

UMLsec is relatively different from the other MDS approaches in the literature given that it concentrates on providing analysis capabilities for security models, rather than on insisting on the security modelling languages. Although clear formal semantics have been defined in (Jan Jürjens, 2001) for the UML fragment used in the UMLsec profile, 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 our definition of *Separation of Security from Business* 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;
• Potentially malicious attacks are modeled in adversary machines, independently from system models;

• Constraints attached to the stereotypes contain criteria for checking whether a security requirement violation is caused by adversaries of the system design;

• Security and recovery mechanisms are defined together with the business logic in the system models.

Due to the fact that the security issues are deeply coupled with system models, the SoC in UMLsec is relatively weak when compared to other
MDS approaches we survey in this paper. 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 diagram 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 is a subsystem specification (see Fig. 4):

- **Use Case Diagram** is used to capture user-system interactions.

- **Activity Diagram** specifies the business workflow of the system.

- **Sequence Diagram** concentrates on component interactions, such as inter-component data flow and protocol.

- **Statechart Diagram** models the dynamic behaviors of a component and the corresponding state changes.

- **Deployment Diagram** is used to model communication links among components such as the client, server, database, etc.

Note that the fragment of UML used by UMLsec as well as the adversary machines are provided a precise semantics using the UML Machines (Jan 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 woven in a concretized model. From the concretized model, a first-order logic theory is generated and security properties can be proved on it 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 model directly from code 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 a theorem prover, called aiCall. A Prolog-based tool is then used to verify the system’s specified 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 for the identification of system design flaws. However, no tool-supported automatic traceability is provided.

**Validation.** UMLsec has been applied for analyzing the security of several industrial applications, such as the development of the biometric authentication system (Jan Jürjens, 2005a; Lloyd and Jürjens, 2009), the Common Electronic Purse Specifications (CEPS) project (Jan Jürjens, 2004), a web-based banking application Jan Jürjens (2005c); Houmb and Jürjens (2003), the embedded system described in (Jan 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 (Torsten 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 these new
Figure 5: SecureUML metamodel (Basin et al., 2006)

languages need to be defined as new UML profiles. Given its generality SecureUML can be applied to many domains. Nonetheless, several concrete examples presented in the SecureUML literature refer to web applications.

**Security Concerns.** The SecureUML approach is focuses 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 that where security should be enforced and 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, in (Basin et al., 2006) the authors 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 modelling 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 a naturally embedded in the framework as can be seen by the explicit separate modelling of the business and the security RBAC concerns. Basin et al. use in their work two main UML-based modeling languages for system modelling: simplified UML Class Diagrams and simplified UML Statecharts.
Composing the business and security models is achieved by a dialect language, linking the system modelling language to the access control RBAC language. This dialect language is also defined as a UML profile and allows specializing the class of Resource and Action 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 for example Execution or Activation. The dialect language specializes the Resource and Action classes in the SecureUML in Fig. 5 into the State and StateMachineAction concepts of the statechart-like system modelling 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 modelling 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.

**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. Both instances of the code generation technique map the security language’s concepts into built-in security mechanisms in both EJB and .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’s 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 (David A. 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 modelling 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 also by Lodderstedt (2003) in his Ph.D. thesis. Finally, Manuel Clavel et al. (2008) reports about the applicability of SecureUML to an industrial case study. The conclusion is that the SecureUML approach allows better understanding of the manipulated concepts, early analysis and fault detection, reusability and evolvability. He also mentions that while access control proved relevant in the case study, it is merely one of the several security concerns that their industrial partner demonstrated interest in.

4.3. Sectet

Application Domains. Sectet (Alam et al., 2007a; Fernández Medina et al., 2006; 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) that is 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 denied;

- **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 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 subject, role and permission, extended by dynamic constraints defined on the permission. The role and the subject (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 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, such as roles and their hierarchies, which can be expressed in three kinds of workflow views (as shown in Fig. 8):

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

- **Local Workflow**: The local workflow represents an intra-organizational workflow which is assumed to be the execution of a business process by a particular service component. Thus a local workflow model can be 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 public 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 be seen in Fig. 8, an integrated platform independent application model 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, i.e. 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 in SECTET-PL to form a platform-independent application model (PIM). These MMT transformations rely on the transformation language.
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 (Fernández Medina et al., 2006; Hafner et al., 2006) 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 ModelSec approach using an example taken from (Eduardo Fernández Medina and Mario Piattini, 2005) of a web application for the management of medical patients. 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. The ModelSec approach allows dealing with other security concerns as is shown in the metamodel in Fig. 9. 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.
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.

Fig. 9 depicts the metamodel of the security requirement language for ModelSec. This metamodel is used to build a DSL in which concrete requirement models can be designed. According to the metamodel, a complete security requirements cluster consists of:

- **Asset**: A physical or logical object which may be exposed to threats;
- **Threat**: Assets can be damaged by a threat;
- **Safeguard**: An impediment to a risk, which can be a measure or an action against the risk;
- **Contingency Plan**: A set of safeguards recommended for reducing risks;
- **Security Requirement**: The security concerns which the system should implement or guarantee.
Figure 10: Overview of the MODELSEC approach

Note that in the security requirements metamodel in Fig. 9 the SecurityRequirementsCluster class is used to attach a set of security requirements to an asset.

Modeling Approach and Separation of Security from Business. MODELSEC follows an MDA process developed based firstly on a requirements metamodel which we do not present here but can be seen in (Sánchez et al., 2009). The security requirement metamodel in Fig. 9 specializes the requirement metamodel by integrating security concerns in it. MODELSEC provides a DSL called SecML (Security Modeling Language) for modeling security requirements and several languages such as use cases or class dia-
grams to represent the business logic of the system.

In figure 10 we graphically depict the ModelSec approach. ModelSec follows the principle of separation of security concerns from application as from the very beginning of the development lifecycle security requirements and business requirements are already separated. The security models are synchronized with the business model at several steps of the process.

From the top level requirement models security design models and business design models are obtained via model transformations. These design models are instances of the corresponding design metamodels, as detailed in (Sánchez et al., 2009), which extend the security and function requirement metamodels by adding the possibility to represent design decisions. Thus, 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.

A security implementation model including platform specific information for a given platform is then produced from the security design model from which the code for the security infrastructure can be generated. The application code is directly generated from the business design model.

The ModelSec approach as described in (Sánchez et al., 2009) is however very sparse regarding details on how to integrate the business model with the security model or on how the produced XACML 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. Verification has not been addressed in the ModelSec approach.
Traceability. The MODELSEC approach does not mention about traceability support.

Validation. MODELSEC has been illustrated using only the academic case study in (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 cryptography is rarely the target of attacks, but the protocol is often subject to attacks known as third-party intruder; i.e. an attacker that can intercept messages to read them (thus breaching confidentiality of data), delete or forge 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 generalised to other context, SecureMDD targets applications with the following components. 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 (like a health center to access data for a patient, or a bank to check that an amount is actually available), the Terminal makes use of identified protocols that need to be trusted to ensure the integrity of the whole system.

The authors 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 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 application in this domain. However, the authors promote the specification of application-specific properties that are, in their opinion, preferable to better guarantee the security of an application. For example, Moebius et al. (2010) model how a copy card works, and ensure that “the provider of a copying service does not lose money”.

33
Strictly speaking, SECUREMDD does not use a dedicated language for specifying security properties. Instead, they are specified directly using a logic (the Dynamic Logic) tailored to ASMs, to which UML diagrams are generated for verification purposes.

**Modeling Approach.** SECUREMDD combines UML profiles for the static part and UML sequence and activity diagrams for capturing behaviour. MEL, the Dsl created by the authors, allows to define fine-grained behaviours over UML diagrams and interact with cryptographic protocols. It has a textual representation, probably more suitable for engineers for early experimentations when designing the application; but it also has a model representation, aimed at being integrated with UML diagrams for enabling full verification over the whole system.

Since SECUREMDD enables formal verification by means of theorem-proving, all languages receive a formal semantics: a formal semantics for the various UML diagrams used for designing the application has been defined in Kiv\(^1\), but also a semantics for Java, the target language chosen for generating code from UML. These semantics are expressed in terms of ASMs, the approach the authors have chosen for performing the verification task.

Figure 11 depicts the general approach for SECUREMDD. The design process starts with the creation of an UML model for the application, generally following functional requirements. 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 behaviour and the interactions between those components. The authors provide concrete examples of such modelling artefacts: Moebius et al. (2009b) shows such diagrams for the copy card application: the interactions between a terminal and a smart card for securely loading money on the card are fully explained; and Moebius et al. (2012) provides several diagrams for another project, the german electronic health card.

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

\(^1\)Web page for the Kiv Theorem-Prover: http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/
Moebius et al. (2012) also proposes a methodology for designing applications in an iterative 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, but also security property specifications, are modeled and handled can quickly become a challenge on itself. But an iterative development 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.** 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.** SecureMDD makes extensive use of transformations to reach executable code from platform-independent models that capture early requirements. Figure 11 illustrates the points where transformation techniques are used. MMT techniques are used to transform the platform-independent models representing the weaved model into two different models: on the left, a Java model aimed to be executed, 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): the specific built-in data types and values, the configuration settings, the serialisation mechanism for protocol exchange messages, the encryption/decryption specific algorithms (e.g., JavaCard requires that all objects are allocated at the beginning of the execution, and does not provide garbage collection: these critical algorithms cannot be identical on the card, and on the server), etc.

From these models, MMT techniques are used for generating the actual code necessary for execution and verification, since it is the normal entry format for Java and the ASM interpreter and verifier. These transformations are implemented with XPand, a statically typed template language focusing on the generation of textual artefacts. 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 accesses and the user interfaces are left to programmers to reach the expected quality of such components (Moebius et al., 2009c,a).

**Verification.** In SecureMDD, the 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 an 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 delay of several weeks 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 the algebraic specifications 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 itself). 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’ behaviour 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 (in particular, use case related to the parts, like using the user interface, whose code is not automatically generated).

**Traceability.** SecureMDD uses a generative approach: once the composed model is verified, i.e. security properties are checked to be enforced within the system, the relevant part of the code corresponding to the core, security-related part of the system is generated. Therefore, all security-related flaws are caught earlier, and the approach does not require *per se*
Tool Support. The authors do not specifically describe the tool support; however, several papers provide insights on the modelling artefacts, the transformation chains and the verification tasks (in particular, (Moebius et al., 2009c, 2012).

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)). Then, it evolved towards a medium-sized case study, the copy card application (Moebius et al., 2010), where more advanced security properties (with a particular emphasis for 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), which gave a positive result on the applicability of the approach that goes beyond the applicability range of other approaches. These results were obtained in a relatively short time (from 2009 to 2012)\textsuperscript{2}.

5. Discussion

5.1. Synthesis of the Evaluation Section

Table 2 provides a synthetic view of the evaluation section. The columns list the MDS taxonomy entries defined in Section 3.2. Each row deals with one selected MDS approaches. This comparison table allows us to draw the following conclusions:

1) Regarding 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.

2) For 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

\textsuperscript{2}The webpage of the project provides details about other projects that were, to our knowledge, not yet published: https://www.informatik.uni-augsburg.de/de/lehrstuehle.swt/se/projects/secureMDD/.
properties. The others approaches, i.e. UMLsec, SECTET, and ModelSec are open to deal with multiple security concerns.

3) Regarding modeling paradigm, all approaches combine the model-driven architecture paradigm with domain-specific modeling paradigm. One exception: UMLsec applies multi-paradigm modeling methodology (MPM). Different views of a system are modeled by different UML diagrams and finally all these views are composed to form the complete specification of the system.

4) Regarding modeling language, only ModelSec uses a DSL based on a non-UML-based language. The other approaches used DSLs defined as UML profiles.

5) Regarding the principle of separation of security concerns from business logic, only UMLsec does not fully follow it. Indeed, in UMLsec security-related information is partially contained in system models. The other MDS approaches we have analyzed either encapsulate security concerns in one specific kind of model artifact (e.g. SECREMDD), or clearly separate security concerns from business logic in the metamodel (e.g. SECTET).

6) Regarding model transformations, ModelSec defines its own MMT tool by extending the Ruby programming language and applies 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.

7) Regarding 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 system models.
Table 2: Comparison of the Evaluated MDs Approaches

<table>
<thead>
<tr>
<th>Application Domains</th>
<th>Security Concerns</th>
<th>Modeling Approach</th>
<th>(*)SoC</th>
<th>Model Transformation</th>
<th>Verification</th>
<th>Traceability</th>
<th>Tool Support</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Paradigm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modeling Language</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMLsec</td>
<td>web applications,</td>
<td>confidentiality,</td>
<td>MPM</td>
<td>UML profiles</td>
<td>o</td>
<td>X</td>
<td>compiler</td>
<td>high</td>
</tr>
<tr>
<td></td>
<td>embedded systems,</td>
<td>integrity,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>distributed systems</td>
<td>authenticity,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>authorization,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>freshness,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>information flow,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-repudiation,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>fair exchange</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecureUML</td>
<td>web applications</td>
<td>access control</td>
<td>MDA</td>
<td>UML profiles</td>
<td>✓</td>
<td>X</td>
<td>compiler</td>
<td>medium</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+ DSM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sectet</td>
<td>e-government,</td>
<td>integrity,</td>
<td>MDA</td>
<td>UML profiles</td>
<td>✓</td>
<td>QVT</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>e-health, e-education</td>
<td>confidentiality,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-repudiation,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>access control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ModelSec</td>
<td>web applications,</td>
<td>privacy,</td>
<td>MDA</td>
<td>SecML (tailored DSL)</td>
<td>✓</td>
<td>RubyTL</td>
<td>X</td>
<td>low</td>
</tr>
<tr>
<td></td>
<td>databases</td>
<td>integrity,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>authentication,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>availability,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-repudiation,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>auditing,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>access control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SecureMDD</td>
<td>smart card and</td>
<td>cryptography</td>
<td>MDA</td>
<td>UML profiles</td>
<td>✓</td>
<td>QVT</td>
<td>X</td>
<td>high</td>
</tr>
<tr>
<td></td>
<td>service applications</td>
<td>(secrecy, secrecy,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>integrity,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>confidentiality,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>application-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>specific security</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>properties</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note:
Support (√): Partially support (o); Doesn’t support (X);
(*)SoC: Separation of security concerns from business logic;
MPM (Multi-Paradigm Modeling); MDA (Model-Driven Architecture); DSM (Domain-Specific Modeling);
8) In what concerns 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.

9) Mds inherits from Mde, so it is not surprising that tool support is provided by all the evaluated Mds approaches.

10) Finally, UMLsec, SECTET and SecureMDD are ranked at a high level for validation because each of them was experimented using a series of case studies, including 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 at a medium level. 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.

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 modelling effort.

Breu et al. (2008) emphasizes 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 in the heart of AOM (Aspect-Oriented Modeling), no evaluated MDS approach 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 AOM paradigm, but we will attempt to propose some 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.

5.3.2. Security Concerns and Corresponding Metamodel

According to the security concern metamodels for the evaluated MDS approaches in Section 4 (Fig. 5, Fig. 7 and Fig. 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 modelling. 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 is 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 paper 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 (Jan Jürjens, 2004), Secure Software Architecture (Yu and Liu, 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) (no reference given in their work for this tool), – a LTL/CTL general-purpose model-checker, and Alloy – a SAT-based solver for relational specifications (Jackson, 2011). Neither can be considered as an MDS approach.

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.

Another systematic review on MDS, conducted by Nguyen et al. (2013), is closer to our work: it explicitly targets MDS approaches. The paper (Nguyen et al., 2013) is a systematic literature review in which a review protocol is clearly defined. In their review protocol, a “search” and “selection” strategy is employed in order to select the most significant MDS papers from a large set of candidates. Based on the review protocol, they conduct a rigorous search-review-select process to select a final set of 80 primary MDS papers from more than 10,000 candidate ones. According to their evaluation criteria and data extraction strategy, they extract and synthesize the extracted data from these 80 papers. In fact, their evaluation criteria is contained in our taxonomy for MDS evaluation. Based on the synthesis result, the authors present and discuss the obtained statistics against each evaluation criterion. For example, w.r.t the security concern(s) criterion, their result shows that most current MDS approaches focus on addressing authorization, especially access control. The paper also briefly mentions and discusses five main MDS approaches selected from their review process. However, due to the limitation of a systematic review, their discussion on those five MDS approaches is very short and narrow. In contrast, our paper deeply describes, evaluates, and discusses a set of well-recognized MDS approaches, rather than presenting a set of statistics on a large set of papers.

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 in MDS.
7. Conclusion

In this paper we have presented the main advances in MDS 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. We finish the paper by a general discussion about the current state of MDS, its limitations and open issues.

This paper provides a broad view of the field and is intended as an introduction to MDS for students, researchers or practitioners.

References


Alam, M., Hafner, M., Breu, R., 2006b. Constraint based role based access control (CRBAC) for restricted administrative delegation constraints in


Aspect-Oriented Software Development VII. Springer-Verlag Berlin Hei-
delberg, volume 6210 of Lecture Notes in Computer Science, pp. 272–320.

Model Unweaving, in: 5795, L. (Ed.), In 12th International Conference
on Model Driven Engineering Languages and Systems (MODELS 2009),

Klein, J., Fleurey, F., Jézéquel, J.M., 2007. Weaving Multiple Aspects in Se-
quence Diagrams. Transactions on Aspect-Oriented Software Development

Klein, J., Hlouet, L., Jézéquel, J.M., 2006. Semantic-based Weaving of Scen-
arios, in: proceedings of the 5th International Conference on Aspect-
Oriented Software Development (AOSD’06), ACM, Bonn, Germany. pp.
27–38.

Driven Architecture: Practice and Promise. Addison-Wesley Longman
Publishing Co., Inc., Boston, MA, USA.

Kramer, M.E., Klein, J., Steel, J.R.H., Morin, B., Kienzle, J., Barais, O.,

Lang, U., Schreiner, R., 2008. Model Driven Security Management: Mak-
ing Security Management Manageable in Complex Distributed Systems,
in: Modeling Security Workshop in association with MODELS, Toulouse,
France.

de Lara, J., Vangheluwe, H., 2002. AToM³: A Tool for Multi-formalism and
Meta-Modelling, in: FASE ’02: Proceedings of the 5th International Con-
ference on Fundamental Approaches to Software Engineering, Springer-
Verlag. pp. 174–188.

System Using UMLsec and JML, in: Model Driven Engineering Languages
and Systems. Springer Berlin Heidelberg, volume 5795 of Lecture Notes in
Computer Science, pp. 77–91.


