Model-driven Development Meets Security: An Evaluation of Current Approaches

Kresimir Kasal
SBA Research
kkasal@sba-research.org

Johannes Heurix
Vienna University of Technology
heurix@ifs.tuwien.ac.at

Thomas Neubauer
Vienna University of Technology
neubauer@ifs.tuwien.ac.at

Abstract

Although our society is critically dependent on software systems, these systems are mainly secured by protection mechanisms during operation instead of considering security issues during software design. Deficiencies in software design are the main reasons for security incidents, resulting in severe economic consequences for (i) the organizations using the software and (ii) the development companies. Lately, model-driven development has been proposed in order to increase the quality and thereby the security of software systems. This paper evaluates current efforts that position security as a fundamental element in model-driven development, highlights their deficiencies and identifies current research challenges. The evaluation shows that applying special-purpose methods to particular aspects of the problem is more suitable than applying generic ones, since (i) the problem can be represented on the proper abstraction level, (ii) the user can build on the knowledge of experts, and (iii) the available tools are more efficient and powerful.

1. Introduction

As our modern society is critically dependent on software systems, the importance of software security is constantly growing [1]. For example, companies depend on applications to administer customer data, payment information and inventory tracking. But not only companies have a need for secure software: consumers also use software to communicate with friends or family, to check their banking accounts and to search for resources available on the Internet. Threats resulting from security breaches range from defeating copy protection mechanisms to attacks such as malicious intrusions into systems that control crucial infrastructure (cf. [2]). Software vulnerabilities, arising from deficiencies in the design or implementation of the software (e.g., due to increasing complexity) are one of the main reasons for security incidents (cf. [3]). These deficiencies are often caused by the increasing complexity of software systems. This is addressed with principles like abstraction, modularization, and separation of concerns, concepts which are all widely used. Although the object-oriented paradigm is mostly employed nowadays, principles like encapsulation, polymorphism, and inheritance are insufficient, and a paradigm change is necessary [4]. For this reason, as a successor of the computer-aided software engineering (CASE) approach, model-driven development (MDD) has been suggested to improve the quality of complex software systems [4], [5]. MDD is used to design abstractions, i.e., platform-independent concepts, which are then translated into more accurate ones that are adjusted to a particular platform. In a further step, such platform-specific models are transformed into production code. In such a development process, models and mappings between them have to be maintained instead of just the generated code. Aspect-oriented software development (AOSD) is an emerging approach with the goal of promoting advanced separation of concerns (cf. [6], [7]). It allows multiple concerns (e.g., security, functionality) to be expressed separately and unifies them into a working system in an automated way. Because of good characteristics in tackling software complexity, model-driven engineering was utilized to develop secure information systems. Juerjens was the first to propose a combination of model-driven development and security using UMLsec (cf. [8]). Subsequently, many proposals dealing with integrating security and modeling languages followed and were summarized under the term model-driven security (cf. [9]). It represents an approach where security is applied together with model-driven architecture [4] and focuses on building secure software systems by specifying models together with their security requirements. At the other end of the spectrum, researchers have proposed formal languages, called specification languages, to represent policies, models, and system descriptions. Such languages are based on mathematical logic systems and have also been applied to the field of information security, for instance for specifying formal security policies and for analyzing cryptographic security protocols [10]. A great number of modeling and specification approaches for describing secure information systems are available, and the question arises which method to use for which problem. When intending to apply model-driven security, or at least to

analyze a model of a system, it is fundamental to know which security mechanisms and security requirements can be modeled by a certain technique and whether an appropriate toolchain exists. As there is a multitude of available modeling approaches, it can become tedious to identify the most suitable method for solving the problem at hand. There is no common comparison framework to contrast the different methods with each other with regard to security and to identify the most suitable approach.

Therefore, this work defines a taxonomy for model-driven security based on the work of Khwaja and Urban (cf. [11]). In particular, we extend Khwaja and Urban's comparison framework with security mechanisms that can be modeled using a certain specification method. In this way, we will answer the research questions (i) which approaches are applicable for solving which development problems and (ii) what specific features characterize these techniques. In the scope of this paper we regard evaluation as the "systematic assessment of the operation and/or the outcomes of a program or policy, compared to a set of explicit or implicit standards, as a means of contributing to the improvement of the program or policy" (cf. [12]). We use a combination of a testing program approach and an objectives-based approach (cf. [13]). The objectives used for the evaluation are taken from literature. This evaluation provides management decision makers such as chief security officers or software developers with a funded decision-making basis for the selection of model-driven security approaches. As literature does not so far provide any evaluations focusing on the comparison of model-driven security approaches, and there is as yet no common comparison framework to contrast the different methods to each other, this paper provides a major contribution to the research area of software engineering.

2. The Evaluation Taxonomy

This section identifies and describes several dimensions in model-based security. The identified dimensions are orthogonal but can still affect each other. For example, if the modeling paradigm is aspect-oriented, one of the artifacts that the method requires is a set of transformation rules that are needed for model weaving. The proposed taxonomy is influenced by the comparison framework developed by Khwaja and Urban (cf. [11]). The framework was intended for evaluation of specification techniques and was already used by Villarroel et al. (cf. [14]) to evaluate development methods for secure information systems. Nevertheless, Khwaja and Urban's comparison framework did not cover aspects such as security mechanisms that can be modeled by a specific technique, and it did not classify the distribution of modeled systems, the artifacts that have to be provided by the modeler, or the applied modeling paradigm. Furthermore, there was no differentiation between several possible dimension instantiations that can occur in system verification. Therefore, these issues are handled in the proposed taxonomy. The taxonomy will provide a classification method that can easily be applied by a practitioner when comparing model-driven security methods in order to choose the appropriate one and consists of the following dimensions:

Paradigm: This dimension is concerned with the modeling paradigm. In case of model-driven security, two alternatives are possible: In single, possibly hierarchical models, crosscutting concerns are modeled in each place where they are needed, which amounts to a significant redundancy in the overall model. By contrast, in models conforming to aspect-oriented development, crosscutting concerns are described in a separate model to be subsequently woven into (or integrated with) the primary model using so called weaving rules, thus eliminating redundancy.

Artifacts: There are three forms of artifacts: (i) Static models describe the static structure of a system such as classes and associations between them; (ii) Dynamic models describe the behavior of a system including interactions or states of entities; (iii) Transformation rules define how models are transformed in the MDA approach and how aspects are woven into aspect-oriented models.

Formality: Formal expressions allow for precise, unambiguous and verifiable specifications of models. Design patterns, for instance, are semi-formal models specifying the system's functionality by describing how entities are assembled and interact with each other to form the desired system. Metamodels are grammars describing how valid models are built. Examples of formal modeling techniques are automata, state machines, or logics and calculus systems.

Distribution: This dimension deals with the existence of distributed components and their interoperability. Differences are made between single or multi-process systems, where the latter may be distributed over multiple machines acting as autonomous and possibly mobile agents. Different instances of this dimension include client/server constructs, P2P-architectures or multiple agent systems.

Granularity: Coarse granularity allows abstracting from details to get a more complete picture of the system, e.g., describing the system as a composition of interacting subsystems where each subsystem consists of several other components that are left outside the scope of the model. Fine-grained models offer a more detailed view, including elements such as classes and interacting functions, but may also be of much higher complexity.

Executability: If the model is executable, it contains enough information to be verified without the need to be enriched with additional information, since its semantics is represented in a mathematically precise and

unambiguous way. Otherwise, the model could not be executed by a machine. In such a case, it needs to be supplemented with additional data during the transformation process. With executable models, extracting test cases from them and making them executable is a plausible technique for validating the system.

Verification: The system's verification can be handled in different ways: While manual testing is error-prone and tedious, automated test case generation is more preferable where we distinguish between (i) deriving both test cases and production code from the same model (ii) and test-specific models being independent from the system built manually. Model checking verifies conformance to a specific requirement, while theorem proving involves verifying whether a theory (system specification) entails a logic formula (requirement).

Tool support: If available, tool support includes assisting the user in the modeling process of the system, generating code, checking the system specification's syntax and consistency, as well as checking whether a system is consistent with its specification (verification) and completely fulfills the user requirements (validation).

Applicability: In this dimension we differentiate between several application domains for which systems can be specified by applying a particular technique. Examples for application domains are information systems, Web applications, e-commerce systems, embedded systems, etc.

Security mechanisms: In this section, security modeling techniques are categorized according to security mechanisms (and thus, indirectly, security requirements) that can be represented and modeled by a particular method. Possible instances for security mechanisms are access control, security protocols and intrusion detection mechanisms. Security mechanisms enforce security requirements. A security aspect represents a particular set of behavior needed by a certain system, as used in aspect-oriented software development (AOSD). A security aspect can represent a security mechanism.

2. Model-driven Security Approaches

This section presents selected model-driven security and formal method approaches, starting with UMLsec, a hierarchical methodology, followed by interesting aspect-oriented approaches. Finally, general and special-purpose formal techniques will be discussed.

2.1. UMLsec (Juerjens 2002)

UMLsec is a very general and powerful technique. It enhances UML's expressiveness by applying security-related stereotypes, tags, and security constraints. These are used to encapsulate knowledge on prudent security engineering such that developers need not be specialized in the domain of security [3]. UMLsec is both a modeling language and a methodology, since the corresponding tool suite allows for iterative refinement and adaptation of the system models. UMLsec, as a modeling language, allows the specification of requirements regarding confidentiality, integrity, non-repudiation, and non-interference (secure information flow). These requirements are expressed as stereotypes and tagged values and are translated into constraints that evaluate the security properties of the model, since the author provides a formal semantics for the fragment of UML that is needed for UMLsec. A system is composed of subsystems which are in turn composed of further subsystems or components that can be modeled in the form of class diagrams or state charts. UMLsec can also be applied to model aspects and systems' crosscutting concerns separately. In such a case, the modeler would have to provide transformation (weaving) rules that specify and determine how specific models have to be integrated [15]. By applying UMLsec, the system can be described on several levels of granularity. Even if there is no consensus on whether UML is an architecture description language (ADL) (cf. [16]), it can also be used to model system architectures, since package and deployment diagrams can be expressed by using UML and therefore also by UMLsec. Formal semantics was provided [3] to formally analyze the behavior of interacting components. Because interactions can be specified in UML, distributed systems can be modeled as well [17]; Juerjens demonstrated this with the TLS security protocol [3].

In UMLsec the modeler has to provide static and dynamic models including secrecy (confidentiality), integrity, non-interference, non-repudiation, data authenticity (can a piece of data be traced back to its original source?), and entity authenticity (can a protocol participant be identified?) as security requirements. Although Juerjens provided a formal basis and thus the foundation for executable UML modeling to simulate whole systems, UMLsec lacks support for exact, traceable and fully automated transition from models to implementation code, thus denying executability. In the meantime, this problem has been solved in the form of a rich toolset making it possible to formally verify the designed models [18]: With state-of-the-art model checkers and theorem provers, this toolset allows the automatic analysis of exported UML models formatted in XML Metadata Interchange (XMI) format. Although UMLsec was initially intended to tackle the problem of designing secure information systems, the author recently also addressed the development of secure embedded systems (cf. [19]). As this approach evolves, further application areas may emerge.

2.2. Secure Software Architectures by Using Aspects (H. Yu et al. 2005)

Yu et al. [20] apply software architecture models (SAM) to define the system's software architecture and the required security aspects. The approach is a formal method for aspect-oriented modeling at an architectural level. SAM is a development framework based on two complementary formalisms: predicate transition nets (also referred to as high-level Petri nets) and temporal logic. Petri nets are used to visualize and describe the high-level static structure of the system, as well as to model the architecture's behavior. Linear temporal logic formulas (LTL) are used to specify the required security properties (that is, security requirements, such as constraints set on the information flow between the interacting components). A consequence of expressing security properties in temporal logic is that policies (i.e., sets of security properties or requirements) are expressed on a very low abstraction level. Expressible policies include safety and liveness properties and variations of these (cf. [21]). In SAM, a hierarchical set of compositions is used to describe the system. Each composition consists of a set of components, a set of connectors, and a set of constraints that have to be satisfied by the interacting components (cf. [7]). The approach is well suited for modeling distributed architectures. In the problem domain model, a precise description of the system's functionality is given. Once this model is established, it is divided into the base architecture model and the security aspect model by applying separation of concerns. The base architecture model defines the software architecture of the targeted application, including basic functional modules and their connections. In this model, no security properties (requirements) are specified; these are specified in the security aspect model, along with vulnerabilities, threats, and provided mechanisms that enforce security policies. Thus, security aspects in this context represent components which implement security-relevant features and mechanisms of information systems. A secure architecture model, the result of merging the base architecture model with the security aspect model, is the model where security policies are enforced. As Petri nets are the basis for the modeling formalism, model checking could also be applied to verify security properties (which are represented by LTL formulas). However, the approach lacks any tool support and no significant further work has been done in order to enhance the proposed method.

2.3. A Model-Based Aspect-Oriented Framework for Building Intrusion-Aware Software Systems (Zhu et al. 2008)

Zhu et al. [22] propose a model-based, aspect-oriented framework for building intrusion-aware software systems. Such a system includes a group of intrusion detection aspects (IDAs) which can automatically detect intrusions. The authors developed a UML profile with aspect-oriented extensions to model attacks and thereby intrusion detection aspects (IDAs) responsible for detecting these attacks. The modeler has to provide static and dynamic views of the system's aspects. Class diagrams are used to represent the system's static attributes, and state machine diagrams are used to represent the dynamic views of intrusions, which describe how the attacker intrudes into the system. The attack scenario models are then transformed into programs (i.e., code is generated for the IDAs) and subsequently woven into the primary program. After weaving, the aspects act as intrusion detection components to automatically identify attacks against the target system. The method is not limited to a specific application domain and the framework can also be used to make distributed systems intrusion-aware. Several open source tools like ArgoUML, AspectJ, and Novasoft Metadata Framework are used to build the framework. When modeling intrusions, the level of formality is high due to state machines describing the attacks. Nevertheless, when considering the remaining application, which is written in an ordinary programming language and for which no model is available, the system is too complex to be verified formally. Therefore, the resulting intrusion-aware application needs to be tested manually by applying a set of attacks from the Web security threat classification released by the Web Application Security Consortium (WASC) [23].

2.4. Automated Validation of Internet Security Protocols and Applications

In [24], the authors propose a tool, called AVISPA, intended to speed up the development of security protocols and to improve their security. The approach provides a language called the High-Level Protocol Specification Language (HLPSL) which is used to describe the protocols and their intended security requirements, and a number of analysis tools to formally validate them [10]. The authors state that the approach provides a modular and expressive formal language for specifying security protocols and properties, and integrates several different back-ends that implement a variety of automatic analysis techniques ranging from protocol falsification to abstraction-based verification methods for both finite and infinite numbers of sessions [24]. The language offers an expressive formalism that allows specification of roles, control flows, data structures as well as security requirements [24]. Providing four separate analysis back-ends, the specification validation is tackled from different angles. Upon termination, the analysis result is presented, stating whether the problem could be solved, whether the problem could not be solved due to exhausted resources (e.g., memory) or some other reason that prevented the tool from solving the problem.

In general, the method offers a high level of formality, since HLPSL is based on Lamport's Temporal Logic of

Actions [25]. The user has to provide a dynamic model of the system's behavior which is represented by a distributed system consisting of interacting processes with messages sent to and received from each other. The model is not executable, since it is an abstraction of the protocol and not its implementation. Authenticity, integrity, and confidentiality can be analyzed with this approach. However, as with all methods based on state exploration, the size and the complexity of analyzed systems are severely limited by the state explosion problem (cf. [26]).

2.5. Symbolic Model Verifier

The Symbolic Model Verifier (SMV) is a model-checking system that can be used for analyzing designs of synchronous and asynchronous process systems. It provides a language for describing finite automata, and it can directly check the validity of temporal logic formulas (that is, linear temporal logic, or LTL for short, and computation tree logic, or CTL for short). The tool uses a textual description of the system's dynamic model and the corresponding specification which is expressed in LTL and CTL terms. On termination, it produces either 'true' if the specification holds or a trace showing why the required property is violated. SMV programs consist of one or more modules, which can declare variables and assign values to them. Usually, assignments give the initial value of a variable (e.g., init(var) := 0), whereas the variable's next value is specified in terms of expressions comprising the current value (e.g., next(var) := ((var + 1) mod 3)) [27], thereby modeling state transitions. Values can also be nondeterministic, in case the environment is influencing the system. In SMV, processes can be represented by modules that can be composed synchronously or asynchronously. In the latter case, the modules run at different speeds, and they are interleaving arbitrarily. Such asynchronous compositions can be used for describing communication protocols, asynchronous circuits, and other systems whose actions are not synchronized with a global clock [27].

In general, the proposed method offers a high level of formality, since it is based on temporal logic. It is well suited for modeling distributed systems, and the user has to provide a model of the system's dynamic behavior. The granularity of modeled systems can vary: On the one hand, processes that communicate with each other can be modeled, which can describe a view of the system's architecture. On the other hand, the method can be applied for modeling finite state machines, such as Mealy automata. Executable software systems cannot be modeled, but security protocols can. Properties like authenticity, integrity, confidentiality and non-repudiation can be verified. Of course, these have to be transformed into temporal logic formulas first (that is, combinations of safety and liveness properties, since all properties can be traced to them [21]) to be analyzable. Lastly, as with all model checking methods, the size and the complexity of analyzed systems are severely limited by the state explosion problem (cf. [26]).

2.4. Alloy

Alloy is a declarative modeling language based on first-order logic, extended with relational logic operators [28]. The language was primarily designed for modeling software designs. Models written in the Alloy language can be analyzed using the so-called Alloy Analyzer, a model-finder built on a SAT (satisfiability problem) solver to simulate models and check their properties. Hereafter, we use the term Alloy to refer to both the language and the tool. The key elements of the approach are a logic, a language, and an analysis, which are introduced below [28].

- In [28], the authors describe Alloy as a first-order relational logic, which provides the building blocks of the language. All logical structures are represented as relations, and all structural properties are expressed with relational operators. States and executions are both described using constraints.
- The language adds a syntax to the underlying first-order relational logic. To support classification, the Alloy language supports typing, sub-typing and compile-time type-checking. Furthermore, the language's module system allows a reuse of generic declarations and constraints.
- Literally speaking, the analysis of Alloy models is a form of constraint solving, either by finding an instance of a model or by finding a counterexample for a given property. An instance is an example of the specified model, in which both the facts and the predicate hold. To make instance finding feasible, a user-specified scope is defined that limits the size of the analyzed instances. Within this bound, the analyzer translates the constraint into a Boolean formula and solves it using a commercial SAT solver [28]. The solution is then presented to the user.

In general, the proposed method is suitable for modeling static and dynamic aspects of software systems. Furthermore, it offers a high level of formality, since the language is based on first-order relational logic. The Alloy language is abstract enough to model the problem domain's specific entities, as well as to model distributed systems, since message transmissions can be represented as dynamic operations. The modeled systems are not executable but are also not bound to a specific application area, since Alloy is expressive enough to capture several problem domains. As the approach is based on first-order logic, security requirements such as authenticity, integrity, non-repudiation, and confidentiality can be expressed. These have to be specified by the user as first-order formulas.

3. Evaluation of Model-driven Security Approaches

Tables I provides a detailed overview of the surveyed methods, answering the question which modeling approaches are applicable for solving which problems. In summary, the conducted classification revealed that UMLsec is the most generally applicable approach focused on security, since all the other methods are either limited to modeling single security mechanisms (e.g., role-based access control), or they are general enough to model security as well but offer no security-specific language elements. The aspect-oriented approaches concentrate on just a single security aspect and lack the adaptability of the other methodologies, thus are unable to support the development of secure real-world applications on their own where usually several security aspects are of equal importance. Alloy is an example of such a language which is indeed very expressive but does not provide established rules of prudent security engineering to make them available for users who may not be experts in security. In such a case, the user has to model all the security aspects of the problem domain, which often requires a deep understanding of security (e.g., cryptographic protocols).

Likewise, the Symbolic Model Verifier (SMV) model-checking system can be applied for analyzing dynamic behavior of parallel executing processes and can be used for the analysis of cryptographic security protocols as well. However, the level of abstraction offered by the SMV modeling language is far lower than the level adequate for describing security protocols. As a result, modeling security protocols in the SMV language is more complex than in AVISPA, since there are much more details to consider. Therefore, even if generally applicable methods (e.g., UMLsec, Alloy) can be applied to a broader range of security problems than special-purpose methods, this does not imply that they are more adequate. First, it depends on the particular problem which method fits best. And second, we have made the experience that picking the adequate special purpose method and applying it to the particular problem is more efficient and leads to better results, since (i) the problem can be represented on the proper abstraction level, (ii) the user can build on the knowledge of experts, and (iii) the available tools are more efficient and powerful.

Dimension	**Juerjens**	**H. Yu et al.**	**Zhu et al.**	**AVISPA**	**SMV**	**Alloy**
Paradigm	hierarchical	aspect-oriented	aspect-oriented	hierarchical	hierarchical	hierarchical
Artifacts	static and dynamic models	static and dynamic models weaving rules	static and dynamic models weaving rules	dynamic models	dynamic models	static and dynamic models
Formality	metamodels constraints	high-level Petri nets temporal logic	metamodels state machines	temporal logic of actions	temporal logic	first-order logic
Distribution	yes	yes	no	yes	yes	yes
Granularity	packages, classes	components and connectors	classes	processes	processes	classes
Executability	no	no	no	no	no	no
Verification	model checking theorem proving	model checking	no	model checking theorem proving	model checking	model finding
Tool support	yes	no	no	yes	yes	yes
Applicability	information systems embedded systems	widely applicable	widely applicable	security protocols	synchronous and asynchronous systems	widely applicable
Security mechanisms and requirements	confidentiality integrity non-repudiation non-interference authenticity access control	safety liveness	intrusion detection	confidentiality integrity authenticity	safety liveness	confidentiality integrity authenticity non-repudiation

Table 1: Evaluation results

5. Conclusion

In recent years, model-driven development has been introduced in order to increase the quality and thereby the security of software systems. This paper presented an evaluation of current efforts that position security as a fundamental element in model-driven development. Our evaluation revealed that approaches that analyze implementations of modeled systems are still missing. Due to the fact that implementations are not generated automatically from formal specifications, verification of running code is reasonable. A further insight was that all the aspect-oriented approaches modeled only a single security aspect. So even if the presented techniques worked well for modeling a single security mechanism, it has not been shown by anyone how adequate the AOSD principle is for developing secure real-world applications. Therefore, modeling several security aspects and combining them with the primary model is one of the next steps that the modeling community has to take. Further work will focus on the evaluation of how complex the weaving rules may become and how difficult it might be to verify a system consisting of several security aspects if more than one aspect is considered. Successfully modeling and thus generating a secure system including several security mechanisms, such as a security protocol (e.g., Needham-Schroeder) and access control (e.g., role-based access control), would provide the necessary confidence that the AOSD paradigm is suitable for development of complex and secure real-world applications.

References

[1] P. T. Devanabu and S. Stubblebine, “Software engineering for security: a roadmap,” in ICSE ’00: Proceedings of the Conference on The Future of Software Engineering. ACM, 2000, pp. 227–239.

[2] The Economist, “Cyberwarfare is becoming scarier,” The Economist (US), 2007.

[3] J. Juerjens, Secure Systems Development with UML. Springer, 2005.

[4] E. Fernandez-Medina, J. Juerjens, J. Trujillo, and S. Jajodia, “Model-driven development for secure information systems,” Information and Software Technology, 2008.

[5] D. C. Schmidt, “Model-driven engineering,” IEEE Computer, vol. 39, no. 2, 2006.

[6] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, “Aspect-oriented programming,” pp. 220– 242, 1997.

[7] J. Dehlinger and N. Subramanian, “Architecting secure software systems using an aspect-oriented approach: A survey of current research,” Iowa State University, Tech. Rep., 2006.

[8] J. Juerjens, “UMLsec: Extending UML for secure systems development,” in UML ’02: Proceedings of the 5th International Conference on The Unified Modeling Language, 2002.

[9] D. Basin, J. Doser, and T. Lodderstedt, “Model driven security for process-oriented systems,” in SACMAT ’03: Proceedings of the eighth ACM Symposium on Access control models and technologies. ACM, 2003, pp. 100–109.

[10] L. Vigano, “Automated security protocol analysis with the AVISPA tool,” Electronic Notes in Theoretical Computer Science, vol. 155, pp. 61–86, 2006.

[11] A. Khwaja and J. Urban, “A synthesis of evaluation criteria for software specifications and specification techniques,” International Journal of Software Engineering and Knowledge Engineering, 2002.

[12] C. H. Weiss, Evaluation: Methods for Studying Programs and Policies. Prentice-Hall, 1998.

[13] H. E. R., “Assumptions underlying evaluation models,” Educational Researcher, 1978.

[14] R. Villarroel, E. Fernandez-Medina, and M. Piattini, “Secure information systems development - a survey and comparison,” Computers & Security, 2005.

[15] J. Fox and J. Juerjens, “Introducing security aspects with model transformations,” in ECBS ’05: Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems. IEEE Computer Society, 2005.

[16] D. Coleman, G. Booch, D. Garlan, S. Iyengar, C. Kobryn, and V. Stavridou, “Is UML an architectural description language,” 1999.

[17] B. Best, J. Juerjens, and B. Nuseibeh, “Model-based security engineering of distributed information systems using UMLsec,” in ICSE ’07: Proceedings of the 29th international conference on Software Engineering, 2007.

[18] J. Juerjens, “UML analysis tools, http://mcs.open.ac.uk/jj2924/umlsectool/index.html,”

[19] J. Juerjens, “Developing secure embedded systems: Pitfalls and how to avoid them,” 2007.

[20] H. Yu, D. Liu, X. He, L. Yang, and S. Gao, “Secure software architectures design by aspect orientation,” in ICECCS ’05: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems. IEEE Computer Society, 2005, pp. 47–55.

[21] B. Alpern, B. Alpera, F. B. Schneider, and F. B. Schneider, “Recognizing safety and liveness,” Distributed Computing, vol. 2, pp. 117–126, 1987.

[22] Z. J. Zhu and M. Zulkernine, “A model-based aspect-oriented framework for building intrusion-aware software systems,” Information and Software Technology, 2008.

[23] WASC, “Threat classification,” Web Application Security Consortium, Tech. Rep., 2008. [Online]. Available: http://www.webappsec.org/projects/threat

[24] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. Heam, O. Kouchnarenko, J. Mantovani, S. Moedersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Vigano, and L. Vigneron, "The AVISPA tool for the automated validation of internet security protocols and applications," in Lecture Notes in Computer Science 3576, 2005.
[25] L. Lamport, "The temporal logic of actions," ACM Transactions on Programming Languages and Systems, vol. 16, pp. 872–923, 1994.
[26] A. Valmari, "The state explosion problem," in Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets. London, UK: Springer Verlag, 1998, pp. 429–528.
[27] M. Huth and M. Ryan, Logic in Computer Science. Cambridge University Press, 2004.
[28] D. Jackson, Software abstractions: Logic, Language, and Analysis. The MIT Press, 2006.

Acknowledgments

This work was supported by grants of the Austrian Government's BRIDGE Research Initiative (contract 824884), the FIT-IT Research Initiative (contract 816158) and was performed at the research center Secure Business Austria funded by the Federal Ministry of Economy, Family and Youth of the Republic of Austria and by the City of Vienna.