

Chapter 2

Security and Dependability Engineering

Jan Jürjens

Abstract The current state of the art in security-critical ambient systems is far from satisfactory: New security vulnerabilities are discovered on an almost daily basis. To improve this situation, there has recently been a lot of work on techniques and tools supporting the development of trustworthy security-critical software, in particular for dynamic systems in an ambient environment. This chapter gives an overview over the field of security and dependability engineering, with an emphasis on ambient system security, and on current advances based on model-based development using UML and providing strong assurance results. We give examples for security flaws found in industrial software using such tools and shortly discuss some open research issues.

2.1 Ambient System Security

As computing devices become smaller and cheaper, their use increasingly pervades all aspects of everyday life, at work and at home [20]. Ubiquitous devices such as smart-cards, RFID tags, networked sensors, and personal digital assistants fulfill many useful and convenient functions. The increasing use of such ambient technology however can also raise security and privacy concerns for their direct users and owners, as well as other people indirectly affected by their use. Ambient technology can help realise and enforce security and privacy requirements; for example, one can label important devices (such as a laptop) with RFID tags to allow owners with suitable sensors to track such devices. On the other hand, ambient technology's pervasiveness may also pose threats to security and privacy concerns of other individuals in the surrounding environment. One therefore needs to investigate relationships and trade-offs between usability and security/privacy, in order to maximise usage opportunities and minimise the security and privacy risks.

Jan Jürjens
The Open University, Milton Keynes, GB, <http://www.jurjens.de/jan>

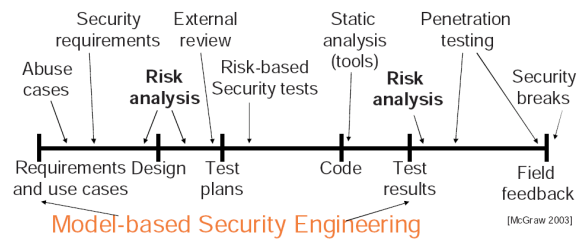


Fig. 2.1 Secure system lifecycle [66]

Specifically, one needs to develop techniques and tools that ensure continuing compliance to security requirements for pervasive systems, to investigate security issues that may arise in such environments, and how these might be dealt with using available security technology. In particular, one needs to investigate a notion of *adaptive security* in this context, which includes for example questions such as whether security mechanisms may have to adapt to the exact location of an individual in an office building (in the sense of *context-dependent* or *location-based* security). Another aspect of this topic is the pervasive interfaces between the human users and the computers, and on how to integrate these with the adaptive security mechanisms in a secure way.

Security requirements elicitation for ambient devices like smart cards or RFID tags is difficult since they are characterized by limited resources, high mobility and (desired) high security level in spite of untrusted environment. Also, such a device has to adapt itself to application requirements or modification of its environment. In other words they must withstand longevity. In order to cope with change one needs to define a system that is essentially autonomic and self-adjusting. For example, one can design a system that is self-aware in order to realize that the environment is changing, and self-adjusting, to be able to account for it. It should have mechanisms to ensure that the program functions correctly during and after adaptations. Because of the inherent higher complexity, this is particularly difficult to achieve for adaptive software. There is an additional challenge: requirements themselves might change.

Problems: Attacks against computer networks, which the infrastructures of modern society and modern economies rely on, cause substantial financial damage. Due to the increasing interconnection of systems, such attacks can be waged anonymously and from a safe distance. Thus networked computers need to be secure. The high-quality development of security-critical systems is difficult. Still, many systems are developed, deployed, and used over years that contain significant security weaknesses.

Causes: While tracing requirements during software development is difficult enough, enforcing security requirements is intrinsically subtle, because one has to take into account the interaction of the system with motivated adversaries that act independently. Thus security mechanisms, such as security protocols, are notoriously hard to design correctly, even for experts. Also, a system is only as secure as its weakest part or aspect. Security is compromised most often not by breaking dedicated mechanisms such as encryption or security protocols, but by exploiting

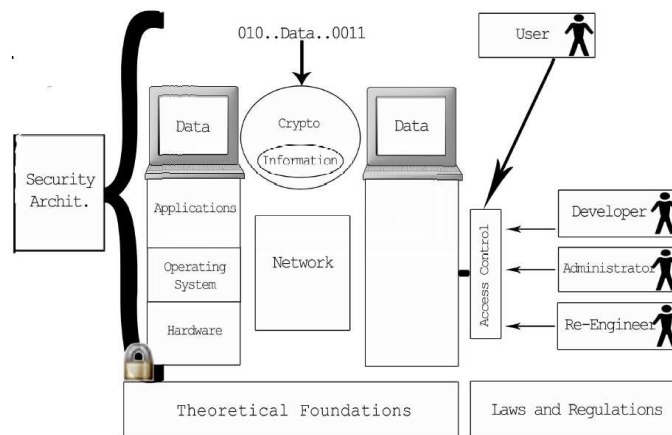


Fig. 2.2 Secure system layers

weaknesses in the way they are being used [4]. Thus it is not enough to ensure correct functioning of security mechanisms used. They cannot be “blindly” inserted into a security-critical system, but the overall system development must take security aspects into account in a coherent way [75]. In fact, according to [78], 85% of Computer Emergency Response Team (CERT) security advisories could not have been prevented just by making use of cryptography. Building trustworthy components does not suffice, since the interconnections and interactions of components play a significant role in trustworthiness [78].

State of the Art in Practice: In practice, the traditional strategy for security assurance has been *penetrate and patch*: It has been accepted that deployed systems contain vulnerabilities. Whenever a penetration of the system is noticed and the exploited weakness can be identified, the vulnerability is removed. Sometimes this is supported by employing friendly teams trained in penetrating computer systems, the so-called “tiger teams”. However, this approach is not ideal: Each penetration using a new vulnerability may already have caused significant damage, before the vulnerability can be removed. It would thus be preferable to consider security aspects more seriously in earlier phases of the system life-cycle, before a system is deployed, or even implemented, because late correction of requirements errors costs up to 200 times as much as early correction [16]. Also, security concerns must inform every phase of software development, from requirements engineering to design, implementation, testing, and deployment, and academic approaches trying to improve the security during development should be tightly integrated with software development approaches already used in industry [22].

Some other challenges for using sound engineering methods for secure systems development exist. For example, the boundaries of the specified components with the rest of the system need to be carefully examined, for example with respect to implicit assumptions on the system context [34]. Lastly, a more technical issue is that formalized security properties are not in all approaches preserved by refinement, which is the so-called *refinement problem*. Since an implementation is necessarily a

refinement of its specification, an implementation of a secure specification may, in such a situation, not be secure, which is clearly undesirable.

A truly secure software engineering approach thus needs to take both dimensions of the problem into account:

- it needs to integrate the different system lifecycle phases (cf. Fig. 2.1),
- it also needs to take into account the different architectural levels of abstraction of a security-critical system (cf. Fig. 2.2) in a demonstrably sound, trustworthy, and cohesive way.

This is certainly a very challenging task that will still pose some highly interesting scientific challenges until it may be finally achieved.

2.2 Current Advances

In this section, we give an overview on some approaches for secure software engineering, with an emphasis on model-based development using UML. Since the area of secure software engineering has grown quickly over the last few years, we cannot hope to give a complete overview but just present some examples which we believe to be representative (although this is necessarily a personal judgement). We will in particular explain how the research developed within the SERENITY project fits within the wider context (cf. [63]).

Of course, there are also approaches for secure software engineering outside of or predating model-based development with UML (cf. for example [75, 24, 61, 78, 22, 4] for some examples and overviews), but because of space restrictions, we cannot consider those in detail here.

2.2.1 Model-based Development

Generally, when using model-based development (Fig. 2.3a), the idea is that one first constructs a model of the system. Then, the implementation is derived from the model: either automatically using code generation, or manually, in which case one can generate test sequences from the model to establish conformance of the code regarding the model. The goal is to increase the quality of the software while keeping the implementation cost and the time-to-market bounded. For security-critical systems, this approach allows one to consider security requirements from early on in the development process, within the development context, and in a seamless way through the development cycle: One can first check that the system fulfills the relevant security requirements on the design level by analyzing the model and secondly that the code is in fact secure by generating test sequences from the model. However, one can also use the security analysis techniques and tools within a traditional software engineering context, or where one has to incorporate legacy systems that were not developed in a model-based way. Here, one starts out with the source code. One then extracts models from the source code, which can then again be analyzed

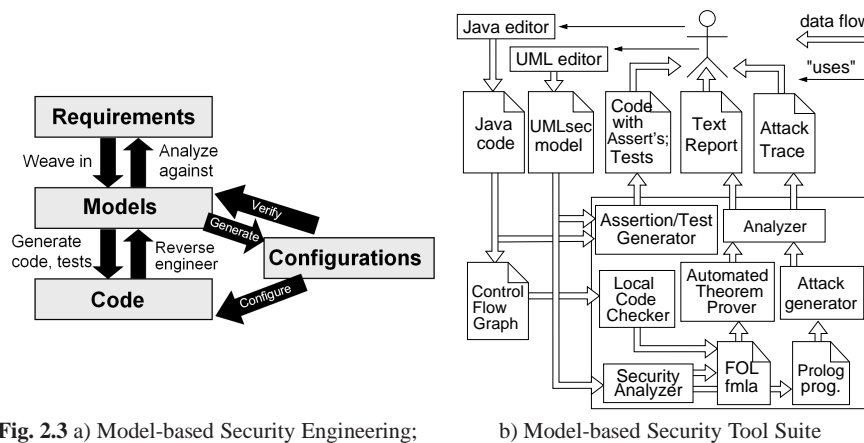


Fig. 2.3 a) Model-based Security Engineering;

b) Model-based Security Tool Suite

against the security requirements. Using model-based development, one can also incorporate the configuration data (such as user permissions) in the analysis, which is very important for security but often neglected.

For example, in the Model-based Security Engineering (MBSE) approach based on the UML extension UMLsec, [47, 50, 51], recurring security requirements (such as secrecy, integrity, authenticity and others) and security assumptions on the system environment, can be specified either within a UML specification, or within the source code (Java or C) as annotations (Fig. 2.3b). This way we encapsulate knowledge on prudent security engineering as annotations in models or code and make it available to developers who may not be security experts. The UMLsec extension is given in form of a UML profile using the standard UML extension mechanisms. *Stereotypes* are used together with *tags* to formulate the security requirements and assumptions. *Constraints* give criteria that determine whether the requirements are met by the system design, by referring to a precise semantics of the used fragment of UML. The security-relevant information added using stereotypes includes security assumptions on the physical level of the system, security requirements related to the secure handling and communication of data, and security policies that system parts are supposed to obey. The semantics of the fragment of UML used for UMLsec is defined in [50] using so-called *UML Machines*, which is a kind of state machine with input/output interfaces similar to Broy's Focus model, whose behavior can be specified in a notation similar to that of Abstract State Machines (ASMs), and which is equipped with UML-type communication mechanisms. On this basis, important security requirements such as secrecy, integrity, authenticity, and secure information flow are defined.

After an early paper on the UML extension UMLsec for secure software development [46], a number of approaches have been developed each targeted at certain facets of model-based development of security-critical systems using UML, several of them initially presented at the workshop series CSDUML (Critical Systems Development using Modeling Languages).

Security requirements modeling: [26] proposes a method determining role-based access rights. Use cases are extended with rights specifications and the rights of a

role are derived from the use cases. The method thus enforces the design principle of least privilege. [19, 37] formulates a vision for the requirements engineering community towards providing a “bridge between the well-ordered world of the software project informed by conventional requirements and the unexpected world of anti-requirements associated with the malicious user”. [32, 33, 64] proposes an extension of the *i**/Tropos requirements engineering framework to deal with security requirements. The Tropos Requirements Engineering methodology is also extended to cover security aspects in [69]. [80] presents an approach to eliciting security requirements using use cases which extends traditional use cases to also cover misuse. [70] uses a combination of UMLsec and Tropos to get a transition from the security requirements to the design phase. [85] presents an executable misuse case modeling language which allows modelers to specify misuse case scenarios in a formal yet intuitive way and to execute the misuse case model in tandem with a corresponding use case model. [91] presents an approach for the transformation of security requirements to software architectures. [6] discuss the use of requirements-engineering techniques in capturing security requirements for a Grid-based operating system. [25] examines how conceptual modeling can provide support for analyzing security trade-offs, using an extension to the *i** framework. [29] presents an approach which integrates security and usability into the requirements and design process, based on a development process and a UML meta-model of the definition and the reasoning over the system’s assets.

Security patterns: [90] proposes a UML based method that enables developers to specify several candidate system behaviors that satisfy the security requirements, using patterns, and shows an application of the method to a real implemented system, the Environmentally Conscious Product (ECP) design support system. A methodology to build secure systems using patterns is presented in [27]. A main idea in the proposed methodology is that security principles should be applied at every stage of the software lifecycle and that each stage can be tested for compliance with those principles. Another basic idea is the use of patterns at each stage. A pattern is an encapsulated solution to a recurrent problem and their use can improve the reusability and quality of software. [74] compares several security patterns to be used when dealing with application security.

SERENITY: The SERENITY approach is based on the notion of Security and Dependability Patterns [63]. They include a functional description of the proposed security solution, a semantic description of the security requirements addressed by it, and descriptions of the assumptions on the context in which the pattern can be used.

Automated security verification: So far only few of the UML based approaches for secure software development come with automated tools to formally verify the UML design for the relevant security requirements. One of these is again the UMLsec approach. The UMLsec tool-support (cf. Fig. 2.3a) can be used to check the constraints associated with UMLsec stereotypes mechanically, based on XMI output of the diagrams from the UML drawing tool in use [49, 84, 51, 55]. They generate logical formulas formalizing the execution semantics and the annotated security requirements. Automated theorem provers and model checkers automatically establish whether the security requirements hold. If not, a Prolog-based tool auto-

matically generates an attack sequence violating the security requirement, which can be examined to determine and remove the weakness. Since the analysis that is performed is too sophisticated to be done manually, it is also valuable to security experts. There is also a framework for implementing verification routines for the constraints associated with the UMLsec stereotypes. Thus advanced users of the UMLsec approach can use this framework to implement verification routines for the constraints of self-defined stereotypes.

Other approaches for verifying UML models for security properties emerge. For example, [7] explains an approach in which queries about properties of an RBAC policy model are expressed as formulas in UML's Object Constraint Language and evaluated over the metamodel of the security-design language, based on the rewriting logic Maude. Also, [81] presents a tool for verifying UML class and state machine diagrams against linear temporal logic formulas using Spin, which is planned to be applied to security properties.

Model construction and development: Having a formally based design notation allows one to precisely formulate and investigate non-trivial questions that need to be solved to enable trustworthy secure software development. For example, to support stepwise development, it has been shown that within UMLsec, secrecy, integrity, authenticity, and secure information flow are preserved under refinement and the composition of system components (under suitable assumptions) [50]. Similarly, it has been shown that layering of security services (such as layered security protocols) is sound, again only under certain assumptions. The same applies to the application of security design patterns, or the use of aspect-oriented modeling techniques. Related approaches have been reported in [77, 76].

SERENITY: To support the Security and Dependability Patterns used in the project, the SERENITY approach also provides Security and Dependability Schemes which allow the users to combine existing security solutions to more complex ones [63]. This is supported by automated tools for classification, selection, and composition of security patterns.

Aspect-Oriented Security Modeling: [72, 41] propose to use aspect-oriented modeling for addressing access control concerns. Functionality that addresses a pervasive access control concern is defined in an aspect. The remaining functionality is specified in a so-called primary model. Composing access control aspects with a primary model then gives a system model that addresses access control concerns.

Model-based Security Risk Assessment: [23] uses UML for the risk assessment of an e-commerce system within the CORAS framework for model-based risk assessment. This framework is characterized by an integration of aspects from partly complementary risk assessment methods. It incorporates guidelines and methodology for the use of UML to support and direct the risk assessment methodology as well as a risk management process based on standards such as AS/NZS 4360 and ISO/IEC 17799. It uses a risk documentation framework based on RM-ODP together with an integrated risk management and system development process based on UP and offers a platform for tool inclusion based on XML. In another approach [9, 58] use UML for risk-driven security analysis which focusses on the assessment of risk and analysis of requirements for operational risk management.

Secure business processes and Service-oriented architectures: A business process-driven approach to security engineering using UML is presented in [62]. The idea is to use UML models in an approach centered on business processes to develop secure systems. A model-based security engineering approach for developing service-oriented architectures is proposed in [21]. The approach is applied on a standard for service-oriented architectures from the Automotive domain (OSGi).

Access control policies: [10, 17] show how UML can be used to specify access control in an application and how one can then generate access control mechanisms from the specifications. The approach is based on role-based access control and gives additional support for specifying authorization constraints. [60, 59] demonstrate how to deal with access control policies in UML. The specification of access control policies is integrated into UML. A graph-based formal semantics for the UML access control specification permits one to reason about the coherence of the access control specification. An aspect-oriented approach to specifying access control in UML is presented in [93]. [67] presents an approach for the specification and refinement of access control rules, including proof rules for verifying that an access control policy is correctly implemented in a system, and preservation of access control by refinement of event systems.

SECTET: [3] presents usage scenarios for access control in contemporary healthcare scenarios and shows how to unify them in a single security policy model. Based on this model, the SECTET [2] framework for Model Driven Security is then specialized towards a domain-specific approach for healthcare scenarios, including the modelling of access control policies, a target architecture for their enforcement, and model-to-code transformations. [1] extends the SECTET approach to take into account operating system level and application level security mechanisms to realize security-critical application and services for healthcare scenarios.

Health information systems: There have been several approaches using UML for security aspects in developing health-care systems. [14] presents an approach based on formal models where security services can be integrated into advanced systems architectures enabling semantic interoperability in the context of trustworthiness of communication and co-operation to support application security challenges such as privilege management and access control. The approach covers domains, service delegation, claims control, policies, roles, authorisations, and access control. [65] presents an approach based on model-based design techniques and high-level modeling abstractions which provides a framework to rapidly develop, simulate, and deploy clinical information system (CIS) prototypes. It includes a graphical design environment for developing formal system models and generating executable code for deployment.

Secure database design: An approach to designing the content of a security-critical database uses the Object Constraint Language (OCL) which is an optional part of the Unified Modeling Language (UML). More specifically, [28] presents the Object Security Constraint Language V.2. (OSCL2), which is based in OCL. This OCL extension can be used to incorporate security information and constraints in a Platform Independent Model (PIM) given as a UML class model. The information from the PIM is then translated into a Platform Specific Model (PSM) given as a multilevel relational model. This can then be implemented in a particular Database

Management System (DBMS), such as Oracle9i Label Security. These transformations can be done automatically or semi-automatically using OSCL2 compilers.

Smart-card based applications: [38, 68] present a method for the development of secure smartcard applications which includes UML models enriched by algebraic specifications, and dynamic logic for JavaCard verification. The approach is implemented in the KIV specification and verification system.

Secure information flow: [39] provides support for the use of UML with secrecy annotations so that the code produced from the UML models can be validated by the Java information flow (Jif) language-based checker. [79] provides an approach which can analyze secure information properties in UML sequence diagrams.

2.2.2 Code-level Assurance against High-level Security Requirements

Even if specifications exist for the implemented system, and even if these are formally analyzed, there is usually no guarantee that the implementation actually conforms to the specification. To deal with this problem, we can use the following approach: After specifying the system in the given notation (such as UMLsec) and verifying the model against the given security goals, we make sure that the implementation correctly implements the specification with techniques such as those explained below. Note that in addition it is often necessary to use dedicated tools to detect specialized weaknesses (such as buffer overflows), although this is not in scope of the current overview.

Run-time Security Monitoring: A simple and effective alternative is to insert security checks generated from the specification that remain in the code while in use, for example using the `assertion` statement that is part of the Java language [11]. These assertions then throw security exceptions when violated at run-time. In a similar way, this can also be done for C code.

SERENITY: The SERENITY approach provides dynamic runtime verification mechanisms that can monitor various security properties dynamically based on event calculus [63]. For example, they can monitor whether the assumptions made by a given security pattern is satisfied at the execution of the system. To achieve this, [82] proposes to use formal patterns that formalize frequently recurring system requirements as *security monitoring patterns*. Also, evolution tools record the operational data relevant for the Security and Dependability Patterns to obtain feed-back that can help improving the patterns.

Model-based Test Generation: For performance-intensive applications, it may be preferable not to leave the assertions active in the code. This can be done by making sure by extensive testing that the assertions are always satisfied, for example by generating the test sequences automatically from the specifications. Since complete test coverage is often infeasible, an approach that automatically selects those test cases that are particularly sensitive to the specified security requirements is sketched in [87, 53] (with respect to the formal semantics underlying UMLsec). Other work on testing crypto-protocols includes [36].

Formally verifying cryptoprotocol implementations: For highly non-deterministic systems such as those using cryptography, testing can only provide assurance up to a certain degree. For higher levels of trustworthiness, it may therefore be desirable to establish that the code does enforce the security properties by a formal verification of the source code. There have recently been some approaches towards formally verifying implementations of crypto-protocols against high-level security requirements such as secrecy, for example [57, 35, 52, 13]. These works so far have aimed to verify implementations which were constructed with verification in mind (and in particular fulfill significant expectations on the way they are programmed) [35, 13], or deal only with simplified versions of legacy implementations [57, 52]. In related work, [71] investigates under which conditions it is sound to abstract from marshalling and unmarshalling operations on transmitted messages when verifying protocol specifications.

2.2.3 Analyzing Security Configurations

There have also been some first steps towards linking model-based security engineering approaches with the automated analysis of security-critical configuration data. For example, a tool that automatically check SAP R/3 user permissions for security policy rules formulated as UML specifications are presented in [40]. Because of its modular architecture and its standardized interfaces, the tool can be adapted to check security constraints in other kinds of application software, such as firewalls or other access control configurations.

2.2.4 Industrial Applications

An overview on industrial applications of model-based security engineering in practice can be found in [5]. We list some examples below.

German Health Card architecture: Ongoing work for the German health telematics platform using a model-driven architectural framework and a security infrastructure based on Electronic Health Records and multifunctional Electronic Health Cards is presented in [15]. A security analysis of the German Health Card Architecture using UMLsec is reported in [54].

Electronic purses: UMLsec was applied to a security analysis of the Common Electronic Purse Specifications (CEPS), a candidate for a globally interoperable electronic purse standard supported by organizations representing 90 % of the world's electronic purse cards (including Visa International). Three significant security weaknesses were found in the purchase and load transaction protocols, improvements to the specifications were proposed, and it was shown that these are secure [56]. There was also a security analysis of a prototypical Java Card implementation of CEPS. A method for the development of secure smartcard applications which includes UML models and is implemented in the KIV specification and ver-

ification system [38, 68] was applied to the specification of the Mondex electronic purse.

Intranet information systems: An application of UMLsec to information systems in an intranet at BMW is reported in [12]. There the use of single-sign-on mechanisms was central, so the application of UMLsec was targeted to demonstrating that it was used correctly within the system context.

Biometric authentication: For a project with an industrial partner, UMLsec was chosen to support the development of a biometric authentication system at the specification level, where three significant security flaws were found [51]. It was also applied to the source-code level for a prototypical implementation constructed from the specification.

Web-based banking application: In a project with a German bank, model-based security engineering was applied to a web-based banking application to be used by customers to fill out and sign digital order forms [50]. The personal data in the forms must be kept confidential, and orders securely authenticated. The system uses a proprietary client authentication protocol layered over an SSL connection supposed to provide confidentiality and server authentication. Using the MBSE approach, the system architecture and the protocol were specified and verified with regard to the relevant security requirements.

2.3 Outlook

Given the current unsatisfactory state of computer security in practice, model-based security engineering seems a promising approach, since it enables developers who are not experts in security to make use of security engineering knowledge encapsulated in a widely used design notation. Since there are many highly subtle security requirements which can hardly be verified with the “naked eye”, even security experts may profit from this approach. Thus one can avoid mistakes that are difficult to find by testing alone, such as breaches of subtle security requirements, as well as the disadvantages of the “penetrate-and-patch” approach. Since preventing security flaws early in the system life-cycle can significantly reduce costs, this gives a potential for developing securer systems in a cost-efficient way. Model-based security engineering has been successfully applied in various industrial projects. The approach has been generalized to other application domains such as real-time and dependability. Experiences show that the approach is adequate for use in practice, after relatively little training. As a consequence, model-based security engineering is now also considered an important emerging technology by industrial think-tanks (cf. e.g. [30]).

Due to space restriction, the current overview can only provide very limited detail or completeness. More comprehensive overviews on model-based security engineering and secure software engineering include [43, 73].

Some examples for open problems that remain:

Tracing security requirements: From a practical point of view, the construction of trustworthy security-critical systems would be significantly facilitated if one

would have a practically feasible approach for tracing security requirements through the system lifecycle phases (cf. Fig. 3.1). A first step in that direction is presented in [92].

Preservation of security properties: Despite some early advances into this question [44, 45] there is so far relatively little known about the preservation of security properties when using design and analysis techniques such as the modular composition or decomposition, refinement or abstraction, or horizontal resp. vertical layering of system parts (cf. Fig. 3.2).

Security verification of legacy systems: A major open problem is to verify complex legacy implementations against high-level security properties in a practically feasible way. Again, some steps in that direction were reported in Sect. 2.2.2.

Security vs. other non-functional requirements / feature interaction: Another open problem is how to reconcile security requirements with other non-functional requirements, which may be orthogonal or even in conflict. First examples regarding performance properties can be found in [18, 31, 89].

Further recommendations for future research, and for improvements of secure software engineering in practice, can be found e.g. in [88, 86].

References

1. Agreiter B, Alam M, Hafner M, Seifert J-P, and Zhang X (2007). Model driven configuration of secure operating systems for mobile applications in healthcare. In Sztipanovits et al. [83].
2. Alam M, Hafner M, and Breu R (2007). Model-driven security engineering for trust management in SECTET. *Journal of Software*, 2(1).
3. Alam M, Hafner M, Memon M, and Hung P (2007). Modeling and enforcing advanced access control policies in healthcare systems with SECTET. In Sztipanovits et al. [83].
4. Anderson R (2001). *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, New York.
5. Apvrille A and Pourzandi M (2005). Secure software development by example. *IEEE Security & Privacy*, 3(4):10–17.
6. Arenas A, Aziz B, Bicarregui J, Matthews B, and Yang EY (2008). Modelling security properties in a grid-based operating system with anti-goals. In *ARES* [42]: 1429–1436.
7. Basin DA, Clavel M, Doser J, Egea M (2007). A Metamodel-Based Approach for Analyzing Security-Design Models. *MoDELS 2007*: 420–435.
8. Breu R, Burger K, Hafner M, Jürjens J, Popp G, Wimmel G, Lotz V (2003). Key Issues of a Formally Based Process Model for Security Engineering. In *Sixteenth Intern. Conference on Software & Systems Engineering & their Applications (ICSSEA 2003)*.
9. Baldwin A, Beres Y, Shiu S, and Kearney P (2006). A model based approach to trust, security and assurance. *BT Technology Journal*, 24(4):53–68.
10. Basin DA, Doser J, and Lodderstedt T (2006). Model driven security: From UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1): 39–91.
11. Bauer A and Jürjens J (2008). Security protocols, properties, and their monitoring. In Bart De Win, Seok-Won Lee, and Mattia Monga, editors, *SESS*: 33–40. ACM.
12. Best B, Jürjens J, and Nuseibeh B (2007). Model-based security engineering of distributed information systems using UMLsec. In *ICSE*. ACM.
13. Bhargavan K, Fournet C, Gordon AD, and Tse S (2006). Verified interoperable implementations of security protocols. In *CSFW*: 139–152. IEEE Computer Society.

14. Blobel B, Nordberg R, Davis JM, and Pharow P (2006). Modelling privilege management and access control. *International Journal of Medical Informatics*, 75(8): 597–623.
15. Blobel B and Pharow P (2007). A model-driven approach for the german health telematics architectural framework and security infrastructure. *International Journal of Medical Informatics*, 76(2–3): 169–175.
16. Boehm BW (1981). *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ.
17. Brucker AD, Doser J, and Wolff B (2006). A model transformation semantics and analysis methodology for SecureUML. In *MoDELS 2006*, volume 4199 of LNCS: 306–320. Springer.
18. Buchholtz M, Gilmore S, Haenel V, and Montangero C (2005). End-to-end integrated security and performance analysis on the DEGAS Choreographer Platform. In *FM 2005*, volume 3582 of LNCS: 286–301. Springer.
19. Crook R, Ince DC, Lin L, and Nuseibeh B (2002). Security requirements engineering: When anti-requirements hit the fan. In *RE 2002*: 203–205. IEEE.
20. Daskala B and Maghiros I (2007). Digital Territories – Towards the protection of public and private space in a digital and Ambient Intelligence environment. Institute for Prospective Technological Studies (IPTS).
21. Deubler M, Grünbauer J, Jürjens J, and Wimmel G (2004). Sound development of secure service-based systems. In *ICSOC 2004*: 115–124. ACM.
22. Devanbu P and Stubblebine S (2000). Software engineering for security: a roadmap. In *The Future of Software Engineering (ICSE 2000)*: 227–239.
23. Dimitrakos T, Ritchie B, Raptis D, Aagedal JØ, den Braber F, Stølen K, and Houmb SH (2002). Integrating model-based security risk management into ebusiness systems development: The CORAS approach. In *Second IFIP Conference on E-Commerce, E-Business, E-Government (I3E 2002)*: 159–175. Kluwer.
24. Eckert C and Marek D (1997). Developing secure applications: A systematic approach. In *13th International Conference on Information Security (SEC 1998)*: 267–279.
25. Elahi G and Yu E (2007). A goal oriented approach for modeling and analyzing security trade-offs. In *ER 2007*, volume 4801 of LNCS: 375–390. Springer.
26. Fernandez EB and Hawkins JC (1997). Determining role rights from use cases. In *Workshop on Role-Based Access Control*: 121–125. ACM.
27. Fernandez EB, Larrondo-Petrie MM, Sorgente T, and VanHilst M (2006). A methodology to develop secure systems using patterns. In H Mouratidis and P Giorgini, editors, *Integrating security and software engineering: Advances and future vision*, chapter 5: 107–126. IDEA Press.
28. Fernández-Medina E and Piattini M (2004). Extending OCL for secure database development. In *UML 2004*, LNCS: 380–394. Springer.
29. Flechais I, Mascolo C, and Sasse MA (2007). Integrating security and usability into the requirements and design process. *International Journal of Electronic Security and Digital Forensics*, 1(1):12–26.
30. Model-driven security: Enabling a real-time, adaptive security infrastructure. Gartner Briefing G00151498, 21 Sep. 2007.
31. Gilmore S, Haenel V, Kloul L, and Maidl M (2005). Choreographing security and performance analysis for web services. In *EPEW/WS-FM 2005*, volume 3670 of LNCS: 200–214. Springer.
32. Giorgini P, Massacci F, and Mylopoulos J (2003). Requirement engineering meets security: A case study on modelling secure electronic transactions by VISA and Mastercard. In I.-Y. Song, S. W. Liddle, T. W. Ling, and P Scheuermann, editors, *22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of LNCS: 263–276. Springer.
33. Giorgini P, Massacci F, Mylopoulos J, and Zannone N (2005). Modeling security requirements through ownership, permission and delegation. In *RE*: 167–176. IEEE Computer Society.
34. Gollmann D (2000). On the verification of cryptographic protocols – a tale of two committees. In S Schneider and P Ryan, editors, *Workshop on Security Architectures and Information Flow*, volume 32 of ENTCS. Elsevier.
35. Goubault-Larrecq J and Parrennes F (2005). Cryptographic protocol analysis on real c code. In *VMCAI’05*, LNCS. Springer.

36. Gürgens S and Peralta R (2000). Validation of cryptographic protocols by efficient automated testing. In James N. Etheredge and Bill Z. Manaris, editors, *FLAIRS Conference*: 7–12. AAAI Press.
37. Haley CB, Laney RC, Moffett JD, and Nuseibeh B (2008). Security requirements engineering: A framework for representation and analysis. *IEEE Trans. Software Eng.*, 34(1):133–153.
38. Haneberg D, Reif W, and Stenzel K (2002). A method for secure smartcard applications. In Hélène Kirchner and Christophe Ringeissen, editors, *AMAST*, volume 2422 of *Lecture Notes in Computer Science*: 319–333. Springer.
39. Heldal R and Hultin F (2003). Bridging model-based and language-based security. In E Sneekenes and D Gollmann, editors, *8th European Symposium on Research in Computer Security (ESORICS 2003)*, volume 2808 of *LNCS*: 235–252. Springer.
40. Höhn S and Jürjens J (2008). Rubacon: automated support for model-based compliance engineering. In Robby, editor, *ICSE*: 875–878. ACM.
41. Houmb SH, Georg G, France RB, Bieman JM, and Jürjens J (2005). Cost-benefit trade-off analysis using BBN for aspect-oriented risk-driven development. In *ICECCS*: 195–204. IEEE Computer Society.
42. IEEE. 3rd Int Conference on Availability, Reliability and Security (ARES 2008), 2008.
43. Jayaram KR and Mathur A (2005). Software engineering for secure software – state of the art: A survey. Technical Report CERIAS-TR-2005-67, SERC-TR-279, CERIAS, Purdue.
44. Jürjens J (2000). Secure information flow for concurrent processes. In C Palamidessi, editor, *CONCUR 2000 (11th International Conference on Concurrency Theory)*, volume 1877 of *LNCS*: 395–409. Springer.
45. Jürjens J (2001). Secrecy-preserving refinement. In *International Symposium on Formal Methods Europe (FME)*, volume 2021 of *LNCS*: 135–152. Springer.
46. Jürjens J (2001). Towards development of secure systems using UMLsec. In H Hußmann, editor, *4th International Conference on Fundamental Approaches to Software Engineering (FASE)*, volume 2029 of *LNCS*: 187–200. Springer. Also Oxford University Computing Laboratory TR-9-00 (November 2000), <http://web.comlab.ox.ac.uk/oucl/publications/tr/tr-9-00.html>.
47. Jürjens J (2002). UMLsec: Extending UML for secure systems development. In *5th Int Conf on the Unified Modeling Language (UML)*, *LNCS*. Springer.
48. Jürjens J (2002). Formal Semantics for Interacting UML subsystems. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS 2002)*, IFIP, Kluwer: 29–43.
49. Jürjens J, Shabalin P (2004). Automated Verification of UMLsec Models for Security Requirements. In *7th Intern. Conference on The Unified Modeling Language (UML 2004)*, *Lecture Notes in Computer Science*: 142–155. Springer.
50. Jürjens J (2005). *Secure Systems Development with UML*. Springer.
51. Jürjens J (2005). Sound methods and effective tools for model-based security engineering with UML. In *27th Int Conf on Softw Engineering*. IEEE.
52. Jürjens J (2006). Security analysis of crypto-based Java programs using automated theorem provers. In S Easterbrook and S Uchitel, editors, *21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006)*. ACM.
53. Jürjens J (2009). A domain-specific language for cryptographic protocols based on streams. To appear, *Journal of Logic and Algebraic Programming (JLAP)*: 54–73.
54. Jürjens J and Rumm R (2008). Model-based security analysis of the German Health Card architecture. *Methods of Information in Medicine*, vol. 47, 5: 409–416. Special section on Model-based Development of Trustworthy Health Information Systems.
55. Jürjens J and Shabalin P (2007). Tools for secure systems development with UML. *Intern. Journal on Software Tools for Technology Transfer*, 9(5–6):527–544. Invited submission to the special issue for FASE 2004/05.
56. Jürjens J, Wimmel G (2001). Security Modelling for Electronic Commerce: The Common Electronic Purse Specifications. In *Towards the E-Society: E-Commerce, E-Business, and E-Government*. Intern. Federation for Information Processing (IFIP), Kluwer Academic Publishers: 489–506. First IFIP Conference on E-Commerce, E-Business, and E-Government (I3E 2001).

57. Jürjens J and Yampolskiy M (2005). Code security analysis with assertions. In D.F. Redmiles, T Ellman, and A Zisman, editors, 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005): 392–395. ACM.
58. Kearney P and Brügger L (2007). A risk-driven security analysis method and modelling language. *BT Technology Journal*, 25(1).
59. Koch M and Parisi-Presicce F (2006). UML specification of access control policies and their formal verification. *Software and System Modeling*, 5(4):429–447.
60. Kolarczyk S, Koch M, Löhr K-P, and Pauls K (2006). SecTOOL – supporting requirements engineering for access control. In Günter Müller, editor, *ETRICS*, volume 3995 of *Lecture Notes in Computer Science*: 254–267. Springer.
61. Lotz V (1997). Threat scenarios as a means to formally develop secure systems. *Journal of Computer Security*, 5(1):31–68.
62. Maña A, Montenegro JA, Rudolph C, and Vivas JL (2003). A business process-driven approach to security engineering. In *DEXA Workshops*: 477–481. IEEE Computer Society.
63. Maña A, Rudolph C, Spanoudakis G, Lotz V, Massacci F, Melideo M, and López-Cobo J-M (2006). Security engineering for Ambient Intelligence: A manifesto. In H Mouratidis, editor, *Integrating Security and Software Engineering: Advances and Future Vision*. Idea Group.
64. Massacci F, Mylopoulos J, and Zannone N (2007). Computer-aided support for secure tropes. *Autom. Softw. Eng.*, 14(3):341–364.
65. Mathe J, Duncavage S, Werner J, Malin B, Ledeczki A, and Sztipanovits J (2007). Implementing a model-based design environment for clinical information systems. In Sztipanovits et al. [83].
66. McGraw G (2006). *Software Security: Building Security In*. Addison Wesley.
67. Méry D and Merz S (2007). Specification and refinement of access control. *J. UCS*, 13(8):1073–1093.
68. Moebius N, Haneberg D, Reif W, and Schellhorn G (2007). A modeling framework for the development of provably secure e-commerce applications. In *ICSEA: 8*. IEEE Computer Society.
69. Mouratidis H, Giorgini P, and Manson GA (2003). Integrating security and systems engineering: Towards the modelling of secure information systems. In J Eder and M Missikoff, editors, 15th International Conference on Advanced Information Systems Engineering (CAiSE 2003), volume 2681 of *LNCS*: 63–78. Springer.
70. Mouratidis H, Jürjens J, and Fox J (2006). Towards a comprehensive framework for secure systems development. In 18th International Conference on Advanced Information Systems Engineering (CAiSE 2006), *LNCS*. Springer.
71. Pironti A, Sisto R (2008). Soundness Conditions for Message Encoding Abstractions in Formal Security Protocol Models. In *ARES 2008*: 72–79.
72. Ray I, France RB, Li N, and Georg G (2004). An aspect-based approach to modeling access control concerns. *Information & Software Technology*, 46(9):575–587.
73. Redwine S (2007). Introduction to modeling tools for software security. In: *Build Security In – Setting a Higher Standard for Software Assurance*. Software Engineering Institute (SEI), Carnegie Mellon University. Available at <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/tools/modeling/698-BSI.html>.
74. Rosado DG, Fernández-Medina E, Piattini M, and Gutiérrez C (2006). A study of security architectural patterns. In *ARES*: 358–365. IEEE Computer Society.
75. Saltzer J and Schroeder M (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308.
76. Santen T (2006). Stepwise development of secure systems. In Janusz Górski, editor, *SAFE-COMP*, volume 4166 of *Lecture Notes in Computer Science*: 142–155. Springer.
77. Santen T, Heisel M, and Pfizmann A (2002). Confidentiality-preserving refinement is compositional – sometimes. In Dieter Gollmann, Günter Karjoth, and Michael Waidner, editors, *ESORICS*, volume 2502 of *Lecture Notes in Computer Science*: 194–211. Springer.
78. Schneider F, editor (1999). *Trust in Cyberspace*. National Academy Press, Washington, DC. Available at <http://www.nap.edu/readingroom/books/trust>.
79. Seehusen F and Stølen K (2006). Information flow property preserving transformation of UML interaction diagrams. In David F. Ferraiolo and Indrakshi Ray, editors, *SACMAT*: 150–159. ACM.

80. Sindre G and Opdahl AL (2005). Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44.
81. Siveroni I, Zisman A, and Spanoudakis G (2008). Property specification and static verification of UML models. In *3rd International Conference on Availability, Reliability, and Security (ARES'08)*.
82. Spanoudakis G, Kloukinas C, and Androutsopoulos K (2007). Towards security monitoring patterns. In *SAC: 1518–1525*. ACM.
83. Sztipanovits J, Breu R, Ammenwerth E, Bajcsy R, Mitchell JC, and Pretschner A, editors (2007). *Workshop on Model-based Trustworthy Health Information Systems (MOTHIS@Models)*.
84. UMLsec group. Security analysis tool, 2004. <http://www.umlsec.org>.
85. Whittle J, Wijesekera D, and Hartong M (2008). Executable misuse cases for modeling security concerns. In *ICSE 2008*.
86. Whyte B and Harrison J (2008). *Secure software development – a white paper*. Knowledge Transfer Network on Cyber Security, UK. Available at http://www.ktn.qinetiq-tim.net/content/files/groups/seuresoft/SSDSIG_softwareSecurityFailures.pdf.
87. Wimmel G and Jürjens J (2002). Specification-based test generation for security-critical systems using mutations. In *International Conference on Formal Engineering Methods (ICFEM)*, volume 2495 of LNCS: 471–482. Springer.
88. Wirsing M (2008). *Software engineering for secure software-intensive systems*. Consultation meeting on "Engineering Secure Software Systems" in the context of the preparation of the EU FP7 ICT work programme 2009-2010, Brussels. Presentation available at ftp://ftp.cordis.europa.eu/pub/fp7/ict/docs/security/20080423-martin-wirsing-lmu-munich_en.pdf.
89. Woodside M, Petriu DC, Petriu DB, Xu J, Israr T, Georg G, France R, Bieman JM, Houmb SH, and Jürjens J (2008). Performance analysis of security aspects by weaving scenarios from UML models. *Journal of Systems and Software*, vol. 82, 1: 56-74.
90. Yoshioka N, Honiden S, and Finkelstein A (2004). Security patterns: A method for constructing secure and efficient inter-company coordination systems. In *EDOC: 84–97*.
91. Yskout K, Scandariato R, De Win B, and Joosen W (2008). Transforming security requirements into architecture. In *ARES [42]: 1421–1428*.
92. Yu Y, Jürjens J, and Mylopoulos J (2008). Traceability for the maintenance of secure software. In *24th International Conference on Software Maintenance (ICSM)*. IEEE.
93. Zhang G, Baumeister H, Koch N, and Knapp A (2005). Aspect-oriented modeling of access control in web applications. In *6th International Workshop on Aspect-Oriented Modeling*.