Key Issues of a Formally Based Process Model for Security Engineering

Ruth Breu, Klaus Burger, Michael Hafner
Research Group "Quality Engineering"
Universität Innsbruck, Institut für Informatik
Technikerstraße 13, A-6020 Innsbruck
{Ruth.Breu, Klaus.Burger, m.hafner}@uibk.ac.at

Jan Jürjens, Gerhard Popp, Guido Wimmel
Software & Systems Engineering
Technische Universität München , Institut für Informatik
Bolzmannstraße 3, D-85748 Garching
{Jan.Juerjens, Gerhard.Popp, Guido.Wimmel@in.tum.de

Volkmar Lotz
Siemens AG
Corporate Technology, Security, Munich
Otto-Hahn-Ring 6, D-81730 München
volkmar.lotz@siemens.com

Abstract: *In this paper we outline a new process model for security engineering. This process model extends object oriented, use case oriented software development by systematic security requirements elicitation and realization. In particular, we integrate the modeling of security requirements, threat and risk analysis on the one hand with the modeling of business processes, use cases and the construction of the software architecture on the other hand. Since formal methods play a special role in security engineering we characterize their usage within the process model presented.*

Key-Words: *Process Model for Security Engineering, Security Process Model, Risk Analysis, Object Oriented Software Development*

Introduction

Today, evolving new technologies like Web Services or component based frameworks and a whole bunch of novel electronic devices just give us a small impression of how IT systems will work and will be integrated in our life in a few years. In any way we will be faced with worlds of dynamically interconnected systems interchanging data and messages. However, new business models between partners cooperating over the web will only evolve and get economic importance if a major prerequisite is fulfilled: trust. Companies will only be willing to let their systems cooperate with other systems if these systems guarantee in some way the confidential treatment of sensitive data, integrity of interchanged documents or the controlled handling of business protocols.

This means that security will play a more and more important role within systems development. Today's process models like the Unified Process [15, 20] or Catalysis [13] treat security aspects as non-functional requirements among others. Our claim is that security is a requirement which has to be considered in all stages of development and which needs particular modeling techniques to be captured. A first step towards this aim was done in [30] with a mapping of the requirements of the Common Criteria for Security Engineering [6] to the phases of a generic software development process.

Some general aspects can be identified to be crucial for security engineering.

Separation of requirements and measures. The mere technical treatment of security causes a series of problems. First, IT security is not limited to technical measures but has its impact on work processes and system services. Moreover, quality assurance is a crucial aspect for security critical systems. Security measures can only be checked for correctness if the requirements are known. This is particularly important since security requirements are often realised by a combination of several measures (e.g. think of confidential data that is secured by database protection, secure communication lines and the careful design of user interfaces).

Traceability. Traceability means the ability to trace system properties along different levels of abstraction supporting controlled system modification and extension. Since security related aspects generally are spread over the requirements specification and the implementation, traceability plays a particularly important role in the lifecycle of a security critical system, from business process modelling until test, documentation and maintenance.

Correctness. A further aspect concerns correctness. The statement of some security requirement is not of much sense if the system implementation is not checked against it. This means that validation and verification plays a major role in security engineering. For highly critical systems, e.g. technical control systems, banking and health care systems, the usage of formal methods is advisable and sometimes mandatory (e.g. for certification with respect to the higher assurance levels of the Common Criteria).

Completeness. Security is inherently connected with completeness. Developers always try to consider all possible threats of a system. In combination with the aspects stated above this implies the importance to capture the complete set of requirements during the development.

In our paper we sketch the main features of our development process for security engineering. Our basis is the kernel of a process model which is oriented towards object oriented and use case oriented development with UML.

We both enhance the artifacts of this process model by security-specific aspects, define additional artifacts and consider the iterative and incremental process as a whole. Our main aim is to integrate concepts of object oriented development and already existing concepts of security engineering, like the approaches of [1, 17, 18, 22, 33, 10, 26] and others.

The structuring of the following sections is as follows. In section 2 we outline the principles of the design process our approach is based on. Section 3 presents the activities and artifacts of our process model. Section 4 discusses the role of formal methods in security engineering and, finally, in section 5 a conclusion is drawn.

1. Basic Concepts of an Object Oriented Software Process

Far away from presenting a whole software development process, we just present in this section the key concepts of the object oriented process we are referring to in the subsequent sections. We use a general approach which can be easily mapped to any of the established process models, such as the Unified Process, Catalysis or the V-Model [29].

2.1 The Core Object Oriented Process

Figure 1 depicts the main activities and artifacts of the core process together with the dependencies. Below we shortly characterise the artifacts to which we will refer in the rest of the paper as *core artifacts* and summarise the ideas of iterative software development.

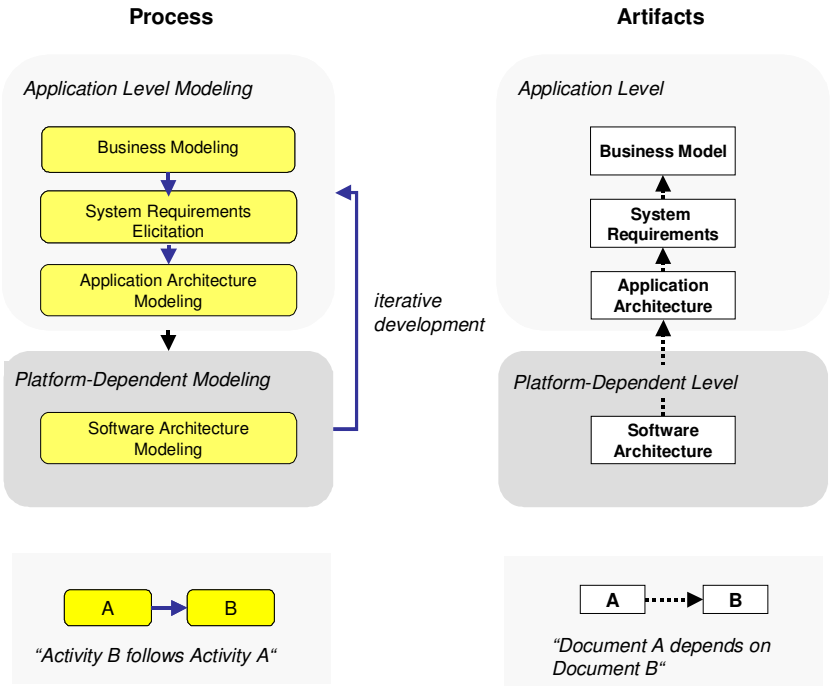

Figure 1 - The Core Object Oriented Process

In order to clarify our ideas we will in the sequel use the well-known example of the development of an automated teller machine (ATM). For the presentation of an application of our process model to a real project we refer to [3].

The **Business Model** captures the organizational environment of the IT-system. The model describes

- who is acting in the business domain (*actors*)
- what *activities* the actors are performing
- which *objects* the activities need as input and which objects they produce as output

Actors in the ATM example are the *customer* and the *bank clerk*. Examples for activities are *withdraw money* (performed by the customer) and *deposit cash* (performed by the clerk). These activities are part of the business process *operate ATM*. Examples for objects in the application domain are the *customer,* the *card* and the *account*.

Actors, activities and objects are modeled by several diagrams. In the UML context this comprises

- *activity diagrams* modeling business processes and activities
- *class diagrams* modeling the organizational structure of the company and the structure of static concepts (customer, card etc.)

In its system view the business model focuses on the work processes and is independent of the IT-system.

The **System Requirements** describe a black box view of the system to be developed based on the concept of *use cases*. Each use case corresponds to a coherent interaction between some actor and the system. The use cases together (also called the use case model) describe the whole functionality of the system. Basic concepts of the use case model are

- the *actors* interacting with the system
- the *use cases*
- the *objects* being involved in the use cases

In extension to business modeling the actors in the use case model might not only represent human beings but also external systems. For instance, actors in the use case model of the ATM comprise the *customer* and the *bank clerk*, but also the *host* system to which the ATM connects. Sample use cases are *withdraw money* or *show daily transactions*.

The System Requirements basically consist of two descriptions,

- the use case diagram together with the textual description of use cases
- the class model describing the static concepts

Commonly, the class model of the System Requirements is the same or a refined version of the class model of the Business Model. The textual description of a use case comprises sections for pre- and postconditions of the use case, for the main steps and interactions when performing the use case and sections for exceptions and variants.

The **Application Architecture** refines the level of description. More precisely, the system is divided into a set of *logical components*. Each component is responsible for a portion of the system structure and behaviour. Interfaces enable the independent development of the system components.

Concerning the design of the system behavior the textual descriptions of the use cases are refined into *scenarios* in the Application Architecture. The scenarios describe the use cases as message flows between objects. That way, an object oriented view of the whole system is achieved. In the UML context the Application Architecture typically consists of the following diagram types:

- one or several component diagrams containing components, interfaces and the related classes
- a series of sequence diagrams each one related to some use case
- state diagrams modeling complex processes or class interfaces.

The Business Model, the System Requirements and the Application Architecture have in common that they rely on an application oriented system view and are independent of any technical platform.

The **Software Architecture** is based on an implementation oriented view of the system. In this model the hardware and software platform is chosen and roughly desribed. This comprises typically

- the design of the system's network structure
- the definition of the database structure
- the design of the software components with their interfaces (e.g. GUI, application kernel)

Besides the artifacts themselves, their sequence and interdependencies form the main characteristics of a process. In this respect, **iterative development** [20] is one of the most important concepts in modern process models. Unlike in the waterfall model the artifacts are not elaborated completely one after another but in an iterative way. The basic idea is that the system is not implemented as a whole but in increments, starting with a kernel functionality and ending with the complete system. Each iteration is managed in a waterfall-like way resulting in a new executable increment. For instance, for the System Requirements this means that not all use cases are specified in detail in a first step but only the kernel ones, and other use cases are specified in later stages of design.

2. Core Concepts of a Process Model for Security Engineering

In this section we present the concepts of our process model. The core idea is the introduction of a micro-process which we call *Security Analysis*. In section 3.1 we clarify the basic idea of the micro-process and its integration in the core process. Section 3.2 is devoted to the security enhanced artifacts.

3.1 The Security Analysis Process

Concerning the treatment of security related aspects in the software lifecycle we pursue a five step approach which we call *Security Analysis* (see Table 1).

1. *Security Requirements Elicitation* – Specify security requirements in the context of the core artifacts.
2. *Threats Modeling* – Gather potential threats related with the security requirements.
3. *Risk Analysis* – Estimate the occurrence of every threat and its potential harm either quantitatively or qualitatively. This provides the basis for the decision whether a threat has to be countered or not.
4. *Measures Design* – Design appropriate measures taking into account the result of the risk analysis and integrate the description of these measures into the core artifacts
5. *Correctness Check* – Check the chosen measures (formally or informally) against the specified requirements and decide what requirements still wait for realisation.

Table 1 – The Security Analysis Process

We illustrate these five steps by the scenario in Figure 2.

1. The non-repudiation of the activity *Fill cash tray* (performed by the bank clerk) is identified as a security requirement in the business model of the ATM.
2. We associate this requirement with the following threat.
   The bank clerk fills the cash tray of the ATM and takes some money off.
3. We both estimate the probability of this threat and the possible damage for the bank as high.
4. The measures to counter the threat both involve the business level and the IT system. On the one hand side the business process is reorganised and improved, e.g. the sum of the cash filled in the ATM is checked by a second clerk (separation of duties). On the other hand we add the functional requirement to the System Requirements that all loadings have to be logged by the system.
5. We check the measures (reorganisation of the business process and logging of loadings) against the requirement of non-repudiation. The result of this check is the new requirement that the two involved clerks must not have access to the logging information.

Figure 2 – Scenario of a Security Analysis

In our process model we treat the Security Analysis as a micro-process which is performed at each level of abstraction and for each increment. This has the following advantages.

- Requirements and measures are each explored and described at the appropriate level of detail. For instance, developers and customers can develop an access policy on the non-technical use case level, whereas software architects think about the authentication of software components on the platform-dependent level.
- Each security requirement can be traced along the levels of abstraction. More precisely, each requirement is transformed into one or several requirements or into some measure at the abstraction level beneath.

The enhanced process involves the following artifacts.

- All artifacts of the core process are also part of the enhanced process. We extend these core artifacts by techniques and methods to express security requirements at the given level of abstraction.
- We define two additional artifacts – the *Application Level Risks* and the *Technical Risks* – documenting threats and risks at the application and platform-dependent level, respectively.

The Risks artifacts complement the specification of requirements by supporting the choice of appropriate measures and the definition of test cases. In the Risks artifacts both external and internal risks (i.e. threats from outside or inside the group of intended system users, respectively) are analysed.

From the viewpoint of a given core artifact (e.g. the System Requirements or the Application Architecture) the Security Analysis results in the series of actions (for each increment) depicted in Table 2.

Step 0:	Develop the core artifact as described in the core process.
Step 1:	Enhance this artifact by security requirements according to given methodological guidelines.
Step 2 and 3:	Analyse related threats and risks in the respective Risks artifact.
Step 4:	Integrate security measures in the core artifact. These security measures may refer to the requirements of the abstraction level above or of the same abstraction level..
Step 5:	Perform the correctness check and eventually add refined requirements to be fulfilled by the abstraction level beneath.

Table 2 – Iterations of a Given Core Artifact

A crucial aspect in this method is the comprehensive specification of security requirements. In our approach we provide systematic checks of the base models with respect to the following settled set of objectives.

- Confidentiality – keeping content from all but those authorised to have it
- Authenticity – establishing the validity of transmission, message or originator
- Data Integrity – prevention of unauthorised modification of data
- Non-Repudiation – guarantee that an entity cannot deny previous commitments of actions
- Availability – ensuring that unauthorised subjects cannot prevent authorised ones from the execution of their functions

In the subsequent section we will demonstrate the application of our process in more detail focusing on the security enhancements of the core artifacts.

Figure 3 summarises the activities and artifacts of the enhanced process, respectively.

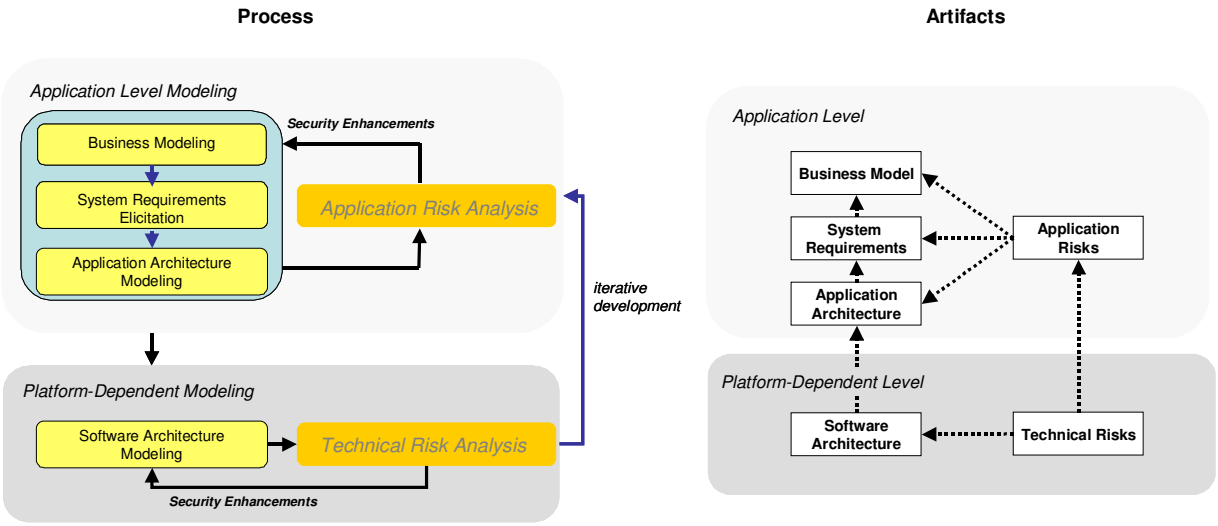

Figure 3 – The Security Process Model

3.2 Security Enhanced Core Artifacts

In the following we briefly describe for each artifact how security requirements are specified, what kinds of threats are captured in the correlated Risks document and what kind of measures the core artifact may comprise. For a more detailed presentation we refer to a series of related publications ([3, 4, 16]).

3.2.1 Business Model

The systematic security check of the given business model comprises the following aspects.

Confidentiality is specified at the business level at a rough level of detail in the class diagram. Each class (or attribute) is provided with one of the keywords public, confidential or secret. Another aspect is related with the object flow between activities. Object flows between different actors require some data exchange. For each such object flow in the business process model we have to check if confidentiality of the object involved is critical.

Data Integrity concerns the access to objects and is captured by the security levels in the class diagram. Moreover, analogously to above, each object flow in the process model is checked for the aspect of data integrity.

As an example, think of the workflow of a loan enquiry (Figure 4). Since the loan form and the offer contain the financial conditions both their confidentiality and integrity is crucial.

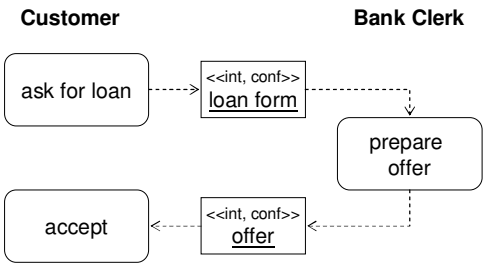

Figure 4: Sample Object Flow with Security Requirements

Authenticity is a requirement which refers to activities and actors in the business model. Each activity has to be checked if authenticity of the executing actor is a critical requirement. In our example of the ATM the customer, of course, has to be authenticated when executing the activities *withdraw* and *check balance*.

Non-repudiation is again a requirement involving activities and actors. For each activity it has to be checked if it is important that the executing actor cannot repudiate the execution of this activity. In the example of the ATM *withdraw* is a good example for non-repudiation.

For each of the security requirements, related threats, like the one described in Figure 2 (item 2), should be described and assessed in the Application Level Risks document. Measures at the level of the business model may involve the reorganisation of work processes typically including the separation of duties and the way of handling objects (e.g. requiring the destruction of certain documents after use).

3.2.2 System Requirements

In the System Requirements document the security requirements of the business model are systematically detailed and put into the context of the use cases and the extended class diagram.

An important activity within the specification of system requirements is the development of a detailed *access policy*. Currently we develop a formally based model for specifying access rights in the context of class diagrams and use case diagrams. Using an extension of OCL predicates (or informal text) the model describes for each actor and each class (or method/attribute) the kind of permission. As an example we specify

The customer has read access to the balance of his own account.

The customer has write access to the balance of his own account if the daily limit is not overdrawn.

Here read and write access means an indirect access through application of the use cases. The access policy is developed in cooperation with customers and/or end-users. If the access rights are specified formally they can be automatically transformed into code. For a more detailed presentation of our specification framework we refer to [4].

Other security relevant aspects are part of the textual description of use cases or are treated in new use cases (such as the use case *log in* in which the authentication takes place). The schematic textual description of use cases is now extended by a section **security** describing the enhanced requirements and measures.

As an example the textual description of the use case *withdraw* is enhanced by the requirements R1 and R2 as shown in Figure 5. Both requirements are refined versions of the security requirements in the business model.

use case withdraw
… (previous textual description of the use case)
security
R1 The withdraw transaction is logged by the system.
R2 The customer has to be authenticated before starting the use case.
R3 ATM and host have to be authenticated before the transaction starts.
R4 The connection to the host must guarantee the confidentiality and integrity of sent and received data.
R5 The transaction cannot be repudiated neither by the ATM nor the host.
R6 The use case must be available 24 hours per day with a maximum of 24 hours breakdown per month.

Figure 5: Security Section of the Use Case Withdraw

Further security requirements contained in the use case description analyse the communication with external systems. In our example the use case *withdraw* includes the communication with the host. Since confidential data is sent over this channel, we add the requirements R3, R4 and R5 to the security section of the use case *withdraw* (cf. Figure 5).

Finally, another requirement which comes into play at this level of abstraction is availability. We add the requirement R6 to the use case *withdraw*.

The security analysis of the use cases in general leads to new threats (like the threat that the ATM is broken down for a few days). These threats have to be integrated in the Application Level Risks document and should be linked with the relevant parts in the System Requirements.

3.2.3 Application Architecture

Based on the security requirements stated in the use case model and the risk analysis the security enhancement of the Application Architecture mainly deals with the design of appropriate measures. This comprises the following steps:

- definition of logical security components and their interfaces
- extension of the sequence diagrams describing the message flow of use case execution by security specific messages

Concerning the security requirements of the use case model the development of the (Security) Application Architecture involves the following kind of measures:

- design of authentication procedures
- access control and access rights management at the application level
- error tracing measures (e.g. for authentication and access control)

- introduction of security protocols for data integrity, confidentiality and non-repudiation

As an example, the use of a challenge-response protocol for authentication or of signatures for data integrity is chosen at this level of abstraction. The check of correctness of these measures against the requirements may require complex mathematical proofs.

For this reason and for the systematic transition from requirements to measures the use of *security patterns* is of great importance in this stage of design. Catalogs of security patterns currently are developed by a number of groups [33, 10, 2, 27]. However, the flexible integration of security patterns into concrete models still poses a number of unsolved problems, e.g. concerning the dynamic behavior of the resulting system and the combination and interference of patterns.

Threats recovered at the level of the Application Architecture relate to the chosen measures. An example for this are the threats created when applying the still very popular perimeter security model of the "mainframe era" to a distributed server environment [19]. Confidence in the correctness of measures such as security protocols can be gained by the use of formal methods (see Section 4).

3.2.4 Software Architecture

This phase starts with the design of the basic architecture (comprising the network structure, database structure, choice of programming languages, frameworks and so on) and the logical components which are distributed across network nodes.

The security enhancement to the Software Architecture (Security Software Architecture for short) is then developed in five consecutive steps as listed in Table 2.

After the mapping of the security requirements stated in the System Requirements (subsection 3.2.2) and the Application Architecture (subsection 3.2.2), the technical threats of this basic architecture are analysed. The technical threats can be gathered independently of the application domain e.g. based on checklists [23, 5, 25]. Examples for technical threats in a concrete architecture are wiretaps, insider abuse of net access or system penetration. After being identified each technical threat is related with the application-level threats cross-checking the technical and the application-dependent level. The third step leads to an estimation of associated risks (taking into account the technical opportunities of the attacker and the possible damage).

In the fourth step the security architecture is designed. Typically this comprises the following aspects:

- realisation of the security measures described in the Application Architecture on the chosen platform
- selection of special hardware devices (such as smart cards, key generators)
- access control of databases
- choice of predefined components or frameworks supporting security (like PGP, J2EE, etc.)

- introduction of measures ensuring the availability of system services and disaster recovery

Finally, the proposed measures have to be evaluated in their compliance with the requirements defined at the beginning of the phase.

For the development of the Software Architecture the security of the basic environment has to be taken into account as well. If the system runs on a platform which provides basic protection (e.g. through firewalls, intrusion detections, virus scanners) the related aspects do not have to be analysed in the context of the project. In the other case the basic protection has to be provided as part of the systems development.

Concerning the systematic transition from the Application Architecture to the Software Architecture the use of frameworks or model driven approaches is the primary choice. Their goal is the realisation of systems in a platform-independent style following the idea that the framework takes over the platform-dependent part of the work. A prominent exponent of such approaches is the J2EE-environment [14]. For the platform-independent realisation of security solutions first limited steps exist [22, 24], but more general approaches still have to be developed.

3. Use of Formal Methods in Security Engineering

Formal methods involve "modeling and analysis where the models and analysis procedures are derived from or defined by an underlying mathematically precise foundation" [21]. Due to the increased need for correctness the benefits of applying formal methods for security critical systems development has been widely acclaimed, for instance, formal security models and system specifications are mandatory to achieve high assurance levels of the Common Criteria. The enforcement of preciseness allows the detection of inconsistent requirements, flaws in protocol and component design and the identification of hidden assumptions.

Formal methods can be applied in all phases of development. In our framework we specifically propose formal reasoning in the Requirements Specification with a formal specification of the security requirements and in the Application Architecture by modeling and verifying particular security-critical subsystems (such as security protocol handlers).

However, there is no need for an all-or-nothing approach by enforcing formal analysis either in all development phases or not at all: Even partial use of formal arguments will help to avoid errors and increase evidence of the validity of security properties and, thus, confidence in the system´s security. Our approach leaves the decision where and when to use formal arguments to the development task at hand by providing a formal foundation, i.e., semantics, to the description techniques used for the development artifacts, in particular its security extensions.

A large number of modeling and verification approaches has been proposed for security-critical systems (see e.g. [11] for an overview), ranging from specifications in general languages such as CSP or TLA to security-specific formalisms such as access control models or security protocol definition languages.

Our approach aims to integrate security aspects into systems development, based on formal methods reflecting the security requirements and threat scenarios. Important concepts are tool support and use of description techniques understandable for non-expert software engineers.

Specifically, we propose to integrate security aspects (such as security requirements and threats) into the general system models. UMLsec, an extension (profile) of the UML, provides this feature via stereotypes and tags that can be attached to UML model elements. Together with UMLsec, a formal semantics of a subset of the UML has been specified, to support proofs of the modeled security properties. For more information on UMLsec, see [18].

A similar aim is pursued by the security extensions of the AUTOFOCUS tool [12]. AUTOFOCUS offers description techniques that are conceptually similar to a subset of the UML, but tailored for models of communicating component-based systems. For AUTOFOCUS, analogous security extensions exist as in UMLsec [32]. Here, formal verification and security-related development steps (such as introducing a security protocol) can be assisted by the tool. Besides, a formal (executable) model of the security-related functionality can be used for *model based test case generation* [31].

We want to stress that the use of formal methods itself is not a guarantee that the system has no vulnerabilities. Care must be taken that the assumptions in the formal models (such as the security of the used cryptographic algorithms) are indeed fulfilled. In addition, implementation-specific issues (e.g. the possibility of buffer overflows), are usually not reflected in the formal models.

4. Conclusion

In the preceding sections we sketched a process model supporting the systematic development of security-critical systems within the framework of object oriented use case based modeling. We defined a micro-process for security analysis which we apply repeatedly in all stages of the design.

The steps of the security analysis are settled and can be partly found in a number of other approaches, e.g. in [8, 28, 9]. What distinguishes us from these approaches is the comprehensive view of the whole design process and a rigorous support of traceability.

Currently the process model is extended and elaborated by our groups in many directions, ranging from the formal modeling of access policies, the tool-supported management of requirements, threats and risks, the platform-independent development of security solutions to the model based test case generation.

The process model presented aims to integrate many of the existing approaches to security engineering which are oriented towards single aspects. Our goal is to develop a tool-supported process appropriate for industrial use. Positive results from pilot projects in an industrial context encourage us to further work.

References

[1] R. Anderson: *Security Engineering*. John Wiley, 2001.

[2] B. Blakley. *Securtiy Design Patterns.* The OpenGroup. 2002 http://www.opengroup.org/security/gsp.htm

[3] R. Breu, K. Burger, M. Hafner, G. Popp. *Core Concepts of a Process Model for Security Engineering*. Submitted.

[4] R. Breu, G. Popp. *Actor-Centric Modeling of Access Rights.* Submitted.

[5] Bundesamt für Sicherheit in der Informationstechnologie. *IT Baseline Protection Manual.* Bonn, 2001. http://www.bsi.de/gshb/english/menue.htm

[6] www.commoncriteria.org

[7] D. Denning, G. Sacco. *Timestamps in key distribution protocols*. Communications of the ACM, 24(8):533-536, 1981.

[8] D. Firesmith. *Security Use Cases*. In: Journal of Object Technology 2(3), 2003. http://www.jot.fm/issues/issue_2003_05/column6

[9] E.B. Fernandez, J.C. Hawkins. *Determining Role Rights from Use Cases*. Proc. ACM Workshop on Role-Based Access Control 1997: 121-125

[10] E. Fernandez, R. Pan. *A Pattern Language for Security Models.* 8th Conference of Pattern Languages of Programs (PloP), 2001.

[11] S. Gritzalis, D. Spinellis, P. Georgiadis. *Security protocols over open networks and distributed systems: Formal methods for their analysis, design, and verification*. Computer Communications, 22(8):695-707, 1999.

[12] F. Huber, S. Molterer, A. Rausch, B. Schätz, M. Sihling, O. Slotosch. *Tool supported Specification and Simulation of Distributed Systems*. In: International Symposium on Software Engineering for Parallel and Distributed Systems, pages 155-164, 1998.

[13] D. D´Souza, A. Wills. *Components and Frameworks with UML – The Catalysis Approach*. Addison-Wesley, 1999.

[14] http://java.sun.com/j2ee/

[15] I. Jacobson, G. Booch, J. Rumbaugh. *The Unified Software Development Process.* Addison-Wesley, 1999.

[16] J. Jürjens, G. Popp, G. Wimmel. *Use Case Oriented Development of Security Critical Systems*. Information Security Bulletin, 8(2): 55-60, March 2003.

[17] J. Jürjens. UMLsec*: Extending UML for secure systems development*. In: J.-M. Jézéquel, H. Hussmann, S. Cook (eds.): UML 2002. Lecture Notes in Computer Science, vol. 2460, Springer, 2002.

[18] J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, to appear.

[19] M. Kis. *Information Security Antipatterns in Software requirements Engineering*. 9th Conference of Pattern Languages of Programs (PloP), 2002.

[20] P. Kruchten. *The Rational Unified Process*. Addison-Wesley, 1999.

[21] N.G. Leveson. *Guest Editor´s Introduction: Formal Methods in Software Engineering*. IEEE Transactions in Software Engineering 16(9), 1990.

[22] T. Lodderstedt, D. Basin, J. Doser. *Secureuml: A uml-based modeling language for model-driven security*. In: J.-M. Jézéquel, H. Hussmann, S. Cook (eds.): UML 2002. Lecture Notes in Computer Science, vol. 2460, Springer, 2002.

[23] J.D. Meier et al*., Improving Web Application Security, Threats and Countermeasures*. Microsoft Corporation, 2003 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp

[24] ObjectSecurity Ltd. *A Model-Driven, Flexible Security-Framework for Distributed Systems.* Cambridge, 2003.

[25] T. R. Peltier. *Information Security Risk Analysis.* Auerbach, 2001.

[26] M. Schumacher, U. Roedig. *Security Engineering with Patterns.* 8th Conference of Pattern Languages of Programs (PloP), 2001.

[27] M. Schumacher. *Security Engineering with Patterns.* PhD Thesis, Lecture Notes in Computer Science, LNCS 2754, Springer, 2003

[28] G. Sindre, A. Opdahl. *Templates for misuse case description.*In: Proc. Seventh International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'2001), 2001.

[29] www.v-modell.iabg.de

[30] M. Vetterling, G. Wimmel, A. Wißpeintner. *Secure Systems Development Based on the Common Criteria.* In Proc. 10th International Symposium on on the Foundations of Software Engineering (FSE-10), 2002.

[31] G. Wimmel, J. Jürjens. *Specification-Based Test Generation for Security-Critical Systems using Mutations*. In 4th International Conference on Formal Engineering Methods (ICFEM 2002), 2002.

[32] G. Wimmel, A. Wißpeintner. *Extended Description Techniques for Security Engineering. In Trusted Information, the New Decade Challenge*. 16th International Conference on Information Security (IFIP/Sec), 2001.

[33] J. Yoder, J. Barcalow *Architectural Patterns for Enabling Application Security* . 4th Conference of Pattern Languages of Programs (PloP), 1997.