Towards Using Security Patterns in Model-based System Development*

Jan Jürjens and Gerhard Popp and Guido Wimmel
Department of Computer Science, Munich University of Technology
Arcisstrasse 21, D-80290 München, Germany
phone: +49-89-28928166 fax: +49-89-28925310
{juerjens,popp,wimmel}@in.tum.de

June 10, 2002

Abstract

We explain how to use patterns in the context of model-based development of security-critical systems in a methodological and formal way. Our approach integrates two complementary notations and explains how to use them within an industrial software development process. We use an extension of the Unified Modeling Language (UML) for security-critical systems development, called UMLsec, to integrate security requirements into a system specification. An industrial-strength CASE tool, AutoFocus, is used to analyze the specifications mechanically, and to generate code or test-sequences. We also sketch different ways of integrating security patterns into system development.

Vers. 27.V.02 of a paper at EuroPLoP02. Current version and other material: www.jurjens.de/jan .

1 Introduction

Patterns [GHJV95, BMR+96] encapsulate design knowledge of software engineers by presenting recurring design problems and standardized solutions. One can use transformations on UMLsec models to introduce patterns within the design process. A goal of this approach is to ensure that the patterns are introduced in a way that has previously been shown to be useful and correct. Also, having a sound way of introducing patterns using transformations can ease formal verification (where desired), since the verification can be performed on the more abstract and simpler level, and one can derive security properties of the more concrete level, provided that the transformation has been shown to preserve the relevant security properties.

In this position paper, we explain how to use patterns in the context of model-based development of security-critical systems in a methodological and formal way.

*This work was partially supported by the German Ministry of Economics within the FairPay project.

Our approach integrates two complementary notations and explains how to use them within an industrial software development process:

- We use an extension of the Unified Modeling Language (UML) for security-critical systems development, called UMLsec, to integrate security requirements into a system specification. Since UML is the industrial standard in object-oriented modelling, one may already have a system specification in UML available, or one could expect developers to be able to construct it.

- An industrial-strength CASE tool, AutoFocus [HMR+98], is used to analyze the specifications mechanically, and to generate code or test-sequences. AutoFocus has a notation that is essentially a simplified fragment of UML, and it is extended to include security requirements in a way parallel to that of UMLsec.

Note that there exists a formal semantics for UMLsec (covering a fragment of UML). Thus in principle one could use UMLsec also for mechanical analysis. However, current UML CASE tools do not offer the needed functionality. Since the translation from UMLsec to the AutoFocus notation is straightforward (a mechanical translation is work in progress), we chose the current approach.

Both approaches can, of course, also be used independently (for example, only the UMLsec approach, if a mechanical analysis is not required, or only the AutoFocus approach, if the specification has to be constructed from scratch by the people analyzing the system for security aspects, anyhow). Therefore, we explain the two approaches independently.

In the remainder of the paper, we first, we explain two complementary approaches using UMLsec and AutoFocus, and then we shortly sketch different ways of integrating security patterns into system development.

An introduction to UMLsec and examples for its use can be found in [Jür01c, Jür02, Jür01b]. The security extension of AutoFocus is explained and used in [WW01, JW01]. Part of the work presented here is an extension of the work in [Jür01a].

2 Transformations for using patterns

In our approach, the application of a pattern p corresponds to a function f_p which takes a UML specification $\mathcal{S}$ and returns a UML specification, namely the one obtained when applying p to $\mathcal{S}$. Technically, such a function (and thus the corresponding pattern) can be presented by defining how it should act on certain subsystem instances, and by extending it to all possible UML specifications in a compositional way: Suppose that we have a set S of subsystems instances such that none of the subsystem instances in S is contained in any other subsystem instance in S. Suppose that for any subsystem instance $\mathcal{S} \in S$ we are given a subsystem instance

$f_p(\mathcal{S})$. Then for any UML specification $\mathcal{U}$, we can define $f_p(\mathcal{U})$ by substituting each occurence of a subsystem instance $\mathcal{S} \in S$ in $\mathcal{U}$ by $f_p(\mathcal{S})$. We demonstrate this with an example.

Figure 1 gives a high level system specification in form of a UML subsystem $\mathcal{C}$ for communication from a sender object to a receiver object, including a class diagram with appropriate interfaces.

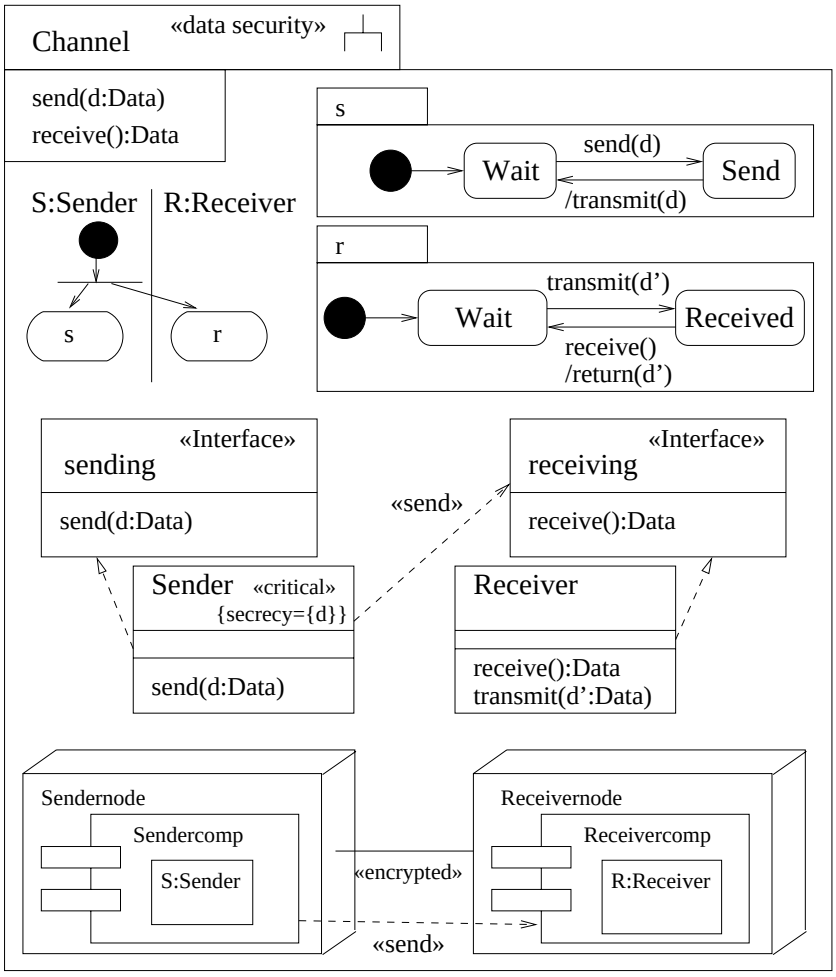

Figure 1: Example: Sender and receiver

In the UML subsystem, the Sender object is supposed to accept a value in the variable d as an argument of the operation send and send it over the «encrypted» Internet link to the Receiver object, which delivers the value as a return value of the operation receive. According to the stereotype «protected» and the associated tag {secrecy}, the subsystem is supposed to preserve the secrecy of the variable d.

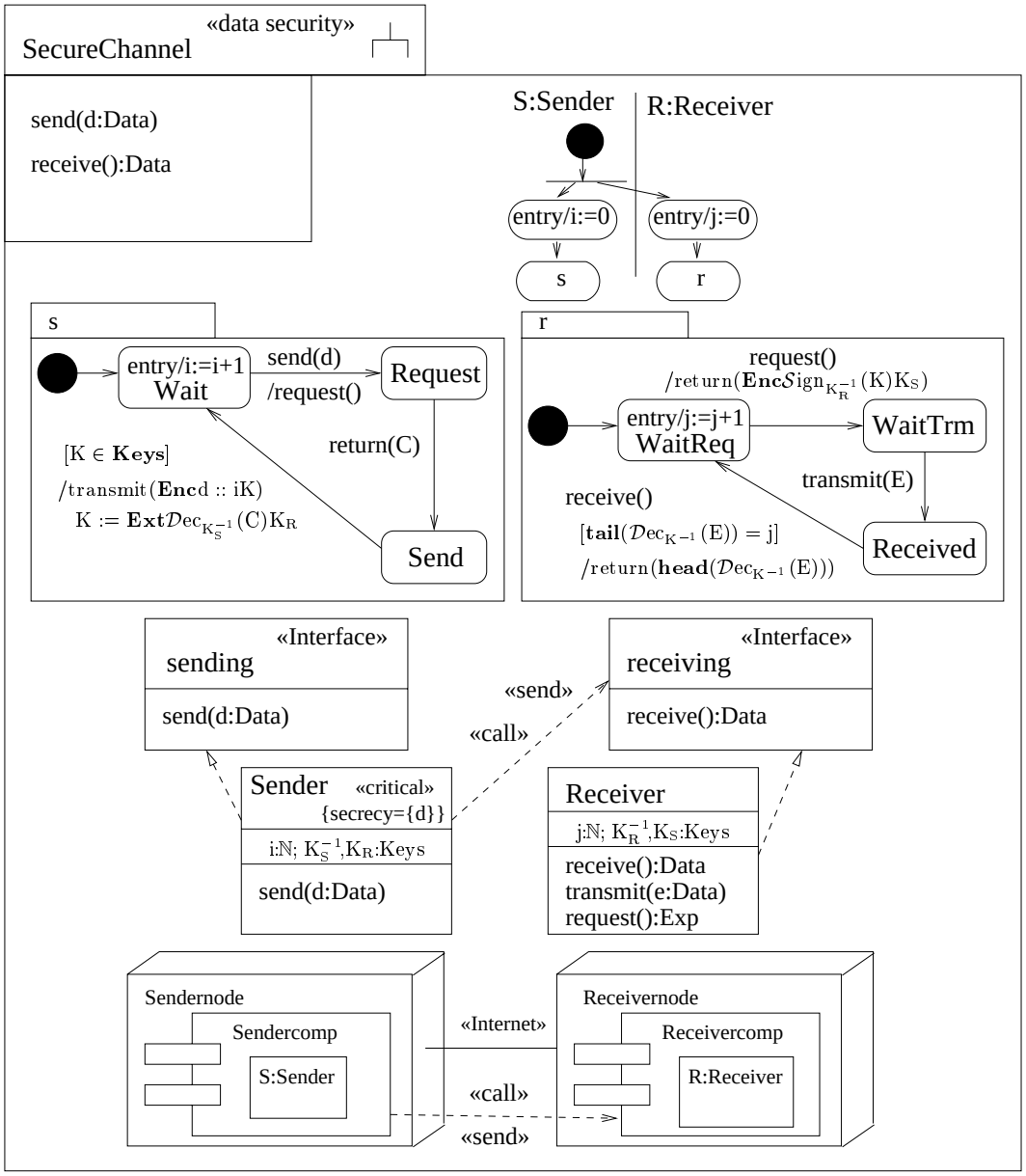

Figure 2: Example: Secure channel

Now assume that we would like to replace the abstract requirement that the communication should happen over an encrypted link by a more concrete specification of the encryption mechanism. By our pattern, our specification is transformed to the specification $\mathcal{C}'$ in Figure 2.

Since we only want to demonstrate the principle of developing a secure channel, we assume for simplicity that sender and receiver already know each other's public keys. The protocol then exchanges a symmetric key using those public keys, since encryption under symmetric keys is more efficient. We assume that the secret keys belonging to the public ones are kept secure. Also, we assume that the protocol is only run once with a given symmetric key.

The behaviour of the sender thus includes retrieving the signed and encrypted symmetric key from the receiver, checking the signature, and encrypting the data under the symmetric key, together with a sequence number (to avoid replay; we thus assume the natural numbers to be in Data: $\mathbb{N} \subseteq \mathsf{Data}$). The receiver first gives out the key with signature, and later decrypts the received data, checking the sequence number.

One can apply this pattern p in a formal way by considering the set S of subsystem derived from the subsystem in Figure 1 by renaming (that is, by substituting any message, data, state, subsystem, node, or component name n by a name m at each occurrence, in a way such that name clashes are avoided). Then f_p sends any subsystem $\mathcal{S} \in S$ to the subsystem derived from that given in Figure 2 by the same renaming. This gives us a presentation of f_p from which the definition of f_p on any UML specification can be derived as indicated above. Since one can show that the subsystem in Figure 2 is secure in a precise sense (see [Jür02]), this gives one a convenient way of reusing security engineering knowledge in a well-defined way within the development context.

3 Channel-oriented transformations

An important aspect of networked security-critical systems is the use of secure information channels. We propose to include abstract security requirements on the channels and relevant threats into the models and use them to induce pattern-based transformations.

For this purpose, we use the notation of the CASE tool AutoFocus [HMR+98], which is conceptually similar to a simplified subset of the UML, but particularly tailored for communicating systems. The AutoFocus tool features simulation, code generation, test sequence generation and formal verification of the modelled systems. In addition, it allows to define and automatically perform model transformations based on a transformation language [Sch01].

We explain our approach at the example of modelling a secure communication link between the parties A and B, as depicted in Figure 3. The models in Figure 3 represent AutoFocus System Structure Diagrams (SSDs), which correspond to UML component diagrams. SSDs show the communication links between the system components, depicted as directed arrows connecting filled and empty circles (output and input ports).

In early system design, we are mostly interested in abstract properties of the channels, which are represented as security tags (annotations) in Figure 3 (a). The tags are part of a security extension to AutoFocus [WW01]. The communication from A to B should be confidential (tag ≪secret≫), meaning only B can read A's messages, and authentic (tag ≪auth≫), meaning if B receives a message on the port connected to A, the message was actually sent by A. The same should hold for communication from B to A. In addition, the model includes the considered threat scenarios: ≪public≫ means the channels can be accessed by an intruder, and ≪node≫ means the components A and B themselves are assumed to be not manipulable.

For the implementation, the channel pair between A and B can now be replaced by an appropriate security pattern, thus transforming the model. A security pattern that supports confidential and authentic communication between A and B is for example the SSL protocol, which can therefore be applied in this case. The SSL pattern consists of SSL protocol components (a client and a server), and their behavioural specifications in form of state machines (which are omitted here). Figure 3 (b) shows the system model after the transformation. The channels between the SSL client and server components remain marked ≪public≫. The SSL protocol pattern should have been analysed separately that it really fulfills the required security properties, possibly using formal verification.

To be able to simulate and verify the system under attack, in an additional pattern transformation step the threat scenarios can be generated (see Figure 3 (c)). In the case of ≪public≫ channels this means replacing these by connections to an Intruder component, which consequently can access and manipulate the messages being transferred on these channels. The behaviour of the intruder can be modelled using state machines or Prolog rules. To achieve maximum security, usually a Dolev-Yao style intruder should be modelled, having the capability to analyse and create cryptographic messages in all possible ways as long as he knows the necessary keys.

4 Use within the software development process

Security patterns, as introduced above, supply us with a solution for problems instantiated through security requirements, needed in the system to develop. Whenever we identify a problem during software development, we may try to apply a suitable security pattern instead of the elaboration of a solution from the scratch.

A software development process is a set of activities needed to transform a user's requirements into a software system (see [JBR99]). The aim of a software development process is to form the requirements from the business model to the intended, error free software system. This overall process is distributed in several single steps and each step has its own related activities.

During the execution of the activities in different steps we gain a more fine-grained picture of the problem. With a pattern language, that is, a set of existing

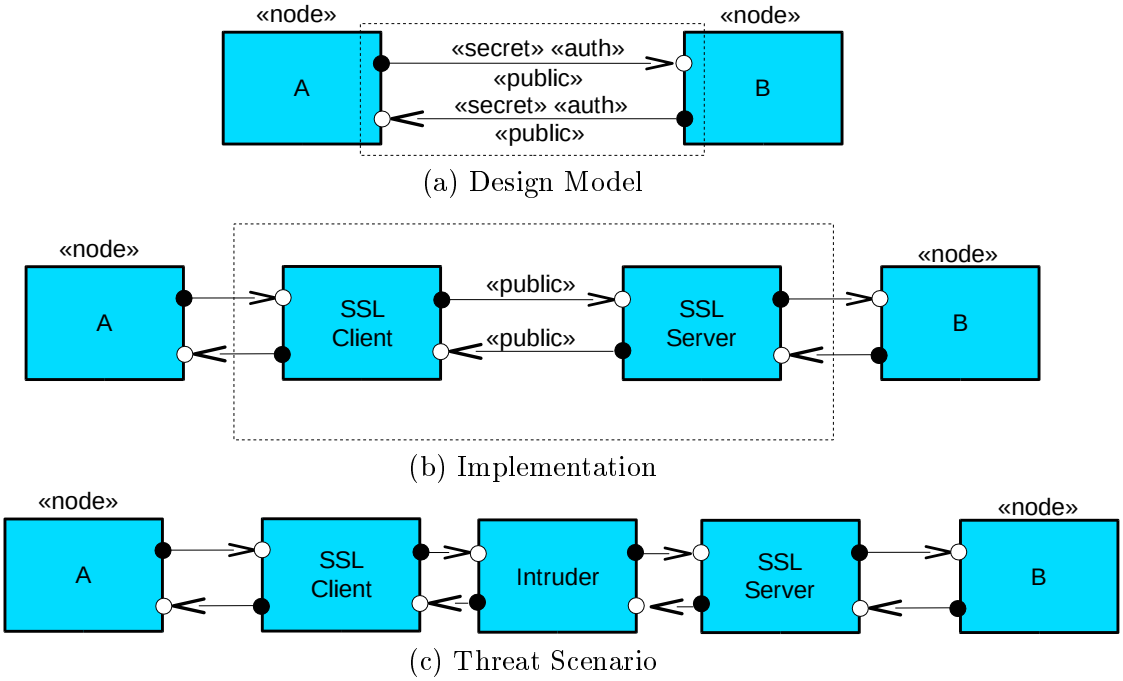

Figure 3: Example: secure communication using SSL

patterns, we can select a pattern suitable for the problem and the current development context, and this pattern provides us with a solution for the current development task.

Within a security software development process we can mainly distinguish two kinds of security patterns, the security analysis patterns and the security design patterns.

The former kind of patterns is used, as the name suggests, at the inception phase ([Kru00]) of the system development. These patterns help us to model security requirements because they provide us with solutions for domain specific analysis. A pattern in this case depicts a solution in form of a part of the analysis document, for example, it supplies us with a part of a protection profile for a bank accounting system, in which we only have to change or to replace specialities of our system, like a concrete protocol name.

The latter kind of patterns, security design patterns, are a specialization of design patterns (see [GHJV95, BMR+96]) for the security domain. Design patterns are mainly executed at the elaboration and construction phase in a software process. During the elaboration of the architecture, these patterns give us solutions in form of complete subsystems, layers, packages or depending classes and they describe structure as well as behaviour. Beside the design solutions, the patterns often come with

a uniform, easy adaptable implementation, and they give guidelines or environments for testing. An example for an security design pattern is the authorization pattern from [FP01].

As a third kind of security patterns one could consider process patterns. In contrast to analysis and design patterns they attempt to describe process parts during system development and do not provide us with an integrated solution. Security process patterns are not available yet. For this reason, we do not discuss process patterns here in detail. Information about process pattern in general is available in [Amb98], [GMP+01].

5 Related Work

Compared to research done using formal methods, relatively little work has been done more generally using software engineering techniques for computer security (for example [EM97], for an overview see [DS00]). [FH97] defines role-based access control rights from object-oriented use cases. [Rud98] presents an abstract formal model for protocols suitable for systematic top-down design. Work on security patterns can be found in [FY00, FP01, SR01, Fer]. Much work was done in the field of software engineering in general, like the waterfall model [Roy89], the V–Modell [WD99], the Spiral Model [Boe86], the Catalysis Apprach [DW98], or the Rational Unified Process [Kru00], just to name a few of them. Summaries of patterns are presented for example in [GHJV95] and [BMR+96]. A study of the combination of these two orthogonal topics hasn't be done yet and is sure an interesting topic for the future.

6 Conclusion and Further Work

We explained how to use patterns in the context of model-based development of security-critical systems. The approach integrates two complementary notations using an extension of the Unified Modeling Language (UML) for security-critical systems development, called UMLsec, and an industrial-strength CASE tool, AutoFocus, used to analyze the specifications mechanically, and to generate code or test-sequences. We also explained how to use our approach within an industrial software development process.

The approach presented here allows to introduce security patterns in a methodological and formal way. Because of the intuitive notations and the existence of substantial tool-support, it seems to be a significant step towards a more methodological way of developing secure systems, promising to reduce the number of vulnerabilities in implemented systems.

References

[Amb98] Scott W. Ambler. *Process Patterns: Building Large–Scale Systems Using Object Technology*. Cambridge University Press, 1998.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern–oriented Software Architecture*. John Wiley and Sons Ltd., 1996.

[Boe86] B. Boehm. A spiral model of software development and enhancement. In *ACM Sigsoft Software Engineering Notes, Vol. 11, No. 4*, 1986.

[DS00] P. Devanbu and S. Stubblebine. Software engineering for security: a roadmap. In *The Future of Software Engineering*, pages 227–239, 2000. Special Volume (ICSE 2000).

[DW98] D. F. D'Souza and A. C. Wills. *Objects, Components, and Frameworks With UML: The Catalysis Approach*. Addison Wesley Publishing Company, 1998.

[EM97] C. Eckert and D. Marek. Developing secure applications: A systematic approach. In *IFIP TC11 13th International Conference on Information Security*, pages 267 – 279. Chapman & Hall, 1997.

[Fer] E.B. Fernandez. Patterns for secure system design.

[FH97] E.B. Fernandez and J.C. Hawkins. Determining role rights from use cases. In *Workshop on Role-Based Access Control*, pages 121–125. ACM, 1997.

[FP01] E.B. Fernandez and R.Y. Pan. A pattern language for security models. In *Conference on Pattern Languages of Programs (PloP 2001)*, 2001. http://jerry.cs.uiuc.edu/plop/plop2001/ accepted_submissions/accepted-papers.html.

[FY00] E.B. Fernandez and X. Yuan. Semantic analysis patterns. In *Procs. of the 19th Int. Conf. on Conceptual Modeling (ER2000)*, pages 183–195, 2000.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GMP+01] M. Gnatz, F. Marschall, G. Popp, A. Rausch, and W. Schwerin. Towards a Living Software Development Process based on Process Patterns. In V. Ambriola, editor, *Proceedings of the Eight European Workshop on Software Process Technology*, number 2077 in LNCS, pages 182–202. Springer, 2001.

[HMR+98] F. Huber, S. Molterer, A. Rausch, B. Schätz, M. Sihling, and 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.

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison Wesley Longman, Inc., 1999.

[Jür01a] J. Jürjens. Transformations for introducing patterns – a secure systems case study. In *WTUML: Workshop on Transformations in UML (ETAPS 2001 Satellite Event)*, 2001.

[Jür01b] Jan Jürjens. Object-oriented modelling of audit security for smart-card payment schemes. In P. Paradinas, editor, *IFIP/SEC 2001 – 16th International Conference on Information Security*. Kluwer, 2001.

[Jür01c] Jan Jürjens. Towards development of secure systems using UML. In H. Hußmann, editor, *Fundamental Approaches to Software Engineering (FASE/ETAPS, International Conference)*, LNCS. Springer, 2001.

[Jür02] J. Jürjens. *Principles for Secure Systems Design*. PhD thesis, Oxford University Computing Laboratory, Trinity Term 2002.

[JW01] Jan Jürjens and Guido Wimmel. Formally testing fail-safety of electronic purse protocols. In *Automated Software Engineering (ASE 2001)*. IEEE Computer Society, 2001.

[Kru00] Philippe Kruchten. *The Rational Unified Process – An Introduction, Second Edition.* Addison Wesley Longman, Inc., 2000.

[Roy89] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *WESCON Technical Papers, Western Electronic Show and Convention, Los Angeles, Aug. 25-28, number 14, 1970*, pages 328–338. Reprinted in Proceedings of the Ninth International Conference on Software Engineering, Pittsburgh, PA, USA, ACM Press, 1989.

[Rud98] C. Rudolph. A formal model for systematic design of key establishment protocols. In *Australasian Conference on Information Security and Privacy (ACISP 98)*, LNCS, 1998.

[Sch01] B. Schätz. The ODL Operation Definition Language and the AutoFocus/Quest Application Framework AQuA Programming Processes. Technical Report TUM-I1101, Technische Universität München, 2001.

[SR01] M. Schumacher and U. Roedig. Security engineering with patterns. In *Procs. of PLoP 2001*, 2001.

[WD99] M. Wiemers W. Dröschel. *Das V–Modell 97.* Oldenbourg, 1999.

[WW01] G. Wimmel and A. Wißpeitner. Extended description techniques for security engineering. In M. Dupuy and P. Paradinas, editors, *Trusted Information: The New Decade Challenge.* IFIP, Kluwer Academic Publishers, 2001. Proceedings of SEC 2001 – 16th International Conference on Information Security.