Developing Secure Systems with UMLsec
From Business Processes to Implementation

Jan Jürjens*
Computing Laboratory, University of Oxford, GB

Abstract

In practice, security of computer systems is compromised most often not by breaking dedicated mechanisms (such as security protocols), but by exploiting vulnerabilities in the way they are employed.

We show how UML (the industry standard in object-oriented modelling) can be used to encapsulate rules of prudent security engineering to make them available to developers without a background in security. UML diagrams can be evaluated wrt. these rules, violations indicated and suggestions for modifications derived. We also show how to use transformations between UML models to introduce patterns by refinement.

This is version 28/9/01 of a paper at VIS'01. Please refer to www.jurjens.de/jan for the current version and more information.

1 Introduction

Many problems with security-critical systems arise from the fact that their developers do not always have a strong background in computer security. This is problematic since in practice, security is compromised most often not by breaking the dedicated mechanisms (such as encryption or security protocols), but by exploiting weaknesses in the way they are being used [And01]. Security mechanisms cannot be "blindly" inserted into a security-critical system, but the overall system development must take security aspects into account.

For instance, in the case of GSM security [Wal00], some examples for security weaknesses arising in this way are

- the failure to acknowledge limitations of the underlying physical security (misplaced trust in terminal identity; false base stations),

*jan@comlab.ox.ac.uk – http://www.jurjens.de/jan – Supported by the Studienstiftung des deutschen Volkes and the Computing Laboratory.

- an inadequate degree of flexibility to upgrade security functions over time and
- lack in the user interface wrt. communicating security-critical information (no indication to the user that encryption is on).

We aim to draw attention to such design limitations during the design phase, before a system is actually implemented.

More specifically, we use a formal core of the Unified Modeling Language (UML [RJB99], the industry-standard in object-oriented modelling) to encapsulate knowledge on prudent security engineering and thereby make it available to developers which may not be specialized in security. Object-oriented systems offer a very suitable framework for considering security due to their encapsulation and modularisation principles.

Currently a large part of effort both in implementing and verifying security-relevant specifications is wasted since these are often formulated imprecisely and unintelligibly [Pau98]. Being able to express security-relevant information in a widely used design notation helps alleviate this problem. Also, this approach can be usefully employed in the context of security certification (e.g. Common Criteria).

For space limitations, we present our results in the framework of a simplified part of UML (in its current version 1.3 [OMG99]), for more details cf. [Jür01e].

After presenting some background and related work in the following subsection, we summarise our use of UML in the next section.

We end with a conclusion and indicate future work.

2 UMLsec Walkthrough

We describe a simplified fragment of UMLsec. For more details and a (preliminary[1]) formal semantics cf. [Jür01e].

UML consists of diagram types describing different views on a system (an excellent introduction is [SP00]).

Use case diagrams describe typical interactions between a user and a computer system (or between different components of a computer system).

Activity diagrams can be used e.g. to model workflow and to explain use cases in more detail.

[1] Note that an official formal semantics for UML is still under development.

Class diagrams define the static structure of the system: classes with attributes and operations/signals and relationships between classes.

Interaction diagrams, which may be sequence diagrams or collaboration diagrams, describe interaction between objects via message exchange. Here we consider sequence diagrams; collaboration diagrams are very similar.

Statechart diagrams give the dynamic behaviour of an individual object: events may cause state in change or actions.

Package diagrams can be used to group parts of a system together into higher-level units.

Deployment diagrams describe the underlying physical layer, we use them to ensure that security requirements on communication are met by the physical layer.

UML offers rich extensions mechanisms in form of labels. These can be either stereotypes (written in double angle brackets such as «*stereotype*») or tag-value pairs (written in curly brackets such as $\{\mathsf{tag}, \mathsf{value}\}$). Using *profiles* or *prefaces* [CKM+99] one can give a specific meaning to model elements marked with these labels. Here we extend the approach in [Jür01e] to give an extension UMLsec of UML making use of such labels to express security requirements. Our results may be used with any process; security-critical systems usually require an iterative approach (e.g. the "Spiral") [And01].

We stress that the aspects not mentioned here (such as association and generalisation in the case of class diagrams) can and should be used in the context of UMLsec; they do not appear in our presentation simply because they are not needed to specify the considered security properties.

Since we are using a formal fragment of UML, we may reason formally, showing e.g. that a given system is as secure as certain components of it (following the simulatability approach, e.g. in [PW00, PW01]). This way, we can reduce security of the system to the security of the employed security mechanisms (such as security protocols). Similarly to work showing to which extent the formal approach to reasoning about security protocols is valid [AJ01], we aim to exhibit the conditions under which protocols can be used securely in the system context. Work on refining security properties [Jür01c] aids in verification applied within the design process.

As a running example we consider an Internet-based business application.

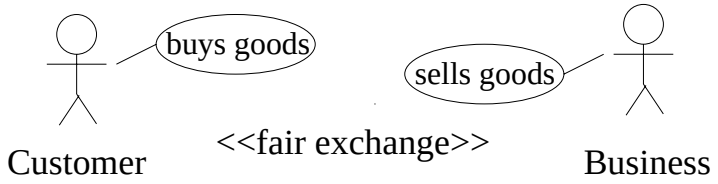

Figure 1: Use case diagram for business application

2.1 Security Requirements Capture with Use Case Diagrams

Use case diagrams describe typical interactions between a user and a computer system (or between different components of a computer system). We use them to capture requirements (in particular security requirements).

To start with our example, Figure 1 gives the use case diagram describing the situation to be achieved: a customer buys a good from a business. The semantics of the stereotype «*fair exchange*» is, intuitively, that the actions "buys good" and "sells good" should be linked in the sense that if one of the two is executed then eventually the other one will be (where these actions are specified on the next more detailed level of specification). This kind of diagram had not been considered in [Jür01e] yet; one can give it a formal semantics following [Ste01] which has to be omitted here.

2.2 Secure Business Processes with Activity Diagrams

Activity diagrams are especially useful to model workflow and to explain use cases in more detail.

Following our example, Figure 2 explains the use case in Figure 1 in more detail. To demonstrate the connection between this diagram and the one in Figure 1, we give two possible diagrams. Both are separated in two *swimlanes* describing activities of different parts of a system (here Customer and Business). Round boxes describe actions (such as Requestgood) and rectangular boxes describe the object flow (such as order[filled]). Horizontal bars (*synchronisation bars*) describe a required synchronisation between two different strands of activity. A diamond describes the merging of two strands of activities. The start state is marked by a full circle, the final state by a small full circle within a circle. In each diagram, two tag-value pairs {fair exchange, buys goods} and {fair exchange, sells goods} are used to mark certain actions. Now any such diagram fulfills the security requirement fair exchange given in the diagram in Figure 1 if for both actions marked with these tag-value pairs it is the case that if one of the actions is executed,

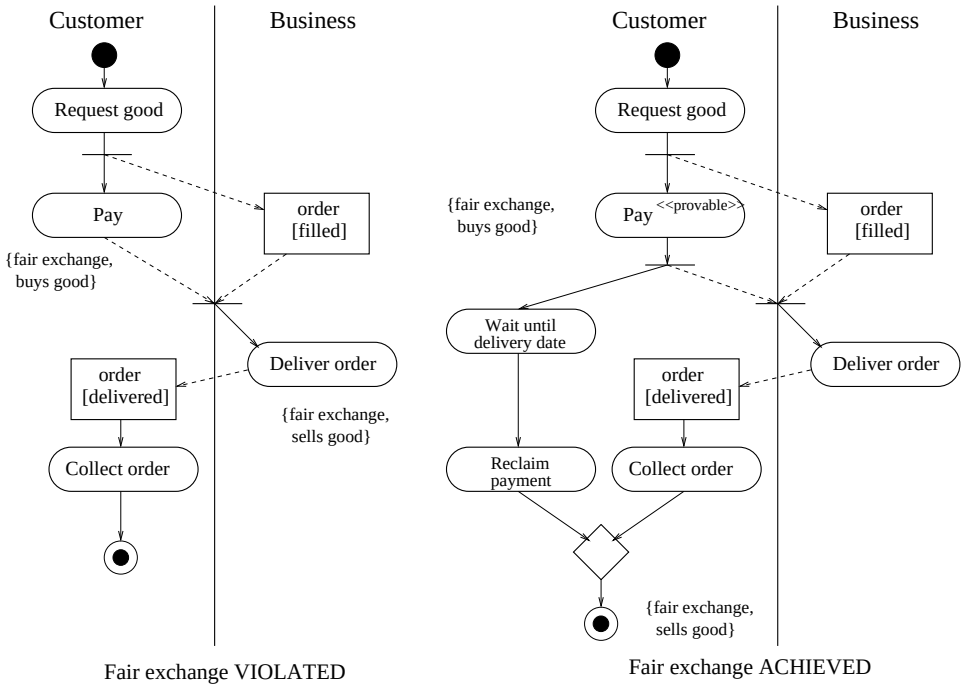

Figure 2: Activity diagram for business application

then eventually the other will be. As one can demonstrate on the level of the formal semantics (such as [BD00], which has to be omitted here), the left diagram does not fulfill this requirement because the Business may never Deliver order. Also one can show that the right diagram does fulfill the requirement (intuitively, because the customer may reclaim the payment if the order is undelivered after the scheduled delivery date), assuming that the customer is able to prove having made the payment (indicated by the stereotype «*provable*»), e.g. by following a fair exchange protocol (e.g. [ASW98]). The use of this kind of diagram in the context of UMLsec is also new here.

2.3 Preservation of sensitivity levels with class diagrams

Class diagrams define the static structure of a system.

As an example, Figure 3 gives a class-level description of a key generator (such as possibly used in the above business application example). The key generator offers the method newkey() which returns a Key for which it

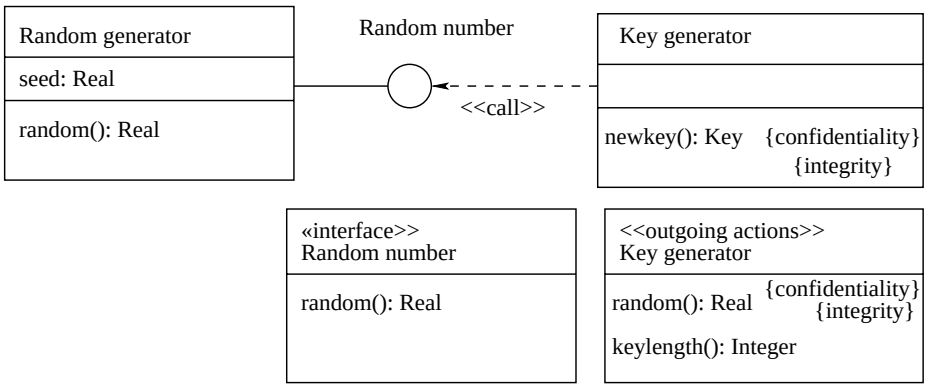

Figure 3: Class diagram for key generator

guarantees confidentiality and integrity.[2] On the other hand, it calls methods random() supposed to return a random number that is required to fulfill integrity and confidentiality (as specified in the model element stereotyped «*outgoing actions*» added in UMLsec). Here, this requirement is however not met by the random generator. As an example, consider the Homebanking-Computer-Interface (HBCI) specifications which in an early version (in the case of the RDH procedure) required the client system to perform key generation without specifying security requirements for the used random number generators. Omissions like this one can be detected using our modelling approach.

Here we assume that the attributes are fully encapsulated by the operations, i. e. an attribute of a class can be read and updated only by the class operations.

2.4 Security-critical message exchange with sequence diagrams

The importance of the underlying physical layer for the security of protocols has been exemplified e.g. in [Gol96]. Thus one should investigate the way security mechanisms (such as protocols) are employed in the system context [Aba00], which in practice offers more vulnerabilities than the mechanisms themselves [And01]. Also one sometimes has to adjust protocols to specific situations, e.g. for resource-bounded applications. As an example, [APS99]

[2] Here we use the convention that where the values are supposed to be boolean values, they need not be written (then presence of the label denotes the value true, and absence denotes false).

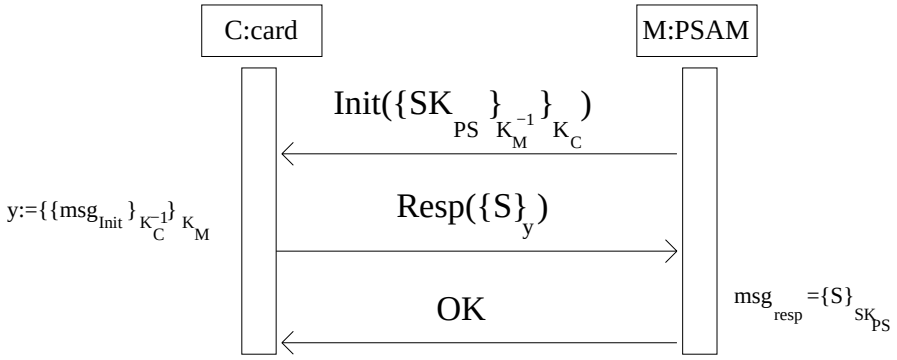

Figure 4: Sequence diagram for CEPS purchase transaction

adjusts TLS (the successor of the Internet protocol SSL) such that the server has to perform less computation. The adjusted protocol contains a drastic flaw (demonstrated in [Jür01c]).

As an example for a situation where the security of a system depends on unstated assumptions on the protocol environment, the security of CEPS transactions depends on the fact that in the immediately envisaged scenario (use of the card for purchases in shops) it is not feasible for the attacker to act as a relay between an attacked card (in a modified terminal) and an attacked terminal. However, this is not explicitly stated, and it is furthermore planned to use CEPS over the Internet [CEP01, Bus.Req.], where an attacker could easily act as such a relay (as demonstrated in [JW01a]).

Sometimes such assumptions are actually made explicit, if rather informally (such as "R_1 should never leave the LSAM in the clear." [CEP01, Tech. Sp. p.187]).

Being able to formulate precisely the assumptions on the protocol context, to reason about protocol security wrt. them, and to be able to communicate them to developers and clients would thus be useful.

In our UML-based approach, security protocols can be specified using message sequence charts. An example from [JW01a] is given in Figure 4. Assumptions on the underlying physical layer (such as physical security of communication links) can be expressed in implementation diagrams (cf. below), and the behaviour of the system context surrounding the protocol can be stated using statecharts and reasoned about as indicated below. In Figure 4, two objects in the system (a card C and a purchase security application module (PSAM) M) exchange messages that involve cryptographic operations such as public-key encryption and signing. The encryption of

the value d (or verification of signature) with the public key K is denoted by $\{d\}_K$, the decryption (or signing) with the private key K^{-1} by $\{d\}_{K^{-1}}$ (for simplicity we assume use of RSA type encryption). Thus M starts by sending to C the message Init with argument a value SK_{PS} (the session key created by the PSAM) signed with M's private key and encrypted with C's public key. C decrypts the received message with its private key and verifies the signature with M's public key. C uses the resulting session key to encrypt a secret S which is then sent back as an argument of the message Resp. M decrypts the received message using the session key and sends back the message OK.

Again one can make use of a formal semantics to reason about such protocol descriptions (for details cf. [Jür01e]).

2.5 Secure state change with statechart diagrams

Statechart diagrams give the dynamic behaviour of an individual object: events may cause state change or actions.

We demonstrate how this kind of diagram can be used in the context of UMLsec with an example involving guards (such as used in Java security) from [Jür01d], in Figure 5. Statechart diagrams consist of states (written as boxes) and transitions between them. The initial state is marked with a transition leading out from a full circle. Transitions between states can be labelled with events, conditions (in square brackets) and actions (preceded by a backslash). An event can be a method of the specified object called by another object (e.g. the transition labelled checkGuard in Figure 5), and the interpretation is that the transition labelled with an event is fired if the event occurs while the object is in the state from which the transition goes out. If a transition is labelled by a condition (formulated in the UML-associated object constraint language (OCL) or otherwise) then it is only fired if the condition is true at the respective moment. If a transition is labelled with an action (which can be to call a method of another object or to assign a value to a local variable), then this action is executed whenever the transition is fired.

Suppose that a certain micropayment signature key may only be used by applets originating at and signed by the site Finance (e.g. to purchase stock rate information on behalf of the user), but this access should only be granted five times a week.

The Java 2 security architecture allows the use of guard objects for this purpose which regulate access to an object [Gon99]. In our example, the guard object can be specified as in Figure 5 (where ThisWeek counts the

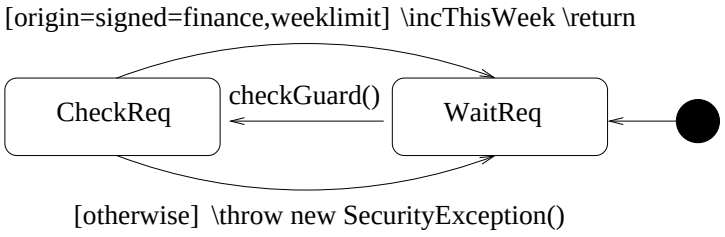

Figure 5: Statechart for guard object

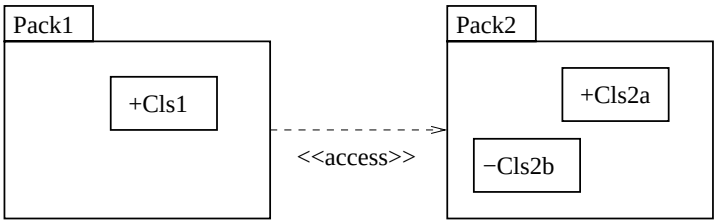

Figure 6: Package diagram with visibility

number of accesses in a given week and weeklimit is true if the limit has not been reached yet). One can then demonstrate using a formal semantics that certain access control requirements are enforced by the guards (for details cf. [Jür01d]).

2.6 Secure visibility with package diagrams

Package diagrams can be used to group parts of a system together into higher-level units. They play a security-relevant role in so far as one can specify visibility of objects within a package wrt. objects outside the package. In Figure 6, the package Pack1 contains the class Cls1 with public visibility. The package Pack2 contains the public class Cls2a and the private class Cls2b. Thus class Cls2b cannot be accessed by the class Cls1 even though the package Pack1 is assumed to access the package Pack2.

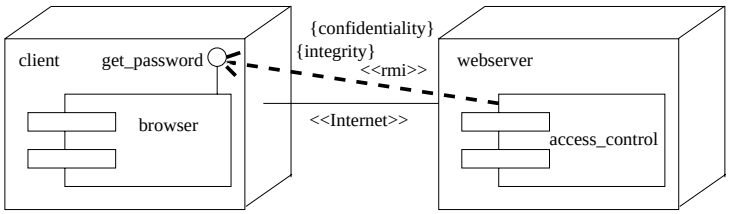

Figure 7: Deployment diagram for physical layer

2.7 Assumptions on the physical layer with deployment diagrams

Deployment diagrams describe the underlying physical layer; we use them to ensure that security requirements on communication are met by the physical layer.

For example, Figure 7 describes the physical situation underlying a system including a client system and a webserver communicating over the Internet. The two boxes labelled Client and Webserver denote *nodes* in the system (i. e. physical entities). The two rectangles labelled browser and access control represent system components residing on the respective nodes. The component browser has an interface offering the method get password. The component access control calls this method over the Internet via remote method invocation. The system specification requires the data exchanged in this invocation to be guaranteed confidentiality and integrity. However, these requirements are not met by the underlying communication link (as usual, this would be treated by using encryption).

3 Refinement for introducing Patterns

Patterns [GHJV95] encapsulate design knowledge of software engineers in the form of recurring design problems. They generally have four core elements: the *pattern name*, the *problem description*, the *solution*, and the *consequences*. We show how to use translations of UMLsec models as refinements for the introduction of patterns within the design process. This is to ensure that the patterns are introduced in a way that has previously 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.

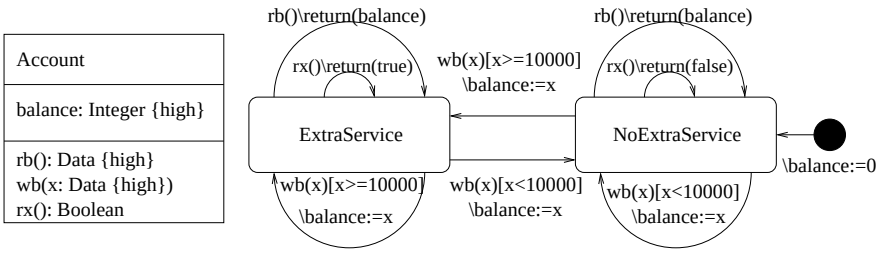

Figure 8: Entry in multi-level database

As an example, we consider the specification of an entry in a multi-level database from [Jür01e]. The object in Figure 8 does not preserve security (the operation rx() leaks information on the account balance, cf. [Jür01e]).

The Wrapper pattern Wrappers are a generic way to augment the security functionality of Commercial Off-The-Shelf (COTS) applications. Here we use wrappers to ensure that objects (that may not be under control of the developer) preserve security.

Thus the pattern takes any UML object model such as the one in Figure 8 and transforms it into a model by adding a wrapper object that controls its interaction with other objects. One such wrapper is given in Figure 9; it ensures that there can be no low read after a high write. The way it works is that instead of calling the original object directly, other objects call the wrapper object. This wrapper objects passes the calls on to the object to be wrapped (and gives back the return values to the clients), unless a non-high read method is called *after* a high write method has been called. The *consequence* of the pattern is that the composition of the original object and the wrapper object preserve security.

Note that refinement of system specification in the presence of security requirements can be subtle [Pfi99]. Motivated by this, [Jür01c] shows that a standard secrecy notion is preserved under refinement.

4 Related Work

This seems to be the first work applying all different UML diagrams to the specification of secure systems. It extends a line of research started in [Jür01e, Jür01b, Jür01d, Jür01a] making systematic use of the UML exten-

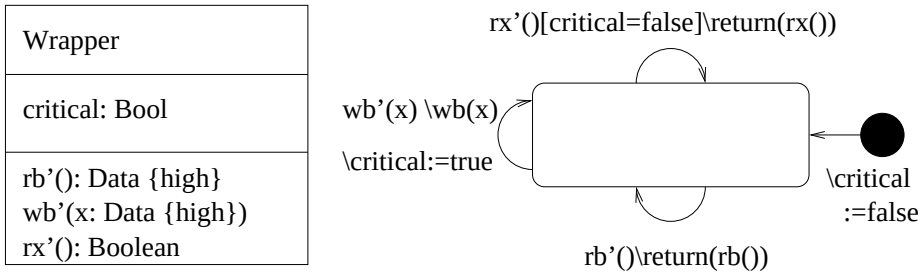

Figure 9: Wrapper Object

sion mechanisms to treat security aspects.

Previously, developers without a background in security depended on applying check-lists (such as in [Pom91]).

There has been much work on security using formal methods (cf. e.g. [Lot00, Jür01c, AJ01]; for an overview cf. [RSG+01]). Less work has been done using software engineering techniques (e.g. [DS00, WW01]).

For background literature on computer security cf. e.g. [Bis95, Brü97, Pfi99, Gol99, Eck00, And01].

5 Conclusion and Future Work

The aim of this work is to use UML to encapsulate knowledge on prudent security engineering and to make it available to developers not specialised in security by highlighting aspects of a system design that could give rise to vulnerabilities.

We showed how the extension UMLsec of UML (obtained using the UML extension mechanisms), can be used to express standard concepts from computer security. These definitions evaluate diagrams of various kinds and indicate possible weaknesses.

It would be very desirable to have tool-support that allows the developer to express security requirements, e.g. similar to [WP00].

Work towards this goal is being undertaken by giving translations (currently being mechanised) from UML into CSP which allow use of the model checker FDR2 to check security properties [BD00, Cri01].

As a case-study we plan to apply our approach to the design of PKIs [FH99].

Furthermore one may go beyond verification and make use of techniques more feasible in practice, such as specification-based testing (cf. e.g. [JW01b]).

Acknowledgements This idea for this line of work arose when doing security consulting for a project during a research visit with M. Abadi at Bell Labs (Lucent Tech.), Palo Alto, whose hospitality is gratefully acknowledged. It has also benefitted from discussions with D. Gollmann, A. Pfitzmann, B. Pfitzmann and others.

References

[Aba00] M. Abadi. Security protocols and their properties. In F. Bauer and R. Steinbrueggen, editors, *Foundations of Secure Computation*. IOS Press, 2000.

[AJ01] M. Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In *Theoretical Aspects of Computer Software (TACS '01)*, LNCS. Springer-Verlag, 2001.

[And01] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001.

[APS99] V. Apostolopoulos, V. Peris, and D. Saha. Transport layer security: How much does it really cost ? In *Conference on Computer Communications (IEEE Infocom)*, New York, March 1999.

[ASW98] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *IEEE Symposium on Security and Privacy*, 1998.

[BD00] C. Bolton and J. Davies. Activity graphs and processes. In *Integrated Formal Methods*, LNCS. Springer-Verlag, 2000.

[Bis95] J. Biskup. *Grundlagen von Informationssystemen*. Vieweg, 1995.

[Brü97] H. Brüggemann. *Spezifikation von objektorientierten Rechten*. Vieweg, 1997.

[CEP01] CEPSCO. Common Electronic Purse Specifications, 2001. Business Requirements vers. 7.0, Functional Requirements vers. 6.3, Technical Specification vers. 2.3, available from http://www.cepsco.com.

[CKM+99] S. Cook, A. Kleppe, R. Mitchell, B. Rumpe, J. Warmer, and A. Wills. Defining UML family members using prefaces. In Ch. Mingins and B. Meyer, editors, *TOOLS'99 Pacific*. IEEE Computer Society, 1999.

[Cri01] C. Crichton. UML statecharts and CSP, 2001. In preparation.

[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).

[Eck00] C. Eckert. *IT-Sicherheit - Konzepte, Verfahren, Protokolle*. R. Oldenbourg Verlag, 2000.

[FH99] D. Fox and P. Horster. Realisierung von Public Key-Infrastrukturen. pages 283–304. Vieweg Verlag, 1999.

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

[Gol96] D. Gollmann. What do we mean by entity authentication ? In *IEEE Symposium on Security and Privacy*, 1996.

[Gol99] D. Gollmann. *Computer Security*. J. Wiley, 1999.

[Gon99] Li Gong. *Inside Java 2 Platform Security – Architecture, API Design, and Implementation*. Addison-Wesley, 1999.

[Hor00] P. Horster, editor. *Systemsicherheit*. Vieweg Verlag, 2000. Conference proceedings.

[Hußß01] H. Hußmann, editor. *Fundamental Approaches to Software Engineering (FASE, 4th International Conference)*, volume 2029 of *LNCS*. Springer-Verlag, 2001.

[Jür01a] Jan Jürjens. Encapsulating rules of prudent security engineering. In *International Workshop on Security Protocols*, LNCS. Springer-Verlag, 2001. To be published.

[Jür01b] Jan Jürjens. Modelling audit security for smart-card payment schemes with UMLsec. In M. Dupuy and P. Paradinas, editors, *Trusted Information: The New Decade Challenge*, pages 93–108. International Federation for Information Processing (IFIP), Kluwer Academic Publishers, 2001. Proceedings of SEC 2001 – 16th International Conference on Information Security.

[Jür01c] Jan Jürjens. Secrecy-preserving refinement. In *Formal Methods Europe (International Symposium)*, volume 2021 of *LNCS*, pages 135–152. Springer-Verlag, 2001.

[Jür01d] Jan Jürjens. Secure Java development with UMLsec. In *I-NetSec 01 - First International IFIP TC-11 WG 11.4 Working Conference on Network Security*. Kluwer Academic Publishers, 2001. To appear.

[Jür01e] Jan Jürjens. Towards development of secure systems using UMLsec. In Hußmann [Hußß01], pages 187–200. Also OUCL TR-9-00 (Nov. 2000), http://web.comlab.ox.ac.uk/oucl/publications/tr/tr-9-00.html.

[JW01a] Jan Jürjens and Guido Wimmel. Security modelling for electronic commerce: The Common Electronic Purse Specifications. In *First IFIP conference on e-commerce, e-business, and e-government (I3E)*. Kluwer Academic Publishers, 2001.

[JW01b] Jan Jürjens and Guido Wimmel. Specification-based testing of firewalls. In *Andrei Ershov 4th International Conference "Perspectives of System Informatics" (PSI'01)*, LNCS. Springer-Verlag, 2001. To be published.

[Lot00] V. Lotz. Ein methodischer Rahmen zur formalen Entwicklung sicherer Systeme. In *[Hor00]*, 2000.

[OMG99] UML Revision Task Force, OMG. UML Specification 1.3. Available at http : //www.omg.org/uml, 1999.

[Pau98] L. Paulson. Inductive analysis of the Internet protocol TLS (transcript of discussion). In B. Christianson, B. Crispo, W.S. Harbison, and M. Roe, editors, *Security Protocols – 6th International Workshop*, number 1550 in LNCS, page 13 ff., Cambridge, UK, April 1998.

[Pfi99] A. Pfitzmann. Sicherheit in Rechnernetzen, 1999. Lecture Notes (in German).

[Pom91] K. Pommenering. *Datenschutz und Datensicherheit*. BI-Wissenschaftsverlag, 1991.

[PW00] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *7th ACM Conference on Computer and Communications Security*, 2000.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its applications to secure message transmissions. In *IEEE Symposium on Security and Privacy*, 2001.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[RSG+01] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.

[SP00] P. Stevens and R. Pooley. *Using UML*. Addison-Wesley, 2000.

[Ste01] P. Stevens. On use cases and their relationships in the Unified Modelling Language. In Hußmann [Huß01], pages 140–155.

[Wal00] M. Walker. On the security of 3GPP networks. In *Advances in Cryptology – EUROCRYPT*, volume 1807 of *LNCS*. Springer-Verlag, 2000.

[WP00] G. Wolf and A. Pfitzmann. Charakteristika von Schutzzielen und Konsequenzen für Benutzungsschnittstellen. *Informatik-Spektrum*, 23(3):173–191, 2000.

[WW01] G. Wimmel and A. Wißpeitner. Extended description techniques for security engineering. In *IFIP SEC*, 2001.