# Use Case Oriented Development of Security-Critical Systems[*]

Jan Jürjens and Gerhard Popp and Guido Wimmel
Department of Computer Science, Munich University of Technology
Bolzmanstraße 3, D-85748 Garching, Germany
phone: +49-89-28917832 fax: +49-89-28917307
{juerjens,popp,wimmel}@in.tum.de

February 20, 2003

**Abstract**

Since the connection of computers over the Internet and the expansion of distributed systems, they are confronted with more and more attacks. To counteract this circumstance, we have to consider security requirements from the beginning of the system development. In early phases of system development, it is common to use a two-part process for the elaboration of the application core and the functional specification in use cases. In this paper, we show an extension of this process for security-critical systems. We show a methodical concept for the development of security-critical systems and the modelling of security aspects in the application core with an extension of the Unified Modelling Language, here UMLsec. Furthermore, we introduce security use cases for the development of security aspects in conjunction with behavioural modelling.

## 1 Introduction

Developers and customers think both about objects (e.g. an account) and about tasks or activities (e.g. selling a book) in the early phases of a software development process. Whereas objects can be modelled in a data model, experience in the field of software engineering shows that the operational concept, i.e. specifying the behaviour with function signatures in the data model, does not meet the requirements on behavioural modelling. For this case, the application of *Use Cases* (see [JCJO92, JBR99]) for the modelling of the dynamic aspects is widely accepted.

The idea of a *Use Case Oriented Development* (see [Bre98, Bre02, Kru00, JBR99, DW98]) is a hybrid process with a functional and object-oriented view of the system. The core of the static system is modeled in a class diagram and the actions in use cases. The

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

integration of the use cases into the static core leads to an object-oriented view with static as well as dynamic aspects.

Whereas the object-oriented development of a data model for the core system is widely agreed upon, we want to consider a more specific definition of a use case than used normally (see [DW98, FS98, Bre98, Bre02]): A (joint action) use case is a joint interaction of a user with a system for the execution of a task.

We describe use cases in an informal way as plain text. The use case description has to answer the following questions:

- Which actor is involved?
- Which data or objects does the actor exchange with the system?
- Which classes of the core system are changed by the execution?
- Which expected behaviour does the system show?
- Which variation of the expected behaviour does exist?

After the elaboration of the use cases and the specification of the core system in a domain model, e.g. in a class diagram, we have the information that is needed for further development steps leading finally to the software product.

Since IT systems become more and more interconnected, they also become exposed to an increasing number of attacks. Thus, we additionally have to observe security aspects in the software development process. Security is a complex non-functional requirement which can only be guaranteed by the interaction of many parts in the system. Leaving security aspects to late stages and not considering them systematically (as it is often done) makes their integration extremely difficult and increases the potential for the final product to contain vulnerabilities.

In this paper, we present a methodology to integrate security aspects into the early phases of development, based on the hybrid process described above. Security requirements are captured within the requirements engineering process and represented both in the respective use case and data models.

Our approach is based on the secure systems extension UMLsec [Jür02b, Jür02a, Jür03] of the Unified Modelling Language, the de-facto industry standard in object-oriented modelling.

In Section 2 we give an overview over the overall two-part process for security development, before we explain in detail the security data modelling in Section 3 and the security use case modelling in Section 4. Section 5 summarizes the integration of the two models and introduces the next development steps. In Section 6 we consider related work before we give a short conclusion in Section 7.

## 2 Methodical Concept for Specifying Security-Critical Systems

The common process in the object-oriented software development establishes a class diagram for the data model. In this diagram, the static data of the domain is collectively modeled with the associations between the data structures. Whereas the class diagram represents only the data structures, the interaction sequences where the user is confronted with the system are initially described by text in the requirements specification (see [Som00]). After a closer specification for each function, use cases are elaborated, which describe, group, divide and extend the functions in a moderate way. After the elaboration of the domain model and the behaviour in use cases, they are connected to an object-oriented system specification.

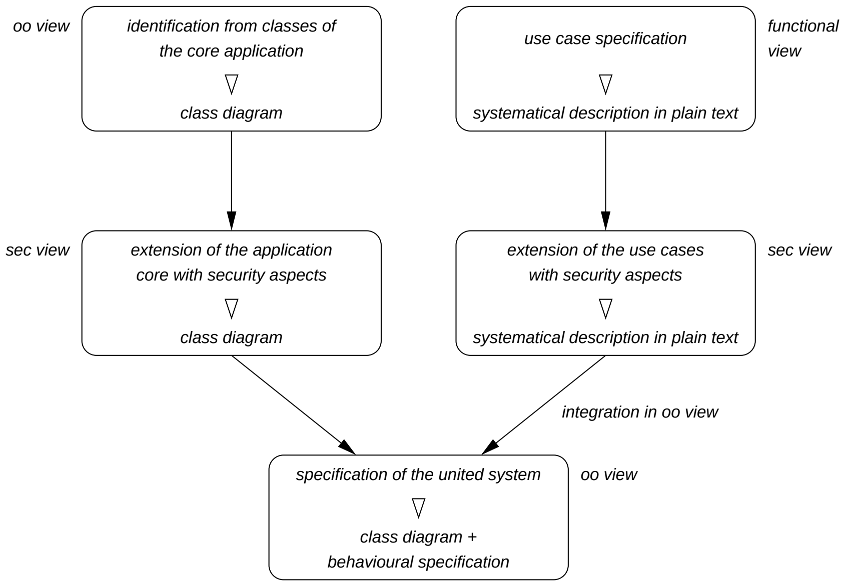
oo view
identification from classes of the core application
class diagram
use case specification
systematical description in plain text
functional view
sec view
extension of the application core with security aspects
class diagram
extension of the use cases with security aspects
systematical description in plain text
sec view
integration in oo view
specification of the united system
class diagram + behavioural specification
oo view

Figure 1: Methodical Concept for Specifying Security-Critical Systems

So far, security aspects are not considered. At most, the requirements specification describes, in a section about non-functional requirements, that the system has to be secure and explains in plain text which requirements should be met.

A systematic way for the integration of security aspects is that we extend both the domain model and the use cases with security information. After the definition of the class diagram, we extend, in a further step, the class diagram with security information. For example, we consider the classes, attributes and relations and decide whether they should be protected. In Section 3 the necessary extension process and the necessary UML-extension are explained.

The two-part process of structure and function development forces a security specification for the functional development as well. For this reason, in addition to extending the class diagram, we also extend the developed use cases with security information. We consider the input and output data as well as the interaction sequences and categorize them into critical and non-critical actions. How we elaborate the security aspects of the use cases is described in Section 4.

In Section 5, we outline the integration of the two security-enriched models into a common object-oriented view. Furthermore we describe the further development steps within the system development.

Figure 1 show the security enriched extended methodical concept. It is derived from [Bre98, Bre02], where security aspects were not considered.

## 3 Static Security Data Modelling with UMLsec

We recall the fragment of UMLsec [Jür02b, Jür02a, Jür03] which concerns static data modelling. UMLsec allows one to express security-related information within the diagrams in a UML system specification. The extension is given in form of a UML profile using the standard UML extension mechanisms. *Stereotypes* are used together with *tags* to formulate security requirements and assumptions on the system environment; *constraints* give criteria that determine whether the requirements are met by the system design.

Stereotypes define new types of modelling elements extending the semantics of existing types or classes in the UML metamodel. Their notation consists of the name of the stereotype written in double angle brackets ≪ ≫, attached to the extended model element. This model element is then interpreted according to the meaning ascribed to the stereotype.

One way of explicitly defining a property is by attaching a *tagged value* to a model element. A tagged value is a name-value pair, where the name is referred to as the *tag*. The corresponding notation is $\{tag = value\}$ with the tag name $tag$ and a corresponding $value$ to be assigned to the tag. Tags can define either data values (DataTags) or references to other model elements (ReferenceTags). If the value is of type Boolean, one usually omits $\{tag = false\}$, and writes $\{tag\}$ instead of $\{tag = true\}$.

| Stereotype | Base Class | Tags | Constraints | Description |
|---|---|---|---|---|
| secrecy | dependency | | | assumes secrecy |
| integrity | dependency | | | assumes integrity |
| high | dependency | | | high sensitivity |
| critical | object | secrecy, integrity, high | | critical object |

Figure 2: UMLsec stereotypes

Another way of adding information to a model element is by attaching *constraints* to refine its semantics.

Stereotypes can be used to attach tagged values and constraints as pseudo-attributes of the stereotyped model elements.

| Tag | Stereotype | Type | Multipl. | Description |
|---|---|---|---|---|
| secrecy | critical | String | * | secrecy of data |
| integrity | critical | String | * | integrity of data |
| high | critical | String | * | high-level message |

Figure 3: UMLsec tags

In Figure 2 we give the relevant fragment of the list of stereotypes from UMLsec, together with their tags and constraints. Figure 3 gives the corresponding tags.

We explain the use of the stereotypes and tags given in Figures 2 and 3.

**critical** This stereotype labels objects that are critical in some way, which is specified in more detail using the corresponding tags. The tags are {secrecy}, {integrity}, and {high}. The values of the first two are the names of expressions or variables (that is, attributes or message arguments) of the current object the secrecy (resp. integrity) of which is supposed to be protected. The tag {high} has the names of messages as values that are supposed to be generally highly sensitive.

**secure dependency** One may use the above stereotypes to enforce the condition of *secure dependency*, which ensures that the ⟪call⟫ and ⟪send⟫ dependencies between (interfaces of) objects respect the security requirements on the data that may be communicated across them, as given by the tags {secrecy}, {integrity}, and {high} of the stereotype ⟪critical⟫. More exactly, the constraint enforced is that if there is a ⟪call⟫ or ⟪send⟫ dependency from an object $C$ to an interface $I$ of an object $D$ then the following conditions are fulfilled.

- For any message name $n$ in $I$, $n$ appears in the tag {secrecy} (resp. {integrity} resp. {high}) in $C$ if and only if it does so in $D$.
- If a message name in $I$ appears in the tag {secrecy} (resp. {integrity} resp. {high}) in $C$ then the dependency is stereotyped ⟪secrecy⟫ (resp. ⟪integrity⟫ resp. ⟪high⟫).

If the dependency goes directly to another object without involving an interface, the same requirement applies to the trivial interface containing all messages of the server object.

As an example, Figure 4 shows a key generation subsystem instance with the requirement ⟪secure dependency⟫. The given specification violates this constraint, since Random generator and the ⟪call⟫ dependency do not provide the security levels for random() required by Key generator. More precisely, the constraint is violated, because the message $random$ is required to be of high level by Key generator (by the tag {high} in Key generator), but it is not guaranteed to be high level by Random generator (in fact there are no high messages in Random generator and so the tag {high} is missing).

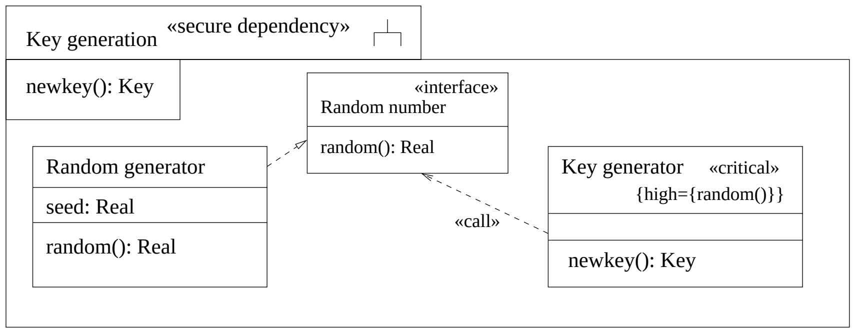

Figure 4: Key generation subsystem instance

## 4 Security Use Case Modelling

In the part about non-functional requirements of the requirements specification, the security requirements are described in plain text for the complete system. These security requirements affect both the data structure, as we have seen above, and the behavioural specification. In the use case specification the functions of the system have been elaborated to use cases. Now we have to map the overall security aspects to the use cases and extend them if necessary. After the mapping we can decide between the following three different use cases.

**The use case is not security-critical,** i.e. no given security requirement has to be implemented in the use case. For instance, in a web-based telephone system the user looks for a phone number in the online directory before he instructs the system to establish a connection. Since phone number directories are publicly accessible, the query is not security-critical.

**It is a security-critical use case,** it has to meet one or more security aspects from the non-functional, overall specification. In our web-based telephone system a user sends a request for a new telephone access. The following security aspects are adopted from the non-functional specification: The data of the request is confidential, its integrity has to be ensured, and repudiation has to be prevented.

**A new use case has to be added to the system** to fulfil the required security aspects. A necessary condition for guaranteeing confidentiality, integrity and non-repudiation is that the user must be logged on the system. For this reason we need an authentication mechanism. This function originally is not part of the functional list in the requirements specification and therefore not elaborated as a use case before security aspects are considered. Such new use cases have a《uses》relationship to the security-critical use cases that rely on the functionality they provide.

USE CASE: ESTABLISH A CONNECTION

**Risk associated with the actor**
The actor establishes a connection without any encryption, so there is no way to identify the sender and the connection can be eavesdropped.

**Security I/O Data**
*Input:* When establishing the connection, the caller selects the phone number of the destination. {*The phone number is critical, because it can be attacked.*}
*Output:* The system sends a message `ready` or `aborted` to the caller. {*The messages are critical, because they can be attacked.*}

**Modified Classes**
Connection

**Added Classes**
Key generation, Key storage, Crypt message

**Modified System Behaviour**
The system has to exchange the data over a secure channel. After the caller selects the phone number, the system generates a session key. The system exchanges the session key with the opposite system in an encrypted way. Afterwards, the system can encrypt and decrypt the data with the known key.

Figure 5: Use Case Extension: Establish a Connection

Whereas for non security-critical use cases there is nothing to do with regards to security, the recovery of security aspects for the second and third use case type may be complex. We suggest the application of a question catalogue for deriving the security aspects from the elaborated standard use cases. The following questions have to be answered for every security-critical and every new use case.

- Which risks are connected with the actor who is involved in or starts a use case?
- Which input and output data of the use case is security-critical when we analyze them with respect to security aspects like authenticity, integrity, confidentiality, availability and non-repudiation?
- Normally, the use case affects classes in the application core. Which classes have to be modified and how strongly does the security aspect affect the classes ? Do we have to change only attributes, or also behaviour of the classes, as far as behaviour is described in this early phase?
- How does the modified system behaviour look like? For example, we may have new error messages, abnormal terminations, reduced operability, etc.
- Do we have additional action sequences due the specification of security aspects?

The elaborated security aspects are described in plain text as well. In contrast to the textual description in the requirements specification this description is more structured. The structure is derived from the questions. In addition to the textual specification we can model the behaviour between the actor and the system in a sequence diagram (see [FS98]), enriched with the necessary security communication between the two participants. [FH97] suggests enriched sequence diagrams in which the security constraints are added to messages and are displayed in curly brackets.

In Figure 5 we show a security extension for the use case “Establish a Connection” over a telephone line. For the system, a non-functional requirement is given: the communication must be secure. In this example, all input and output data are critical, as described textually. Additionally, we have emphasised these security requirements through curly brackets as suggested for interaction diagrams in [FH97].

## 5 Outline of Integration and Further Development Steps

After the integration of the security aspects into both the application core and the use case specification, the two views will be connected before the design process can begin. We extend the class diagram with a class for every use case. This class later holds the behavioural sequence functions.

Furthermore, the mapping of the security aspects takes place in the integration step. Every use case has input and/or output data. If the data structure of this data is not part of the class diagram, we have to add the necessary new classes. Additionally, we have to map the security aspects of the input and output data. For each attribute or the whole class we have to test whether the security specification is the same as in the class diagram. Otherwise we have ignored some security aspects in the preceding static security data modelling or in the security use case modelling. This adjustment brings the opportunity to add missing security aspects and helps therefore to avoid important security problems in both the data model and the functional specification.

For the example given in Section 4 we need a data type for the phone number and one for the return message in the domain model, if the classes do not exist yet. The phone number, as well as the return message are classified as security-critical in Figure 5. For this reason, the added classes have to be classified critical, too. During the design the attributes of the phone number class and the message class must be specified according to the classification of the class. If e.g. the phone number class existed before the integration process started, then we have to check the class whether it is classified critical. If not, we have detected a disagreement and we have to adapt the domain model.

Further, in the integration we may realize that we need special security classes, for example for the implementation of a cryptographic algorithm. We can add them to the class diagram in this early stage and we therefore have the ability to bring more structure into the data model.

After the integration, the integrated class diagram and the enriched use case diagrams are the starting points for the analysis phase. In this phase, a refinement of the behavioural specification from the use cases takes place, the example sequences have to be enriched with all possible alternatives, and the messages within the system have to be considered more closely with respect to the communication between single objects, and not merely between the actor and one system object.

## 6 Related Work

E. B. Fernandez and J. C. Hawkins present in [FH97] an extension of use cases and interaction diagrams to develop distributed system architecture requirements. Among other non-functional requirements they introduce questions for the extension of use cases, too, but without relating the use cases to class diagrams. In contrast to our security use case modelling the paper presents a universal list of questions for requirements elaboration, like e.g. system communication load, fault tolerance, safety, real-time deadlines and security. For these reasons, the questions are less specific than in our context. Furthermore, the presented questions are aimed at the detection of use cases from scratch, whereas in contrast to this, our approach extends existing use cases, with exception of the third use case type given in 4. [BKL02] uses use cases for access control design. [FMMMP02] employs use cases in the context of secure database design and [BPRF99] in the context of security in healthcare. However, these works are mainly focused on application examples for use cases in security-critical systems, not on giving a methodology for their development or a concept for their integration with domain models.

## 7 Conclusion and Further Work

In this paper we suggested a methodology to integrate security aspects from the beginning in a system development process. Through the introduced methodical concept for specifying security-critical systems we consider security aspects in both the static domain model and the functional specification. For the elaboration of the functional aspects we introduced a question catalogue and for the domain model an appropriate UML-extension.

A worthwhile direction of further investigation would be to try to continue combining the approach of use case oriented development with ideas from model-driven development (also called model-driven architecture) by including further kinds of UMLsec diagrams in the approach, beyond the security-annotated class-diagrams used here. As mentioned in Section 4, these could be sequence diagrams, but other kinds of diagrams should also be considered. This would allow one to perform consistency checks between the use cases that are formulated and the other models constructed during the development of the system, or even to verify some behavioural requirements.

Furthermore, more work has to be done for the integration of security use cases in the field of testing and in the field of verifying that the security aspects described in the use cases are met in the system.

Another interesting research aspect is the integration of security use cases into criteria catalogues like the Common Criteria (see [Com99]). Thereby we want to look at the essential documents necessary for a security evaluation, which are connected with the security use cases.

## References

[BKL02] G. Brose, M. Koch, and K.-P. Löhr. Integrating access control design into the software development process. In *Integrated Design and Process Technology (IDPT)*, 2002.

[BPRF99] B. Blobel, P. Pharow, and F. Roger-France. Security and Design Based on a General Conceptual Security Model and UML. In *HPCN Europe*, 1999.

[Bre98] Ruth Breu. *Konzepte, Techniken und Methodik des objektorientierten Entwurfs – Ein integrierter Ansatz*. Habilitationsschrift, Technische Universität München, November 1998.

[Bre02] Ruth Breu. An Integrated Approach to Use Case Based Development, 2002. To be published.

[Com99] Common criteria for information technology security evaluation version 2.1. Technical report, August 1999. Avilable from http://www.commoncriteria.org/ docs/index.html.

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

[FH97] E. B. Fernandez and J. C. Hawkins. Extending use cases and interaction diagrams to develop system architecture requirements. Technical Report TR-CSE-97-47, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, Florida, 1997.

[FMMMP02] E. Fernández-Medina, A. Martínez, C. Medina, and M. Piattini. UML for the design of secure databases: Integrating security levels, user roles, and constraints in the database design process. 2002. In [JCF$^+$02].

[FS98] Martin Fowler and Kendal Scott. *UML Distilled – Applying the Standard Object Modelling Language*. Addison Wesley Longman Inc., 7th edition, June 1998.

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

[JCF$^+$02] J. Jürjens, V. Cengarle, E. Fernandez, B. Rumpe, and R. Sandner, editors. *Critical Systems Development with UML*, number TUM-I0208 in TUM technical report, 2002. UML'02 satellite workshop proceedings.

[JCJO92] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard. *Object-Oriented Software Engineering: A Use–Case Driven Approach*. Addison Wesley Longman, Inc., 1992.

[Jür02a] J. Jürjens. *Principles for Secure Systems Design*. PhD thesis, Oxford University Computing Laboratory, Trinity Term 2002.

[Jür02b] J. Jürjens. UMLsec: Extending UML for secure systems development. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *UML 2002 – The Unified Modeling Language*, volume 2460 of *LNCS*, pages 412–425, Dresden, Sept. 30 – Oct. 4 2002. Springer. 5th International Conference.

[Jür03] J. Jürjens. *Secure Systems Development with UML*. Springer, 2003. To be published.

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

[Som00] Ian Somerville. *Software Engineering*. Addison Wesley, 2000.