1

Introduction

A Need for Security

Modern society and modern economies rely on infrastructures for communication, finance, energy distribution, and transportation. These infrastructures depend increasingly on networked information systems. Attacks against these systems can threaten the economical or even physical well-being of people and organizations. There is widespread interconnection of information systems via the Internet, which is becoming the world's largest public electronic marketplace, while being accessible to untrusted users. Attacks can be waged anonymously and from a safe distance. If the Internet is to provide the platform for commercial transactions, it is vital that sensitive information (like credit card numbers or cryptographic keys) is stored and transmitted securely.

Problems

Developing secure software systems correctly is difficult and error-prone. Many flaws and possible sources of misunderstanding have been found in protocol or system specifications, sometimes years after their publication or use. For example, the observations in [Low95] were made 17 years after the well-known Needham–Schroeder authentication protocol had been published in [NS78]. Many vulnerabilities in deployed security-critical systems have been exploited, sometimes leading to spectacular attacks. For example, as part of a 1997 exercise, an NSA hacker team demonstrated how to break into US Department of Defense computers and the US electric power grid system, among other things simulating a series of rolling power outages and 911 emergency telephone overloads in Washington, DC, and other cities [Sch99a]. While there are of course many more recent examples of security breaches, this particular example also shows that there is more to be concerned about than website defacements and creditcard misuse.

Computer breaches do significant damage, as a study by the Computer Security Institute shows: Ninety percent of the respondents detected com-

puter security breaches within the last 12 months. Forty-four percent of them were willing and able to quantify the damage. These 223 firms reported $455,848,000 in financial losses [Ric03].

Causes

Firstly, enforcing security requirements is intrinsically subtle, because one has to take into account the interaction of the system with motivated adversaries that act independently. Thus security mechanisms, such as security protocols, are notoriously hard to design correctly, even for experts. Also, a system is only as secure as its weakest part or aspect.

Secondly, risks are very hard to calculate: security-critical systems are characterized by the fact that the occurrence of a successful attack at one point in time on a given system dramatically increases the likelihood that the attack will be launched subsequently at another system. This problem is made worse by the existence of the Internet as a mass communication medium that is currently largely uncontrolled and enables fast and anonymous distribution of information on successful exploits.

Thirdly, many problems with security-critical systems arise from the fact that their developers, who employ security mechanisms, do not always have a strong background in computer security. This is problematic since, in practice, security is compromised most often not by breaking dedicated mechanisms such as encryption or security protocols, but by exploiting weaknesses in the way they are being used [And01]: According to A. Shamir, the Israeli state security apparatus is not hampered in its investigations by the fact that suspects may use encryption technology that may be virtually impossible to break. Instead, other weaknesses in overall computer security can be exploited [Sha99]. As another example, the security of Common Electronic Purse Specifications (CEPS) [CEP01] transactions depends on the assumption that it is not feasible for the attacker to act as a relay between an attacked card and an attacked terminal. However, this is not explicitly stated, and it is furthermore planned to use the CEPS over the Internet, where an attacker could easily act as such a relay. This is investigated in Sect. 5.3. As a last example, [Wal00] attributes the failures in the security of the mobile phone protocol GSM among other reasons to the failure to acknowledge limitations of the underlying physical security, such as misplaced trust in terminal identity and the possibility to create false base stations.

Thus it is not enough to ensure correct functioning of security mechanisms used. They cannot be "blindly" inserted into a security-critical system, but the overall system development must take security aspects into account in a coherent way [SS75]. More specifically, one can say that "those who think that their problem can be solved by simply applying cryptography don't understand cryptography and don't understand their problem" (R. Needham). In fact, according to [Sch99a], 85% of Computer Emergency Response Team

(CERT) security advisories [CER] could not have been prevented just by making use of cryptography. Thus, given the current state of software security, just using encryption to protect communication still leaves most weaknesses unresolved, and has been compared to using an armored car to deliver credit card information "from someone living in a cardboard box to someone living on a park bench" [VM02]. Building trustworthy components does not suffice, since the interconnections and interactions of components play a significant role in trustworthiness [Sch99a].

Lastly, while functional requirements are generally analyzed carefully in systems development, security considerations often arise after the fact. Adding security as an afterthought, however, often leads to problems [Gas88, And01]. Also, security engineers get little feedback about the secure functioning of their products in practice, since security violations are often kept secret for fear of harming a company's reputation.

It has remained true over the last 30 years since the seminal paper [SS75] that no coherent and complete methodology to ensure security in the construction of large general-purpose systems exists yet, in spite of very active research and many useful results addressing particular subgoals [Sch99a], as well as a large body of security engineering knowledge accumulated [And01]. Such a methodology would allow the computer security engineer to construct a system in a way similar to how a civil engineer would build a bridge. In contrast, today ad hoc development leads to many deployed systems that do not satisfy important security requirements. Thus a sound methodology supporting secure systems development is needed.

Traditional Approaches

In practice, the traditional strategy for security assurance has been "penetrate and patch": It has been accepted that deployed systems contain vulnerabilities. Whenever a penetration of the system is noticed and the exploited weakness can be identified, the vulnerability is removed. Sometimes this is supported by employing friendly teams trained in penetrating computer systems, the so-called "tiger teams" [Wei95, McG98].

For many systems, this approach is not ideal: Each penetration using a new vulnerability may already have caused significant damage, before the vulnerability can be removed. For systems that offer strong incentives for attack, such as financial applications, the prospect of being able to exploit a discovered weakness only once may already be enough motivation to search for such a weakness. System administrators are often hesitant to apply patches, *especially* in critical systems, since applying the patch may disrupt the service [And01]. Having to create and distribute patches costs money and leads to loss of customer confidence. Patches may contain security threats themselves, such as the FunLove virus in a Microsoft hotfix distributed in April 2001 [Mic01, The01].

It would thus be preferable to consider security aspects more seriously in earlier phases of the system life-cycle, before a system is deployed, or even implemented, because late correction of requirements errors costs up to 200 times as much as early correction [Boe81].

The difficulty of designing security mechanisms correctly has motivated quite successful research using mathematical concepts and tools to ensure correct design of small security-critical components such as security protocols, including [MCF87, BAN89, Mea91, Low96, Pau98b, AG99]. The goal is to establish crucial requirements on the specification level through formalization and proof, which may be mechanically assisted or even automated. Note that it is not possible to actually prove a system secure in an absolute sense: Proofs can only be performed with respect to models which are necessarily abstractions from reality. Attackers can always try to go beyond the limitations of a given model to still attempt an attack. Nevertheless, a model-based security analysis is useful, because certain attacks can be prevented and the required effort for successful attacks increased. Also, often problems with a specification are detected just by trying to make it sufficiently precise for formal analysis [Gol03a].

Unfortunately, due to a perceived high cost in personnel training and use, formal methods have not yet been employed very widely in industrial development [Hoa96, Hei99, KK04]. To increase industry acceptance in the context of security-critical systems, it would be beneficial to integrate security requirements analysis with a standard development method, which should be easy to learn and to use [CW96]. Also, security concerns must inform every phase of software development, from requirements engineering to design, implementation, testing, and deployment [DS00b].

Some other challenges for using sound engineering methods for secure systems development exist. Currently a large part of effort both in analyzing and implementing specifications is wasted since these are often formulated imprecisely and unintelligibly, if they exist at all [Pau98a]. If increased precision by use of a particular notation brings an additional advantage, such as automated tool support for security analysis, this may however be sufficient incentive for providing it. Since software developers cannot expect to learn a particular formal method to do this, because of limited resources in time and training, one needs to instead use the artifacts that are at any rate constructed in industrial software development. Examples include specification models in the Unified Modeling Language (UML). Also, the boundaries of the specified components with the rest of the system need to be carefully examined, for example with respect to implicit assumptions on the system context [Gol00, Aba00]. Lastly, a more technical issue is that formalized security properties are not always preserved by refinement, which is the so-called *refinement problem* [RSG+01]. Since an implementation is necessarily a refinement of its specification, an implementation of a secure specification may, in such a situation, not be secure, which is clearly undesirable. Also, it hinders the use of stepwise development, where one starts with an abstract specification and refines it in several steps

to a concrete specification which is implemented, allowing mistakes to be detected early in the development cycle, and thus leading to considerable savings: Without preservation of security by refinement, developing secure systems in a stepwise manner requires one to redo the security analysis after each refinement step. Hence, we need formalizations of security requirements that are indeed preserved under refinement.

Model-Based Security Engineering with UML

Towards a solution of the problems mentioned in the previous sections, we propose an approach for model-based security engineering using the Unified Modeling Language (UML) [RJB99, UML03]. We explain our motivation firstly for choosing this kind of approach and secondly for using the UML notation.

Generally, in model-based development, as represented in Fig. 1.1, the idea is to first construct a model of a system, which should be as close to human intuition as possible and is typically relatively abstract. In a second step, the implementation is derived from the model: either automatically using code generation, or manually, in which case one can still generate test sequences from the model to establish conformance of the code regarding the model. The goal is to increase the quality of the implemented code while keeping the implementation cost and the time-to-market bounded.

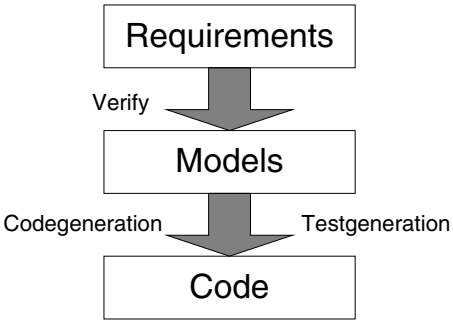

Fig. 1.1. Model-based development

For security-critical systems, this approach allows one to consider security requirements from early of in the development process, within the development context, and in a seamless way throught the development cycle. Using the model-based approach, one can, firstly, establish that the system fulfills the relevant security requirements on the design level, by analyzing the model. Secondly, one can check that the code is also secure by generating test sequences from the model.

UML now offers an, as such probably unprecedented, opportunity as a notation for a high-quality model-based development of security-critical systems that is feasible in an industrial context:

- As the de facto standard in industrial modeling notations, a large number of developers are trained in UML, and this number is still growing because UML is widely taught at universities. Thus, a UML specification may already be available for security analysis, or less difficult to obtain than other notations.
- UML provides graphical, intuitive description techniques with multiple views of a system through different kinds of diagrams. It offers standard extension mechanisms (such as stereotypes, tags, constraints, and profiles) which one can use to tailor the notation to a specific application domain.
- Compared to previous notations with a user community of comparable size, UML is relatively precisely defined, since [UML03] defines syntax and semantics of the UML notation in a relatively high degree of detail, although not entirely formal.
- A variety of tools exist that provide the basic functionality required to use UML, such as the drawing of UML diagrams.

Note that although UML was developed to model object-oriented systems, one may use it just as well to analyze systems that are not object-oriented, by thinking of objects as components and not making use of object-oriented features, such as inheritance.

To exploit this opportunity, however, some challenges remain: One needs to adapt the UML to the application domain of security-critical systems and advance its correct use in this application domain. One has to develope advanced tool support for secure systems development with UML, such as automatic analysis of UML specifications with respect to security requirements. This requires dealing with conflicts between flexibility and unambiguity in the meaning of UML models.

This book aims to contribute to overcoming these challenges. More specifically, it presents the UML extension UMLsec for secure systems development. The UMLsec extension:

- allows one to evaluate UML specifications for security weaknesses on the design level,
- encapsulates established rules of prudent security engineering in the context of a widely known notation, and thus makes them available to developers who may not be security experts,
- allows the developer to consider security requirements from early on in the system development process, and
- involves little additional overhead, since the UML diagrams can serve as system documentation, which is always desirable to have, and sometimes required (for example, for security certifications).

1.1 Overview

The UMLsec extension

We present an extension of the UML [UML03] for secure systems development, called UMLsec. Recurring security requirements (such as secrecy, integrity, and authenticity) are offered as specification elements by the UMLsec extension. The properties are used to evaluate diagrams of various kinds and to indicate possible vulnerabilities. One can thus verify that the stated security requirements, if fulfilled, enforce a given security policy. One can also ensure that the requirements are actually met by the given UML specification of the system. UMLsec encapsulates knowledge on prudent security engineering and thereby makes it available to developers who may not be experts in security.

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, by referring to a precise semantics mentioned below.

The extension has been developed based on experiences on the model-based development of security-critical systems in industrial projects involving German government agencies and major banks, insurance companies, smart card and car manufacturers, and other companies. Note that an extension of UML to an application domain such as security-critical systems that aims to include requirements from that application domain as stereotypes, as opposed to just adding specific architectural primitives, can probably never be fully complete: It would then have to incorporate all existing design knowledge on security-critical computing systems, which fills countless books. Therefore, here we focus on providing a core profile that includes the main security requirements. We expect this to be extended with additional, more specific concepts (for example, from more specialized application domains such as mobile security).

We list the requirements on a UML extension for secure systems development and discuss how far our extension meets these requirements. We explain the details of the extension by means of examples, demonstrate how to employ the extension for enforcing established rules of secure systems design and show how to use UMLsec to apply security patterns.

Applications

To validate our approach using UMLsec for secure systems development, we investigate the degree to which it is suitable for enforcing established rules of prudent security engineering. We consider several case studies:

- We demonstrate stepwise development of a security-critical system with UMLsec as the example of a secure channel design, together with a mathematically precise verification.

- We uncover a flaw in a variant of the handshake protocol of the Internet protocol TLS proposed in [APS99], suggest a correction, and verify the corrected protocol.
- We apply UMLsec to a security analysis of CEPS, a candidate for a globally interoperable electronic purse standard. We discover flaws in the two central parts of the specifications (the purchase and the load protocol), propose corrections, and give a verification of the corrected versions.
- We show how to use UMLsec to correctly employ advanced Java 2 security concepts such as guarding objects in a way that allows formal verification of the specifications.
- We also report on a project with a major German bank, where we applied our ideas about model-based development of security-critical systems to a web-based banking application.

There are further applications in industrial development projects which because of space limitations can only shortly be mentioned.

Tool Support

For the ideas that we present in this book to be of benefit in practice, it is important to have advanced tool support to assist in using them. We present the necessary background to construct such tool support, as well as the tool suite that has been developed [JSA+04]. The developed tools can be used to check the constraints associated with UMLsec stereotypes mechanically, based on XMI output of the diagrams from the UML drawing tool in use, and using sophisticated analysis engines that as model-checkers and automated theorem provers. For this, the developer creates a model using a UML drawing tool capable of XMI export and stores it as an XMI file. The file is imported by the UMLsec analysis tool (for example, through its web interface) which analyses the UMLsec model with respect to the security requirements that are included. The results of the analysis are given back to the developer, together with a modified UML model, where the weaknesses that were found are highlighted.

We also explain a framework for implementing verification routines for the constraints associated with the UMLsec stereotypes. The goal is that advanced users of the UMLsec approach should be able to use this framework to implement verification routines for the constraints of self-defined stereotypes. In particular, the framework includes the UMLsec tool web interface, so that new routines are also accessible over this interface. The idea behind the framework is to provide a common programming framework for the developers of different verification modules. A tool developer should be able to concentrate on the implementation of the verification logic and not be required to implement the user interface.

Furthermore, we present research on linking the UMLsec approach with the automated analysis of security-critical data arising at runtime. Specifically, we present research on the construction of a tool which automatically checks

the SAP R/3 configuration for security policy rules, such as separation of duty. The permissions are given as input in an XML format through an interface from the SAP R/3 system, the rules are formulated as UML specifications in a standard UML CASE tool and output as XMI, as part of the UMLsec framework mentioned above. The tool then checks the permissions against the rules using an analyzer written in Prolog. Because of its modular architecture and its standardized interfaces, the tool can be adapted to check security constraints in other kinds of application software, such as firewalls or other access control configurations.

As noted, for example, in [Fow04], the ultimate benefit in software development is not "pretty pictures", but the running implementation of a system. We present some approaches for linking UML models to implementations, such as model-based testing. The aim is to ensure that the benefits gained from the model-based approach on the level of the system model, such as increased confidence in satisfaction of critical requirements, actually carry over to the implemented system.

To provide tool support for analyzing UMLsec models with respect to the security properties included as predefined constraints, tool developers need to formulate the properties in a mathematically precise way. This is only possible if the UML specification they refer to also has a mathematically precise meaning. In particular, this concerns the behavioral aspects, since many security requirements refer to the system behavior. For this goal, we provide a precise execution semantics for a simplified part of UML using so-called *UML Machines*. These are based on Abstract State Machines which give a mathematically rigorous yet rather flexible framework for modeling computing systems [Gur95]. *UML Machine Systems* allow us then to build up UML Machine specifications in a modular way and to treat external influences on the system beyond the planned interaction, such as attacks on insecure communication links. This allows a rather natural modeling of potential adversary behavior and to define different kinds of adversary strengths. On this basis, important security requirements such as secrecy, integrity, authenticity, and secure information flow are defined. To support stepwise development, we show secrecy, integrity, authenticity, and secure information flow to be *preserved* under refinement. Because of the modular way UML Machines are defined, they give a formal framework for formally analyzing security-critical systems in their own right, independently of the UML notation.

Based on this, we provide a precise semantics for a simplified core of UML that allows one to use a more focussed kind of UML subsystems to group together several diagrams. The precise semantics for a restricted version of subsystems incorporates the precise semantics of the diagrams contained in a subsystem in a way that allows them to interact by exchanging messages. The statechart semantics which is part of it is based on part of the statechart semantics from [BCR00]. The motivation is to concentrate on a core of UML for which it is feasible to construct and use advanced tool support. The UMLsec case studies mentioned above demonstrate that our choice of a subset

of UML is useful. We also consider some helpful concepts, such as consistency between diagrams, different kinds of refinement of and equivalence between UML specifications, and the use of rely-guarantee specifications.

Via UML Machines and UML Machine Systems we make use of the presented treatment of security-critical systems. In particular, UML specifications can be evaluated using the attacker model, which incorporates the possible attacker behaviors, to find vulnerabilities.

1.2 Outline

Here is an outline of the following chapters:

Chapter 2: For a short "walk-through" to highlight the UMLsec approach, we consider a simplified model of an Internet-based business application as a running example.

Chapter 3: Some background information is recalled that is needed in the remainder of the book.

Chapter 4: After discussing requirements on a UML extension for secure systems development, we present the UMLsec profile. We show how to formulate security requirements on a system and security assumptions on the underlying layer in UMLsec. It is explained how to evaluate the system specification against the security requirements, by referring to the precise semantics sketched in Chap. 3. We demonstrate how to employ the extension for enforcing established rules of secure systems design and how to use UMLsec in order to apply security patterns.

Chapter 5: At the example of a secure channel design, we demonstrate stepwise development of a security-critical system with UMLsec. We uncover a flaw in a variant of the handshake protocol of the Internet protocol TLS proposed in [APS99], suggest a correction, and verify the corrected protocol. Furthermore, we use UMLsec for a security analysis of CEPS, a candidate for a globally interoperable electronic purse standard. We discover three flaws in the two central parts of the specifications, propose corrections, and give a verification. We show how to use UMLsec to correctly employ advanced Java 2 security concepts such as guarded objects.

Chapter 6: The necessary background for developing tool support for UMLsec is explained. We present a tool which automatically checks a UMLsec model with respect to the security requirements associated with the UMLsec stereotypes, based on XML output of industrial UML drawing tools. A framework is presented which allows advanced users to conveniently include verification routines for the constraints of self-defined stereotypes. As an instance of this framework, we present a tool which links the UMLsec approach with the automated analysis of security-critical data arising at runtime, such as permissions in SAP R/3 systems. We explain approaches for linking UML models to implementations, such as model-based testing.

Chapter 7: We introduce *UML Machines* and *UML Machine Systems* and define notions of refinement and rely-guarantee specifications. We explain how we use UML Machines to specify security-critical systems. In particular, we give definitions for secrecy, integrity, authenticity, and secure information flow, and give equivalent internal characterizations to simplify verification. We show secrecy, integrity, authenticity, and secure information flow to be preserved under refinement.

Chapter 8: We use UML Machines and UML Machine Systems to give a precise semantics for a simplified part of UML. This semantics is used to give consistency conditions for different diagrams in a UML specification. Also, we define notions of refinement and behavioral equivalence, and investigate structural properties, such as substitutivity. We consider rely-guarantee properties for UML specifications and their structural properties.

Chapter 9: An account of other approaches to security engineering with a similarly sound basis is given.

Chapter 10: We conclude with a critical evaluation of the approach we presented and an outlook on future developments.

Appendices: We explain how to adjust our approach to the upcoming version UML 2.0, give the formal definition of UML Machine rules and the proofs for the statements from Chaps. 5, 7, and 8.

1.3 How to Use this Book

Being the first book on the topic of secure systems development with UML, this book was written with two audiences in mind:

- researchers and graduate students interested in UML, computer-aided software engineering or formal methods, and IT security, who may use the book as background reading for their own research in using UML for critical systems development, or in building advanced tool support for UML
- advanced software developing professionals as the intended users of the approach proposed in this book.

Some basic knowledge in computer security and UML would be helpful. This knowledge is recalled in Sections 3.1 and 3.2, and pointers to background reading are given.

For the benefit of the second group, we deferred the material on the semantics of UML to the end of the book in Chaps. 7 and 8. These can then be left out by people who are not interested in constructing advanced tool support for UML by themselves. The information in Sect. 3.3 about the used semantics of UML is sufficient to understand the remaining chapters.

Note that the UML extension proposed in this book aims to offer assistance also to developers who are not security experts, for example, by enabling them to use security mechanisms in a secure way. Nevertheless, parts of the book

are concerned with advanced applications, such as cryptoprotocol analysis, for which some background knowledge in security would be helpful.

The material in this book has been used extensively for teaching students, as well as researchers and software developers. For example, full-day tutorials for practitioners have been delivered based on the material in Chaps. 3 and 4 and Sects. 6.2 and 6.4. For a two-day course, one can also include Chap. 5. A Masters-level student course could also cover Chaps. 7 and 8.

Additional material is given on a website [Jür04] associated with this book which is continuously being updated. It includes the following material:

- Slides and audio recordings from the tutorials and courses based on this book.
- Other learning and teaching material, including exercises and answers.
- A web interface for a tool which analyzes UMLsec models for security requirements. These models can be written using an industrial UML modeling tool and uploaded over the Internet.

2

Walk-through: Using UML for Security

For a quick impression of what this book is about, we give a short "walk-through" through a small part of the UMLsec notation to highlight the UMLsec approach, considering a simplified model of an Internet-based business application as a running example. For readers who find themselves lacking background on computer security and on the Unified Modeling Language (UML), it is briefly recalled in Chap. 3. The UMLsec extension is then defined and explained in more detail in Chap. 4, as well as the examples shown in this chapter.

A central idea of the UMLsec extension is to define labels for UML model elements, the so-called *stereotypes*, which, when attached, add security-relevant information to these model elements. This security-relevant information can be of the following kinds:

- Security assumptions on the physical level of the system, such as the «Internet» stereotype shown below.
- Security requirements on the logical structure of the system (such as the «secrecy» stereotype) or on specific data values (such as the «critical» stereotype).
- Security policies that system parts are supposed to obey, such as the «fair exchange», «secure links», «data security», or «no down – flow» stereotypes.

In the first two cases, the stereotypes simply add some additional information to a model. They can be attached to any diagram of the relevant kind. In the third case, there are constraints associated with a stereotype that have to be fulfilled by a diagram so that it can justifiably carry the stereotype. If such a stereotype is attached to a diagram which does not meet this constraints, this results in an incorrect model, as in the case of the «secure links», «data security», and «no down – flow» stereotypes below. This prompts the tool support available for UMLsec [JSA+04], described in Chap. 6, to automatically point out the mistake, which should then be corrected by the developer.

2.1 Security Requirements Capture with Use Case Diagrams

Use case diagrams are commonly used to describe typical interactions between a user and a computer system in requirements elicitation. They may also be used to capture security requirements.

To start with our example, Fig. 2.1 shows a use case diagram describing the following situation: a customer buys a good from a business. The trade should be performed in a way that prevents both parties from cheating. We include this requirement in the diagram by adding a stereotype «fair exchange» to the subsystem containing the use case diagram. A more detailed explanation of what the requirement represented by this stereotype means in this specific situation, and of the activities associated with the use cases, is given in the following subsection.

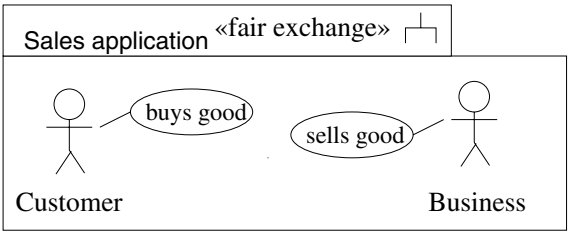

Fig. 2.1. Use case diagram for business application

2.2 Secure Business Processes with Activity Diagrams

Activity diagrams can be used to model workflow and to explain use cases in more detail. Similarly, they can be used to make security requirements more precise.

Following our example, Fig. 2.2 explains the use case in more detail by giving the business process realizing the above two use cases. The requirement «fair exchange» is now formulated by referring to the activities in the diagram. Intuitively, the actions listed in the tags {start} and {stop} should be linked in the sense that if one of the former is executed then eventually one of the latter will be. This property can be checked automatically.

This would entail that, once the customer has paid, either the order is delivered to the customer by the due date, or the customer is able to reclaim the payment on that date.

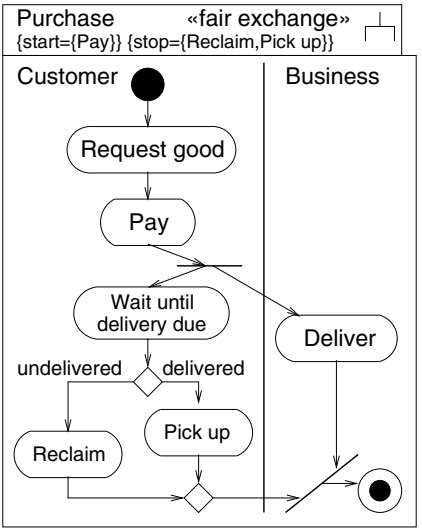

Fig. 2.2. Purchase activity diagram

2.3 Physical Security Using Deployment Diagrams

Deployment diagrams are used to describe the physical layer of a system. We use them to check whether the security requirements on the logical level of the system are enforced by the level of physical security, or whether additional security mechanisms (such as encryption) have to be employed.

Continuing with our example, the business application is part of an e-commerce system, which is supposed to be realized as a web application. The payment transaction involves transmission of data to be kept secret (such as credit card numbers) over Internet links. This information on the physical layer and the security requirement is reflected in the UML model in Fig. 2.3.

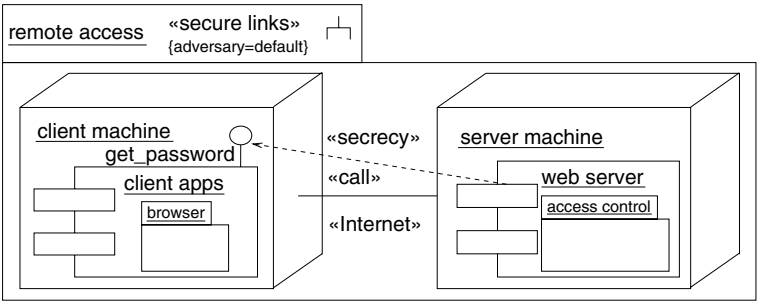

Fig. 2.3. Example *secure links* usage

We then use the stereotype « secure links » to express the demand that security requirements on the communication are met by the physical layer. More precisely, for each dependency stereotyped « secrecy » between subsystems or classes on different nodes n, m, and any communication link between n and m with some stereotype s, the threat scenario arising from the stereotype s with regard to an adversary of a given strength should not violate the secrecy requirement on the communicated data. This constraint will be defined more precisely and explained in detail in Chap. 4. For now we only note that in the given diagram, this constraint associated with the stereotype « secure links » is already violated when considering standard adversaries, because plain Internet connections can be eavesdropped easily, and thus the data that is communicated does not remain secret. For this adversary type, the stereotype « secure links » is thus applied wrongly to the subsystem, which is pointed out automatically by the UMLsec tool presented in Chap. 6.

2.4 Security-Critical Interaction with Sequence Diagrams

Sequence diagrams are used to specify interaction between different parts of a system. Using UMLsec stereotypes, we can extend them with information giving the security requirements relevant to that interaction. For example, this enables one to see whether cryptographic session keys exchanged in a key exchange protocol remain confidential from possible adversaries.

With regard to our example, based on the security analysis in the previous subsection we decide to create a secure channel for the sensitive data that has to be sent over the untrusted networks, by making use of encryption. As usual, we first exchange symmetric session keys for this purpose. Let us assume that, for technical reasons, we decide not to use a standard and well-examined protocol such as SSL but instead a customized key exchange protocol such as the simplified one in Fig. 2.4. The goal is to exchange a secret session key K, using public keys $\mathsf{K_C}$ and $\mathsf{K_S}$, which is then used to encrypt the secret data s before transmission. Here $\{\mathsf{M}\}_\mathsf{K}$ is the encryption of the message M with the key K, $\mathcal{S}ign_\mathsf{K}(\mathsf{M})$ is the signature of the message M with K, and :: denotes concatenation. A detailed explanation of the figure and the protocol can be found in Sect. 5.2.

Note that the UMLsec model of the protocol given in Fig. 2.4 is similar to the traditional informal notation, for example, used in [NS78]. In that notation, the protocol would be written as follows:

$$
\begin{aligned}
&C \to S : \mathsf{N_i}, \mathsf{K_C}, \mathcal{S}ign_{\mathsf{K_C^{-1}}}(\mathsf{C} :: \mathsf{K_C}) \\
&S \to C : \{\mathcal{S}ign_{\mathsf{K_S^{-1}}}(\mathsf{k_j} :: \mathsf{N_i})\}_{\mathsf{K_C}}, \mathcal{S}ign_{\mathsf{K_{CA}^{-1}}}(\mathsf{S} :: \mathsf{K_S}) \\
&C \to S : \{\mathsf{s_i}\}_{\mathsf{k_j}}.
\end{aligned}
$$

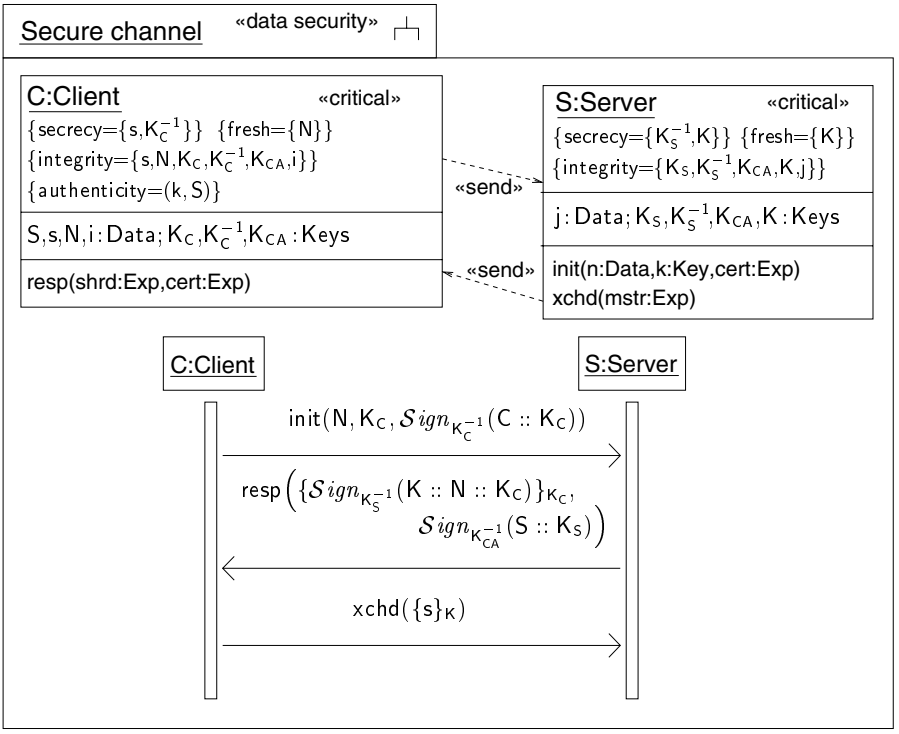

Fig. 2.4. Key exchange protocol

We argue in Sect. 5.2 that the traditional notation needs to be interpreted with care and that the UMLsec notation can be seen to be more precise and to lead over more easily to an implementation.

One can now again use stereotypes to include important security requirements on the data that is involved. Here, the stereotype « critical » labels classes containing sensitive data and has the associated tags {secrecy}, {integrity}, {authenticity}, and {fresh} to denote the respective security requirements on the data. The constraint associated with « data security » then requires that these requirements are met with respect to the given adversary model. We assume that the standard adversary is not able to break the encryption used in the protocol, but can exploit any design flaws that may exist in the protocol, for example by attempting so-called "man-in-the-middle" attacks. This is made precise for a generic adversary model in Sect. 3.3.4. Technically, the constraint then enforces that there are no successful attacks of that kind. Note that it is highly non-trivial to see whether the constraint holds for a given protocol. However, using well-established concepts from formal methods applied to computer security in the context of UMLsec, it is possible to verify this automatically.

2.5 Secure States Using Statechart Diagrams

Statechart diagrams, showing the changes in state throughout an object's life, can be used to specify security requirements on the resulting sequences of states and the interaction with the object's environment.

As the last station in our quick walk-through, we now assume that for privacy reasons, it should remain secret how much money a customer spends at the website. We thus consider the simplified specification of the customer account object in Fig. 2.5. The object has a secret attribute money containing the amount of money spent so far by a given customer. It can be read using the operation rm() whose return value is also secret, and increased by placing an order using the operation wm(x). If the object is in the state ExtraService since the amount of money spent already is over 1000, there is special functionality offered at the website providing the customer with complimentary extra services. There is an associated operation rx() to check whether this functionality should be provided. In the specification shown in Fig. 2.5, this operation is not assumed to be secret.

Now we use the stereotype « no down-flow » to indicate that the object should not leak out any information about secret data, such as the money attribute. Unfortunately, the given specification violates this requirement, since partial information about the input of the secret operation wm() is leaked out via the return value of the non-secret operation rx(). Thus the model carries the stereotype illegitimately. Again this can be detected automatically, and it is another example for a constraint which is infeasible to verify without tool support, for specifications of the size arising in practice.

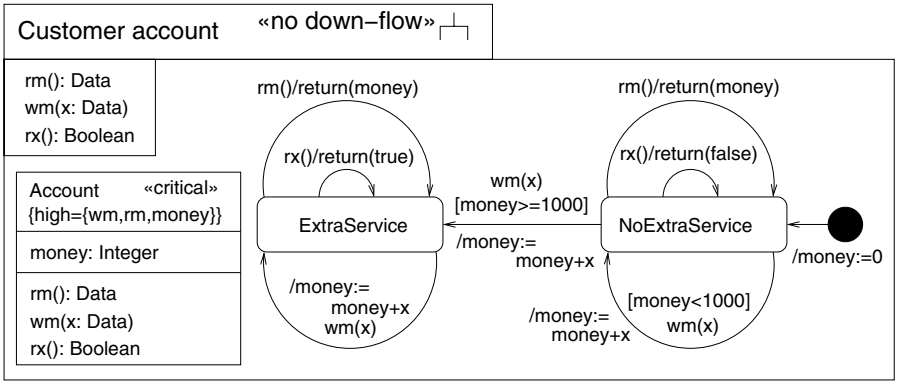

Fig. 2.5. Customer account data object

http://www.springer.com/978-3-540-00701-2

Secure Systems Development with UML
Jürjens, J.
2005, XX, 316 p. 79 illus., Hardcover
ISBN: 978-3-540-00701-2