Using UMLsec and Goal Trees for Secure Systems Development

Jan Jürjens*
Computing Laboratory, University of Oxford, GB and
Software & Systems Engineering, Dep. of Informatics, TU München, Germany
juerjens@in.tum.de

ABSTRACT

Today many software systems need to take into account security considerations. While Software Engineering has been quite successful in ensuring that systems satisfy non-functional requirements such as dependability, less work has been done wrt. security requirements.

In this work we present a software engineering method aiming to facilitate secure systems development, which is based on an extension of UML called UMLsec.

Keywords

Secure systems development, Unified Modeling Language (UML), goal trees, formal semantics, UMLsec

1. INTRODUCTION

While functional requirements are generally analyzed carefully in systems development, security considerations often arise after the fact. Also, the relationship between non-functional requirements and software architectures is only very poorly understood. In particular regarding security, there has been significant research in verifying formal specifications of small security critical components against security requirements. However, security concerns must inform every phase of software development, from requirements engineering to design, implementation, testing and deployment [5], because adding security as an afterthought often leads to problems [2]. With our approach, we try to improve on this situation by integrating security requirements analysis with a standard development process.

Extending the work in [15], we give an extension of the Unified Modeling Language (UML [22], the de facto industry-standard in object-oriented modelling) to include modelling of security related features such as confidentiality, access control etc.. Advantages of this approach include

- a unified design of systems and security policies,
- modularity and possibility for reuse
- the opportunity to make use of existing or future tools from general systems engineering.

This approach is useful, since 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 [2]. 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 [23], 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),
- 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 simplified formal core of UML (in its current version 1.4 [22]) to give an extension, called UMLsec, using the standard UML extension mechanisms that encapsulates knowledge on prudent security engineering and thereby make it available to developers who may not be specialized in security.

We use a combination of a use-case driven process [11] with a goal-directed approach [3]. This enables us to make use of results from object-oriented analysis for functional requirements and to complement these with the formulation of goals especially suited for non-functional requirements.

Currently a large part of effort both in implementing and verifying security-relevant specifications is wasted since these are often formulated imprecisely and unintelligibly. Being able to express security-relevant information in a widely used design notation helps alleviate this problem.

After presenting some background on security and software engineering in the following subsection, we summarize our use of UML in the subsequent section. We end with

*http://www.jurjens.de/jan

SAC 2002, Madrid, Spain

pointers to related work, a conclusion and indication of future work.

1.1 Security Requirements

During the requirements engineering stage in the development of a software system one identifies the functional and non-functional requirements of the system. Functional requirements describe which services a system should offer. Non-functional (or quality) requirements describe qualities of these services, such as fault tolerance, performance and security aspects [3] (for example, "secrecy" is called non-functional because it describes not *what* a system should do (e.g. send some data), but *how* it should do it (e.g. send the data in a way that prevents illegitimate access)).

While much research in software engineering research was focussed on specifying functional requirements, non-functional requirements have received less systematic attention. There are standard industrial software engineering development processes (e.g. ISO-12207) that do consider non-functional requirements, but these are often formulated in rather general terms. Here we pay special attention to distributed object-oriented systems.

2. A PROCESS FOR SECURE SYSTEMS DEVELOPMENT WITH UMLSEC

2.1 Designing Secure Systems

It is still largely the case that developers rely mostly on their intution in developing secure systems, without much systematic help or guidance.

Following [3], we address the following questions regarding our proposed design process.

(1) *How can a variety of well-known and lesser-known security techniques be made available to the designer through systematic search ?* Our approach aims to encapsulate design knowledge to facilitate reuse [12]. One can use a variety of security notions (such as confidentiality, integrity, authenticity) to express security requirements [15]. Transformations to introduce patterns aid the selection of specific security mechanisms, such as guards controlling access to objects [16].

(2) *How can interactions among potentially conflicting or synergistic requirements be managed systematically ?* We can make use of various extensions of the UML that evaluate system designs wrt. different requirements, such as performance requirements.

(3) *How can the nature of relationships between design decisions be represented ? How can the effect of each design decision be systematically evaluated ?* Design decisions are evaluated in a systematic way using conditions on the diagrams in a UML specification [15]. Since UML diagrams allow various views on a system, one can evaluate design decision regarding these (including e.g. the physical environment).

(4) *How can security requirements be systematically integrated into the design, together with other types of non-functional requirements ?* Using UML makes integration into the design and the general development process relatively straightforward, since many developers know UML and can use it without too much overhead (after learning the security-specific information given in accordance with the standard UML extension mechanisms (such as stereotypes)).

(5) *What drives design actions ? What representational structures are appropriate for systematically recording the results of such actions ?* As pointed out in [6], a goal-oriented approach to requirements (such as [3]) may work better wrt. non-functional requirements than use-case driven approaches (such as [11]). However, [18] points out that goal-oriented analysis and object-oriented analysis complement each other. Thus one can fruitfully employ the goal tree approach to non-functional requirements [3] in our work using UML. Specifically, we propose to combine a use-case driven approach as in [11] for the functional requirements with a goal-driven approach as in [3] for the security requirements. This takes account of the fact that security requirements (such as confidentiality) often apply to specific functions (e.g. a certain value) of a system, rather than the system as a whole, because applying the security requirement to the whole system may be infeasible [21].

As remarked in [2], security-critical systems often require an iterative development (such as the Spiral).

2.2 Using UMLsec

UMLsec consists of the following diagram types describing different views on a system: Use case diagrams, Activity diagrams, Class diagrams, Sequence diagrams, Statechart diagrams, and Deployment diagrams (for an introduction to UML cf. [20]):

UML offers rich extension mechanisms in the 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 {tag = value}). Using *profiles* one can give a specific meaning to model elements marked with these labels. Here we give an extension UMLsec of UML making use of such labels to express security requirements.

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.

As mentioned, we use a combination of a use-case driven process with a goal-directed approach. More specifically, we develop a security goal tree alongside the development of the system specification. The security goals are refined in parallel by giving more system details in subsequent design phases. The process is in general iterative; here we only explain one iteration due to space limitations. To keep things simple, our examples employ relatively simple goal trees.

Since we are using a formal fragment of UML, we may reason formally, showing for example that a given system is as secure as certain components of it. This way, we can reduce security of the system to the security of the employed security mechanisms (such as security protocols). We aim to exhibit the conditions under which protocols can be used securely in the system context.

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

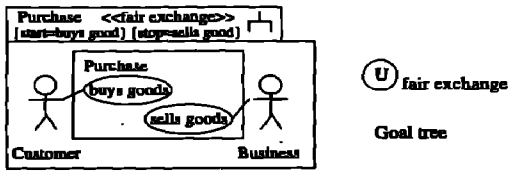

Figure 1: Use case diagram with goal tree

2.3 Requirements Capture

We use use cases to capture security requirements. To start with our example, Figure 1 gives the use case diagram describing the situation to be achieved together with the (yet trivial) goal tree: a customer buys a good from a business in a way that should provide a fair exchange of the good against the payment. The semantics of the stereotype «*fair exchange*» is, intuitively, that the actions "buys goods" and "sells goods" 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). The "U" in the goal tree stands for "undetermined" – it is not yet known whether the goal will be *satisficed* ("S") or *denied* ("D") [3].

2.4 Analysis

We use activity diagrams to explain use cases in more detail. Following our example, Figure 2 explains the use case in Figure 1 and the associated goal tree in more detail by giving an activity diagram and a refined goal tree. The activity diagram is separated in two *swim-lanes* describing activities of different parts of a system (here Customer and Business). Two tag-value pairs {start = Pay} and {stop = final} 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, then eventually the other will be. As one can demonstrate on the level of the formal semantics [13, 17], the activity 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.

2.5 Design: class diagrams

Pursuing our example, Figure 3 gives a class-level description of a key generator (such as used in the fair exchange protocol mentioned above). The key generator offers the method newkey() which returns a Key for which it guarantees confidentiality and integrity.[1] 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 specifiying security requirements for the used random number generators. Omissions like this one can be detected using our modelling approach.

[1]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).

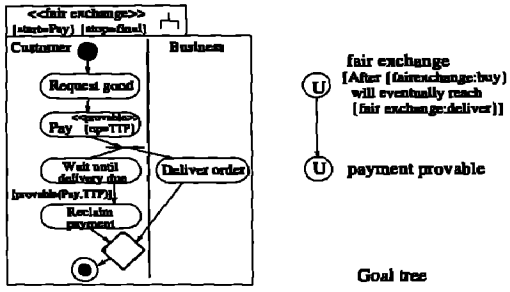

Figure 2: Activity diagram and goal tree

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.

2.6 Design: sequence diagrams

In our UML-based approach, security protocols can be specified using message sequence charts. Further to our example, we assume that the payment is undertaken using the purchase protocol from the Common Eletronic Purse Specifications (CEPS). In particular, the security then relies on the correctness of the purchase transactions protocol, whose abstract specification 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 [13, 17].

2.7 Design: statechart diagrams

Statechart diagrams give the dynamic behaviour of an individual object: events may cause state in 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), 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

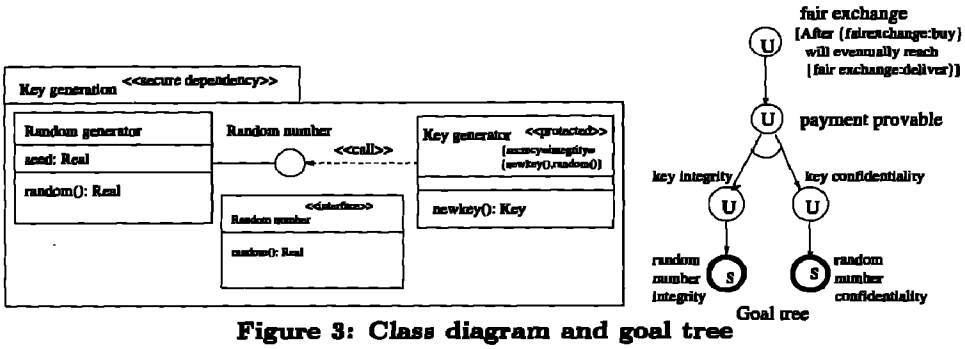

Figure 3: Class diagram and goal tree

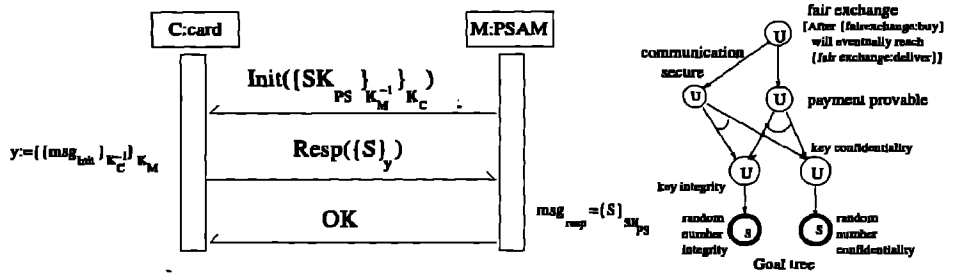

Figure 4: Sequence diagram and goal tree

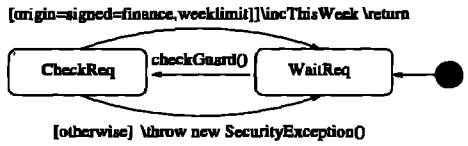

Figure 5: Statechart for guard object

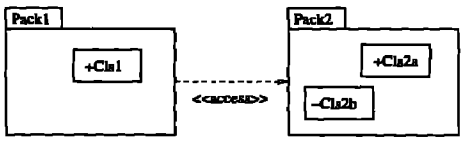

Figure 6: Package diagram with visibility

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. In our example, the guard object can be specified as in Figure 5 (where ThisWeek counts the 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. [14]).

2.8 Design: 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.

2.9 Implementation: 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

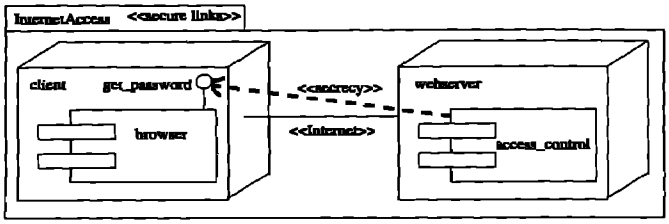

Figure 7: Deployment diagram for physical layer

the underlying communication link (as usual, this would be treated by using encryption).

3. RELATED WORK

This work extends a line of research started in [15], making use of results on a formal semantics for UML in [17]. [5] suggests to make systematic use of the UML extension mechanisms to treat security aspects. So far, this topic does not seem to have been addressed otherwise.

There has been much work on security using formal methods (for an overview regarding security protocols cf. [19]). Less work has been done using software engineering techniques [5]. Similar in spirit is the work in [7] (defining role-based access control rights from object-oriented use cases). Also relevant is work on security patterns (such as [8]). Note e.g. work on trusted software engineering in [4]. There has been a lot of work on formal methods for object-orientation, cf. e.g. [10, 9].

4. 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 specialized 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. We demonstrated a process to use with these diagrams, which is a combination of use-case driven and goal-directed approaches.

Work towards tool-support is being undertaken by giving translations from UML into Abstract State Machines (ASMs) [17] which allows use of the associated tools to check security properties.

5. REFERENCES

[1] ACM. *Symposium of Applied Computing*, 2002.

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

[3] L. Chung. Dealing with security requirements during the development of information systems. In *CAiSE '93, 5th Int. Conf. Advanced Information Systems Engineering*. Springer-Verlag, 1993.

[4] P. Devanbu, P. Fong, and S. Stubblebine. Techniques for trusted software engineering. In *20th International Conference on Software Engineering*, pages 126–135, 1998.

[5] 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).

[6] W. Emmerich. Software engineering and middleware: a roadmap. In *ICSE - Future of SE Track*, pages 117–129, 2000.

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

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

[9] P. Gibson. Formal object oriented requirements: simulation, validation and verification. In *Modelling and Simulation: A tool for the next millenium, vol II.*, pages 103–111. SCS, 1999. European Simulation Multi-conference (ESM99).

[10] P. Gibson and D. Méry. Fair objects. In *Object-oriented technology and computing systems re-engineering*. Horwood Publishing, 1999.

[11] I. Jacobsen, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1998.

[12] J. Jürjens. Encapsulating rules of prudent security engineering. In *International Workshop on Security Protocols*, Lecture Notes in Computer Science. Springer-Verlag, 2001. To be published.

[13] J. Jürjens. *Principles for Secure Systems Design*. PhD thesis, Oxford University Computing Laboratory, 2001. In preparation.

[14] J. 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.

[15] J. Jürjens. Towards development of secure systems using UMLsec. In H. Hußmann, editor, *Fundamental Approaches to Software Engineering (FASE, 4th International Conference, Part of ETAPS)*, volume 2029 of *Lecture Notes in Computer Science*, pages 187–200. Springer-Verlag, 2001. Also OUCL TR-9-00 (Nov. 2000), http://web.comlab.ox.ac.uk/oucl/publications/tr/tr-9-00.html.

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

[17] J. Jürjens. A UML statecharts semantics with message-passing. In *Symposium of Applied Computing* [1].

[18] J. Mylopoulos, L. Chung, and E. Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1):31–37, 1999.

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

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

[21] R. Stevens, P. Brook, K. Jackson, and S. Arnold. *Systems Engineering*. Prentice Hall, 1998.

[22] UML Revision Task Force. OMG UML Specification v. 1.4. OMG Document ad/01-09-67. Available at http : //www.omg.org/uml, Sept. 2001.

[23] M. Walker. On the security of 3GPP networks. In *Advances in Cryptology – EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

6. BIOGRAPHY

Jan Jürjens' biography is given at the end of [17] in the current volume.