

Using Model-based Security Analysis in Component-oriented System Development

A Case-based Evaluation

Gyrd Brændeland
SINTEF & University of Oslo, Norway
gyb@sintef.no

Ketil Stølen
SINTEF & University of Oslo, Norway
kst@sintef.no

ABSTRACT

We propose an integrated process for component-based system development and security risk analysis. The integrated process is evaluated in a case study involving an instant messaging component for smart phones. We specify the risk behaviour and functional behaviour of components using the same kinds of description techniques. We represent main security risk analysis concepts, such as assets, stakeholders, threats and risks, at the component level.

Categories and Subject Descriptors

D.2 [Software]: Software Engineering—*requirements/ specifications*

General Terms

Security, Theory

Keywords

Case studies, Security Risk Analysis

1. INTRODUCTION

If the brakes on your car fail it might not be caused by sloppy work at the garage, but rather by a virus from the latest music file downloaded by your car's filesharing software. Well, maybe not today but this is not a far fetched scenario. There are cars on the market today running dynamic component platforms centralising a wide range of functionalities, such as entertainment functions, positioning systems and car controls. They allow continuous installation of new services or upgrades. With strict "time-to-market" requirements and short life-time expectancies for software technology, products such as cars, smart phones and mobile devices in general are increasingly sold as upgradable products.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

QoP'06, October 30, 2006, Alexandria, Virginia, USA.
Copyright 2006 ACM 1-59593-553-3/06/0010 ...\$5.00.

But how do we know if we can trust a new component that we download into a system? The increased application of component technology makes systems more flexible but also gives rise to new security concerns and calls for documentation of security risks at the component level. A component-oriented development process must reflect this.

The purpose of security risk analysis is to decide upon the necessary level of asset protection against security risks, such as a confidentiality or integrity breach. For convenience we will often use *security analysis* as a short term for *security risk analysis*.

We propose an integrated process for component-based system development and security analysis that builds on already existing techniques for specification and security analysis. The CORAS method [5] combines security analysis methods with UML-inspired systems modelling. CORAS has no particular support for component-oriented development. We believe, however, that UML models are also well suited to document and structure security analysis results at the component level. In CORAS and other current approaches to security analysis in system development, UML models are mainly used as input for identifying threats and structuring the security analysis process. We wish to use system development and modelling techniques such as UML not only as input for the security analysis, but also for the purpose of documenting analysis results, in order to support composition of such results.

We evaluate the integrated process in a case study involving an instant messaging (**im**) component for smart phones. The **im** component should allow users to interact in chat sessions and exchange media files with buddies, organised in a peer-to-peer fashion. It should be possible to deploy and run the service on smart phones, laptops etc. running a dynamic component platform.

The case we present is inspired by a tutorial on developing a chat service using OSGi (The Open Source Gateway initiative) – a standardised computing environment for networked services [23]. It is a fictitious example, but nevertheless represents a realistic case for component development that is of practical relevance. Due to lack of space we focus on a few selected parts that illustrate the characteristics of component-based security analysis and the structure of an integrated process. We refer to Brændeland and Stølen [2] for a presentation of the full evaluation. We evaluate the integrated approach with regard to the following criteria. In an integrated component-oriented process it should be

possible to:

1. Identify stakeholders at the component level.
2. Identify, value and represent assets at the component level.
3. Represent threats towards assets and analyse component behaviour at the component level, in relation to various threats.
4. Capture the notion of a risk at the component level in terms of impact and probability of an event.

Furthermore it should be possible to:

5. Compose individual components documented according to 1–4, into composite components, and deduce the security risk level of the composition from the security risk documentation of its constituents.

This paper addresses the intersection of security risk analysis and component-oriented development. As these notions may mean different things in different settings and communities, we begin by explaining our notions of component and security risk analysis in Sections 2 and 3. The evaluation of the integrated process is structured into four sections (Sections 5 to 8) corresponding to the early stages in a component-based development process. In Section 9 we attempt to place our work in relation to ongoing research within related areas and finally, in Section 10, we discuss to which extent the evaluation criteria are met.

2. THE COMPONENT MODEL

Figure 1 shows our conceptual model of a component. An interface is a contract with a client, describing a set of

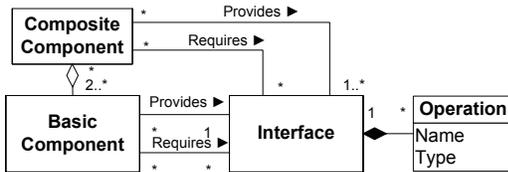


Figure 1: Conceptual model of a component

behaviours provided by a component object. It defines a list of operations, their signatures and semantics.

A component is a contract with the realiser. It describes provided interfaces and component dependencies in terms of required interfaces. By required interface we mean the calls the component needs to make, in order to implement the operations described in the provided interfaces. We distinguish between basic components and composite components. A basic component provides only one interface. We obtain components with more than one provided interface by the composition of basic components. Composite components can also be composed to obtain new composite components.

3. SECURITY ANALYSIS CONCEPTS

By security risk analysis we mean risk analysis applied to the domain of information and communication technology (ICT) security. ICT security includes all aspects related to defining, achieving and maintaining confidentiality,

integrity, availability, non-repudiation, accountability, authenticity and reliability of ICT [9]. The definitions used in this section are all from the ISO standard for managing ICT security (ISO/IEC 13335-1 FDIS) [9].

There are many forms and variations of security analysis. The CORAS security analysis process is *asset-oriented*. Assets are parts or features of a system which are considered sufficiently valuable to require some degree of protection. Concepts to consider in asset-oriented security analysis include: A *vulnerability* is a weakness of an asset or group of assets that can be exploited by one or more threats. An *information security incident* refers to any unexpected or unwanted event that might cause a compromise of business activities or information security. A *threat* is a potential cause of an incident that may result in harm to a system or organisation. A *risk* refers to the potential that a given threat will exploit vulnerabilities of an asset or group of assets and thereby cause harm to the organisation. It is measured in terms of a combination of the *probability* of an event and its *impact* (consequence). Impact is the result of an information security incident.

4. AN INTEGRATED APPROACH

In the following we investigate how the CORAS process can be integrated into a component-based system development process as described by Cheesman and Daniels [3]. They describe a process for specification and development of components using UML diagrams, illustrated by the left-hand side of Figure 2. The grey boxes represent workflows

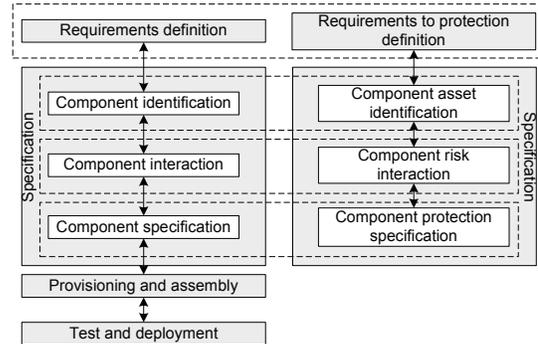


Figure 2: Integrated process

as defined in the Rational Unified Process [11]. Each workflow produces an artifact that is used as input in the next workflow.

The component-oriented process starts by describing the overall system level requirements, such as functional requirements and quality of service requirements. During the requirements workflow the system is viewed as a black box, any internal behaviour of sub-components are hidden. According to [3], the requirements workflow should deliver a business concept model and a set of use cases to the specification workflow. During the specification workflow the system is decomposed into basic components that are refined further, independently of each other. It entails identifying provided and required interfaces, describing component dependencies and specifying how the basic components can be fitted together into a composite component that refines the original requirements. The output from the specification

workflow is used in the provisioning workflow to determine what components to build or buy, in the assembly workflow to guide the correct integration of components, and in the test workflow as input to test scripts.

We aim, in particular, to integrate security analysis into the early stages of system development and focus, as indicated by Figure 2, on the requirements and specification workflow. While the requirements definition captures the quality of service and functional requirements, the requirements to protection specify the acceptable level of security risk, i.e., what may be tolerated with respect to security risks, i.e., what may be tolerated with respect to security risks, as further explained in section 5. In component-based security analysis we need to describe stakeholders and assets at the component level and in Section 6 we extend the component identification to achieve this. In Section 7 we augment the specification of the component interactions with specifications of the component risk interactions. Even if the basic component specifications is verified to refine the system requirements this of course does not mean that the requirements to protection are fulfilled. As explained further in Section 8 in addition to specifying the ordinary component logic we must also characterise its way of protection.

We use a combination of UML 2.0 notation [18], sequence diagrams as defined in STAIRS [8, 7] and CORAS diagrams [5] in the component specifications.

5. REQUIREMENTS

In accordance with Figure 2 we first give the requirements definition as described by Cheesman and Daniels [3]. Then we present the requirements to protection definition.

5.1 Requirements definition

The purpose of requirements definition is to describe what services the system should provide, allocate responsibilities for the various services and to decide upon the system boundary [3]. We use a UML use case diagram shown in Figure 3 to clarify the software boundary, identify the actors interacting with the system, and list those interactions. A use case diagram is usually used in combination with a

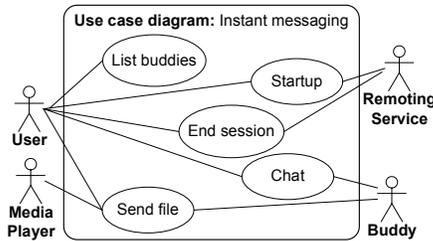


Figure 3: Use case diagram

textual description capturing the details of each use case. We can also specify details of the use cases in sequence diagrams. The diagram to the right in Figure 4 captures the details of the Startup use case. When a user starts the **im** client, the client searches for other peers. After starting the client, the user can register or login. Registration is specified as optional since this happens only the first time a user launches an **im** component. If a login is successful the **im** service is registered.

To the left in Figure 4 is an interaction overview diagram showing the flow of interactions of the **im** component. After startup the user can either check for other buddies, send

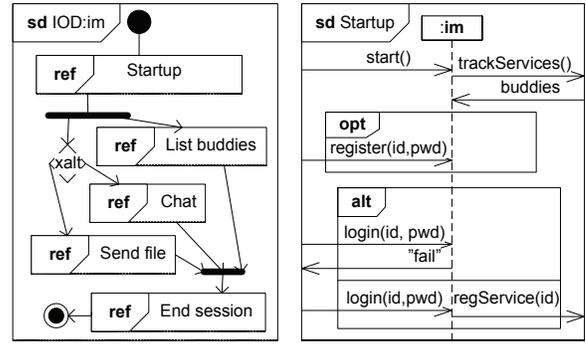


Figure 4: Interactions of the **im** component

messages or send files. We use the **xalt** construct, as defined in STAIRS [8, 7], to express that the choice between sending messages and exchanging files is mandatory, i.e., that all implementations of the **im** component must offer both possibilities.

5.2 Requirements to protection definition

The requirements to protection definition specify what the stakeholder may tolerate with respect to risk towards an asset. Within risk analysis such requirements are often referred to as risk acceptance criteria. Prior to defining the risk acceptance criteria it is necessary to identify stakeholders and identify and value assets.

The use case diagram in Figure 3 serves as input to the identification of stakeholders and their assets. In a use case the actors interact with the system as a whole, not some specific part of it. The actors identified in a use-case correspond to component clients or suppliers, represented through provided and required interfaces.

Figure 5 show examples of identified assets. We identify

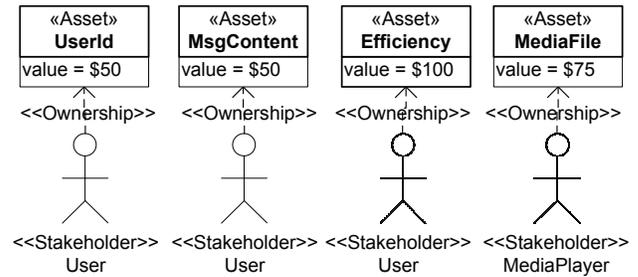


Figure 5: Asset identification

UserId, MsgContent and Efficiency as assets of the actor User and MediaFile as an asset of the MediaPlayer¹. The UserId may be of value for the User if she employs the same id for many different services and access to the UserId could provide access to these.

We can now define risk acceptance criteria on behalf of the stakeholders for each asset. A risk acceptance criterion may for example be that “The probability of a risk reducing the MediaFile asset with 30 dollars should not be higher than 0.25.”

¹The CORAS UML profile introduces the stereotypes **«Stakeholder»** and **«Asset»** to model stakeholders and asset.

6. COMPONENTS AND THEIR ASSETS

In accordance with Figure 2, we first conduct component identification, thereafter we identify the assets of each component.

6.1 Component identification

Component identification is the first stage of the specification workflow. It entails decomposing the system into basic components and identifying component interfaces. A use case indicates the types of operations that the system should offer through interfaces. We group the responsibility for the Startup, Chat, List buddies and End session and assign it to one interface that we call IChat, as illustrated in Figure 6². The responsibility for the Send file use case

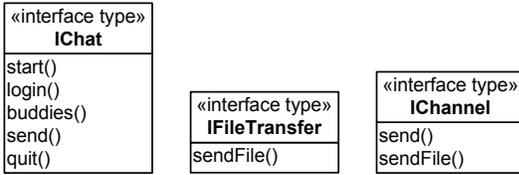


Figure 6: Interface types

is assigned to a separate interface IFileTransfer. We identify an interface IChannel that a Buddy can use for sending messages or files to a client.

We must also identify business interfaces, if any, i.e., the interfaces handling information managed by the system. We assume that we have already defined a business concept model and refined this into a business type model providing the necessary input for this task. The only type of information that the **im** component is itself responsible for managing, is the UserId. Hence, we identify one business interface IUserMgr. Other types of information will be handled by external components.

We also need to establish whether there are existing interfaces and systems that are part of the environment into which the **im** service will be deployed. As illustrated in Figure 7, the **im** component requires an interface for remote services, such as registration and tracking, and interfaces for displaying messages and playing music files. Buddies registered through the IRemote interface will be other Channels providing interfaces that the Chat and FileTransfer components employ for sending messages and files, respectively. We indicate component dependencies using the

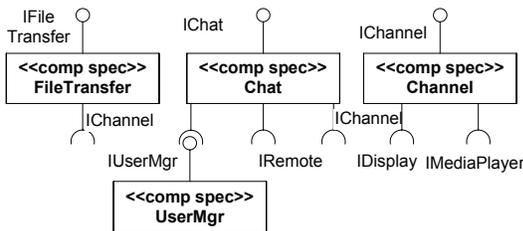


Figure 7: Basic components

²Cheesman and Daniels [3] introduce the stereotypes «interface type» and «comp spec», rather than applying the predefined UML modelling elements «interface» and «component», which are used for modelling the implementation level.

ball and socket notation of UML component diagrams. The component dependencies are detailed further as part of the component interaction specification.

6.2 Component asset identification

During the requirements workflow we identified system level stakeholders and assets. We now decompose and identify stakeholders and assets at the level of basic components. The actor MediaPlayer corresponds to the provider of the IMediaPlayer interface and the actor RemotingService corresponds to the provider of the IRemote interface. The actor User corresponds to the providers and clients of the external interfaces: IFileTransfer, IChat and IDisplay. The assets UserId and MsgContent can be affected by operations of the IChat interface and we assign these assets to the client of that interface. We assign the Efficiency asset to the provider of the IDisplay interface which the Channel component requires to convey received messages.

Above we identified the component UserMgr which provides an interface that the Chat component employs to handle user information. Hence the UserMgr and Chat components are stakeholders of each other. We assign an asset UserId to the requirer of the UserMgr interface, i.e. the Chat component.

7. INTERACTIONS

As indicated by Figure 2, there are two kinds of interactions to be specified. We first describe the wanted interactions and then the risk interactions.

7.1 Component interaction

We now have to decide how the various sub-components work together in order to deliver the required functionality. This entails specifying which operations the Chat component calls from the IUserMgr interface in order to perform the login operation and so on. For example, as illustrated in Figure 8 when sendFile is called, the IChannel component calls a method to play the music file. As explained below,

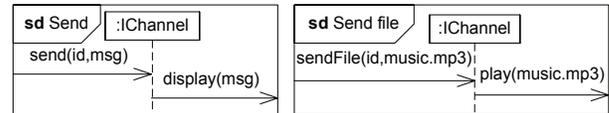


Figure 8: Interactions of the channel component

this is not without risk.

Interface behaviour can be seen as an abstraction over basic component interactions. A provided interface represents the view of the user, who does not need to know how an operation is implemented. Formally we obtain the provided interface of a basic component by filtering away the interactions on the required interface [1].

7.2 Component risk interaction

In order to describe the risk interactions, we first identify risks and estimate their risk levels. CORAS uses structured brainstorming inspired by HazOp [16] for identifying threats related to the functional behaviour of a system. A structured brainstorming is a methodical “walk-through” of the object of analysis. Experts on different aspects of the system identify threats and exploitable vulnerabilities. We

use the system specification diagrams as input to the structured brainstorming session. The result is documented in CORAS threat diagrams as the one in Figure 9. Threat diagrams are used to structure the chain of events leading from a threat to an unwanted incident. A vulnerability is

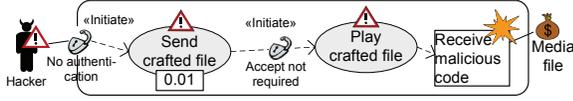


Figure 9: Threat related to file transfer

depicted as an open lock. The vulnerability “Accept not required” refers to that the **im** component automatically plays music without consulting the user. This may be exploited to send a specially crafted music file designed to exploit possible buffer overflow vulnerabilities in a media player. A threat scenario is depicted as a use case decorated with a warning sign and an information security incident as a box decorated by a small explosion symbol. A scenario may lead to another scenario, and this is shown by use of the stereotype \ll Initiate \gg . In this case, “Send crafted file” initiates the threat scenario “Play crafted file” and the information security incident “Receive malicious code” that relates to the asset “MediaFile”.

To determine the risk level of the identified risks we estimate their consequence and probability values. Threat diagrams are used as input to brainstorming sessions aiming to calculate impacts and probabilities of risks. In order to estimate the probability of a hacker successfully sending a crafted media file, we must know both the probability of an attack and the probability of the success of an attack (degree of vulnerability), given an attempt. The probability of a single threat scenario can for example be estimated from historical data. While the number of mobile device threats continues to increase, the number of reported threats is still relatively small [22]. We therefore estimate the probability of an attack to be fairly low (between 0 and 0.01).

In order to estimate the success of an attack we must consider vulnerabilities and the effectiveness of control mechanisms if any such exists. As the **IChannel** component is not designed to check validity of file names and automatically opens incoming music files, the probability that the **IChannel** component attempts to play a crafted file is 1. We multiply the probability of an attack with the probability of its success to obtain the probability of the unwanted incident. Hence the probability that the **IMediaPlayer** receives a crafted file is the same as the probability of the threat.

The risk interaction is finally formalised in STAIRS [8, 7]. In a component setting we represent threats as lifelines. A threat may initiate a threat scenario by calling an operation of one of the components interfaces. Figure 10 shows the interactions of a hacker sending a crafted mp3 file and the **IChannel** component receiving the crafted file.

We represent an asset as a lifeline that has a value. In an interaction we represent the reduction of asset value by a special message called **reduce**, which takes as argument the amount by which the asset value should be reduced, see Figure 10. As the sequence diagram illustrates an example run, we assume the initial asset value have been set elsewhere and parameterise the specification with the asset value k of type $\$$. The value of an asset at a given point in time is computed from its initial value and all occurrences of **reduce**, with the

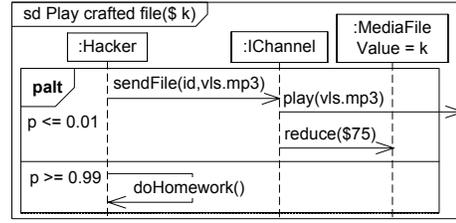


Figure 10: A hacker sends a crafted mp3 file

asset as receiver, up to that point. The value of an asset can not go below zero.

A risk is measured in terms of its probability and its impact. The manifestation of a risk in an interaction is the transmission of a **reduce** event to an asset. In order to decide the probability of a risk one must know the combined probability of the interactions leading to the transmission of the **reduce** message. We use the **palt** operator of probabilistic STAIRS [17], which is a generalisation of the **xalt** operator for mandatory choice, to specify probabilistic choice, as illustrated in Figure 10.

8. SPECIFICATION

During the final stage of the specification workflow, we finalise the component specifications and verify that the requirements, including the requirements to protection, are met by the component specifications. In accordance with Figure 2, we first describe what this stage entails with regard to component specification and then with regard to the protection specification.

8.1 Component specification

Component specification is the final stage of the specification workflow. During this stage we describe how the basic components can be fitted together into a composite component that refines the original requirements. In a composite component we hide local interactions, so the interactions between the **Chat** and **UserMgr** sub-components become hidden. We build composites using the basic composition operators of STAIRS. For the full details of the semantics of composite component specifications, see [1].

8.2 Component protection specification

We must also combine the component security risk specifications using the composition operators of probabilistic STAIRS to check whether their composition fulfils the requirements to protection, i.e., the risk acceptance criteria given for each asset. We refer to [1] for the full details of composite security risk specifications. If the identified risks are found to violate the risk acceptance criteria, we need to identify protection mechanisms to ensure that the requirements to protection are met. A protection specification should refine the system requirements. STAIRS distinguish between two special cases of refinement, called narrowing and supplementing. Narrowing reduces the set of positive traces and supplementing categorises inconclusive traces as either positive or negative.

In order to reduce the probability of the risk we presented for the **Channel** component, we can remove the behaviour where an incoming file is automatically opened (narrowing) and supplement with alternative behaviour requiring that

the user should be consulted before opening an incoming file. The refined specification constitutes the component protection specification.

9. RELATED WORK

The need for conducting risk analysis in the early phases of system development is widely recognised in the security community, and several approaches to this end have been proposed [15, 6, 4, 19, 20, 14, 13].

There are also several proposals for including security requirements into the requirements phase, such as for example in SecureUML [12] and in UMLsec [10]. SecureUML is a method for modelling access control policies and their integration into model-driven software development. SecureUML is based on role-based access control and models security requirements for well-behaved applications in predictable environments. UMLsec is an extension to UML that enables the modelling of security-related features such as confidentiality and access control. These methods focus on specifying security properties of systems which is orthogonal to what we do. They include no notion of risk or probability. Rather than specifying security properties of systems, we focus on analysing security risks and specifying those risks, i.e., we integrate the documentation of the probability that unwanted behaviour may occur into component specifications. Specifications using either of the above approaches can, however, be used as input to further security risk analysis.

One approach that takes into account both system level behaviour and hardware specific information is the model-based performance risk analysis method by Cortelessa et al. [4]. They combine performance related information of interaction specifications with hardware characteristics, in order to estimate the overall probability of performance failures. Their approach is based on a method for architectural-level risk analysis using UML.

The idea to apply specialised use-cases for the purpose of threat identification was first proposed by McDermott and Fox [14, 13]. Sindre and Opdahl [19, 20] later explained how to extend use-cases with mis-use cases as a means to elicit security requirements.

The use of threat diagrams in CORAS to structure the chain of events leading from a threat to an unwanted incident is inspired by Fault Tree Analysis (FTA) and Event Tree Analysis (ETA) [21]. FTA is a top-down approach to calculate the probability of an unwanted incident. ETA is a bottom-up approach to calculate impacts of events.

The novelty of our approach lies in the usage of system development techniques such as UML and STAIRS not only as input for the security analysis, but also as a means for documenting security analysis results. We identify, analyse and document security risks at the component level, thus allowing for the shifting risks depending on the type of environment that a component interacts with.

10. SUMMARY AND CONCLUSION

We have presented a case-based evaluation of the feasibility of applying model-based security analysis in order to integrate security analysis in component-oriented development. We claim that the integrated approach presented here meets the requirements of Section 1 in the following sense:

1. By limiting the notion of a stakeholder to that of a

component client or supplier, we manage to identify and represent stakeholders at the component level

2. We identify assets for each stakeholder and specify them as lifelines with a value. We represent the reduction of asset value as the reception of a special kind of reduce message.
3. We represent threats as lifelines that may interact with a component and initiate a threat scenario.
4. We represent a risk as a probabilistic interaction leading to the reduction of an asset value.
5. The component security analysis results are documented using sequence diagrams in probabilistic STAIRS. We obtain the risk level of a composite component by composing the security risk specifications of its sub-components, using the composition operators of probabilistic STAIRS.

Acknowledgements

The research on which this paper reports has been funded by the Research Council of Norway via the two research projects COMA 160317 (Component-oriented model-based security analysis) and SECURIS (152839/220).

11. REFERENCES

- [1] G. Brændeland and K. Stølen. A semantic paradigm for component-based specification integrating a notion of security risk. To appear in Proceedings of the fourth international Workshop on Formal Aspects in Security and Trust (FAST'06), 2006.
- [2] G. Brændeland and K. Stølen. Using model-based security analysis in component-oriented system development. a case-based evaluation. Technical Report 342, University of Oslo, Department of Informatics, 2006.
- [3] J. Cheesman and J. Daniels. *UML Components. A simple process for specifying component-based software*. Component software series. Addison-Wesley, 2001.
- [4] V. Cortellessa, K. Goseva-Popstojanova, K. Appukkutty, A. Guedem, A. E. Hassan, R. Elnaggar, W. Abdelmoez, and H. H. Ammar. Model-based performance risk analysis. *IEEE Transactions on Software Engineering*, 31(1):3–20, 2005.
- [5] F. den Braber, T. Dimitrakos, B. A. Gran, M. S. Lund, K. Stølen, and J. Ø. Aagedal. *UML and the Unified Process*, chapter The CORAS methodology: model-based risk management using UML and UP, pages 332–357. IRM Press, 2003.
- [6] K. Goseva-Popstojanova, A. E. Hassan, A. Guedem, W. Abdelmoez, D. E. M. Nassar, H. H. Ammar, and A. Mili. Architectural-level risk analysis using UML. *IEEE Transactions on Software Engineering*, 29(10):946–960, 2003.
- [7] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, University of Oslo, Department of Informatics, 2004.
- [8] Ø. Haugen and K. Stølen. STAIRS – steps to analyze interactions with refinement semantics. In *UML*,

- volume 2863 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2003.
- [9] ISO/IEC. *Information Technology – Security techniques – Management of information and communications technology security – Part 1: Concepts and models for information and communications technology security management*, 2004. TR 13335–1.
- [10] J. Jürjens, editor. *Secure systems development with UML*. Springer, 2005.
- [11] P. Kruchten, editor. *The rational unified process. An introduction*. Addison-Wesley, 2004.
- [12] T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *UML*, volume 2460 of *Lecture Notes in Computer Science*, pages 426–441. Springer, 2002.
- [13] J. P. McDermott. Abuse-case-based assurance arguments. In *ACSAC*, pages 366–376. IEEE Computer Society, 2001.
- [14] J. P. McDermott and C. Fox. Using abuse case models for security requirements analysis. In *ACSAC*, pages 55–. IEEE Computer Society, 1999.
- [15] G. McGraw. *Software security: Building security in*. Software security. Adison-Wesley, 2006.
- [16] F. Redmill, M. Chudleigh, and J. Catmir. *System safety: HazOp and software HazOp*. Wiley, 1999.
- [17] A. Refsdal, K. E. Husa, and K. Stølen. Specification and refinement of soft real-time requirements using sequence diagrams. In *FORMATS*, volume 3829 of *Lecture Notes in Computer Science*, pages 32–48. Springer, 2005.
- [18] J. Rumbaugh, I. Jacobsen, and G. Booch. *The unified modeling language reference manual*. Addison-Wesley, 2005.
- [19] G. Sindre and A. L. Opdahl. Eliciting security requirements by misuse cases. In *37th Technology of Object-Oriented Languages and Systems (TOOLS-37 Pacific 2000)*, pages 120–131. IEEE Computer Society, 2000.
- [20] G. Sindre and A. L. Opdahl. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, 2005.
- [21] Standards Australia, Standards New Zealand. *Information security risk management guidelines*, 2004. HB 231:2004.
- [22] Symantec. Symantec internet security threat report. Trends for July 05–December 05, March 2006.
- [23] T. Watson and P. Kriens. OSGi component programming. Tutorial held at Eclipsecon 2006, 2006.