UNIVERSITY OF OSLO
Department of Informatics

Using model-driven risk analysis in component-based development

Research report 342

Gyrd Brændeland

Ketil Stølen

ISBN 82-7368-298-6
ISSN 0806-3036

December 2010

Using model-driven risk analysis in component-based development

Gyrd Brændeland Ketil Stølen
SINTEF and University of Oslo

March 2, 2011

Abstract

Modular system development causes challenges for security and safety as upgraded subcomponents may interact with the system in unforeseen ways. Due to their lack of modularity, conventional risk analysis methods are poorly suited to address these challenges. We propose to adjust an existing method for model-based risk analysis into a method for component-based risk analysis. We also propose a stepwise integration of the component-based risk analysis method into a component-based development process. By using the same kinds of description techniques to specify functional behaviour and risks, we may achieve upgrading of risk analysis documentation as an integrated part of component composition and refinement.

Contents

1 Introduction

When your computer crashes just as you are about to take a last print out on your way home it is annoying. But when software glitches cause erratic behaviour of critical functions in your car it may be serious. Products such as cars, laptops, smart phones and mobile devices in general are not sold as finished products, but rely more and more on software components that may be upgraded several times during their lifetime. The problems faced by Toyota in explaining what caused the problem with the sticky accelerators (Ahrens, 2010) illustrate how upgrades may interact with the rest of the system in unforeseen ways. The flexibility offered by modular development facilitates rapid development and deployment, but causes challenges for risk analysis that are not addressed by current methods. By *risk* we mean the combination of the consequence and likelihood of an unwanted event. By *risk analysis* we mean the process to understand the nature of risk and determining the level of risk (ISO, 2009b).

A widely adopted requirement to components is that they need to be distinguished from their environment and other components, in order to be independently deployable. This distinction is provided by a clear specification of the component interfaces and by encapsulating the component implementation (Crnkovic and Larsson, 2002). The same principles of modularity and composition should apply to risk analysis methods targeting component-based systems. A modular understanding of risks is a prerequisite for robust component-based development and for maintaining the trustworthiness of component-based systems. Ultimately, one cannot have component-based development without a modular understanding of risks.

To understand the risks related to deploying a new component or upgrading an existing one is challenging: It requires an understanding not only of the risks of the component itself, but also of how its interactions with the system affect the risk level of the system as a whole and how they affect the risk level of each other. An example is the known buffer overflow vulnerability of previous versions of the media player Winamp, which may allow an unauthenticated attacker using a crafted file to execute arbitrary code on a vulnerable system. By default Internet Explorer opens crafted files without prompting the user (Secunia, 2006). Hence, the probability of a successful attack is much higher if a user utilises both Internet Explorer and Winamp, than only one of them.

The likelihood of an unwanted event depends partly on external factors, such as the likelihood that a threat occurs. In conventional risk analyses the parts of the environment that are relevant for estimating the risk-level are therefore often included in the analysis. Furthermore, in existing risk analysis methods (Alberts et al., 1999; Farquhar, 1991; Swiderski and Snyder, 2004), the environment and the time-frame are viewed as constants. These features of existing risk analysis methods make them poorly suited to analyse modular systems whose threats varies with the environment in which the systems exist (Verdon and McGraw, 2004).

1.1 A framework for component-based risk analysis

The purpose of this report is to present a framework for component-based risk analysis. The objective of component-based risk analysis is to improve component robustness and facilitate reuse of risk analysis results to allow composition of overall analyses more efficiently than analysing systems from scratch. A method for component-based risk analysis should adhere to the same principles of encapsulation and modularity as component-based development methods, without compromising the feasibility of the approach or standard definitions of risk (ISO, 2009b,a; Standards Australia, 2004). To ease the analysis of how changes in a component affects system risks, it should be possible to represent risk related concepts such as assets, incidents, consequences and incident probabilities, as an integrated part of component behaviour. To support robust

component-development in practice it should be easy to combine risk analysis results into a risk analysis for the system as a whole.

In order to evaluate the presented framework with regard to the above requirements to component-based risk analysis, we apply it to a running example involving the development and risk analysis of an instant messaging component for smart phones. It is a fictitious example, but nevertheless represents a realistic case for component development that is of practical relevance.

We also use the case to illustrate the various steps of the proposed framework, to explore existing system specification and risk analysis techniques that may be used to carry out the various tasks involved in component-based risk analysis, and to identify areas where further research is needed in order to obtain a full method for component-based risk analysis.

The proposed framework is based on the CORAS method for model-driven risk analysis. The CORAS method (den Braber et al., 2007) combines risk analysis methods with UML-inspired systems modelling. CORAS has no particular support for component-based development. We believe, however, that UML models are well suited to document and structure risk analysis results at the component level.

1.2 Outline

This report is structured as follows: In Section 2 we explain the basic notions of risk analysis in general and we give an overview of the CORAS method for model-driven risk analysis. We also briefly explain our notion of a component and give an overview of the component development process proposed by Cheesman and Daniels. In Section 3 we explain the notion of risk in a component setting. We also present the overall framework for component-based risk analysis and its stepwise integration into a component-based development process.

In sections 4 to 7 we go through the initial steps of the integrated approach. We use a combination of UML 2.0 notation (Rumbaugh et al., 2005) and sequence diagrams in STAIRS (Haugen and Stølen, 2003; Haugen et al., 2004) for the component specifications and CORAS diagrams (den Braber et al., 2003) for the risk analysis documentation. In Section 6.1 we briefly explain the STAIRS approach and how it can be used to capture probabilistic behaviour, which is a prerequisite for representing risks. In Section 6.2 we present an extension to CORAS with facilities for documenting assumptions of a risk analysis, which is used to obtain modularity of risk analysis results. In Section 7.2 we introduce a calculus for composition of risk analysis results.

In Section 8 we attempt to place our work in relation to ongoing research within related areas. Finally, in Section 9, we summarise our findings, discuss the extent to which our requirements to a component-based risk analysis method is met and point out what remains in order to achieve a full method.

2 Background

This report addresses the intersection of risk analysis and component-based development. As these notions may mean different things in different settings and communities, we begin by explaining our notions of risk analysis and component.

2.1 Risk analysis

We explain the concepts of risk analysis and how they are related to each other through a conceptual model, captured by a UML class diagram (Rumbaugh et al., 2005) in Figure 1. The conventional risk concepts are adapted from international standards for risk management (ISO,

2009a,b; Standards Australia, 2004). The associations between the concepts have cardinalities specifying the number of instances of one element that can be related to one instance of the other. The hollow diamond symbolises aggregation and the filled composition. Elements connected with an aggregation can also be part of other aggregations, while composite elements only exist within the specified composition.

There are many forms and variations of risk analysis, depending on the application domain, such as finance, reliability and safety, or security. In finance risk analysis is concerned with balancing potential gain against risk of investment loss. In this setting a risk can be both positive and negative. Within reliability/safety and security risk analysis is concerned with protecting what is already there. The first approach may be seen as offensive risk analysis, while the latter may be seen as defensive risk analysis. This report focuses upon defensive risk analysis.

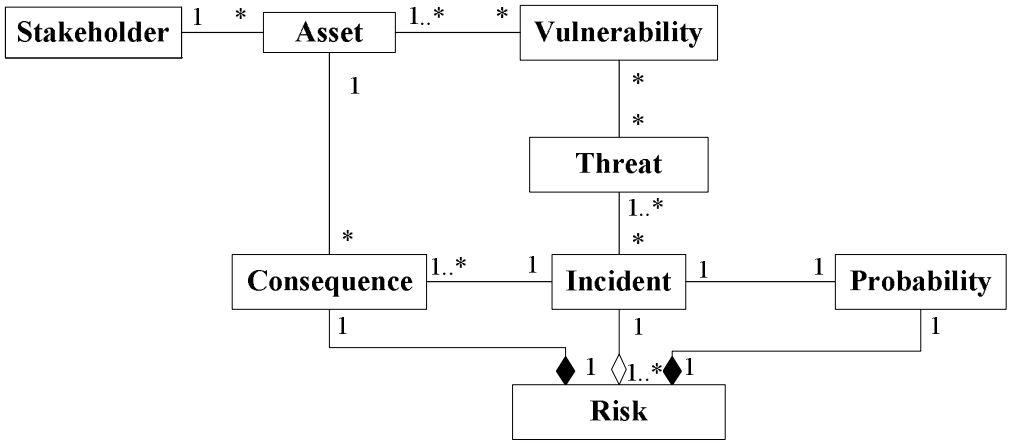

Figure 1: Conceptual model of risk analysis

We explain the conceptual model as follows: *Stakeholders* are those people and organisations who are affected by a decision or activity and on whose behalf the risk analysis is conducted. An *asset* is something to which a stakeholder directly assigns value and, hence, for which the stakeholder requires protection. An asset is uniquely linked to its stakeholder. An *incident* is an event with negative consequences for the assets to be protected. Within the safety domain an incident may for example be a discharge of toxic chemicals or nuclear reactor melt down. In the security domain an incident may be a confidentiality breach, for example due to theft or a human blunder, compromised integrity of information of the system itself or loss of service availability. A *consequence* is the outcome of an event that affects assets. A *vulnerability* is a weakness which can be exploited by one or more threats. A *threat* is a potential cause of an incident. It may be external (e.g., hackers or viruses) or internal (e.g., system failures). Furthermore, a threat may be intentional, that is, an attacker, or unintentional, that is, someone causing an incident by fault or by accident. *Probability* is the extent to which an incident will occur. Conceptually, as illustrated by the UML class diagram in Figure 1, a *risk* consists of an incident, its probability and consequence with regard to a given asset. There may be a range of possible outcomes associated with an incident. This implies that an incident may have consequences for several assets. Hence, an incident may be part of several risks.

2.1.1 Model-driven risk analysis

The *CORAS* method for model-driven risk analysis offers specialised diagrams to model risks. The CORAS modelling language consists of a graphical and a textual syntax and semantics. It

was originally defined as a UML profile, and has later been customised and refined in several aspects, based on experiences from industrial case studies, and by empirical investigations. The CORAS method also consist of a step-by-step description of the risk analysis process, with a guideline for constructing the CORAS diagrams; and the CORAS tool for documenting, maintaining and reporting risk analysis results.

As illustrated by Figure 2, the CORAS process is divided into eight steps (Lund et al., 2010). The first four of these steps are introductory in the sense that they are used to establish a common understanding of the target of the analysis and to make the target description that will serve as a basis for the subsequent steps. This includes all assumptions about the context or setting in which the target is supposed to work as well as a complete list of constraints regarding which aspects of the target should receive special attention, which aspects can be ignored, and so on. The remaining four steps are devoted to the actual detailed analysis. This includes identifying concrete risks and their risk level as well as identifying and assessing potential treatments.

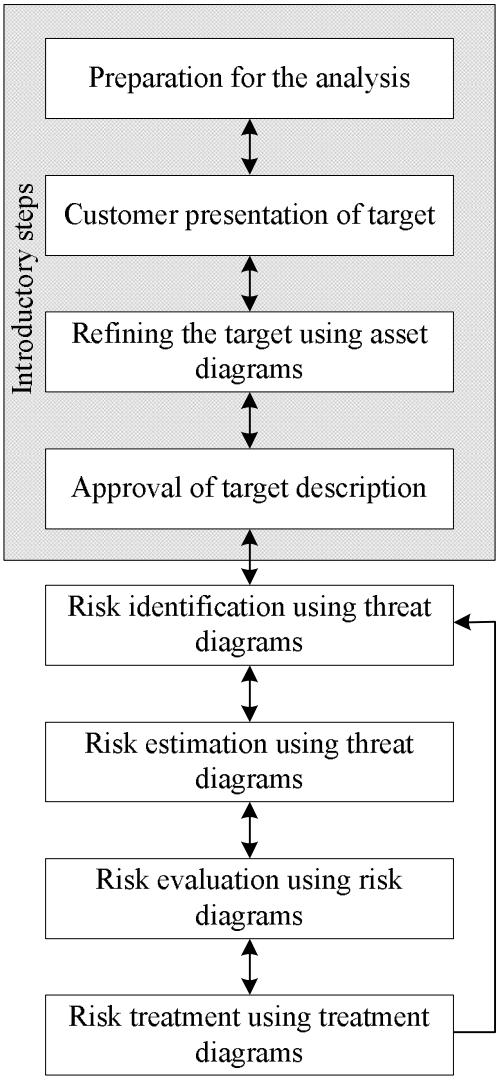

Figure 2: The eight steps of the CORAS method

The risk analysis process is iterative, indicated by the double arrows between each step. During the risk identification step it may for example be necessary to go back and make more

detailed target descriptions. Also the overall risk analysis process is iterative: ideally, the target should be analysed anew, after treatments have been identified, to check whether the treatments have the desired effect and no critical side effects.

2.2 Component-based development

Component-based development encompasses a range of different technologies and approaches. It refers to a way of thinking, or a strategy for development, rather than a specific technology. The idea is that complex systems should be built or composed from reusable components, rather than programmed from scratch. A development technique consists of a syntax for specifying component behaviour and system architectures, and rules (if formal) or guidelines (if informal/semiformal) for incremental development of systems from specifications.

Our component model is illustrated in Figure 3. An *interface* is a contract describing both

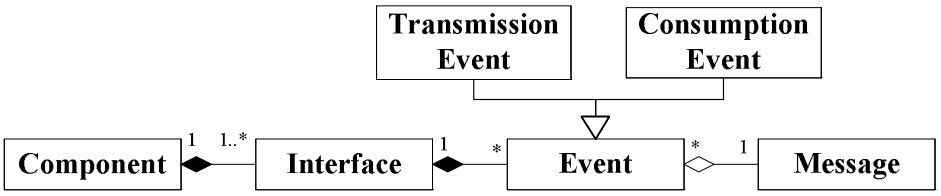

Figure 3: Conceptual component model

the provided operations and the services required to provide the specified operations. A *component* is a collection of interfaces some of which may interact between themselves. Interfaces interact by the transmission and consumption of messages. We refer to the transmission and consumption of messages as events.

Our component model is inspired by the one defined by Cheesman and Daniels (2001), but in order to keep the component model simple and general we have made some adjustments. According to the component definition of Cheesman and Daniels, a component specification is a realisation contract describing provided interfaces and component dependencies in terms of required interfaces. A provided interface is a usage contract, describing a set of operations provided by a component object. As shown in our conceptual component model, we do not distinguish between usage and realisation contracts. An interface is our basic unit of specification.

2.2.1 Model-driven component development

For the case of the presentation we have followed a process for component development using UML diagrams, proposed by Cheesman and Daniels (2001). However, our approach is not restricted to following this particular development process.

UML (OMG, 2007) is a semiformal description and modelling language. By semiformal we mean a technique based on semiformal description and modelling languages, which is seemingly formal but lack a precisely defined syntax, or contains constructs with an unclear semantics (Broy and Stølen, 2001). There is a large number of semiformal development techniques built up around UML, such as for example RUP (Rational Unified Process) (Kruchten, 2004). RUP is a framework for an iterative software development process structured into a series of so called workflows. Each workflow produces an artefact that is used as input in the next workflow. The process proposed by Cheesman and Daniels resembles RUP, but is specifically tailored towards component-based development. A benefit of using a UML-based method is that UML is the de facto industry standard for system specification, and therefore familiar to most system developers. Moreover, since the CORAS risk modelling language is adapted from

UML diagrams, the normal behaviour and the risk behaviour of components can be modelled using the same kind of diagrams.

Figure 4 shows the overall development process proposed by Cheesman and Daniels. The

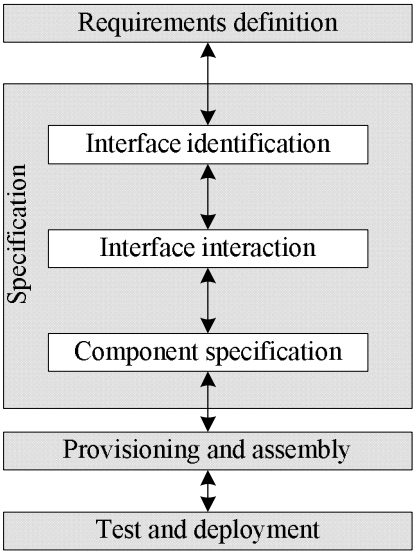

Figure 4: The workflows of the component development process

grey boxes represent workflows as defined in RUP. The process starts by describing the overall component requirements, such as functional requirements and quality of service requirements. During the requirements workflow the component is viewed as a black box, any internal behaviour of sub-parts is hidden. The requirements workflow should deliver a business concept model and a set of use cases to the specification workflow. A business concept model is a conceptual model of the business domain that needs to be understood and agreed. Its main purpose is to create a common vocabulary among the business people involved with in project. A use case describes interactions between a user (or other external actor) and the system, and therefore helps to define the system boundary (Cheesman and Daniels, 2001). During the specification workflow the component is decomposed into interfaces that are refined further, independently of each other. It entails identifying interfaces, describing interface interaction and dependencies and specifying how the interfaces can be fitted together into a component that refines the original requirements. The output from the specification workflow is used in the provisioning workflow to determine what interfaces to build or buy, in the assembly workflow to guide the correct integration of interfaces, and in the test workflow as input to test scripts.

We use sequence diagrams in STAIRS (Haugen and Stølen, 2003; Haugen et al., 2004) to specify the interface interactions in the specification workflow. STAIRS is a formal approach to system development with UML sequence diagrams that supports an incremental and modular development process. STAIRS assigns a formal trace semantics to sequence diagrams and defines refinement relations for specifications and compliance relations for implementations. STAIRS is not part of the process developed by Cheesman and Daniels. The benefit of using STAIRS to specify interface interactions is that it supports incremental development through refinement.

3 Component-based risk analysis and development

We propose to integrate the process of risk analysis into a component-based development process. In order to obtain a method for component-based risk analysis, we need to understand the meaning of risk in a component setting. In particular, we need to understand which risk concepts to include at the component level, without compromising the modularity of our components. In Section 3.1 we present a conceptual model for component-based risk analysis that relates some of the concepts from risk analysis to the conceptual component model. In Section 3.2 we give an overview of our approach for integrating risk analysis into a component-based development process.

3.1 Component-based risk analysis concepts

Figure 5 shows how the conceptual model of risk analysis relates to the conceptual component model. We identify assets on behalf of component interfaces as illustrated in Figure 5. Each

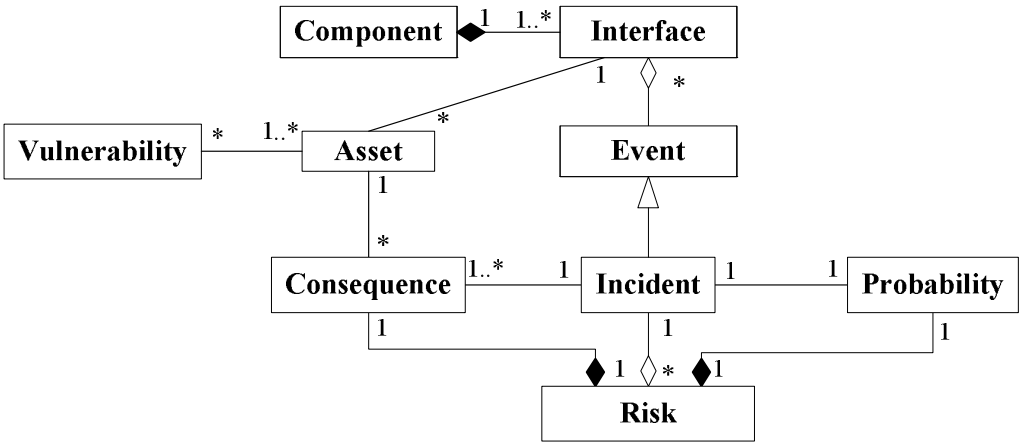

Figure 5: Conceptual model of component-based asset-driven risk analysis

interface has a set of assets. Hence, the concept of a stakeholder is implicitly present in the integrated conceptual model, through the concept of an interface[1]. An incident refers to an event of an interface that harms at least one of its assets. An event is as explained above either the consumption or the transmission of a message by an interface. Moreover, a consequence is a measure on the level of seriousness of an incident with regard to an asset. The concept of a threat is not part of the integrated conceptual model as a threat is something that belongs to the environment of a component.

3.2 Integrating risk analysis into component-based development

As already mentioned, we adapt the CORAS method to make it suitable for component-based risk analysis. We aim, in particular, to integrate risk analysis into the early stages of component development. We have structured the adapted method to correspond with the requirements and specification workflow in the Cheesman and Daniels process.

Figure 6 gives an overview of the integration of the adapted method into the development process of Cheesman and Daniels. For each step in the requirements and specification workflow,

[1]Note that there may be interfaces with no assets; in this case the stakeholder corresponding to the interface has nothing to protect.

we first conduct the development step and then conduct the corresponding risk analysis step.

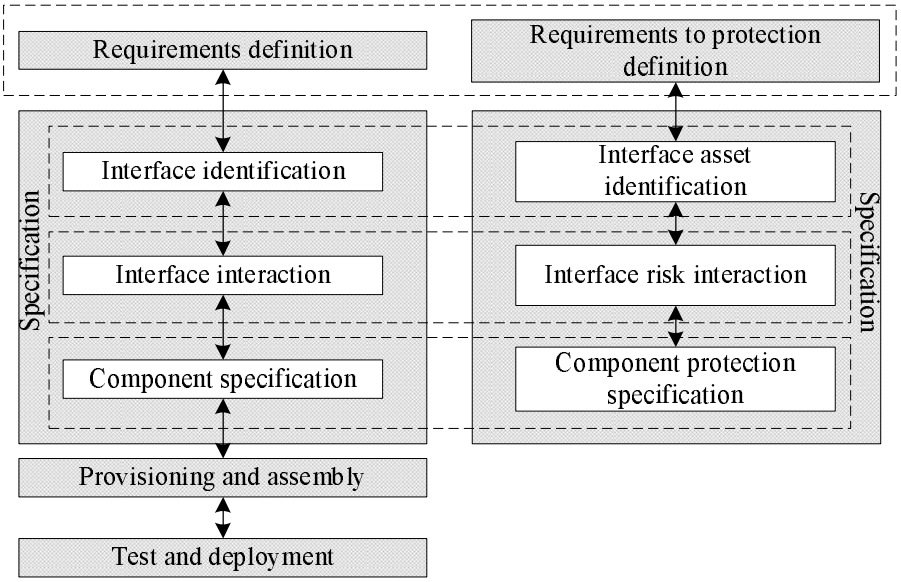

Figure 6: Integrating risk analysis into component-based development

Figure 7 shows how the workflows of the adapted method relate to the steps in the CORAS method. We have left out the steps *Preparation for the analysis* and *Customer preparation of target* of the CORAS method, as they are taken care of by the component specification workflows. We have collapsed the tasks of the introductory steps *Refine the target description using asset diagrams* and *Approval of target description* into one workflow that we call *Requirements to protection definition*. While the requirements definition captures the quality of service and functional requirements, the requirements to protection specify the acceptable level of risk, that is, what may be tolerated with respect to risks.

In component-based risk analysis we need to describe assets at the level of component interfaces. We have therefore included the step *Interface asset identification* as part of the specification workflow of the risk analysis process. This step is not part of the original CORAS process as decomposition of assets is not required in conventional risk analysis.

We augment the specification of the interface interactions with specifications of the interface risk interactions. Even if the component specification is verified to refine the component requirements this of course does not mean that the requirements to protection are fulfilled. In addition to specifying the ordinary component behaviour we must therefore also characterise its way of protection.

In the following sections we go through the initial steps of the integrated approach to component-based development and risk analysis in more detail. The presentation is structured into four sections (Sections 4 through 7.1) corresponding to the early stages in a component-based development process.

We focus our presentation on the points where the component-oriented approach to risk analysis differs from the CORAS process. For a full presentation of the CORAS method we refer to the book *Model driven risk analysis. The CORAS approach* by Lund et al. (2010) and for a presentation of the component-based system development process we refer to the book *UML Components. A simple process for specifying component-based software* by Cheesman and Daniels.

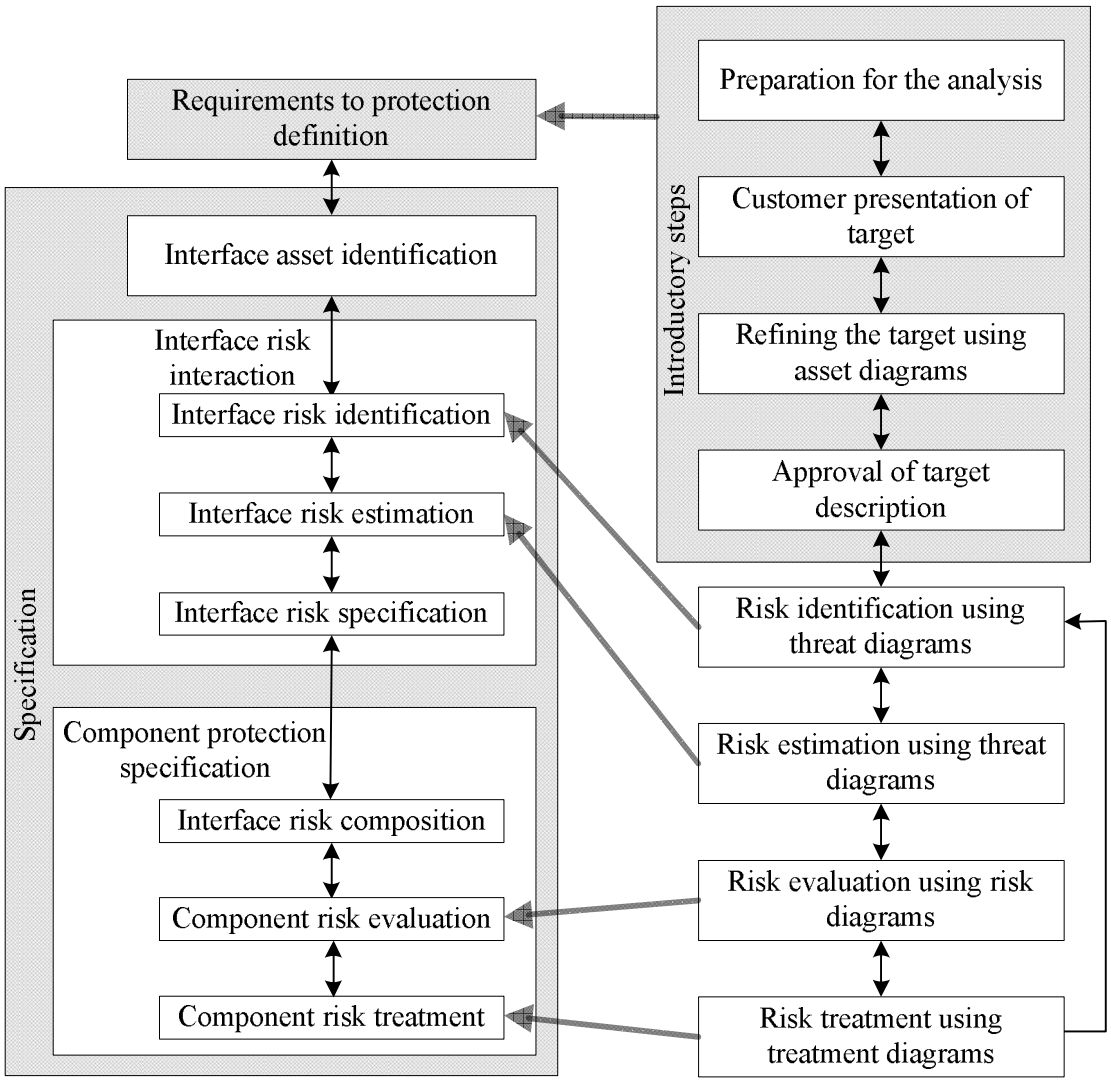

Figure 7: Adapting CORAS into a component-based risk analysis process

4 Requirements

In this Section we explain how to perform the requirements to protection definition step of the component-based risk analysis method, based in the requirements definition. In accordance with the integrated process described in the previous section we first give the requirements definition (Section 4.1), and then present the requirements to protection definition (Section 4.2). As shown in Figure 7, the requirements to protection definition cover the four introductory steps of the CORAS method which includes identifying assets and establishing their required protection level. In Section 4.2 we explain how these tasks may be adapted to comply with the principles of modularity and encapsulation of component development. A summary of the adjustments is given in Table 6.

4.1 Requirements definition

The purpose of requirements definition is to describe what services the component should provide, allocate responsibilities for the various services and to decide upon the component boundary (Cheesman and Daniels, 2001).

4.1.1 Use cases

Cheesman and Daniels employ use cases to identify the actors interacting with the component, and list those interactions. A use case helps to define the boundary of the component. One actor is always identified as the actor who initiates the use case; the other actors, if any, are used by the system (and sometimes the initiating actor) to meet the initiator's goal (Cheesman and Daniels, 2001).

In the following we introduce the case that will serve as a running example throughout the presentation of the integrated component development and risk analysis process.

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 (Watson and Kriens, 2006). It is a fictitious example, but nevertheless represents a realistic case for component development that is of practical relevance.

Example 1 (Use cases for the instant messaging component) The instant messaging component should allow users to interact in chat sessions and exchange media files with buddies, organised in a peer-to-peer fashion, as illustrated in Figure 8. It should be possible to deploy and run the service on smart phones; laptops et cetera running on a dynamic component platform.

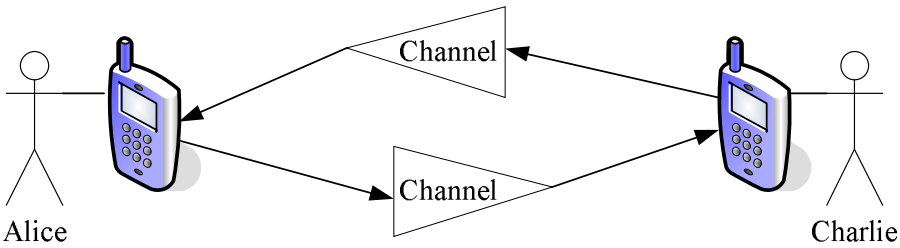

Figure 8: Peer-to-peer instant messaging

Buddies use *Channel* interfaces to interact with each other. A *Channel* is one way. A user of an instant messaging component can receive messages through her own *Channel* and send messages to buddies through their *Channels*.

The UML use case diagram in Figure 9 shows the actors interacting with the instant messaging component. The actor *User* initiates the use cases *User login*, *List buddies*, *Send message*

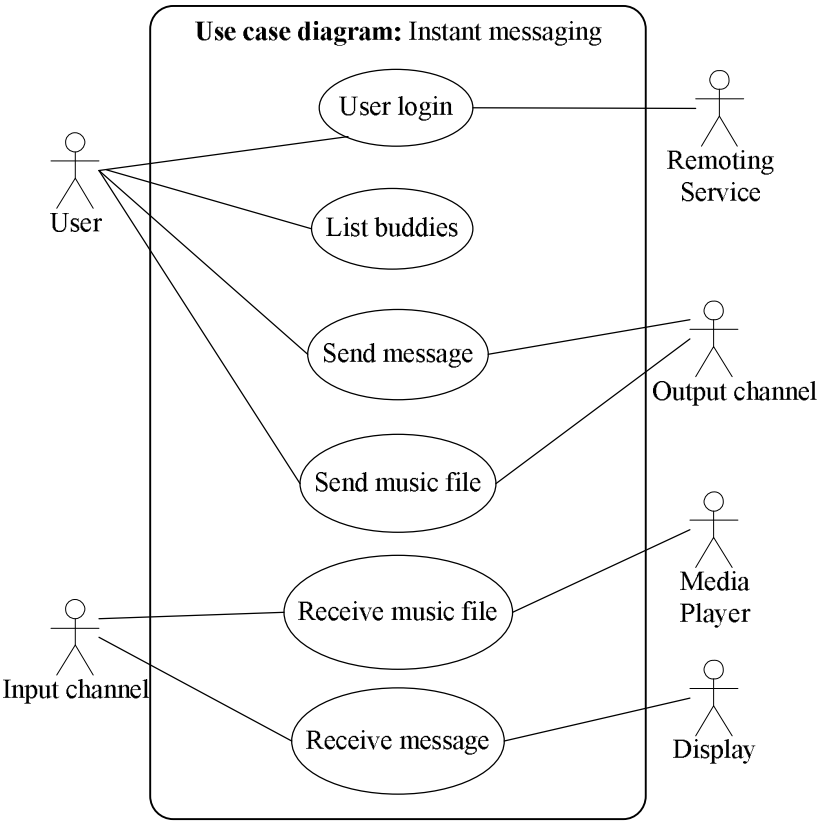

Figure 9: Use case diagram

and *Send music file*. We assume the instant messaging component uses an external service represented through the actor *Remoting service*, which handles discovery of other services and registers the messaging service. In order to perform the actions involved in the *Send message* and *Send music file* use cases the instant messaging component employs an actor *Output channel*.

The actor *Input channel* initiates the use cases *Receive music file* and *Receive message*. To perform the actions involved in the *Receive music file* and *Receive message* use cases the instant messaging component employs services provided by the actors *Media player* and *Display*, respectively.

Cheesman and Daniels break use cases down into steps and use those steps to identify the system operations needed to fulfil the system's responsibilities. For simplicity we leave out the detailed use case descriptions here. □

4.1.2 Business concept model

A business concept model is a conceptual model of the business domain that needs to be understood and agreed. Its main purpose is to create a common vocabulary among the business people involved in the project. Hence, the business concept model should include the informational concepts that exist in the problem domain.

Example 2 (Business concept model for the instant messaging component) The business concept model is a conceptual model of the information that exists in the problem domain. Based in the Use case diagram we identify four main informational concepts: *User*, *Buddy*, *Music file* and *Message*. We use a UML class diagram (Figure 10) to depict the various concepts and

the relations between them. The associations between the concepts have cardinalities specifying the number of instances of one element that can be related to one instance of the other. The

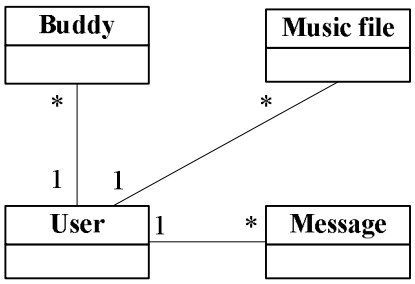

Figure 10: Business concept model

business concept shows the informational concepts of a single instant messaging component. A user of an instant messaging component may send and receive several messages and music files and have several buddies. The associations between *User* and the other informational concepts are therefore one-to-many. □

4.2 Requirements to protection definition

The purpose of the requirements to protection definition is to establish the accepted level of risk towards component assets. An asset is something which is of value and that should be protected from harm.

Prior to establishing the accepted level of risks towards assets, the CORAS method requires that the following sub-tasks are conducted: describing the target of analysis; identifying stakeholders; and identifying and value assets. As explained in Section 2.1 the stakeholders are the asset-owners, on whose behalf the risk analysis is conducted.

The goal of describing the target of analysis is to define the exact boundaries of the component that will be assessed. In conventional risk analysis the target of analysis may be a system, a part of a system or a system aspect. In component-based risk analysis we identify the target of analysis as the component or component interface being analysed.

Due to the overall requirement that the risk analysis results must comply with the same principles of modularity as the component specifications, both components and their associated risk attributes must be self-contained. This means that we cannot have a specification that requires knowledge about external actors or stakeholders. In component-based risk analysis we therefore identify assets on behalf of the component or component interface which is the target of analysis.

During the requirements to protection definition workflow we identify system level assets. After we have identified component interfaces we must assign the system interfaces to the interfaces they belong to and identify business level assets, that is, assets belonging to business interfaces, if any such exists. Since an interface seldom is a human being, however, decisions regarding assets and their protection level must be done by the component owner, or the development team in an understanding with the component owner.

4.2.1 Identify component assets

An asset may be physical, such as data handled by a component interface. It may also be purely conceptual, such as for example the satisfaction of the component user or it may refer to properties of data or of a service, such as confidentiality or availability. We use the use case diagrams and the business concept model in as input to identify component assets.

Example 3 (Assets of the instant messaging component) The result of the asset identification is documented in a CORAS asset diagram shown in Figure 11.

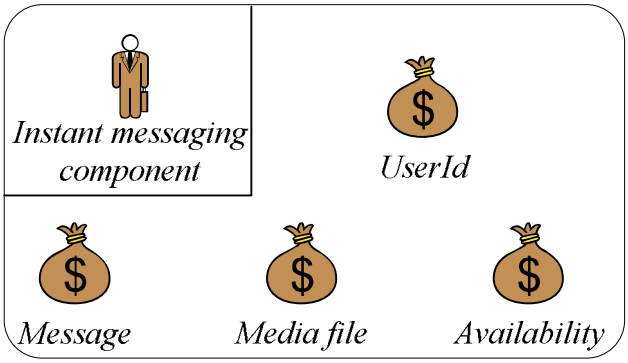

Figure 11: Asset identification

As already we identify assets at component level on behalf of the *Instant messaging component* itself. The business concept model in Figure 10 shows the type of information that exists in the problem domain. We use this to identify three informational assets of value for the instant messaging component: *UserId*, *Message* and *Media file*.

The use case diagram in Figure 9 shows several use cases involving the sending and reception of information. We identify *Availability* as an asset of the component, implying that the timely operation of these use cases is a valuable feature that we wish to protect. □

4.2.2 Establish the level of protection

The protection level for an asset is decided by the requirements to protection. The requirements to protection definition should serve as input to the component protection specification workflow, where we check whether the requirements are fulfilled by the component specification. If the component specification does not fulfil the requirements to protection, it should be revised. A risk is the potential of an incident to occur. The risk level of a risk is a function of the likelihood of the incident to occur and its consequence (Lund et al., 2010).

Likelihood values are given as frequencies or probabilities. Consequence, in terms of harm to one or more component assets, is a measure on the level of seriousness of an incident. Likelihood and consequence scales may be qualitative (e.g., *Unlikely*, *Likely* and *Minor*, *Moderate*, *Major*), or quantitative (e.g., 0.1, 0.8 and 100$). Qualitative values can be mapped to concrete values. A *Minor* consequence with regard to availability may for example correspond to at most one minute delay in response time, while a *Major* consequence may correspond to a delay for more than one hour.

A common way to represent a risk function is by a coordinate system with consequence values on the y-axis and likelihood values on the x-axis. Each entry in the matrix represents a risk value.

Example 4 (Likelihood scale and risk functions) For the purpose of this example we use the likelihood scale *Unlikely*, *Possible*, *Likely*, *Almost certain* and *Certain*. Each linguistic term is mapped to an interval of probability values in Table 1. We use the consequence scale *Minor*, *Moderate*, *Major*. For the case of simplicity we do not provide quantitative consequence values in this example.

Tables 2 to 5 define the risk functions for the four assets. We have only two risk values: *high* and *low*, where the grey areas represent high and the white represents low. The risk values decide the requirements to protection: risks with a low risk value are acceptable, while risks

Likelihood	Description
Unlikely	$\langle 0.00, 0.25]$[2]
Possible	$\langle 0.25, 0.50]$
Likely	$\langle 0.50, 0.70]$
Almost certain	$\langle 0.70, 0.99]$
Certain	$\{1.00\}$

Table 1: Likelihood scale for probabilities

with a high risk value are unacceptable and must be treated. So for example we do not accept risks towards the *UserId* asset with consequence *Moderate* and likelihood *Possible* or higher. □

Consequence	Likelihood				
	Unlikely	Possible	Likely	Almost certain	Certain
Minor					
Moderate					
Major					

Table 2: Protection criteria for *UserId*

Consequence	Likelihood				
	Unlikely	Possible	Likely	Almost certain	Certain
Minor					
Moderate					
Major					

Table 3: Protection criteria for *Message*

5 Interfaces and their assets

In this Section we explain how to identify assets at the interface level, based on the requirements to protection definition and the interface identification. In accordance with the integrated process, we first conduct interface identification (Section 5.1); thereafter we identify the assets of each interface (Section 5.2). As illustrated in Figure 7, the interface asset identification step is not part of the original CORAS process.

5.1 Interface identification

Interface identification is the first stage of the specification workflow. It entails decomposing the component into interfaces. Cheesman and Daniels (2001) distinguish between two layers of a component: the system layer and the business layer. The system layer provides access to the services of the component. It acts as a facade for the layer below. The business layer implements the core business information and is responsible for the information managed by the component. The use case diagram from the requirements workflow guides the identification of system interfaces and the business concept model guides the identification of business interfaces.

[2]We use $\langle a, b\rangle$ to denote the open interval $\{x \mid a < x < b\}$.

Consequence	Likelihood				
	Unlikely	Possible	Likely	Almost certain	Certain
Minor					
Moderate					
Major					

Table 4: Protection criteria for *Availability*

Consequence	Likelihood				
	Unlikely	Possible	Likely	Almost certain	Certain
Minor					
Moderate					
Major					

Table 5: Protection criteria for *Media file*

Summary of component-based risk analysis: workflow 1

- **Objective:** Establish the accepted level of risk towards component assets.
- **Input documentation:** The business concept model and use cases delivered from the requirements definition.
- **Output documentation:** Asset diagrams, likelihood scale and for each direct asset; a consequence scale, risk function and requirements to protection.
- **Adaptations to CORAS:**
  1. The target of analysis is the component itself.
  2. We identify assets on behalf of the component.

Table 6: Requirements to protection definition

5.1.1 Identifying system interfaces and operations

A use case indicates the types of operations that the component should offer through interfaces. Cheesman and Daniels (2001) propose to define one system interface per use case. For each step in the use case description they consider whether there are system responsibilities that need to be modelled. A system responsibility is represented by one or more operations of the responsible system interface.

Example 5 (Instant messaging system interfaces) As explained in Section 4 there are two types of external actors that can initiate use cases: the actor *User* initiates the use cases *User login*, *List buddies*, *Send message* and *Send music file* and the actor *Input channel* initiates the use cases *Receive music file* and *Receive message*. For simplicity we leave out the descriptions of steps involved in a use case in this example and we mostly let one use case correspond to one operation.

We group the two operations corresponding to the use cases *Receive music file* and *Receive message* initiated by *Input channel* into one interface *Channel*. The operations of the interface *Channel* are invoked by other instant messaging components when a buddy attempts to transfer messages or files.

With regard to the operations corresponding to use cases initiated by *User* we consider that sending messages, listing buddies and login are related to chatting and group them together in one interface called *Chat*. The operation corresponding to the use case, *Send music file*, gets its own interface that we call *FileTransfer*.

Figure 12 shows the interface types, that is, the interface name and the list of operations it provides. Inspired by Cheesman and Daniels (2001) we use the stereotypes ≪interface type≫, rather than applying the predefined UML modelling element ≪interface≫, which is used for modelling the implementation level. □

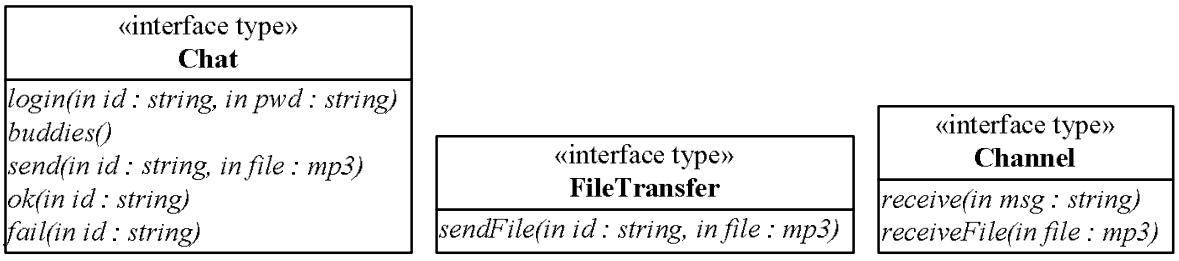

Figure 12: System interface types

5.1.2 Identifying business interfaces

In order to identify business interfaces, Cheesman and Daniels refine the business concept model into a business type model. The purpose of the business type model is to formalise the business concept model to define the system's knowledge of the outside world.

Example 6 (Instant messaging business interfaces) For the purpose of the example, we assume that we have refined the business concept model into a business type model providing the necessary input. The only type of information that the instant messaging component itself manages, is the user id. Hence, we identify one business interface *UserMgr*, shown in Figure 13. Other types of information are handled by external interfaces. The *UserMgr* interface has an operation *validate*, for checking that the user information is correct. □

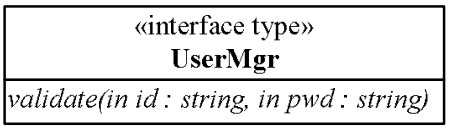

Figure 13: Business interface type

5.1.3 Interface dependencies

It is also necessary to identify existing interfaces that are part of the environment into which the instant messaging component will be deployed. In order to specify interface dependencies we introduce the stereotype ≪interface spec≫. The stereotype ≪interface spec≫ resembles the predefined UML modelling element ≪component≫, but is used to model the specification rather than the implementation level. The interface dependencies are detailed further as part of the interface interaction specification.

Example 7 (Instant messaging dependencies) As already explained, the instant messaging component provides an interface *Channel*, which may receive messages and files from other instant messaging services. The *Channel* interface is one way. In order to implement the operations described by the *Send message* and *Send music file* use cases, the instant messaging component employs the *Channel* interface of the buddy's instant messaging component, as illustrated in the use case diagram in Figure 9.

Hence, both the interfaces *FileTransfer* and *Chat* require an external interface *Channel* to implement their operations. We specify which interfaces an interface requires through the use of sockets in Figure 14. The lollipop at the top of each interface symbolises the connection

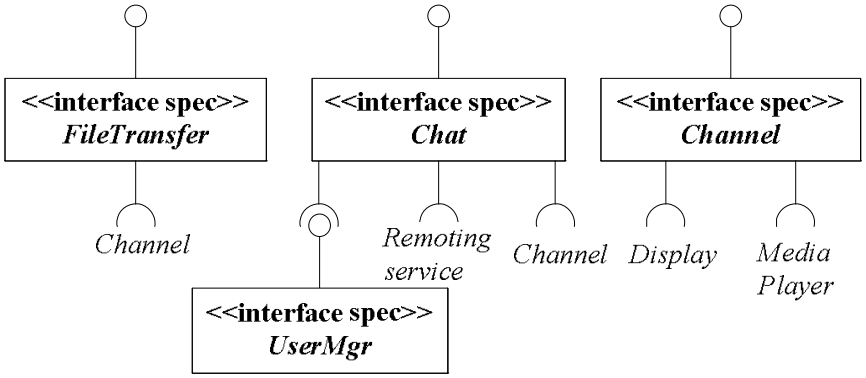

Figure 14: Interface dependencies

point through which other interfaces may employ the services of the interface. For example, the required interface *Channel* indicated by the socket of the *Chat* interface, is the *Channel* interface it requires in order to send messages to buddies. This is not the same as the provided interface *Channel* of the instant messaging component.

We also identified the actor *Remoting service* that the instant messaging component employs to handle discovery of other services and registering the messaging service, and actors *Display* and *MediaPlayer* that the instant messaging component employs to implement the operations described by the *Receive message* and *Receive music file* use cases, respectively. Thus, the *Chat* service requires an interface *Remoting service* for remoting services and the *Channel* interface requires interfaces *MediaPlayer* and *Display* in order to display messages or play music files received from buddies.

In Section 5.1.2 we identified an interface *UserMgr* that is responsible for managing the user id. This interface is used by the interface *Chat* to implement the operations of the *User login* use case. □

5.2 Interface asset identification

In the previous section we decomposed the component into interfaces. The point of this is that we can refine the interface specification independently of each other, before they are fitted together during the component specification step. Such a modular approach facilitates reuse and maintenance of sub-components and interfaces. For the same reason we want to identify and analyse risks at the interface level and then combine the interface risks into a risk picture for the component as a whole during the component protection specification step. In order to be able to analyse risks at the interface level we must decide for each component asset which of the component interfaces it belongs to. According to our conceptual model of component-based risk analysis in Figure 5 the set of component assets is the union of the assets of its interfaces.

We use the interface specifications to guide the process of assigning assets to interfaces. As already explained assets may be physical, such as data handled by a component interface, or properties of data or of a service, such as confidentiality or availability. We use the rule of thumb that assets referring to data are assigned to the interfaces handling the data. Assets referring to properties of data or services are assigned to the interfaces handling the data or contributing to the services for which the properties are relevant.

In order to evaluate risks at the component level, the risk analyst must decide how to compute the harm towards component assets from harm towards its constituent assets.

Example 8 (Assets of the instant messaging interfaces) During the requirements workflow we identified assets on behalf of the instant messaging component. In Section 5.1 we decomposed the instant messaging component into four interfaces: *FileTransfer*, *Chat* and *Channel* and *UserMgr*.

Figures 15 to 17 show the interface assets. The asset *UserId* refers to the informational content of the user ID which is handled both by the *Chat* and the *UserMgr* interfaces. We therefore decompose this asset into two: *Chat UserId* which we assign to the *Chat* interface, and *UserMgr UserId* which we assign to the *UserMgr* interface. The asset *Message* refers to

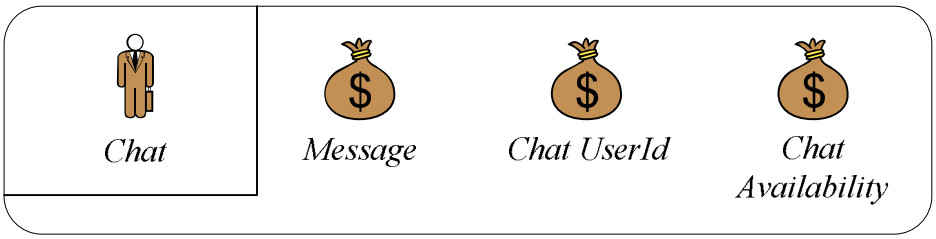

Figure 15: Asset identification for the *Chat* interface

messages sent from a user to a buddy. This task is included in the *Send message* use case, which we assigned to the *Chat* interface. Hence, we assign the asset *Message* to the *Chat* interface. The asset *Media file* refers to media files sent from a buddy to a user. This task is included in the *Receive music file* use case, which we assigned to the *Channel* interface and we assign the *Media file* asset to that interface. The asset *Availability* refers to the time the instant messaging component uses to respond to operation calls. *Availability* is of relevance to all three system interfaces (*FileTransfer*, *Chat*, *Channel*). We therefore decompose this asset into three assets:

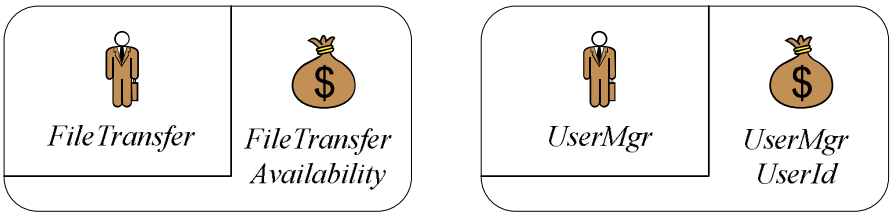

Figure 16: Asset identification for the *FileTransfer* and *UserMgr* interfaces

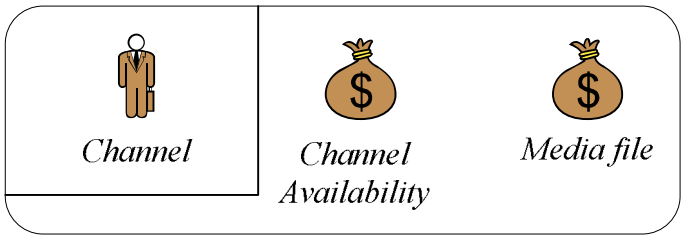

Figure 17: Asset identification for the *Channel* interface

FileTransfer Availability, *Chat Availability* and *Channel Availability* and assign one to each system interface.

As mentioned earlier we must decide how to compute the harm towards component assets from harm towards its constituent assets. For example an incident harming any of the interface assets *Chat UserId* or *UserMgr UserId* will constitute an incident with regard to the component asset *UserId*, since this is the union of the two interface assets. For simplicity, in this example, we have decided that harm towards an interface asset constitute the same level of harm towards the corresponding component asset. Hence, the risk protection matrices for *Chat UserId* and *User Mgr UserId* are the same as the one defined for *UserID* and the risk protection matrices for the *FileTransfer Availability*, *Chat Availability* and *Channel Availability* assets are the same as the *Availability* risk protection matrix. □

Summary of component-based risk analysis: workflow 2, step 1
• **Objective:** Assign assets to interfaces.
• **Input documentation:** The asset diagrams from the requirements to protection workflow and the interface specification diagrams from the interface identification step in the specification workflow.
• **Output documentation:** Interface asset diagrams.
• **Adaptations to CORAS:** This step is not part of the original CORAS process.

Table 7: Interface asset identification

6 Interactions

In this Section we explain how to specify the interface risk interactions, based on the specification of the interface interactions. In accordance with the integrated process described in Section 3 we first describe the normal interactions (Section 6.1), and then the risk interactions (Section 6.2).

The normal interactions describe how each of the interfaces identified in the previous section use other interfaces in order to implement their operations. The risk interactions capture how the normal interactions can be mis-used in order to cause incidents that harm the identified interface assets. In order to specify risk interactions we first identify and estimate risk using threat diagrams. These steps are part of the original CORAS process, but should follow certain conventions in order to comply with the principles of modularity and encapsulation of component development. The integration of risk behaviour as part of the interface specification is not part of the original CORAS method.

6.1 Interface interactions

Cheesman and Daniels (2001) use UML 1.3 collaboration diagrams to specify the desired interactions between component objects. UML 1.3 collaboration diagrams correspond to communication diagrams in UML 2.0. A UML 1.3 collaboration diagram can focus on one particular component object and show how it uses the interfaces of other component objects. According to Cheesman and Daniels sequence diagrams can be used instead of collaboration diagrams. They prefer collaboration diagrams because they show the relationship between diagrams.

We use sequence diagrams in STAIRS (Haugen and Stølen, 2003; Haugen et al., 2004) to specify interface interactions. STAIRS is a formalisation of the main concepts in UML 2.0 sequence diagrams. A sequence diagram shows messages passed between two or more roles (interfaces in our case), arranged in time sequence. An interface is shown as a lifeline, that is, a vertical line that represents the interface throughout the interaction. A message can also come from or go to the environment (that is, outside the diagram). The entry and exit points for messages coming from or going to the environment are called gates (Rumbaugh et al., 2005). The sequence diagram in Figure 18 specifies a scenario in which the *Chat* interface consumes a message *send(id,msg)* and then transmits the message *receive(msg)* to a *Channel* interface.

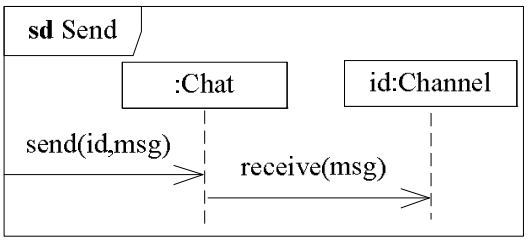

Figure 18: Example of a sequence diagram

In addition to defining semantics for existing UML operators, STAIRS also introduces a new choice operator called xalt. This operator is introduced to allow a distinction between inherent nondeterminism (also called mandatory behaviour) and underspecification in terms of potential behaviour where only one alternative need to be present in a final implementation. For describing potential behaviour, the common UML alt operator is used, while a xalt is used to capture mandatory behaviour and distinguish this from potential behaviour. (Runde, 2007; Refsdal, 2008). Formally, the operands of a xalt result in distinct interaction obligations in order to model the situation that they must all be possible for an implementation.

The following example borrowed from Solhaug (2009) illustrates the difference between alt and xalt: A beverage machine should offer both coffee and tea, where coffee can be offered as americano or espresso. If this is specified by (americano alt espresso) xalt tea, the machine must always offer the choice between coffee and tea since it is represented by inherent nondeterminism. A machine that can only serve espresso if coffee is chosen fulfils the specification since this

alternative is represented by underspecification.

Probabilistic STAIRS (pSTAIRS) is an extension of STAIRS for specifying probabilistic requirements. pSTAIRS introduces a generalisation of the xalt operator, palt, which is meant to describe the probabilistic choice between two or more alternative operands whose joint probability should add up to one. For the purpose of specifying mutually exclusive probabilistic alternatives pSTAIRS also introduces the operator expalt. See Refsdal (2008) for a full description of probabilistic STAIRS.

STAIRS uses denotational trace semantics in order to explain the meaning of a sequence diagram. A trace is a sequence of events. There are two kinds of events: transmission and consumption of a message, where a message is a triple consisting of a signal, a transmitter and a consumer. The set of traces described by a diagram like that in Figure 18 are all positive traces consisting of events such that the transmit event is ordered before the corresponding receive event, and events on the same lifeline are ordered from the top downwards. Shortening each message to the first letter of each signal, we thus get that Figure 18 specifies the trace $\langle !s, ?s, !r, ?r \rangle$ where ? denotes consumption and ! denotes transmission of a message.

Formally we let $\mathcal{H}$ denote the set of all well-formed traces over the set of events $\mathcal{E}$. A trace is well-formed if, for each message, the send event is ordered before the corresponding consumption event, and events on the same lifeline are ordered from the top. An *interaction obligation* (p, n) is a classification of all of the traces in $\mathcal{H}$ into three categories: the positive traces p, representing desired and acceptable behaviour, the negative traces n, representing undesired or unacceptable behaviour, and the inconclusive traces $\mathcal{H} \setminus (p, n)$. The inconclusive traces result from the incompleteness of interactions, representing traces that are not described as positive or negative by the current interaction (Runde et al., 2006). The reason we operate with inconclusive traces is that a sequence diagram normally gives a partial description of system behaviour. It is also possible to give a complete description of system behaviour. Then every trace is either positive or negative.

An interaction obligation with a range of probabilities is called a *probability obligation*, or *p-obligation*. Formally a p-obligation is a pair $((p, n), Q)$ of an interaction obligation (p, n) and a set of probabilities $Q \subseteq [0, 1]$. The assignment of a set of probabilities Q rather than a single probability to each interaction obligation captures underspecification with respect to probability, as the implementer is free to implement the p-obligation with any of the probabilities in Q (Refsdal, 2008). The assignment of an interval of probabilities $\{j, n\}$ to an interaction obligation (p, n), means that in any valid implementation the probability of producing the traces in $\mathcal{H} \setminus n$ should be at least j or equivalently, that the probability of producing traces in n should be at most $1 - j$. The probability of producing traces in $\mathcal{H} \setminus n$ may be greater than n if p-obligations resulting from different probabilistic alternatives have overlapping sets of allowed traces. The semantics of a sequence diagram D in pSTAIRS is a set of p-obligations.

We assume that interface interaction is asynchronous. This does not prevent us from representing systems with synchronous communication. It is well known that synchronous communication can be simulated in an asynchronous communication model and the other way around (He et al., 1990).

Since we use sequence diagrams to specify the operations of an individual interface, we only include the lifelines of the interfaces that an interface employs to implement an operation, that is, of the required interfaces. We adopt the convention that only the interface whose behaviour is specified can transmit messages in a specification of interface operations. Any other lifelines in the specification are required interfaces. This corresponds to the conventions Cheesman and Daniels apply for specifying individual components using UML 1.3 collaboration diagrams.

Example 9 (*Chat* interactions) The diagrams in Figures 19 and 20 specify the *send* and *login* operations of the *Chat* interface, respectively. When a user wants to chat she invokes the

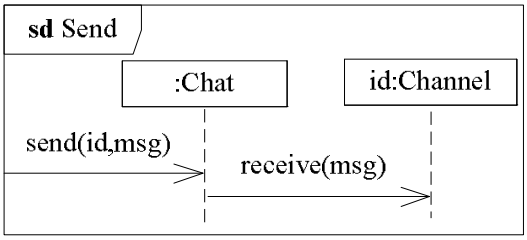

Figure 19: The *send* operation of the *Chat* interface

send operation of the *Chat* interface with the ID of her buddy and message as parameters. The *Chat* interface then calls the operation *receive* of a *Channel* interface with a matching buddy id, as illustrated in Figure 19.

When a user successfully logs on to her instant messaging component, her messaging service is registered at a remoting service. Since the *Chat* interface is a system interface it does not store any user data itself. It uses the business interface *UserMgr* to validate the user data, as illustrated in the sequence diagram in Figure 20.

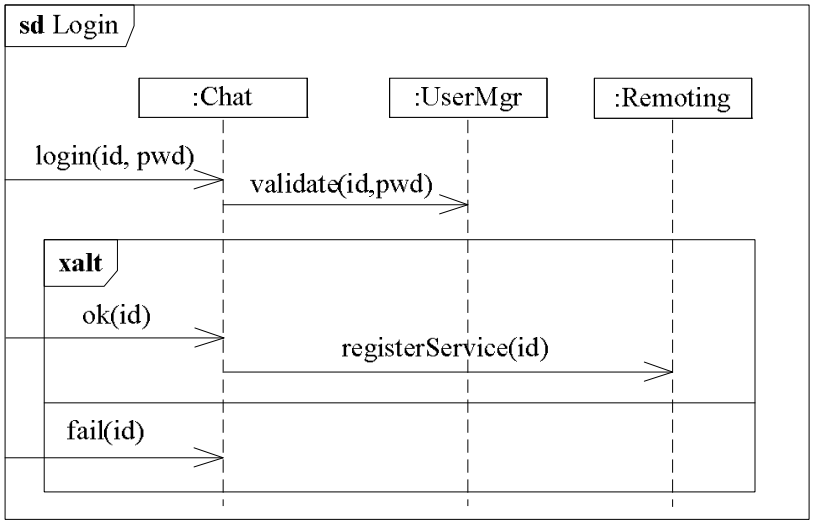

Figure 20: The *login* operation of the *Chat* interface

If the *Chat* interface receives the message *ok(id)* it employs a Remoting service to register the instant messaging service. If the login attempt fails, the instant messaging service is not registered. We use a xalt-operator to specify that an implementation must be able to perform both the alternatives where the login succeeds and where it fails. Due to the assumption that interface interaction is asynchronous, we cannot simply specify the *ok(id)* or *fail(id)* messages as alternative replies to the call *validate(id,pwd)*. Instead we specify these as two separate invocations of an *ok* and a *fail* operation. In reality the interface invoking the *ok* or *fail* operations will be the same as the one who consumed the *validate* operation. Due to our convention that only the specified interface can transmit messages, however, the transmitter of the *ok* and *fail* messages are not included in the sequence diagram. □

Example 10 (*UserMgr* interactions) The *UserMgr* interface handles user information. When it receives the message *validate(id, pwd)*, it should either send the message *ok(id)* or the message *fail(id)* to a *Chat* interface. The conditions under which alternative may be chosen is left unspecified at this point. In the final implementation we would expect the response *ok(id)* if

the password is correct and *fail(id)* otherwise. Such a constraint may be imposed through use of guards, but we have left guards out of the example for the case of simplicity. See Runde et al. (2006) for a discussion on the use of guards in sequence diagrams.

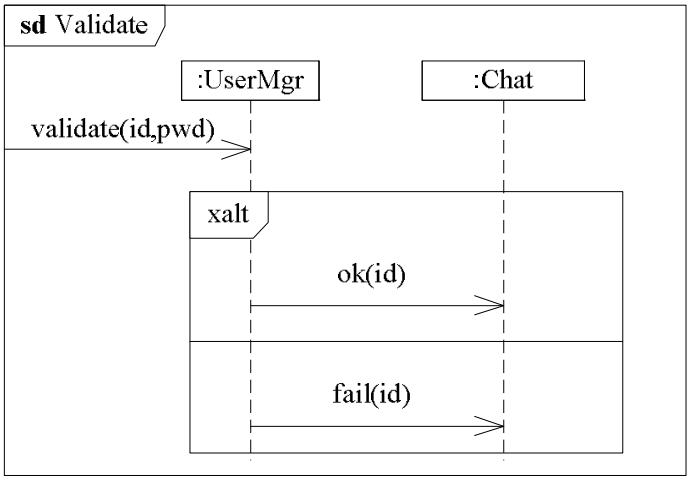

Figure 21: The *validate* operation of the *UserMgr* interface

As explained above, we cannot simply specify the *ok(id)* or *fail(id)* messages as replies to the call *validate(id,pwd)*, due to the assumption that interface interaction is asynchronous. Instead we specify that the *UserMgr* actively invokes an *ok* or a *fail* operation of a *Chat* interface. In reality the interface whose *ok* or *fail* operations are invoked will be the same as the one who invoked the *validate* operation. □

Example 11 (*FileTransfer* interactions) Figure 22 specifies the *sendFile* operation of the *FileTransfer* interface. If a user wants to send a music file to one of her buddies she calls

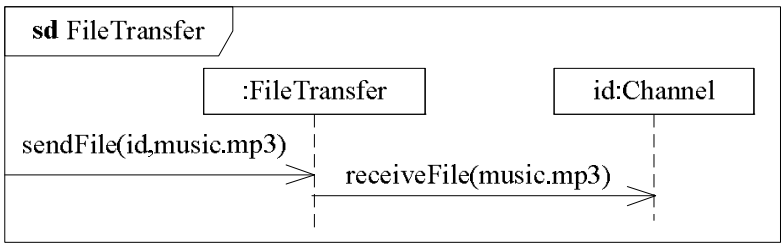

Figure 22: The *sendFile* operation of the *FileTransfer* interface

the operation *sendFile* of the *FileTransfer* interface, with the buddy ID and the music file as parameters. The *FileTransfer* interface must call an operation *receiveFile* of a *Channel* interface with the required buddy id, in order to implement this operation. □

Example 12 (*Channel* interactions) The two diagrams in Figure 23 specify the *receive* and *receiveFile* operations of the *Channel* interface. When the operation to send a message is called the *Channel* interface calls an operation of a *Display* interface that we assume is provided by the environment. When the operation *sendFile* is called with a music file as parameter, the *Channel* interface checks the format of the music file. The *Channel* interface then either calls an operation to play the music file or does nothing. Again, the conditions under which alternative may be chosen is left unspecified.

We use the xalt-operator to specify that an implementation must be able to perform both the alternatives where the format is found to be ok, and where it is not. □

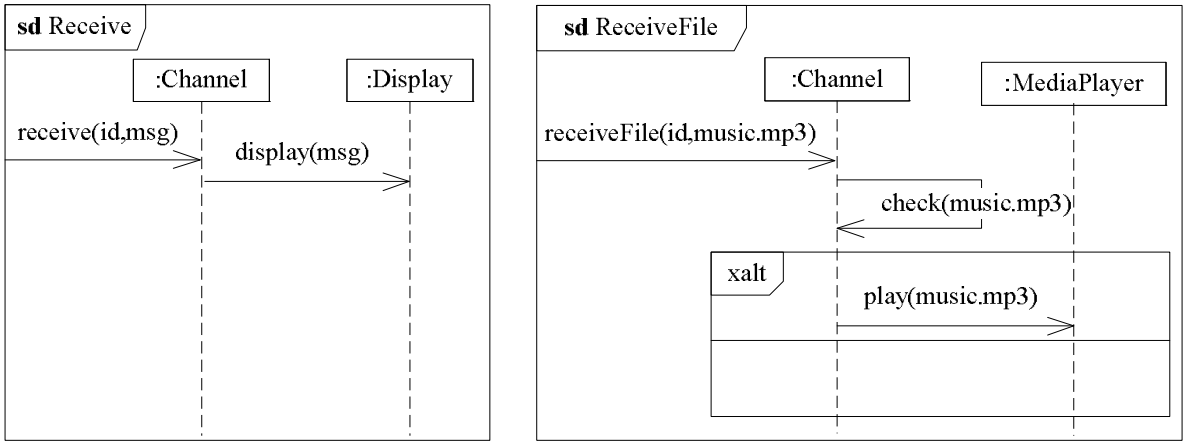

Figure 23: The receive operations of the *Channel* interface

6.2 Interface risk interactions

Risk is the likelihood that an incident occurs. Hence, in order to specify interface risk interactions we need a probabilistic understanding of interface behaviour. Before we can specify the risk interactions, we need to identify interface risks. Risk identification is the topic of the following section.

6.2.1 Identify interface risks

Risk identification involves identifying incidents and measuring their likelihood and consequence values. An incident is caused by a threat exploiting component vulnerabilities. Risk analysis therefore begins by identifying threats towards assets. In conventional risk analysis external threats are often included in the target of analysis. Since we have a component-based approach, we have stated that the target of analysis is a component or a component interface. Since we do not know what type of platform the instant messaging component will be deployed on, we do not know the level of threats it will be exposed to.

In order to facilitate modularity of risk analysis results we document risk analysis results in so called dependent threat diagrams. Dependent threat diagrams extend CORAS threat diagrams (den Braber et al., 2007) with facilities for making assumptions of a risk analysis explicit.

Dependent threat diagrams are inspired by assumption-guarantee reasoning, which has been suggested as a means to facilitate modular system development (Jones, 1981; Misra and Chandy, 1981; Abadi and Lamport, 1995). Dependent threat diagrams transfer the assumption-guarantee style to threat modelling, to support documentation of environment assumptions. Environment assumptions are used in risk analysis to simplify the analysis, to avoid having to consider risks of no practical relevance and to support reuse and modularity of risk analysis results (Lund et al., 2010).

CORAS uses structured brainstorming inspired by HazOp (Redmill et al., 1999) to identify and analyse threats towards assets. A structured brainstorming is a methodical "walk-through" of the target of analysis. Experts on different aspects of the target of analysis identify threats and exploitable vulnerabilities. The same method can be used for interfaces. We use the use case diagram and the sequence diagrams as input to the structured brainstorming sessions. We document the results in dependent threat diagrams.

Threat diagrams (dependent or otherwise) describe how different threats exploit vulnera-

bilities to initiate threat scenarios and incidents, and which assets the incidents affect. The basic building blocks of threat diagrams are as follows: threats (deliberate, accidental and non-human), vulnerabilities, threat scenarios, incidents and assets. A non-human threat may for example be a computer virus, system failure or power failure. A threat scenario is a chain or series of events that is initiated by a threat and that may lead to an incident. Figure 24 presents the icons representing the basic building blocks.

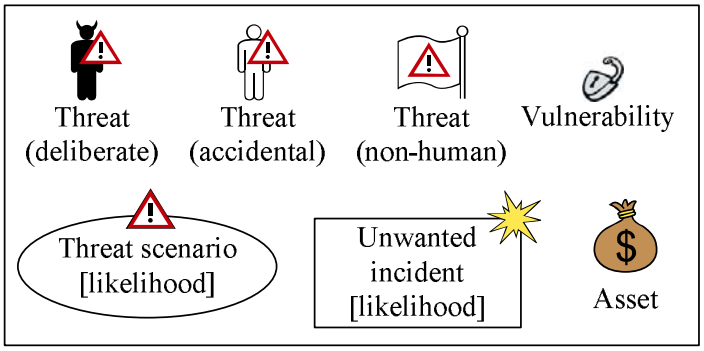

Figure 24: Basic building blocks of a CORAS threat diagram

A CORAS threat diagram consists of a finite set of vertices and a finite set of relations between them. The vertices correspond to the threats, threat scenarios, incidents, and assets. The relations are of three kinds: initiate, leads-to, and impacts. An initiate relation originates in a threat and terminates in a threat scenario or an incident. A leads-to relation originates in a threat scenario or an incident and terminates in a threat scenario or an incident. An impacts relation represents harm to an asset. It originates in an incident and terminates in an asset.

Figure 25 shows an example of a threat diagram. From the diagram we see that a hacker sends a crafted music file. Further, the diagram says that if a hacker sends a crated file, it may lead to the file being played, due to the vulnerability that no acceptance is required of the receiver of the crafted file. Since playing the crafted file means executing it, playing of the crafted file may lead to reception of malicious code embedded in the crafted file. According to the diagram, reception of malicious code harms the asset *Media file*.

Figure 25: Example threat diagram

A dependent CORAS diagram is similar to a basic threat diagram, except that the set of vertices and relations is divided into two disjoint sets representing the assumptions and the target. Figure 26 shows a dependent threat diagram. The only difference between a dependent threat diagram and a normal threat diagram is the border line separating the target from the assumptions about its environment. Everything inside the border line belongs to the target; every relation crossing the border line, like the leads-to relation from *Send crafted file* to *Play crafted file*, also belongs to the target. Everything completely outside the border line, like the threat scenario *Send crafted file* and the threat *Hacker* and the initiate relation from *Hacker* to *Play crafted file*, belongs to the assumptions.

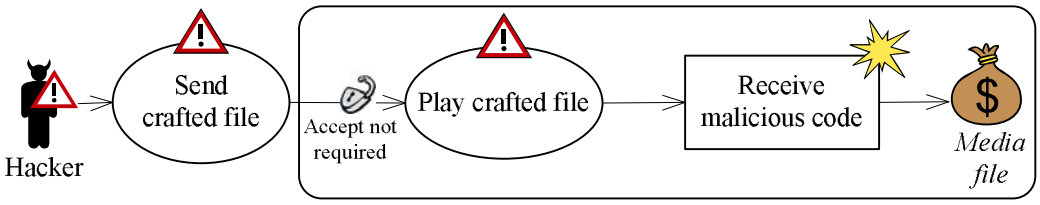

Figure 26: Example dependent threat diagram

Example 13 (Identify *Chat* risks). Figures 27 and 28 show dependent threat diagrams for the *Chat* interface assets. The use case *User login* involves an operation *login*, that we assume

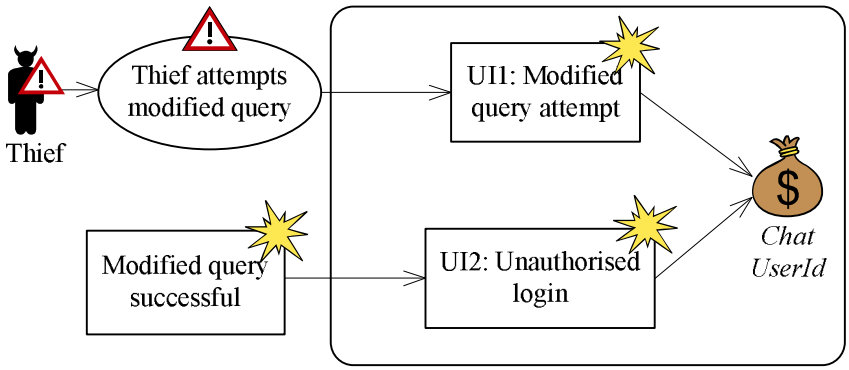

Figure 27: Threat scenario related to the login operation

a deliberate threat *Thief* may attempt without authorisation, by sending a modified query, if the device that the instant messaging component runs on is stolen.

We give the incidents short names to ease their reference in subsequent risk overviews. The incident *Modified query attempt* is therefore described by *UI1: Modified query attempt*, where *UI1* is the ID of the incident.

The *Chat* interface uses a *UserMgr* to check if user data is correct. The *UserMgr* may be implemented as an SQL (structured query language) data base or it may be interacting with an SQL database. If the *UserMgr* is not protected against SQL injection an attacker can modify or add queries by crafting input. An example of a modified query is to write a double hyphen (--) instead of a password. Unless the *UserMgr* is programmed to handle such metacharacters in a secure manner, this has the effect that the test for a matching password is inactivated and the modified query will be accepted.

In the case that the modified query is accepted by the sub-system handling user information, the *Chat* interface will register the *Thief*, resulting in the incident *Unauthorised login*. Since the *Chat* interface uses the *UserMgr* interface to implement the *login* operation, as specified in Figure 20, the incident *Unauthorised login* depends on how the provider of the *UserMgr* interface handles modified queries.

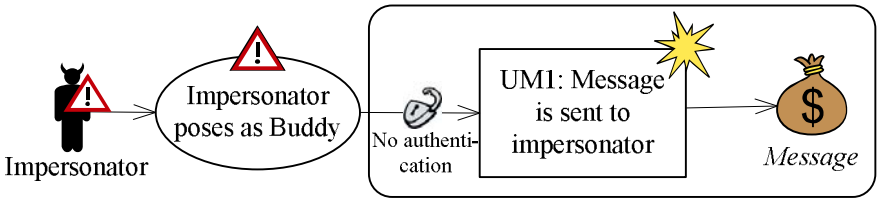

Figure 28: Threat scenario related to the send operation

We assume that a deliberate threat *Impersonator* can act as a buddy leading to the threat scenario *Message is sent to impersonator*, due to the vulnerability *No authentication*, as documented in the diagram in Figure 28. □

Example 14 (Identify *UserMgr* risks). In Figure 29 we document threat scenarios, incidents and vulnerabilities related to the operations of the *UserMgr* interface. Recall that a business interface is responsible for managing the information handled by the system. Since the

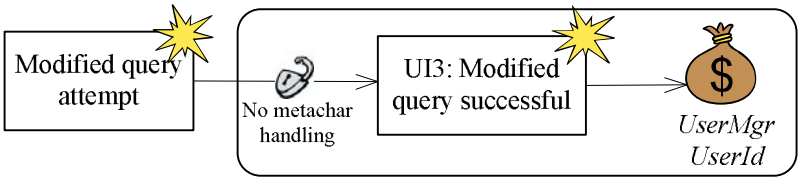

Figure 29: Threat scenario related to the validate operation

UserMgr interface is a business interface it only interacts with system interfaces. The incidents towards the asset of the *UserMgr* interface therefore depends on results from risk analyses of interacting interfaces and we use a dependent threat diagram to state assumptions about the environment of the *UserMgr*.

The dependent diagrams of Figure 27 and 29 illustrate that assumptions in one diagram can be part of the target in another diagram, and vice versa. The assumption *Modified query successful*, for example, is in Figure 27 an assumption about an incident affecting an interacting interface. In Figure 29 the same incident is part of the target. The *UserMgr* interface makes the assumption *Modified query attempt* which is part of the target in Figure 27.

From the diagram we see that the incident *Modified query attempt* in the environment, may lead to the incident *UI3: Modified query successful* due to the vulnerability *No metachar handling*. This vulnerability refers to that the *UserMgr* interface is not specified to check the arguments to the *validate* operation for metacharacters, that is, special characters with a specific meaning in an SQL database, such as hyphens. The *UserMgr* interface is therefore vulnerable to so called modified queries. □

Example 15 (Identify *Channel* risks) In Section 5 we assigned responsibilities for the *Receive music file* and *Receive message* use cases to an interface *Channel* that a buddy can use for sending messages or files to a user. We assume a deliberate threat *Hacker* may exploit

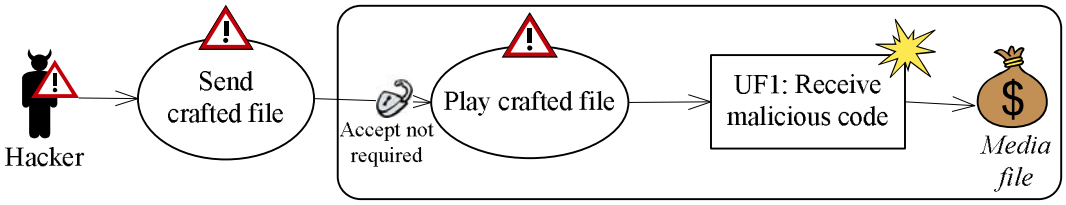

Figure 30: Threat scenario related to the receiveFile operation

the operations described by the *Receive music file* use case, by sending a crafted music file designed to exploit possible buffer overflow vulnerabilities in a media player. When the operation *receiveFile* is called, the *Channel* interface calls an operation from a *MediaPlayer* interface to play the music file, without prompting the user. This vulnerability is denoted *Accept not required* in Figure 30. This vulnerability may be exploited by the threat scenario *Send crafted file*, leading to the threat scenario *Play crafted file*. Exploiting a buffer overflow is about filling a buffer with more data than it is designed for. In this way a pointer address may be overwritten,

thus directing the device to an uncontrolled program address containing malicious code. Thus, the threat scenario *Play crafted file* may lead to the incident *Receive malicious code* harming the asset *Media file*.

The possibility of a buffer overflow is a known vulnerability of older versions of Winamp (cve, 2005). Buffer overflow exploits for other media formats have also been identified in various media players, including Windows Media Player, RealPlayer, Apple Quicktime, iTunes and more (Mannan and van Oorschot, 2004; Sans, 2005). Even though vendors release security patches when vulnerabilities are discovered, a device with a media player which does not make regular use of these patches will still be vulnerable. New vulnerabilities are also frequently discovered, even if old ones are patched.

In Section 5.2 we also identified *Channel Availability* as an asset of the *Channel* interface. The asset *Channel Availability* refers to the time the *Channel* interface uses to respond to operation calls. We have identified and documented two threat scenarios that may lead to incidents causing harm to the *Availability* asset: *Spimming* and *Flooding attack*. The threat scenarios are documented in Figure 31.

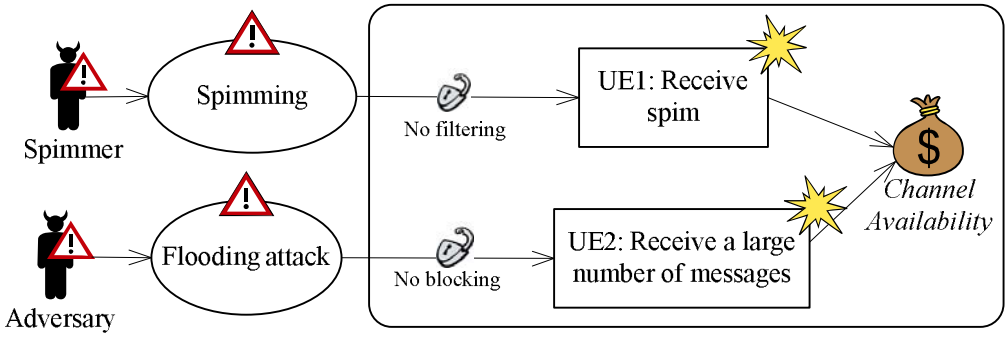

Figure 31: Threat scenarios related to the receive operation

Spimming, or spam over instant messaging is an increasing problem for instant messaging on fixed devices and it is reasonable to assume that this problem will spread to mobile devices. Mobile instant messaging is also vulnerable to denial of service attacks, such as flooding, where a user receives a large number of messages. □

6.2.2 Estimate likelihoods

After having completed the identification and documentation of threat scenarios, incidents and vulnerabilities, we are ready to estimate the risks. A risk is the likelihood of an incident and its consequence for an asset. Risk estimation is to estimate the risk level of the identified incidents. The objective is to determine the severity of the risks which allows us to subsequently prioritise and evaluate the risks, as well as determining which of the risks should be evaluated for possible treatment.

For this task we use dependent threat diagrams defined earlier as input and estimate risk levels by estimating likelihood and consequence values of the identified incidents. In CORAS the risk estimation is conducted as a structured brainstorming session involving personnel with various backgrounds. The result of risk estimation is documented by annotating dependent threat diagrams with likelihood and consequence values. Threat scenarios, incidents, initiate relations and leads-to relations may be annotated with likelihoods. Only impacts relations are annotated with consequence values.

Figure 32 shows the same threat diagram as in Figure 25 annotated with likelihood and consequence values.

Figure 32: Example threat diagram annotated with likelihood and consequence values

The analyst team has estimated that the likelihood of a hacker sending a crafted file lies within the interval that is mapped to *Possible* in Table 1. This is documented in the diagram by annotating the initiate relation from the deliberate threat *Hacker* to the threat scenario *Send crafted file* with this likelihood value. Further, the diagram says that in the case that a hacker sends a crafted file, there is a small likelihood (*Unlikely*) for this leading to the crafted music file being played.

If a diagram is incomplete in the sense that it does not document all the ways in which a threat scenario or incident may happen, we can deduce only the lower bounds of the likelihoods. For the purpose of the example we assume that the diagrams are complete in the sense that no other threats, threat scenarios, or unwanted incidents than the ones explicitly shown lead to any of the threat scenarios or to the unwanted incident in the diagrams. Based on the assumption that the diagram is complete we can calculate the likelihood of this scenario by multiplying the intervals mapping to *Possible* and *Unlikely* as illustrated in Table 8.

Source scenario	**Leads-to**	**Target scenario**
$\langle 0.25, 0.50]$	$\langle 0.00, 0.25]$	$\langle 0.25 \times 0.00, 0.50 \times 0.25] = \langle 0.00, 0.13]$

Table 8: Example of how one may calculate likelihood values

Since playing the crafted file means executing it, it is certain that this will lead to reception of malicious code embedded in the crafted file. We calculate the likelihood value for the incident *Receive malicious code* to be *Unlikely*, using the same procedure as that used to calculate the likelihood of the threat scenario *Play crafted file*. The consequence of the incident *Receive malicious code* with regard to the asset *Media file* is considered to be *Moderate*.

In component-based risk analysis we must take into account that the likelihood of a threat may differ depending on the environment in which a component exists (Verdon and McGraw, 2004). The probability of a risk depends both on the probability of a threat initiating a threat scenario, and the probability that a threat scenario leads to an incident. The latter probability gives a measure on the degree of vulnerability of a component towards a threat. At the component or interface level, we can only know how the component will react given an attack and we may estimate the likelihood of a threat scenario leading to a new threat scenario or an incident, that is, the likelihood of leads-to relations. In order to estimate the likelihood that an attack is successful we must consider vulnerabilities and the effectiveness of control mechanisms if any such exists. We annotate the leads-to relations with likelihood values indicating the effectiveness of control mechanisms.

Example 16 (Likelihood estimation for *Chat* risks) Figures 33 to 34 show the dependent threat diagrams from Example 13 annotated with conditional likelihoods.

We do not know the likelihood of theft, since that depends on the type of platform the instant messaging component exists on. The likelihood of theft is probably higher for a portable device like a cell phone, than on a personal computer. We therefore parameterise the conditional likelihood of the initiate relation from the deliberate threat *Thief* to the threat scenario *Thief*

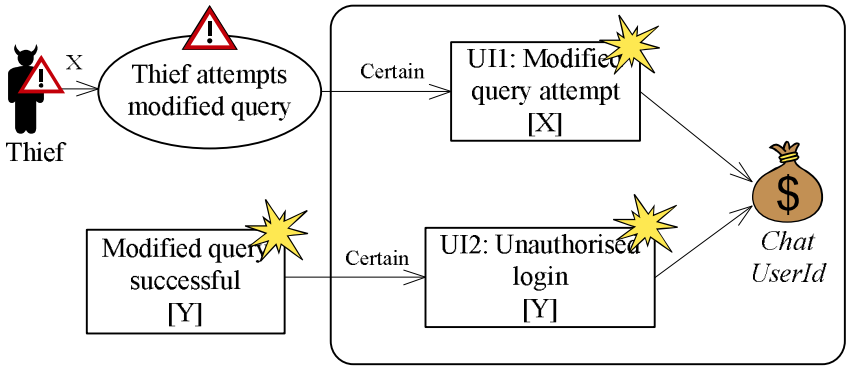

Figure 33: Likelihood estimation for login related threats

attempts modified query in the assumption. The possibility to generalise assumptions through parameterisation facilitates reuse and thereby modularity of analysis results.

A login attempt is simply passed on to the *UserMgr* interface. The leads-to relation from the threat scenario *Thief attempts modified query* to the incident *Modified query attempt* is therefore annotated with conditional likelihood *Certain*. This means that the probability of the latter given the former is 1.0, as defined in Table 1.

As already mentioned, if a diagram is incomplete, we can deduce only the lower bounds of the likelihood values on threat scenarios and incidents. For the purpose of the example we assume that the diagrams are complete in the sense that no other threats, threat scenarios or incidents than the ones explicitly shown lead to any of the threat scenarios or the incident in the diagrams. We can therefore obtain the likelihood of the threat scenario *Modified query attempt* by multiplying the likelihood of the threat scenario *Thief attempts modified query* with the conditional likelihood on the leads-to relation leading from *Thief attempts modified query* to *Modified query attempt*. Hence the likelihood of the incident *Modified query attempt* is the same as the likelihood of the threat scenario leading to it, namely X.

By assumption the probability of the incident *Unauthorised login* depends only on the probability of the incident *Modified query successful* in the environment. We assume that if a modified query is successful with probability Y this will lead to an unauthorised login with probability *Certain*. Hence the probability of the incident *Unauthorised login* is Y.

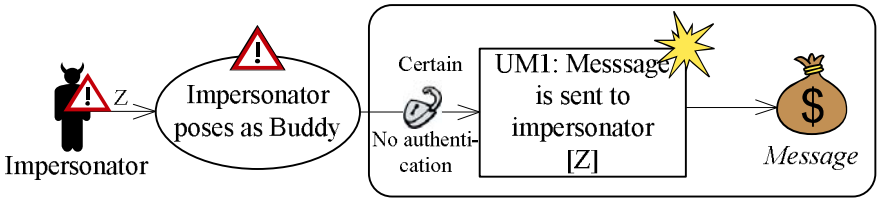

Figure 34: Likelihood estimation for sending of messages to impersonator

Since there is no authentication of buddies we estimate the likelihood of the threat scenario *Impersonator poses as buddy* leading to the threat scenario *Message is sent to impersonator* to be *Certain*. □

Example 17 (Likelihood estimation for *UserMgr* risks) Figure 35 shows the dependent threat diagram from Figure 29 annotated with conditional likelihoods.

At this point the *UserMgr* interface is not specified to handle metacharacters used in modified queries. That is, the interface is not specified to filter input for special characters. As

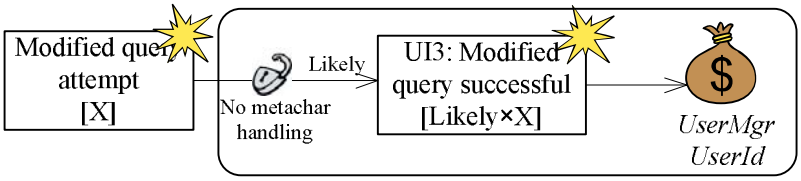

Figure 35: Likelihood estimation for successful modified query

explained in Example 14 this may be exploited by a malicious user to craft a request in a specific way. We estimate it to be a fair chance that a modified query will inactivate the test for password. However, we estimate that there is a slight possibility that a modified query will be rejected by the regular password check. We estimate the likelihood of the incident *Modified query attempt* leading to the incident *Modified query successful* to be *Likely*. That is, the probability of a successful modified query is $Likely \times X$. □

Example 18 (Likelihood estimation for *Channel* risks) Figures 36 and 37 show the dependent threat diagrams from Example 15 annotated with likelihood values.

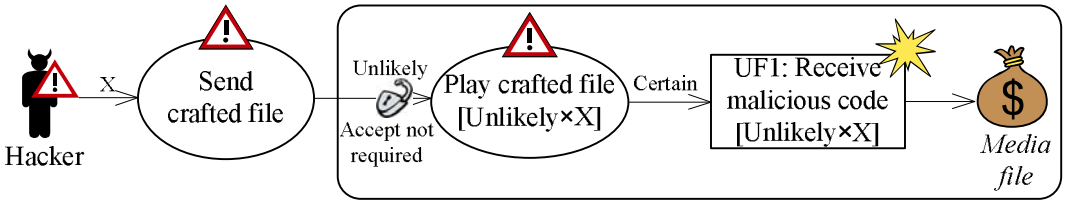

Figure 36: Likelihood estimation for reception of malicious code

As explained in Example 10 we have left unspecified at this point how the check for validity of file names is performed. Even if a check of the file name is implemented, there is a possibility that a file is crafted in a manner that may go undetected. We estimate the likelihood of the threat scenario *Send crafted file* in Figure 36 leading to the threat scenario *Play crafted file* to be *Unlikely*. We assume that similar calculations have been done for the other threat scenarios in the dependent threat diagram in Figure 37.

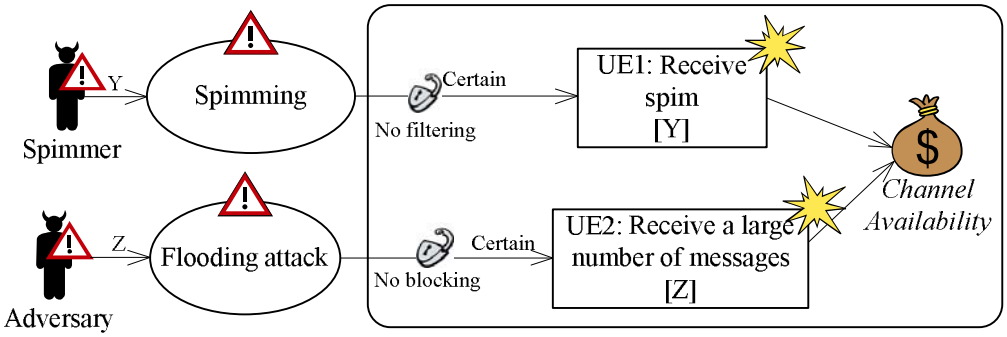

Figure 37: Likelihood estimation for spimming and flooding

6.2.3 Estimate consequences

The next step is to assign consequence values to incidents. The consequence values are taken from the asset's consequence scale that is defined during the requirements to protection work-

flow. In CORAS the consequence of an incident with regard to a specific asset is decided by the asset stakeholder. As explained in Section 4.2, in component-based risk analysis we identify assets on behalf of the component or component interface which is the target of analysis. Decisions regarding assets and their protection level must therefore be done by the component owner, or the development team in an understanding with the component owner.

Example 19 (Consequence estimation) Figures 38 and 39 show the dependent threat diagrams from Example 16 annotated with consequence values.

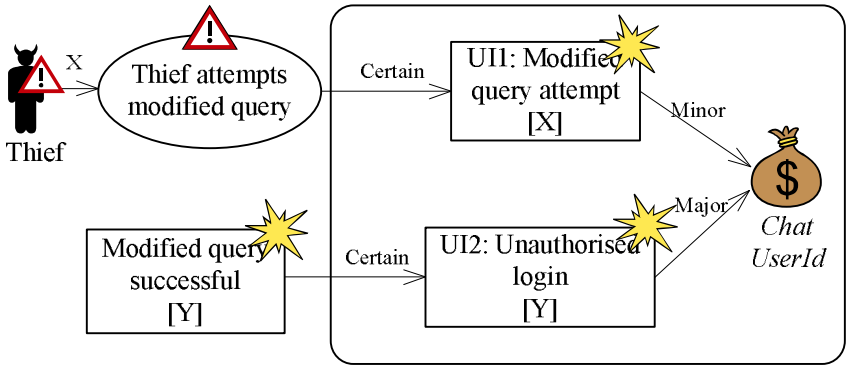

Figure 38: Consequence estimation for login related threats

We have assigned the consequence value *Minor* to the impacts relation from the incident *Modified query attempt*, to the asset *Chat UserId*, and the consequence value *Major* to the impacts relation from the incident *Unauthorised login* to the same asset. The rationale for this is that a modified query in itself is not so serious with regard to the asset *Chat UserID*, whereas an unauthorised login implies that the integrity of the *Chat UserId* is corrupted which is a major incident for this asset.

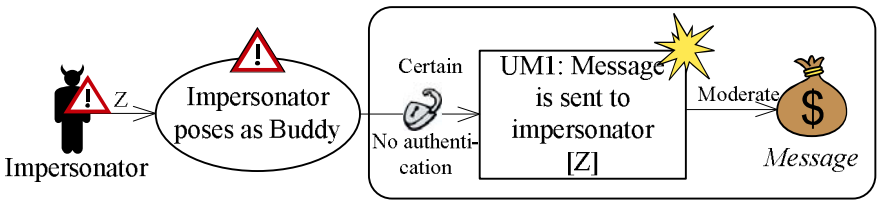

Figure 39: Consequence estimation for sending of messages to impersonator

The severity of the incident *Message is sent to impersonator* is considered to be in the middle with regard to the asset *Message* and the impacts relation from this incident is assigned the consequence value *Moderate*. In a full risk analysis the exact meaning of the consequence values may be explained by mapping each linguistic term to concrete values.

Figure 40 shows the dependent threat diagram from Figure 35 annotated with consequence values.

Figures 41 and 42 show the dependent threat diagrams from Example 18 annotated with consequence values. □

6.2.4 Specifying risk interactions

During the interface risk interaction of the specification workflow, we identified risks related to each of the operations specified in the sequence diagrams in Section 6.1. The identified

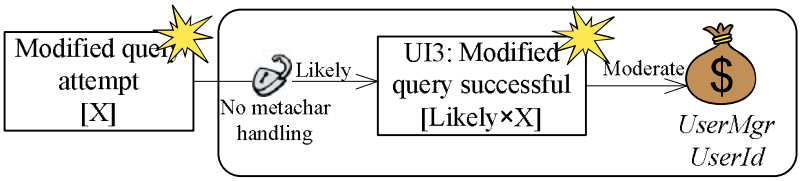

Figure 40: Consequence estimation for modified query success

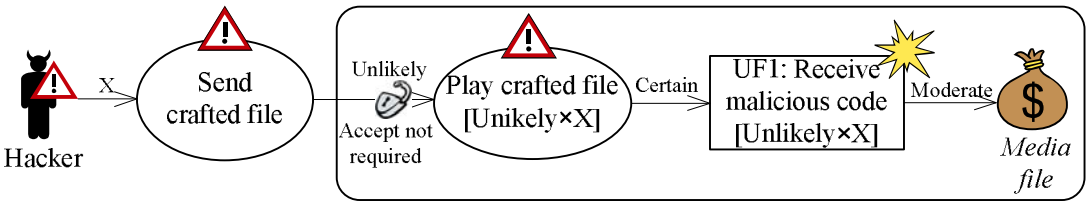

Figure 41: Consequence estimation for reception of malicious code

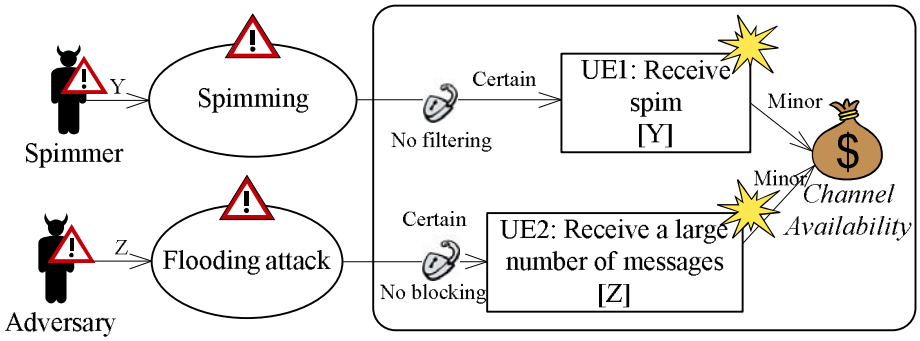

Figure 42: Consequence estimation for spimming and flooding

risks were documented in dependent threat diagrams. We are, however, not finished with documenting risks, as we want to specify component and interface risks as an integrated part of the component specification, using the same type of specification techniques. The motivation for this is that we want to be able to update our knowledge about the risk behaviour when a component-based system is upgraded with a new component. We therefore specify the details of the threat diagrams using sequence diagrams in STAIRS, just as we did for the use cases in Section 6.1.

We use the dependent threat diagrams from the interface risk interaction task described in Section 6.2.3, together with the sequence diagrams specifying interface interactions, as input to specifying the interface risk interactions.

The risk interactions capture how the normal interactions can be mis-used in order to cause incidents that harm the interface assets. This implies that incidents are events that are allowed within the specified behaviour but not necessarily intended. Hence, the only thing that can be added in the specification of risk interactions are alternative behaviours where the arguments of the operations differ from the intended ones.

A leads-to relation from a diagram element v_1 to another diagram element v_2 in a threat diagram, means that v_1 may lead to v_2. If we assign the probability 0.5 to the relation going from v_1 to v_2 it means that the conditional likelihood that v_2 will occur given that v_1 has occurred is 0.5. That is, there is a 0.5 probability that v_1 will lead to v_2. This implies that there is also a 0.5 probability that v_1 will *not* lead to v_2. In a threat diagram we do not include the alternative behaviour that does not represent threats or incidents, as they are not relevant for documenting risks. For example in the dependent threat diagram describing the threat scenario related to the *UserMgr* interface we assigned the likelihood *Likely* on the leads-to relation from the incident *Modified query attempt* to the incident *Modified query successful*. The likelihood value *Likely* is mapped to the interval $\langle 0.50, 0.70]$ in the likelihood scale in Table 1. This means that the conditional likelihood of *Modified query attempt* not leading to the incident *Modified query successful* should be in the interval $[0.30, 0.50]$.

When we use a probabilistic sequence diagram to explain the details of a threat diagram, the alternative behaviour that does not represent threats or incidents is also included. The alternatives that a particular event happens or not happens are mutually exclusive. We use the expalt operator of probabilistic STAIRS (Refsdal, 2008) to specify mutually exclusive probabilistic choice in a sequence diagram. Since the probabilities of all operands of a palt or an expalt must add up to one (Refsdal, 2008), we must include both the alternative that the events corresponding to the incident *Modified query successful* happen, and that they do not happen. This case is illustrated in Example 21. In general, each leads-to relation to an element v, in a dependent threat diagram, which has a probability or interval of probabilities lower than 1, is described by one expalt operator in the corresponding sequence diagram. The first operand of the expalt represents the alternative where v occurs and the other operand represents the alternative where v does not occur.

If we have assigned the probability 1.0 to a relation going from a diagram element v_1 to another element v_2 it implies that v_1 always leads to v_2. Hence, given that v_1 has occurred, the probability that v_2 does not happen is zero. In this case we do not need to use the expalt operator when specifying the risk interactions in a sequence diagram. This case is illustrated in Example 20.

The current version of STAIRS has no facilities for documenting assumptions about the environment. Furthermore, the formal semantics of STAIRS as defined by Haugen and Stølen (2003) and Haugen et al. (2004) does not include constructs for representing vulnerabilities, incidents or harm to assets. This means that some of the information documented in the dependent threat diagrams will be lost when we interpret the risk interactions in sequence diagrams. In

order to obtain the complete risk analysis documentation we therefore need both the threat diagrams that document the consequences of incidents, and the sequence diagrams, that relate risk behaviour to the overall interface behaviour. In order to provide a fully integrated risk analysis and interface specification we need a denotational trace-semantics that captures risk relevant aspects such as assets and incidents formally. For an approach to represent risk behaviour in a denotational trace semantics see Brændeland and Stølen (2006, 2007). Furthermore, we need to map syntactical specifications using sequence diagrams to a formal semantics for risk behaviour. This is a task for future work.

Example 20 (*Chat* risk interactions) The two sequence diagrams in Figure 43 illustrate the relation between normal interactions and risk interactions. The sequence diagram to the left specifies the normal behaviour related to the *login* operation of the *Chat* interface.

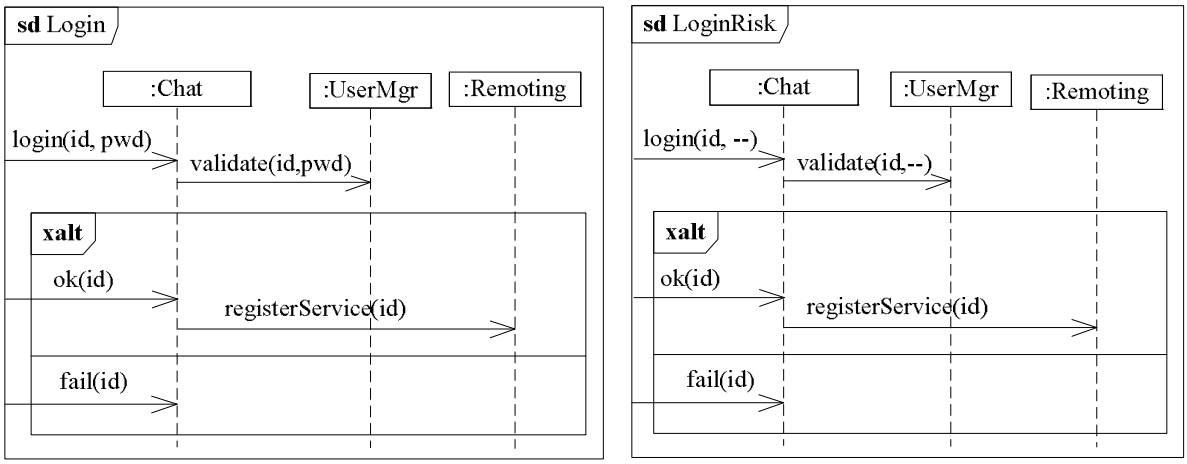

Figure 43: Normal interaction versus risk interaction of the *Chat* interface

The sequence diagram to the right specifies the risk interactions related to the *login* operation. According to the dependent threat diagram in Figure 38 the *login* operation of the *Chat* interface can be exploited to perform a modified query, that is, a specially crafted query aiming to inactivate the password validity check. One example of a modified query is to write a double hyphen (- -) instead of a password. This scenario is specified in the lower operand of the xalt in the sequence diagram in Figure 43. As already explained, the only things that can be added when specifying risk interactions are alternative behaviours where the arguments of the operations differ from the intended ones.

A login attempt to the *Chat* interface is simply passed on to the *UserMgr* interface. We therefore estimated the likelihood of the threat scenario *Thief attempts modified query* leading to the incident *Modified query attempt* to be *Certain*, corresponding to 1.0. This means that in this case the attack will always be successful in the sense that it will lead to the incident *Modified query attempt*. The transmission of the message *validate(id,- -)* from the *Chat* interface to the *UserMgr* interface, corresponds to this incident.

The other threat scenarios, incidents and relations documented in Figures 38 and 39 can be formalised in a similar fashion. □

Example 21 (*UserMgr* risk interactions) Figure 44 shows the details of the target scenario in the dependent threat diagram for the *UserMgr* interface in Figure 40.

We estimated the likelihood of the incident *Modified query attempt* leading to the incident *Modified query successful* to be *Likely*, corresponding to the interval $\langle 0.50, 0.70]$.

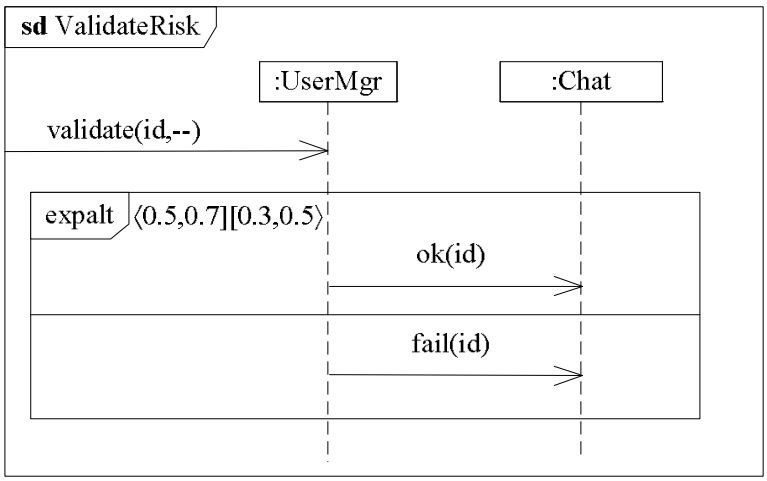

Figure 44: Modified query to the *UserMgr* interface

The first operand of the expalt operator represents the case that constitutes the threat scenario: that a modified query attempt is successful. The incident *Modified query successful* in Figure 29 corresponds to the transmission of the message *ok(id)* in the sequence diagram in Figure 44. The first probability set $\langle 0.50, 0.7]$ belongs to this alternative, which means that this alternative should take place with a probability of at most 0.7. In a probabilistic sequence diagram we can only assign probability values to whole scenarios, not single events. The probability interval $\langle 0.50, 0.7]$ refers to the probability set of the traces that the first operand gives rise to.

The second operand represents the alternative where the modified query fails. The second probability set $[0.3, 0.50]$ belongs to this alternative, which means that the probability of an attack not happening is at least 0.3 and at most 0.50.

Note that the probability set $\langle 0.50, 0.7]$ in Figure 44 does not tell the whole story with regard to the probability of the incident *Modified query successful*. The total probability depends on the probability of an attack, of which this diagram says nothing.

The sequence diagram in Figure 45 shows an example of a complete risk interaction that includes both the external threat and the affected interfaces of the instant messaging component.

Assuming the probability of a modified query attempt from the thief to be in the interval $[0, 0.4]$, we use the information from the risk interaction diagrams in Figure 43 and Figure 44 to construct the combined risk interaction diagram for the *Thief*, the *Chat* interface, *UserMgr* interface and the *Remoting* interface. The first two operands of the expalt operator, together represent the case where the thief attempts a modified query. In the first operand the attempted modified query is successful and the thief is registered. The probability interval $\langle 0, 0.28]$ refers to the probability set of the traces corresponding to this scenario. This interval is obtained by multiplying the probability interval for the modified query attempt with the probability interval for the alternative where the modified query is successful, corresponding to the upper operand in of the expalt in Figure 44: $[0, 0.4] * \langle 0.5, 0.7] = \langle 0 * 0.50, 0.4 * 0.7] = \langle 0, 0.28]$

The second operand represents the case where the attempted modified query fails, and the message *fail(id)* is sent to the *Chat* interface. This scenario is assigned the probability interval $\langle 0, 0.2]$, obtained by multiplying the probability interval for the modified query attempt with the probability interval for the alternative where the modified query fails, corresponding to the lower operand in of the expalt in Figure 44.

The third operand represents the case where the thief does not attempt a modified query. For our purpose it is not important to know what the thief does, if she does not attempt a

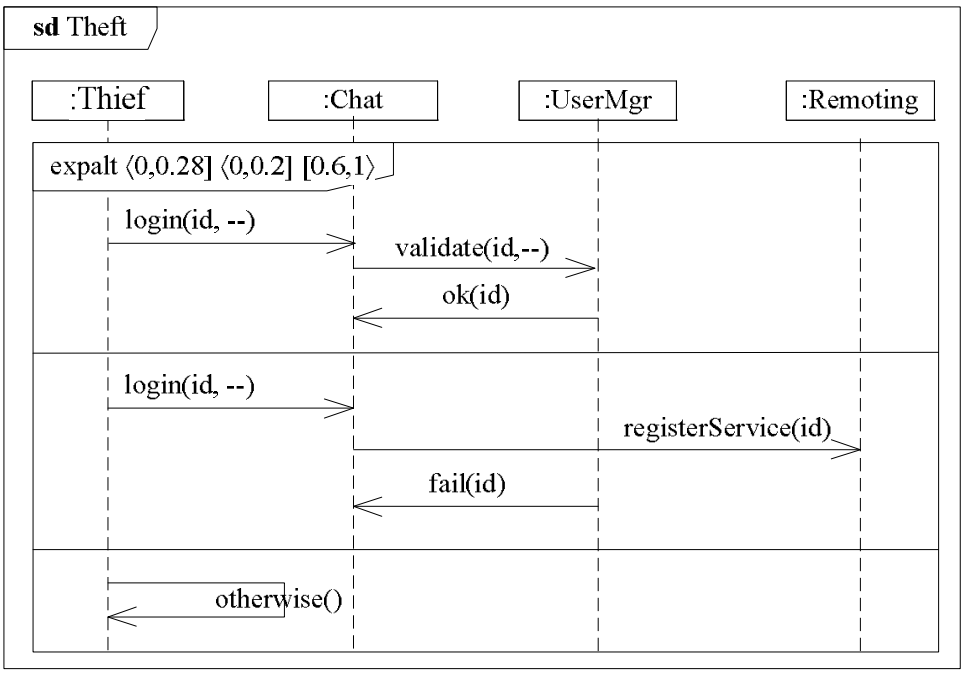

Figure 45: Thief attempts modified query

modified query. This alternative is therefore represented by the generic message *otherwise*, sent from the thief to itself. □

Example 22 (*Channel* risk interactions) The sequence diagram in Figure 46 specifies the sequence of events involved in the target in Figure 41.

The traces obtained by the first operand of the expalt operator represent the case that constitutes the threat scenario: that a crafted music file is sent to the *MediaPlayer* interface. The incident *Receive malicious code* in Figure 41 corresponds to the transmission of the message *play(vls.mp3)* in the sequence diagram in Figure 46. Since the *Channel* interface is specified to check validity of file names, we estimated the probability of this alternative to be only *Unlikely*. The second operand represents the case where the attack fails, that is, where the file is discarded. □

7 Specification

In this Section we explain how to check whether the requirements defined in Section 4, are met by the component specification. In accordance with the integrated process, we first describe how the interfaces can be fitted together into a component providing the required functional behaviour (Section 7.1), and then explain how to combine the interface risk analysis results to check compliance with the protection requirements (Section 7.2). Risk analysis composition is not part of the original CORAS process.

7.1 Component specification

Component specification is the final stage of the specification workflow. During this stage we describe how the interfaces can be fitted together into a component that refines the original requirements.

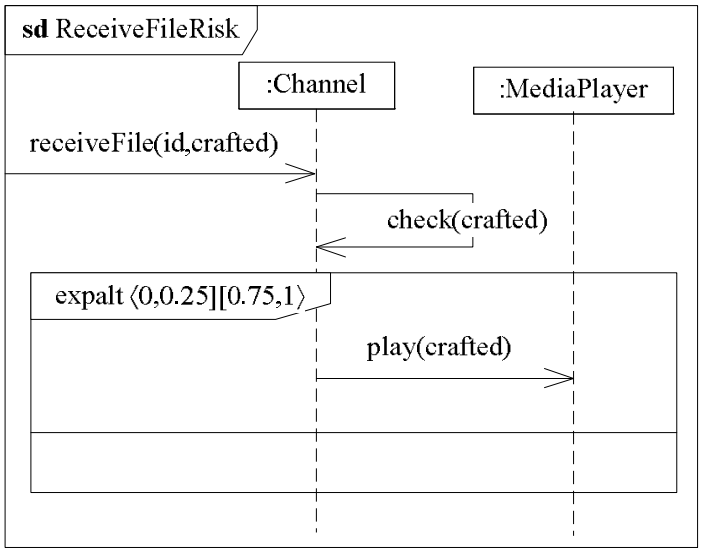

Figure 46: The *Channel* interface receives a crafted mp3 file

Summary of component-based risk analysis: workflow 2, step 2
• **Objective:** Identify interface risks and determine their severity in terms of likelihood and consequence.
• **Input documentation:** The use case diagrams from the requirements workflow, the likelihood scale and consequence scale from the requirements to protection workflow and the interface asset diagrams and initial sequence diagrams, showing normal interactions, from the specification workflow.
• **Output documentation:** Dependent threat diagrams documenting interface risks and sequence diagrams specifying both normal interactions and risk interactions.
• **Adaptations to CORAS:** The specification of risk interactions in sequence diagrams is not part of the original CORAS method.

Table 9: Interface risk interaction

Refinement refers to a development step from an abstract specification to a more concrete specification. Within formal methods the correctness of a development step is verifiable in a formally defined semantics. We use UML use case diagrams and class diagrams, that have no formal semantics, to specify the component requirements. Rather than following a formal refinement step we therefore use the diagrams from the requirements workflow to *guide* the specification of the interfaces and their operations. The details of the use cases for each interface, are captured using sequence diagrams in STAIRS (Haugen and Stølen, 2003; Haugen et al., 2004), for which we have a formal semantics. In this section we use STAIRS sequence diagrams to specify component behaviour that involve interactions between interfaces.

A sequence diagram specifying the operations of an individual interface may be seen as an abstraction over the component specification. If we want to obtain the p-obligations of an interface i from the specification of a component c, we proceed as follows:

- Let $[\![c]\!]$ be the denotational representation of c.
  1. Remove all events from $[\![c]\!]$ in which i is neither a consumer nor a transmitter.
  2. For each transmission event in which i is the consumer, substitute the transmitter with a fresh input gate.

For a formal treatment of substitution of lifelines and handling of gates, see Runde et al. (2006) and Haugen et al. (2004), respectively. The above procedure is illustrated in Example 23.

Example 23 (Combining instant messaging interfaces) As illustrated in Figure 14 the only interfaces of the instant messaging component that interact among themselves are the *Chat* and *UserMgr* interfaces. The two sequence diagrams in Figure 47 specify the complete interaction between the *Chat* and *UserMgr* interfaces. That is, both the normal interactions specified in the sequence diagrams in Figure 20 and Figure 21 and the risk interactions specified in Figures 43 and 44.

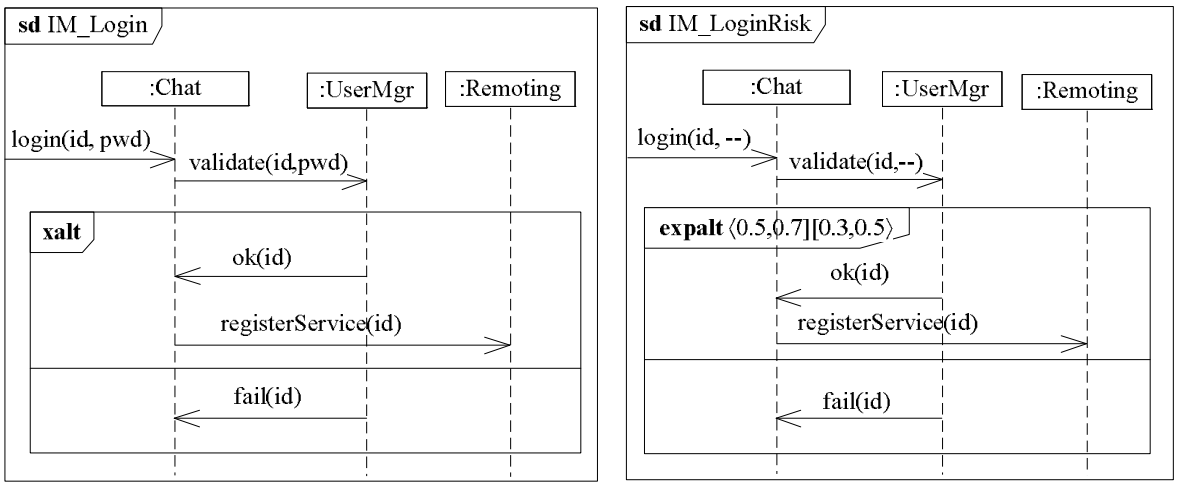

Figure 47: Interaction among the instant messaging interfaces

The denotation of the sequence diagram *User login* in Figure 20, which specifies the *login* operation of the *Chat*, interface can be obtained by following the procedure described above. Let $[\![\mathit{IM\_Login}]\!]$ denote the sequence diagram to the left in Figure 47. All events in $[\![\mathit{IM\_Login}]\!]$ have the interface *Chat* as consumer or transmitter, so no events are removed. Substitute *UserMgr* with the fresh input gate names i_1 and i_2 in $[\![\mathit{IM\_Login}]\!]$.

Until now we have left the gate names implicit in the sequence diagrams. Figure 48 shows the sequence diagram from 20 where we have included the names of two input gates: j_1 and j_2. The set of p-obligations obtained by applying the procedure described above to the denotational

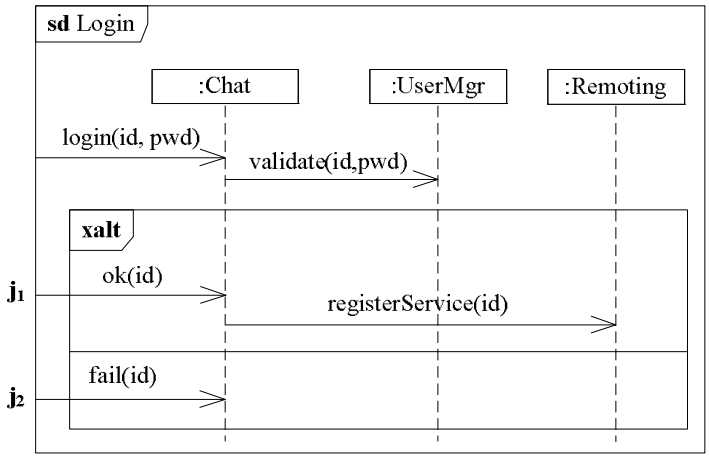

Figure 48: The sequence diagram from Figure 20 with explicit input gates

representation of *IM_Login* is the same as the denotational representation of *User login* with j_1 and j_2 substituted by i_1 and i_2. The denotation of the sequence diagram *Validate* in Figure 21 may be obtained in a similar manner. □

7.2 Component protection specification

We must also check whether the component specification fulfils the requirements to protection specification, that is, whether any of the identified risks has a high risk level and needs to be treated. As part of the *Component specification* step described in Section 7.1, we specified both the normal interactions and the risk interactions of a component, using STAIRS. However, as explained earlier, the current version of STAIRS has no facilities for documenting assumptions about the environment. In order to obtain the complete risk analysis documentation we therefore need dependent threat diagrams that document the consequences of incidents and the assumptions on which risk analyses results depend, in addition to sequence diagrams. As we saw in the example with the *Chat* interface and the *UserMgr* interface, when component interfaces interact with each other their risks may also depend on each other. In order to obtain a risk picture for the whole component we must in this step combine the dependent threat diagrams.

7.2.1 Reasoning about dependent diagrams

As explained in Section 6.2.1 the extension of CORAS with dependent threat diagrams was motivated by the need to support modular risk analysis. This is achieved by facilities for making the assumptions of a risk analysis explicit through diagrams drawn in an assumption-guarantee-style. Assumption-guarantee-style diagrams are particularly suited to document risks of open components that interact with and depend on an environment.

We have previously presented a semantics for so called dependent risk graphs and a calculus for reasoning about them (Brændeland et al., 2010). Dependent CORAS threat diagrams can be interpreted as dependent risk graphs and reasoned about in the calculus for risk graphs. We distinguish between two types of risk graphs: basic risk graphs and dependent risk graphs. A basic risk graph consists of a finite set of vertices and a finite set of relations between them. A vertex is denoted by v_i, while a relation from v_i to v_j is denoted by $v_i \rightarrow v_j$. A relation $v_1 \rightarrow v_2$

between vertices (threat scenarios) v_1 and v_2 means that v_1 may lead to v_2. Both vertices and relations between them are assigned likelihood values.

For a basic risk graph D to be well-formed, we require that if a relation is contained in D then its source vertex and destination vertex are also contained in D:

$$v \rightarrow v' \in D \Rightarrow v \in D \wedge v' \in D \tag{1}$$

A dependent risk graph is a risk graph with the set of vertices and relations divided into two disjoint sets representing the assumptions and the target. We write $A \rhd T$ to denote the dependent risk graph where A is the set of vertices and relations representing the assumptions and T is the set of vertices and relations representing the target. For a dependent risk graph $A \rhd T$ to be well-formed we have the following requirements:

$$v \rightarrow v' \in A \;\Rightarrow\; v \in A \wedge v' \in A \cup T \tag{2}$$
$$v \rightarrow v' \in T \;\Rightarrow\; v' \in T \wedge v \in A \cup T \tag{3}$$
$$v \rightarrow v' \in A \cup T \;\Rightarrow\; v \in A \cup T \wedge v' \in A \cup T \tag{4}$$
$$A \cap T \;=\; \emptyset \tag{5}$$

Note that (4) is implied by (2) and (3). This means that if $A \rhd T$ is a well-formed dependent risk graph then $A \cup T$ is a well-formed basic risk graph.

Since a risk graph has only one type of vertex and one type of relation, we must translate the vertices and relations of a CORAS diagram into a risk graph in order to make the risk graph calculus applicable. We interpret a set of threats $t_1, \ldots, t_n$ with initiate relations to the same threat scenario s as follows: The vertex s is decomposed into n parts, where each sub-vertex $s_j, j \in [1..n]$ corresponds to the part of s initiated by the threat t_j. We combine a threat t_j, initiate relation i_j with likelihood P_j and sub-vertex s_j into a new threat scenario vertex: *Threat t_j initiates s_j with likelihood P_j*. We interpret a set of incidents $u_1, \ldots, u_n$ with impacts relations $i_1, \ldots, i_n$ to the same asset a as follows: The vertex a is decomposed into n parts, where each sub-vertex $a_j, j \in [1..n]$ corresponds to the part of a harmed by the incident u_j. The impacts relation from u_j is interpreted as a relation with likelihood 1. Each sub-vertex a_j is interpreted as the threat scenario vertex: *Incident u_j harms asset a with impact i_j*. Figure 49 illustrates how the threat diagram in Figure 38 can be interpreted as a dependent risk graph following these steps. For the full procedure for instantiating the risk graphs in CORAS see Brændeland et al. (2010).

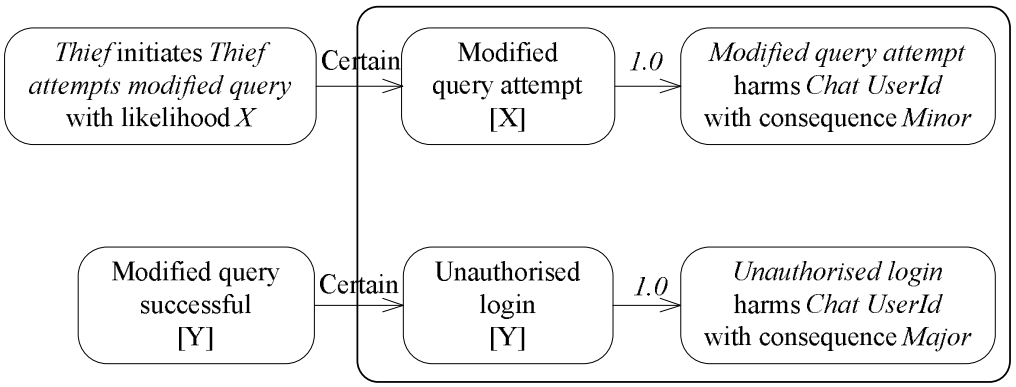

Figure 49: Representing the threat diagram in Figure 38 as a dependent risk graph

Given two sub-graphs D, D', we let $i(D, D')$ denote D's interface towards D'. This interface is obtained from D by keeping only the vertices and relations that D' depends on directly. We

define $i(D, D')$ formally as follows:

$$i(D, D') \stackrel{\text{def}}{=} \{v \in D \mid \exists v' \in D' : v \rightarrow v' \in D \cup D'\} \cup \{v \rightarrow v' \in D \mid v' \in D'\} \tag{6}$$

Let for example

$$\begin{aligned} A_1' &= \{Tmq\} \\ T_1' &= \{Tmq \xrightarrow{Certain} Ma, Ma(X), Ma \rightarrow MahC, MahC\} \\ A_1'' &= \{Ms(Y)\} \\ T_1'' &= \{Ms \xrightarrow{Certain} Ul, Ul(Y), Ul \xrightarrow{UlhC}, UlhC\} \end{aligned}$$

represent different sub-graphs of the dependent risk graph in Figure 49, based on the abbreviations in Table 10. Then $i(A_1', T_1') = \{TT\}$ and $i(T_1', T_1'') = \emptyset$.

Tmq	=	*Thief* initiates *Thief attempts modified query* with likelihood X.
Ma	=	*Modified query attempt*
$MahC$	=	*Modified query attempt* harms *Chat UserId* with consequence *Minor*
Ms	=	*Modified query successful*
Ul	=	*Unauthorised login*
$UlhC$	=	*Unauthorised login* harms *Chat UserId* with consequence *Major*

Table 10: Abbreviations of vertices

We do not consider the incident IDs such as *UI3* to be part of the incident names. The unwanted incident *Modified query successful* in assumption A_1'' in Figure 49 is therefore considered the same as the unwanted incident in the assumption in Figure 38.

A dependent risk graph on the form $A \rhd T$ means that all sub-graphs of T that only depends on the parts of A's interface towards T that actually holds, must also hold. The semantics of a dependent risk graph $A \rhd T$ is defined by:

$$[\![\ A \rhd T\ ]\!] \stackrel{\text{def}}{=} \forall T' \subseteq T : [\![\ i(A \cup T \setminus T', T')\ ]\!] \Rightarrow [\![\ T'\ ]\!] \tag{7}$$

Note that if the assumption of a dependent graph $A \rhd T$ is empty (i.e. $A = \emptyset$) it means that we have the graph T, that is the semantics of $\emptyset \rhd T$ is equivalent to that of T.

7.2.2 Combining interface risks into component risks

In the following examples we illustrate how to deduce the validity of a combined threat diagram obtained from two dependent threat diagrams of two interacting interfaces. We explain only the subset of the calculus for dependent risk graphs that we need in the examples. The rules of the calculus are of the form

$$\frac{P_1 \quad P_2 \quad \ldots \quad P_i}{C}$$

where $P_1, \ldots, P_i$ is referred to as the premises and to C as the conclusion. The interpretation is as follows: if the premises are valid so is the conclusion.

In order to reason about dependencies we first explain what is meant by dependency. The relation $D \ddagger D'$ means that D' does not depend on any vertex or relation in D. This means that D does not have any interface towards D' and that D and D' have no common elements:

Definition 1 (Independence)

$$D \ddagger D' \Leftrightarrow D \cap D' = \emptyset \wedge i(D, D') = \emptyset$$

Note that $D \ddagger D'$ does not imply $D' \ddagger D$.

The following rule allows us to remove part of the target scenario as long as it is not situated in-between the assumption and the part of the target we want to keep.

Rule 2 (Target simplification)

$$\frac{A \rhd T \cup T' \quad T' \ddagger T}{A \rhd T}$$

The following rule allows us to remove a part of the assumption that is not connected to the rest.

Rule 3 (Assumption simplification)

$$\frac{A \cup A' \rhd T \quad A \ddagger A' \cup T}{A' \rhd T}$$

Example 24 (Target and assumption simplification) In Examples 13 and 14 we saw that the risks related to the *login* operation of the *Chat* interface and the *validate* operation of the *UserMgr* interface depend on each other. In order to obtain the combined risks of these operations we shall combine the dependent threat diagrams from Figures 38 and 40 into a new dependent diagram. Let $A'_1 \cup A''_1 \rhd T'_1 \cup T''_1$ represent the diagram in Figure 49. We assume that the dependent diagram in Figure 49 is correct, that is: we assume the validity of

$$A'_1 \cup A''_1 \rhd T'_1 \cup T''_1$$

Since $i(T'_1, T''_1) = \emptyset$ and $T'_1 \cap T''_1 = \emptyset$, it follows by Rule 1 that $T''_1 \ddagger T'_1$. Hence, by applying Rule 2 we can deduce

$$A'_1 \cup A''_1 \rhd T'_1 \tag{8}$$

Since we also have $A''_1 \ddagger A'_1 \cup T'_1$, we can apply Rule 3 and deduce

$$A'_1 \rhd T'_1 \tag{9}$$

Using the same procedure we can also deduce the validity of

$$A''_1 \rhd T''_1 \tag{10}$$

□

In order to support the sequential composition of several dependent threat diagrams into a new dependent diagram we need a new rule which is not part of the previously defined basic set of rules. The rule states that if we have two dependent diagrams $A_1 \rhd T_1$ and $A_2 \rhd T_2$ where the vertex v in A_1 leads to a vertex v' in T_1 and the same vertex v' occurs in A_2, and the two dependent diagrams otherwise are disjoint, then we may deduce $A_1 \cup A_2 \cup \{v\} \rhd \{v \rightarrow v', v'\} \cup T_1 \cup T_2$.

Rule 4 (Sequential composition)

$$\frac{A_1 \cup \{v\} \rhd \{v \rightarrow v', v'\} \cup T_1 \quad A_2 \cup \{v'\} \rhd T_2 \quad (A_1 \cup T_1) \cap (A_2 \cup T_2) = \emptyset}{A_1 \cup A_2 \cup \{v\} \rhd \{v \rightarrow v', v'\} \cup T_1 \cup T_2}$$

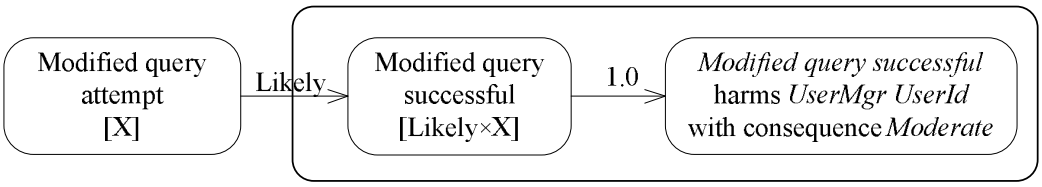

Figure 50: Representing the threat diagram in Figure 40 as a dependent risk graph

where v does not occur in A_1, v' does not occur in A_2, and neither $v \rightarrow v'$ nor v' occurs in T_1. The soundness of this rule is shown in Appendix A.

Example 25 (Combining dependent diagrams) Let $A_2 \triangleright T_2$ represent the diagram in Figure 50, which shows the threat diagram in Figure 40 interpreted as a risk graph. We use the shorthand notations for the translated and transformed elements listed in Table 10.

Let $A_1''' \triangleright T_1'''$ be $A_1'' \triangleright T_1''$ where Y is substituted by $Likely \times X$. Since we have $A_1'' \triangleright T_1''$ by 10, we also have

$$A_1''' \triangleright T_1''' \tag{11}$$

We assume the validity of

$$A_2 \triangleright T_2 \tag{12}$$

By (11), (12) and the fact that $(A_2 \setminus \{Ma\} \cup T_2 \setminus \{Ms([Likely \times X])\}) \cap (A_1''' \setminus \{Ms([Likely \times X])\} \cup T_1''') = \emptyset$, we can apply Rule 4 to deduce

$$A_2 \triangleright T_2 \cup T_1''' \tag{13}$$

By (9), (13) and the fact that $(A_1' \setminus \{Tmq\} \cup T_1' \setminus \{Ma([X])\}) \cap (A_2 \setminus \{Ma([X]) \cup T_2 \cup T_1'''\}) = \emptyset$ we can apply Rule 4 once more to deduce

$$A_1' \triangleright T_1' \cup T_2 \cup T_1'''$$

which corresponds to the combined dependent threat diagram for the *Chat* and *UserMgr* interfaces in Figure 51, given the interpretations above. □

7.2.3 Evaluating component risks

At the component level, we need to know the harm against component assets. The risk value of a risk is the combination of its likelihood and consequence value. The likelihood of an incident is determined from the probability of an attack and the probability of success given an attack. As explained during the risk estimation step when we do a risk analysis of a component that may exist on different platforms during its lifetime, we cannot include the external threats in the analysis. Hence, we are not able to estimate the absolute likelihood values of risks, only the conditional likelihoods that incidents occur given attacks or faults of external systems. The probability of an attack being successful reflects the component vulnerabilities and the effectiveness of control mechanisms.

There are different strategies we can choose for evaluating risks in the lack of absolute likelihood values: One is to evaluate risks with regard to the conditional likelihood values that we have estimated. Another option is to make explicit assumptions about the likelihood values of the assumed threat scenarios and incidents. For example in the dependent threat in Figure 38

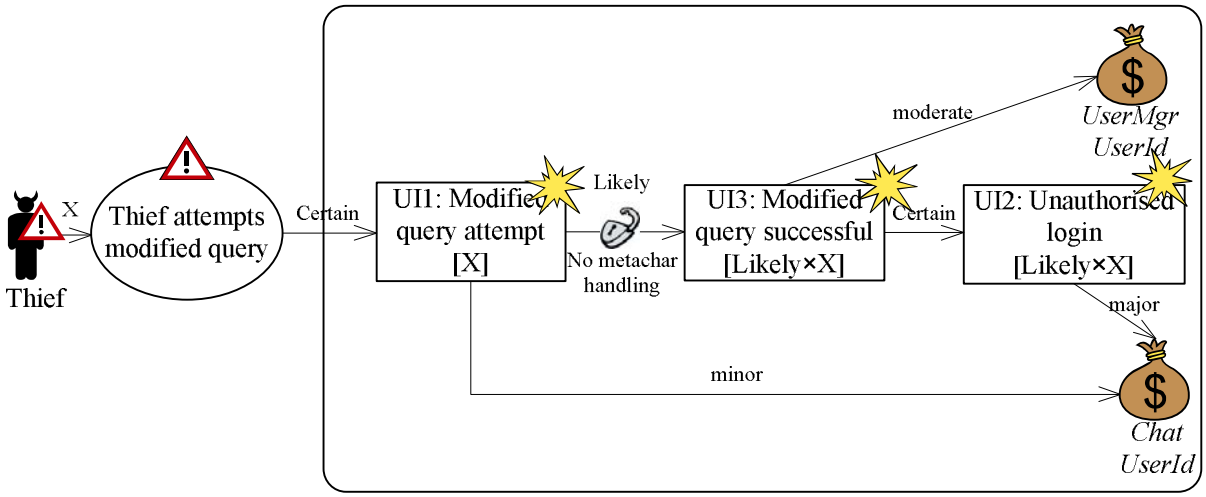

Figure 51: Combined threat diagram for the *Chat* and *UserMgr*

we could say, given that the likelihood of the assumed threat scenario *Thief attempts modified query* is *Likely*, that the likelihood of the incident *Modified query attempt* in the target is also *Likely*. In such a case it is important to make assumptions that are realistic with regard to potential platforms the component may be deployed in. If the assumed likelihood values are too high it may give the impression that the component is less trustworthy than it really is. If the assumed likelihood values are too low it may render the component useless for potential users because we without any real reason impose assumptions that their infrastructure does not fulfil (Lund et al., 2010).

If we want to combine two separate risk analyses that make explicit assumptions about the likelihood values of each other, we need to check that the assumptions of one analysis is consistent with the target of the other and vice versa. If either analysis has made assumptions that are too strong or too weak they must be adjusted to obtain the correct values for the combined system.

Example 26 (Evaluation of instant messaging risks) The combined threat diagram in Figure 51 shows incidents harming interface assets *Chat UserId* and *UserMgr UserId*. In Section 5.2 we decided that harm towards interface assets *Chat UserId* and *UserMgr UserId* implies the same level of harm to the component asset *UserId*.

For the purpose of the example we have chosen to use the latter of the two strategies described above to evaluate the risks. That is, we make explicit assumptions about the likelihood values of the assumed threat scenarios and incidents. In order to make visible the assumptions on which the risk evaluations rely, we include the assumption in the risk evaluation table of each asset.

If we choose to instantiate X with *Unlikely* in the dependent threat diagram in Figure 51 the incidents *UI1* and *UI2* become acceptable, whereas *UI3* is unacceptable. However, we think this assumption is too strict and choose *Likely* instead. We make similar assumptions for the assumed threat scenarios and incidents affecting the risk level of the other component assets as documented in the risk evaluation matrices in Tables 12 to 14.

The incidents affecting *User Id*, and their consequence and likelihood based on the assumed likelihood *Likely* of the assumed threat, are documented in the risk evaluation matrix in Table 11. The risk value of the incidents *UI1*, *UI2* and *UI3* are categorised as unacceptable. We should therefore identify protection mechanisms and revise the component specifications accordingly, to ensure that the requirements to protection are met.

Consequence	Likelihood				
	Unlikely	Possible	Likely	Almost certain	Certain
	Assuming *Thief* initiates *Thief attempts modified query* with likelihood *Likely*				
Minor			UI1		
Moderate		UI2			
Major		UI3			

Table 11: Risk evaluation for *UserId*

Consequence	Likelihood				
	Unlikely	Possible	Likely	Almost certain	Certain
	Assuming *Impersonator* initiates *Impersonator poses as buddy* with likelihood *Likely*				
Minor			UM1		
Moderate					
Major					

Table 12: Risk evaluation for *Message*

Consequence	Likelihood				
	Unlikely	Possible	Likely	Almost certain	Certain
	Assuming *Spimmer* initiates *Spimming* with likelihood *Likely* and *Adversary* initiates *Flooding attack* with likelihood *Likely*				
Minor			UE1 UE2		
Moderate					
Major					

Table 13: Risk evaluation for *Availability*

Consequence	Likelihood				
	Unlikely	Possible	Likely	Almost certain	Certain
	Assuming *Hacker* initiates *Send crafted file* with likelihood *Possible*				
Minor					
Moderate	UF1				
Major					

Table 14: Risk evaluation for *Media file*

One possible treatment, as depicted in the treatment diagram in Figure 52, is to check all possible metacharacters to the *UserMgr* interface. To save space, we have not included a revised

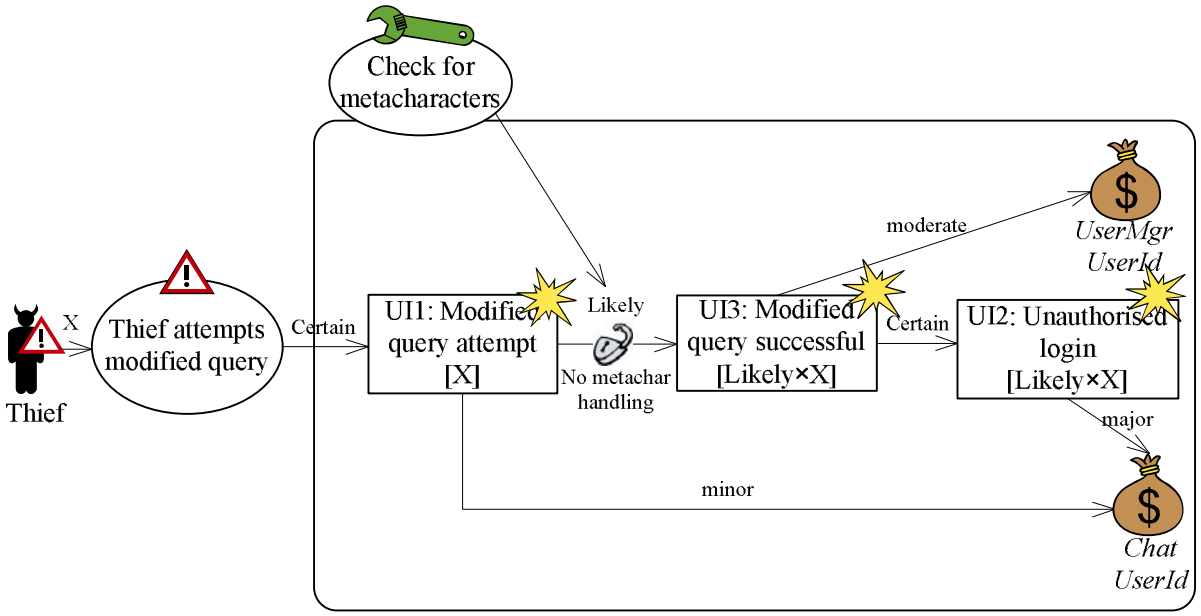

Figure 52: Treatment diagram for the Unauthorised login risk

component specification in this example. □

Summary of component-based risk analysis: workflow 2, step 3
• **Objective:** Combine interface risk specifications into a risk picture for the component as a whole. Decide which of the identified risks are acceptable and which of the risks that must be treated. Identify treatments for the unacceptable risk. Revise the component specification if necessary.
• **Input documentation:** Dependent CORAS threat diagrams with estimated likelihoods and consequences; CORAS risk diagrams; CORAS asset diagrams; the risk evaluation criteria.
• **Output documentation:** Combined dependent threat diagrams documenting component risks risks and sequence diagrams specifying both normal interactions and risk interactions at the component level; CORAS treatment diagrams documenting the identified treatments; revised component specification.
• **Adaptations to CORAS:** Risk composition is normally not a part of the CORAS method. Neither is the specification of risk interactions in sequence diagrams.

Table 15: Component protection specification

8 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

(McGraw, 2006; Goseva-Popstojanova et al., 2003; Cortellessa et al., 2005; Sindre and Opdahl, 2000, 2005; McDermott and Fox, 1999; McDermott, 2001).

There are also several proposals for including security requirements into the requirements phase, such as for example in SecureUML (Lodderstedt et al., 2002) and UMLsec (Jürjens, 2005). A security requirement is a requirement to the protection of information security in terms of confidentiality, integrity, availability, nonrepudiation, accountability and authenticity of information (ISO, 2004). 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.

Our framework for component-based risk analysis and approaches such as SecureUML and UML are complementary and may be used at different stages during a robust development process. While SecureUML and UMLsec may be used for specifying security requirements, our approach may be used to identify and analyse the probability that security requirements are violated. The violation of a security requirement may constitute an unwanted incident, since it may cause harm to system assets.

Jürjens and Houmb (2004) have proposed an approach to risk-driven development of security-critical systems using UMLsec. Their approach uses CORAS for the purpose of identifying, analysing and evaluating risks. The security risks that are found to be unacceptable are treated by specifying security requirements using UMLsec (Jürjens, 2005). Our approach is similar to theirs, in that they propose to combine CORAS with model-driven development in a security critical setting. One difference is that we focus on component-based development which requires a modular approach to risk analysis, whereas Jürjens and Houmb (2004) have no particular focus on component-oriented specification.

The model-driven performance risk analysis method by Cortellessa et al. (2005) takes into account both system level behaviour and hardware specific information. 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 (1999); McDermott (2001). Sindre and Opdahl (2000, 2005) 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 incident is inspired by Fault Tree Analysis (FTA) and Event Tree Analysis (ETA). FTA (IEC, 1990) is a top-down approach that breaks down an incident into smaller events. The events are structured into a logical binary tree, with and/or gates, which shows possible routes leading to the incident from various failure points. ETA (IEC, 1995) is a bottom-up approach to calculate consequences of events. ETA focuses on illustrating the (forward) consequences of an event and the probabilities of these.

CORAS threat diagrams combine the features of both fault tree and event tree. CORAS threat diagrams are more general since they do not have the same binary restrictions and causes does not have to be connected through a specified logical gate. CORAS diagrams may have more than one top vertex and can be used to model assets and consequences. Moreover, in CORAS likelihoods may be assigned to both vertices and relations, whereas in fault trees only the vertices have likelihoods. The likelihood of a vertex in a CORAS diagram can be calculated from the likelihoods of its parent vertices and connecting relations. The possibility to assign likelihoods to both vertices and relations has methodological benefits during brainstorming sessions because it may be used to uncover inconsistencies. Uncovering inconsistencies helps to

clarify misunderstandings and pinpoint aspects of the diagrams that must be considered more carefully. Another difference between fault trees and CORAS threat diagrams is that fault trees focus more on the logical decomposition of an incident into its constituents, and less on the causal relationship between events which is the emphasis in CORAS. The most significant difference between CORAS and other threat modelling techniques for our purpose, however, is the extension of CORAS with so called dependent diagrams, which facilities the documentation of environment assumptions. Dependent threat diagrams are crucial for obtaining the modularity of our approach as discussed in Section 6.2.1. We are not aware of any threat modelling techniques apart from dependent CORAS that are designed to capture context dependencies.

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

9 Conclusion and discussion

We have presented a framework for component-based risk analysis and provided suggestions for integrating it step-by-step into a component-based development process. The proposed approach focuses on integrating risk analysis into the early stages of component development. Integrating security into the requirements and design phase is important for several reasons: First, it aids developers to discover design flaws at an early stage. Second, it aids developers to ensure that components are developed in accordance with the desired protection level and reduces the need for ad hoc integration of security mechanisms after the component has been implemented.

In this section we summarise the steps we have taken in order to adjust CORAS into a method for component-based risk analysis. We also discuss the extent to which the presented framework fulfils the overall requirement of encapsulation and modularity without compromising the feasibility of the approach or the common understanding of risk. We also discuss our findings with regard to further research needed in order to obtain a full method for component-based risk analysis.

The requirement that a component needs to be distinguished from its environment in order to be independently deployable, implies that we do not allow any concepts that are external to a component to be part of the component-based risk analysis framework. With regard to the requirements to protection step we therefore adjusted the CORAS method so that the target of analysis is the component or component interface being analysed. Furthermore, we have no external stakeholders, but identify assets on behalf of the component or component interface which is the target of analysis.

Identifying the target of analysis as the component itself limits the scope of analysis compared to a conventional risk analysis. It means for example that external threats are not included in the analysis as such, even though they affect the overall level of risks. This limitation is discussed further with regard to the risk identification and analysis step.

Identifying assets on behalf of the component or component interfaces has the consequence that tasks that are normally the responsibility of the stakeholders, must be conducted by the component owner, or the development team in an understanding with the component owner. These tasks entail identifying assets, establishing the protection level of assets and deciding the consequence values of incidents.

A component asset may for example be confidentiality of information handled by a component interface. Ultimately a component buyer may be interested in assets of value for *him*, such

as for example the cost of using a component, or his own safety, which are not the same as the assets of the component he is buying. One solution to this may be to identify the component user as a component with its own assets and analyse how incidents towards the assets of the bought component affect assets of the owner. A simpler solution would be to identify the component user's assets as indirect assets with regard to the component asset and evaluate how a risk harming an asset, such as confidentiality of information, affects an asset of the component user such as for example cost of use.

According to our conceptual model of component-based risk analysis in Figure 5 a component is a collection of interfaces. Hence, the set of component assets is the union of the assets of its interfaces. When we decompose the component into interfaces we must therefore decide for each asset which of the component interfaces it belongs to. The assignment of assets to interfaces is not part of the original CORAS method. In order to achieve this task we introduced the following rules of thumb:

- An asset referring to data handled by several component interfaces, is decomposed into one asset for each interface;
- An asset referring to data which is handled by only one interface is assigned to the interface handling it;
- An asset referring to a property of data or a service is decomposed and assigned to interfaces handling the data or contributing to services for which the property is of relevance.

As explained above external threats are not part of a component risk analysis, due to the requirement that a component needs to be distinguished from its environment. This makes sense since a component may be deployed at different platforms during its lifetime and the types of external threats towards a component may change depending on the platform. In order to analyse component risk without including threats in the analysis we use so called dependent threat diagrams that allow us to document assumptions about the environment and parameterise likelihood values arising from external threats.

To obtain the whole risk picture for a component running on a given platform, we must compose the risk analysis of the component with the risk analysis of the platform, or perform an analysis of the platform if none exists. In order to reason about mutually dependent diagrams, that is, where the target of one diagram is the assumption of another, we apply a calculus for so called dependent risk graphs (Brændeland et al., 2010). We can apply the calculus to check that a dependent threat diagram is a valid composition of two or more dependent threat diagrams. The actual carrying out of the proofs are quite cumbersome, however, as we saw in an example. In order to the make use of dependent threat diagrams feasible in a real risk it should be supported by a tool that could perform the derivations automatically or semi-automatically, such as an interactive theorem prover like Isabelle[3].

We use sequence diagrams in STAIRS (Haugen and Stølen, 2003; Haugen et al., 2004; Runde, 2007; Refsdal, 2008), which is a formalisation of the main concepts in UML 2.0 sequence diagram, to specify interface interactions. We want to update risk analysis documentation when a component-based system is upgraded with a new component. To achieve this we would like to specify component risks as an integrated part of the component specification, using the same type of specification techniques. We therefore propose to use STAIRS also for the purpose of specifying risk interactions based on risk analysis documentation in the form of dependent threat diagrams.

Risks are identified in relation to the existing interface specifications. This implies that incidents are events that are allowed within the specified behaviour but not necessarily intended.

[3] `http://isabelle.in.tum.de/`

We can therefore specify component risks by supplementing component specification with incidents that may happen as part of the normal behaviour. However, the current version of STAIRS has no facilities for documenting assumptions about the environment. Furthermore, the formal semantics of STAIRS as defined by Haugen and Stølen (2003) and Haugen et al. (2004) does not include constructs for representing vulnerabilities, incidents or harm to assets. This means that some of the information documented in the dependent threat diagrams is lost in the translation into sequence diagrams. In order to fully integrate risk behaviour in interface specifications we need a denotational trace semantics that captures risk relevant aspects such as assets and incidents formally. For an approach to representing risk behaviour in a denotational trace semantics see Brændeland and Stølen (2006, 2007). Furthermore, we need to map syntactical specifications using sequence diagrams to a formal semantics for risk behaviour. This is a topic for future research within the field of component-based risk analysis.

Due to the preliminary state of the presented framework and the qualitative nature of the criteria that we have evaluated our presented framework against, we have used a case-based example to evaluate our approach. A case-based evaluation can be useful in the early phase of a research process, in the creation of ideas and insights (Galliers, 1992). In order to empirically verify the feasibility of the approach and its influence on component quality and software development progress, further evaluations are necessary.

The CORAS method, which our approach is based on, was compared to six other risk analysis methods in a test performed by the Swedish Defence Research Agency in 2009 (Bengtsson et al., 2009). The purpose of the test was to check the relevance of the methods with regard to assessing information security risks during the different phases of the life cycle of IT systems, on behalf of the Swedish Defence Authority. In the test CORAS got the highest score with regard to relevance for all phases of the life cycle. According to the report the good score of CORAS is due to the well established modelling technique of CORAS for all types of systems and the generality of the method. The applicability of the CORAS language has been thoroughly evaluated in a series of industrial case studies, and by empirical investigations documented by Hogganvik and Stølen (2005a,b, 2006).

We believe the case-based evaluation of the adapted CORAS method for component-based risk analysis shows some promising prospects with regard to improving component robustness. By integrating risk analysis into the development process and documenting risks at the component level, developers acquire the necessary documentation to easily update risk analysis documentation in the course of system changes. Since we specify component risks as an integrated part of a component specification, the analysis of how changes in a component affect system risks becomes straightforward. If we modify one of the component interfaces in the instant messaging example, we only need to analyse how the changes affect that particular interface. The risk level for the instant messaging component can be obtained using the operations for composition described in Section 7.2. Finally, the explicit documentation of assumptions on which risk analysis results depend, facilitates the comparison of component risks independent of context, which is a prerequisite for creating a market for components with documented risk properties.

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-driven risk analysis) and SECURIS (152839/ 220). We would like to thank Bjørnar Solhaug for extensive comments and very helpful suggestions for improvements on an earlier version of this report. We would also like to thank Øystein Haugen for advice on modelling in the UML.

References

Abadi, M. and Lamport, L. (1995). Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534.

Ahrens, F. (2010). Why it's so hard for Toyota to find out what's wrong. The Washington Post.

Alberts, C. J., Behrens, S. G., Pethia, R. D., and Wilson, W. R. (1999). Operationally critical threat, asset, and vulnerability evaluation (OCTAVE) framework, version 1.0. Technical Report CMU/SEI-99-TR-017. ESC-TR-99-017, Carnegie Mellon. Software Engineering Institute.

Bengtsson, J., Hallberg, J., Hunstad, A., and Lundholm, K. (2009). Tests of methods for information security assessment. Technical Report FOI-R–2901–SE, Swedish Defence Research Agency.

Brændeland, G., Refsdal, A., and Stølen, K. (2010). Modular analysis and modelling of risk scenarios with dependencies. *Journal of Systems and Software*, 83(10):1995–2013.

Brændeland, G. and Stølen, K. (2006). Using model-based security analysis in component-oriented system development. In *QoP '06: Proceedings of the 2nd ACM workshop on Quality of protection*, pages 11–18, New York, NY, USA. ACM Press.

Brændeland, G. and Stølen, K. (2007). A semantic paradigm for component-based specification integrating a notion of security risk. In *Formal Aspects in Security and Trust, Fourth International Workshop, FAST*, volume 4691 of *Lecture Notes in Computer Science*, pages 31–46. Springer.

Broy, M. and Stølen, K. (2001). *Specification and development of interactive systems – Focus on streams, interfaces and refinement.* Monographs in computer science. Springer.

Cheesman, J. and Daniels, J. (2001). *UML Components. A simple process for specifying component-based software.* Component software series. Addison-Wesley.

Cortellessa, V., Goseva-Popstojanova, K., Appukkutty, K., Guedem, A., Hassan, A. E., Elnaggar, R., Abdelmoez, W., and Ammar, H. H. (2005). Model-based performance risk analysis. *IEEE Transactions on Software Engineering*, 31(1):3–20.

Crnkovic, I. and Larsson, M. (2002). *Building reliable component-based software systems.* Artech-House.

cve (2005). CVE-2005-2310. National institute of standards and technology. National vulnerability database.

den Braber, F., Dimitrakos, T., Gran, B. A., Lund, M. S., Stølen, K., and Aagedal, J. Ø. (2003). *UML and the Unified Process*, chapter The CORAS methodology: model–based risk management using UML and UP, pages 332–357. IRM Press.

den Braber, F., Hogganvik, I., Lund, M. S., Stølen, K., and Vraalsen, F. (2007). Model-based security analysis in seven steps – a guided tour to the CORAS method. *BT Technology Journal*, 25(1):101–117.

Farquhar, B. (1991). One approach to risk assessment. *Computers and Security*, 10(1):21–23.

Galliers, R. (1992). *Informations systems research*, chapter Choosing Information Systems Research Approaches. Blacwell scientific publications.

Goseva-Popstojanova, K., Hassan, A. E., Guedem, A., Abdelmoez, W., Nassar, D. E. M., Ammar, H. H., and Mili, A. (2003). Architectural-level risk analysis using UML. *IEEE Transactions on Software Engineering*, 29(10):946–960.

Haugen, Ø., Husa, K. E., Runde, R. K., and Stølen, K. (2004). Why timed sequence diagrams require three-event semantics. Technical Report 309, University of Oslo, Department of Informatics.

Haugen, Ø. and Stølen, K. (2003). STAIRS – Steps to Analyze Interactions with Refinement Semantics. In *Proceedings of the Sixth International Conference on UML (UML'2003)*, volume 2863 of *Lecture Notes in Computer Science*, pages 388–402. Springer.

He, J., Josephs, M., and Hoare, C. A. R. (1990). A theory of synchrony and asynchrony. In *IFIP WG 2.2/2.3 Working Conference on Programming Concepts and Methods*, pages 459–478. North Holland.

Hogganvik, I. and Stølen, K. (2005a). On the comprehension of security risk scenarios. In *13th International Workshop on Program Comprehension (IWPC 2005)*, pages 115–124. IEEE Computer Society.

Hogganvik, I. and Stølen, K. (2005b). Risk analysis terminology for IT systems: Does it match intuition? In *Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE'05)*, pages 13–23. IEEE Computer Society.

Hogganvik, I. and Stølen, K. (2006). A graphical approach to risk identification, motivated by empirical investigations. In *Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06)*, volume 4199 of *LNCS*, pages 574–588. Springer.

IEC (1990). *Fault Tree Analysis (FTA)*. IEC. IEC 61025.

IEC (1995). *Event Tree Analysis in Dependability management – Part 3: Application guide – Section 9: Risk analysis of technological systems*. IEC. IEC 60300.

ISO (2004). *Information Technology – Security techniques – Management of information and communications technology security – Part 1: Concepts and models for information and communications technology security management*. ISO/IEC. ISO/IEC 13335-1:2004.

ISO (2009a). *Risk management – Principles and guidelines*. ISO. ISO 31000:2009.

ISO (2009b). *Risk management – Vocabulary*. ISO. ISO Guide 73:2009.

Jones, C. B. (1981). *Development Methods for Computer Programmes Including a Notion of Interference*. PhD thesis, Oxford University.

Jürjens, J., editor (2005). *Secure systems development with UML*. Springer.

Jürjens, J. and Houmb, S. H. (2004). Risk-Driven Development Of Security-Critical Systems Using UMLsec. In *IFIP Congress Tutorials*, pages 21–54. Kluwer.

Kruchten, P., editor (2004). *The rational unified process. An introduction*. Addison-Wesley.

Lodderstedt, T., Basin, D. A., and Doser, J. (2002). SecureUML: A UML-based modeling language for model-driven security. In *Proceedings of the 5th International Conference, UML 2002 – The Unified Modeling Language*, volume 2460 of *Lecture Notes in Computer Science*, pages 426–441. Springer.

Lund, M. S., Solhaug, B., and Stølen, K. (2010). *Model Driven Risk Analysis. The CORAS Approach*. Springer.

Mannan, M. and van Oorschot, P. C. (2004). Secure public instant messaging. In *Proceedings of the Second Annual Conference on Privacy, Security and Trust*, pages 69–77.

McDermott, J. P. (2001). Abuse-case-based assurance arguments. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, pages 366–376. IEEE Computer Society.

McDermott, J. P. and Fox, C. (1999). Using abuse case models for security requirements analysis. In *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC 1999)*, pages 55–. IEEE Computer Society.

McGraw, G. (2006). *Sofware security: Building security in*. Software Security Series. Adison-Wesley.

Misra, J. and Chandy, K. M. (1981). Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426.

OMG (2007). *OMG Unified Modeling Language (OMG UML), Superstructure*. Object Management Group (OMG), 2.1.2 edition.

Redmill, F., Chudleigh, M., and Catmir, J. (1999). *System safety: HazOp and software HazOp*. Wiley.

Refsdal, A. (2008). *Specifying Computer Systems with Probabilistic Sequence Diagrams*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo.

Rumbaugh, J., Jacobsen, I., and Booch, G. (2005). *The unified modeling language reference manual*. Addison-Wesley.

Runde, R. K. (2007). *STAIRS - Understanding and Developing Specifications Expressed as UML Interaction Diagrams*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo.

Runde, R. K., Haugen, Ø., and Stølen, K. (2006). The Pragmatics of STAIRS. In *4th International Symposium, Formal Methods for Components and Objects (FMCO 2005)*, volume 4111 of *Lecture Notes in Computer Science*, pages 88–114. Springer.

Sans (2005). The SANS top 20 list. The twenty most critical internet security vulnerabilities. SANS. `http://www.sans.org/top20/`.

Secunia (2006). Secunia advisory: sa12381.

Sindre, G. and Opdahl, A. L. (2000). 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.

Sindre, G. and Opdahl, A. L. (2005). Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44.

Solhaug, B. (2009). *Policy Specification Using Sequence Diagrams. Applied to Trust Management*. PhD thesis, Faculty of Social Sciences, University of Bergen.

Standards Australia (2004). *Information security risk management guidelines*. Standards Australia, Standards New Zealand. HB 231:2004.

Swiderski, F. and Snyder, W. (2004). *Threat Modeling*. Microsoft Press.

Troelstra, A. S. and Schwichtenberg, H. (2000). *Basic Proof Theory*. Cambridge tracts in theoretical computer science. Cambridge University Press, 2nd edition.

Verdon, D. and McGraw, G. (2004). Risk analysis in software design. *IEEE Security & Privacy*, 2(4):79–84.

Watson, T. and Kriens, P. (2006). OSGi component programming. Tutorial held at Eclipsecon 2006.

A Proofs

Theorem 1

$$\begin{aligned}(((A_1 \cup T_1) \cap (A_2 \cup T_2) = \emptyset) \wedge v \notin A_1 \wedge v' \notin A_2' \wedge (\{v \rightarrow v', v'\} \cap T_1 = \emptyset) \\ \wedge [\![\ A_1 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1\ ]\!] \wedge [\![\ A_2 \cup \{v'\} \triangleright T_2\ ]\!] \Rightarrow) \\ [\![\ A_1 \cup A_2 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1 \cup T_2\ ]\!]\end{aligned}$$

We assume that the three risk graphs: (1) $A_1 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1$; (2) $A_2 \cup \{v'\} \triangleright T_2$; and (3) $A_1 \cup A_2 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1 \cup T_2$ fulfil well-formed requirements (2)-(5).

PROOF:

$\langle 1 \rangle 1$. ASSUME: 1. $(A_1 \cup T_1) \cap (A_2 \cup T_2) = \emptyset$
2. $v \notin A_1$
3. $v' \notin A_2'$
4. $\{v \rightarrow v', v'\} \cap T_1 = \emptyset$
5. $[\![\ A_1 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1\ ]\!]$
6. $[\![\ A_2 \cup \{v'\} \triangleright T_2\ ]\!]$
PROVE: $[\![\ A_1 \cup A_2 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1 \cup T_2\ ]\!]$

$\langle 2 \rangle 1$. $\forall T \subseteq \{v \rightarrow v', v'\} \cup T_1 \cup T_2$:
$[\![\ i(A_1 \cup A_2 \cup \{v\} \cup (\{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus T, T)\ ]\!] \Rightarrow [\![\ T\ ]\!]$

$\langle 3 \rangle 1$. ASSUME: $T \subseteq \{v \rightarrow v', v'\} \cup T_1 \cup T_2$
PROVE: $[\![\ i(A_1 \cup A_2 \cup \{v\} \cup (\{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus T, T)\ ]\!] \Rightarrow [\![\ T\ ]\!]$

$\langle 4 \rangle 1$. ASSUME: $[\![\ i(A_1 \cup A_2 \cup \{v\} \cup (\{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus T, T)\ ]\!]$
PROVE: $[\![\ T\ ]\!]$

$\langle 5 \rangle 1$. LET: $T = T' \cup T''$ such that
$(T' \subseteq \{v \rightarrow v', v'\} \cup T_1) \wedge (T'' \subseteq T_2)$
PROOF: By $\langle 3 \rangle 1$.

$\langle 5 \rangle 2$. $[\![\ i(A_1 \cup A_2 \cup \{v\} \cup (\{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T''), T' \cup T'')\ ]\!]$
PROOF: By assumption $\langle 4 \rangle 1$, $\langle 5 \rangle 1$ and the rule of replacement (Troelstra and Schwichtenberg, 2000).

⟨5⟩3. $[\![\ T' \cup T''\ ]\!]$
⟨6⟩1. $[\![\ T'\ ]\!]$
⟨7⟩1. $[\![\ i(A_1 \cup \{v\} \cup \{v \to v', v'\} \cup T_1 \setminus T', T')\ ]\!] \Rightarrow [\![\ T'\ ]\!]$
PROOF: By assumption ⟨1⟩1.5, ⟨5⟩1 and definition (7).
⟨7⟩2. $[\![\ i(A_1 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1) \setminus T', T')\ ]\!]$
⟨8⟩1. $i(A_1 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1) \setminus T', T') \subseteq$
$i(A_1 \cup A_2 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T''), T' \cup T'')$
⟨9⟩1. $A_1 \cup A_2 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'') =$
$(A_1 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1) \setminus T') \cup (A_2 \cup T_2 \setminus T'')$
PROOF: By assumptions ⟨1⟩1.1, ⟨1⟩1.2, ⟨1⟩1.3, ⟨1⟩1.4 and ⟨5⟩1.
⟨9⟩2. ASSUME: $V \in i(A_1 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1) \setminus T', T')$
PROVE: $V \in i(A_1 \cup A_2 \cup \{v\} \cup$
$(\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T''), T' \cup T'')$
⟨10⟩1. CASE: $V = v_1$, that is V is a vertex.
⟨11⟩1. $v_1 \in A_1 \cup A_2 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'')$
⟨12⟩1. $v_1 \in A_1 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1) \setminus T'$
PROOF: By assumption ⟨9⟩2, assumption ⟨10⟩1 and definition (6).
⟨12⟩2. $v_1 \in (A_1 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1) \setminus T') \cup$
$(A_2 \cup T_2 \setminus T'')$
PROOF: By ⟨12⟩1 and elementary set theory
⟨12⟩3. Q.E.D.
PROOF: By ⟨9⟩1, ⟨12⟩2 and the rule of replacement (Troelstra and Schwichtenberg, 2000).
⟨11⟩2. $\exists v_2 \in T' \cup T'' : \{v_1 \to v_2\} \in (A_1 \cup A_2 \cup \{v\} \cup$
$(\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'')) \cup (T' \cup T'')$
⟨12⟩1. $\exists v_2 \in T' : \{v_1 \to v_2\} \in (A_1 \cup \{v\} \cup (\{v \to v'\} \cup T_1) \setminus T') \cup T'$
PROOF: By assumption ⟨9⟩2, assumption ⟨10⟩1 and definition (6).
⟨12⟩2. LET: $v_2 \in T'$ such that
$v_1 \to v_2 \in (A_1 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1) \setminus T') \cup T'$
PROOF: By ⟨12⟩1.
⟨12⟩3. $v_2 \in T' \cup T''$
PROOF: By ⟨12⟩2 and elementary set theory.
⟨12⟩4. $v_1 \to v_2 \in A_1 \cup A_2 \cup \{v\} \cup$
$(\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'')$
⟨13⟩1. $v_1 \to v_2 \in (A_1 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1) \setminus T') \cup T' \cup (A_2 \cup T_2 \setminus T'')$
PROOF: By ⟨12⟩2 and elementary set theory
⟨13⟩2. Q.E.D.
PROOF: By ⟨9⟩1, ⟨13⟩1 and the rule of replacement (Troelstra and Schwichtenberg, 2000).
⟨12⟩5. Q.E.D.
PROOF: By ⟨12⟩3, ⟨12⟩4 and $\exists$ introduction.
⟨11⟩3. Q.E.D.
PROOF: By ⟨11⟩1, ⟨11⟩2 and definition (6).
⟨10⟩2. CASE: $V = v_1 \to v_2$, that is V is a relation.
⟨11⟩1. $v_1 \to v_2 \in A_1 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1) \setminus T' \wedge v_2 \in T'$
PROOF: By assumption ⟨9⟩2, assumption ⟨10⟩2 and definition (6).
⟨11⟩2. $v_1 \to v_2 \in A_1 \cup A_2 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'')$
⟨12⟩1. $v_1 \to v_2 \in (A_1 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1) \setminus T') \cup (A_2 \cup T_2 \setminus T'')$
PROOF: By ⟨11⟩1, $\wedge$ elimination and elementary set theory.

⟨12⟩2. Q.E.D.
PROOF: By ⟨12⟩1, ⟨9⟩1 and the rule of replacement (Troelstra and Schwichtenberg, 2000).
⟨11⟩3. $v_2 \in T' \cup T''$
PROOF: By ⟨11⟩1, $\wedge$ elimination and elementary set theory.
⟨11⟩4. Q.E.D.
PROOF: By ⟨11⟩2, ⟨11⟩3 and definition (6).
⟨10⟩3. Q.E.D.
PROOF: The cases ⟨10⟩1 and ⟨10⟩2 are exhaustive.
⟨9⟩3. Q.E.D.
PROOF: $\subseteq$-rule.
⟨8⟩2. Q.E.D.
PROOF: By ⟨5⟩2 and ⟨8⟩1.
⟨7⟩3. Q.E.D.
PROOF: By ⟨7⟩1, ⟨7⟩2 and $\Rightarrow$ elimination.
⟨6⟩2. $[\![\ T''\ ]\!]$
⟨7⟩1. $[\![\ i(A_2 \cup \{v'\} \cup T_2 \setminus T'', T'')\ ]\!] \Rightarrow [\![\ T''\ ]\!]$
PROOF: By assumption ⟨1⟩1.6, ⟨5⟩1 and definition (7).
⟨7⟩2. $[\![\ i(A_2 \cup \{v'\} \cup T_2 \setminus T'', T'')\ ]\!]$
⟨8⟩1. $[\![\ i(A_1 \cup A_2 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T''), T' \cup T'')\ ]\!] \wedge [\![\ T'\ ]\!]$
PROOF: By ⟨5⟩2, ⟨6⟩1 and $\wedge$ introduction.
⟨8⟩2. $i(A_2 \cup \{v'\} \cup T_2 \setminus T'', T'') \subseteq$
$i(A_1 \cup A_2 \cup \{v\} \cup (\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T''), T' \cup T'') \cup T'$
⟨9⟩1. ASSUME: $V \in i(A_2 \cup \{v'\} \cup T_2 \setminus T'', T'')$
PROVE: $V \in i(A_1 \cup A_2 \cup \{v\} \cup$
$(\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T''), T' \cup T'') \cup T'$
⟨10⟩1. ASSUME: $V \notin i(A_1 \cup A_2 \cup \{v\} \cup$
$(\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T''), T' \cup T'') \cup T'$
PROVE: $\bot$
⟨11⟩1. $V \notin (A_1 \cup A_2 \cup \{v\} \cup$
$(\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'')) \cup T'$
PROOF: By assumption ⟨10⟩1, definition (6) and elementary set theory.
⟨11⟩2. $V \in (A_1 \cup A_2 \cup \{v\} \cup$
$(\{v \to v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'')) \cup T'$
⟨12⟩1. $V \in A_2 \cup \{v'\} \cup T_2 \setminus T''$
PROOF: By assumption ⟨9⟩1 and definition (6).
⟨12⟩2. $v' \notin T_2$
PROOF: By assumption ⟨1⟩1.6, the assumption that $A_2 \cup \{v'\} \triangleright T_2$ is well-formed and requirement (5).
⟨12⟩3. CASE: $V \in A_2$
⟨13⟩1. Q.E.D.
PROOF: By ⟨12⟩1, assumption ⟨12⟩3 and elementary set theory.
⟨12⟩4. CASE: $V = v'$
⟨13⟩1. CASE: $v' \in T'$
⟨14⟩1. Q.E.D.
PROOF: By assumption ⟨13⟩1 and elementary set theory.
⟨13⟩2. CASE: $v' \notin T'$
⟨14⟩1. $v' \notin T''$
PROOF: By ⟨12⟩2, ⟨5⟩1 and elementary set theory.

⟨14⟩2. Q.E.D.
PROOF: By assumption ⟨13⟩2, ⟨14⟩1, assumption ⟨12⟩4 and elementary set theory.
⟨13⟩3. Q.E.D.
PROOF: The cases ⟨13⟩1 and ⟨13⟩2 are exhaustive.
⟨12⟩5. CASE: $V \in T_2 \wedge V \notin T''$
⟨13⟩1. $T_2 \cap T' = \emptyset$
⟨14⟩1. $v \rightarrow v' \notin T_2$
PROOF: By ⟨12⟩2, the assumption that $A_2 \cup \{v'\} \triangleright T_2$ is well-formed and requirement (3).
⟨14⟩2. Q.E.D.
PROOF: By assumption ⟨1⟩1.1, ⟨12⟩2, ⟨14⟩1 and elementary set theory.
⟨13⟩2. Q.E.D.
PROOF: By assumption ⟨12⟩5, ⟨13⟩1 and elementary set theory.
⟨12⟩6. Q.E.D.
PROOF: The cases ⟨12⟩3, ⟨12⟩4 and ⟨12⟩5 are exhaustive.
⟨11⟩3. Q.E.D.
PROOF: By ⟨11⟩1, ⟨11⟩2 and $\perp$ introduction.
⟨10⟩2. Q.E.D.
PROOF: Proof by contradiction.
⟨9⟩2. Q.E.D.
PROOF: $\subseteq$ rule.
⟨8⟩3. Q.E.D.
PROOF: By ⟨8⟩1 and ⟨8⟩2.
⟨7⟩3. Q.E.D.
PROOF: By ⟨7⟩1, ⟨7⟩2 and $\Rightarrow$ elimination.
⟨6⟩3. Q.E.D.
PROOF: By ⟨6⟩1 and ⟨6⟩2.
⟨5⟩4. Q.E.D.
PROOF: By ⟨5⟩1, ⟨5⟩3 and the rule of replacement.
⟨4⟩2. Q.E.D.
PROOF: $\Rightarrow$ introduction
⟨3⟩2. Q.E.D.
PROOF: $\forall$ introduction.
⟨2⟩2. Q.E.D.
PROOF: By ⟨2⟩1 and definition (7).
⟨1⟩2. Q.E.D.
PROOF: $\Rightarrow$ introduction.

B Key terms and definitions

Interface. A contract describing a set of provided operations and the services required to provide the specified operations.

Component. A collection of interfaces some of which may interact between themselves.

Event. The transmission or consumption of a message by an interface.

Asset. An item or a feature of value to an interface for which it requires protection.

Incident. An event of an interface that harms at least one of its assets.

Risk. The combination of the consequence and likelihood of an incident.

Risk analysis. The process to understand the nature of risk and determining the level of risk.

Component-based risk analysis. A process for analysing separate parts of a system or two systems independently with means for combining separate analysis results into an overall risk picture for the whole system.

Risk modelling. Techniques used to aid the process of identifying and estimating likelihood and consequence values.

Risk graph. A structured representation of incidents, their causes and consequences.

Dependent risk graph. A risk graph that is divided into two parts; one part that describes the target of analysis and one part describes the assumptions on which the risk estimates depend.