Component-based risk analysis

Doctoral Dissertation by

Gyrd Brændeland

Submitted to the Faculty of Mathematics and Natural Sciences at the University of Oslo in partial fulfillment of the requirements for the degree Ph.D. in Computer Science

March 2011
Abstract

Component-based system development causes challenges for security and safety as upgraded components may interact with a system in unforeseen ways. To analyse the risks related to deploying a new component or upgrading an existing one requires an understanding not only of the risks of the component itself, but also of how its interactions with the system affects the risk level of the system as a whole and how the system affects the risk level of the component. In conventional risk analysis the parts of the environment that are relevant for estimating the risk-level are often included as part of the target of analysis. Furthermore, in existing risk analysis methods [3, 30, 142], the environment and the time frame are viewed as constants. These features of existing risk analysis methods make them poorly suited to analyse component-based systems whose vulnerabilities and threats varies with the environment in which they exist [150].

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, which are the most relevant for component-based development, 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. The purpose of defensive risk analysis is to gather sufficient knowledge about vulnerabilities, threats, consequences and probability, in order to establish the appropriate protection level for existing assets of to the system, service or business to be analysed.

A modular understanding of risks is a prerequisite for robust component-based development and for maintaining the trustworthiness of modular systems. In order to properly address risks related to component-based systems we propose a component-based approach to risk analysis, which is based on the same principles of modularity and composition as component-based development. The purpose of the approach is to support the integration of risk analysis into component-based development. The approach consists of: (1) a framework for component-based risk analysis; (2) a modular approach to risk modelling; (3) a formal foundation for modular risk modelling; (4) and a formal component model integrating the notion of risk. The framework for component-based risk analysis provides a process for analysing separate parts of a system independently with means for combining separate analysis parts into an overall picture for the whole system. It applies the modular risk modelling approach for the purpose of identifying, analysing and documenting component risks. The component model with a notion of risk provides a formal foundation for integrating risk analysis into component-based development.
Acknowledgements

Ketil Stølen is a thorough and meticulous supervisor who gives his students plenty of feedback. He is highly knowledgeable within the field of theoretical computer science and possesses a scientific curiosity for always exploring new ideas and areas of research. As a PhD student within his field of interest one therefore never feels isolated scientifically. Ketil Stølen had the initial idea that there was some interesting research to be done within the field of component-based security risk analysis. He also had several ideas for directions for the work and for improvements as the work progressed, especially regarding the formal notation for modular risk modelling presented in this thesis. I have learnt a lot during my years as Ketil Stølen’s student, about formal specification and reasoning and about the process of doing scientific research. For this I thank him, and also for continuing to supervise me long after the funding ran out and for never giving up on me. I would also like to thank Ketil Stølen for inspiring me to start running, which often helped to clear up my thoughts. A number of the most difficult proofs where solved while running around in the nearest hills.

I am grateful to Atle Refsdal who co-authored two of the papers in this thesis. The formal notation for modular risk analysis presented in this thesis builds upon the work Refsdal did on probabilistic STAIRS in his PhD. Refsdal also contributed by painstakingly reviewing the most important proofs of the properties of the formal notation and offered good advise about the structuring of formal proofs.

I thank Ida Hogganvik Grøndahl with whom I shared offices during the first years of my PhD. She taught me most of what I know about CORAS when we worked together on the research project SECURIS, and she was responsible for the syntax of CORAS in its present form through the work she did on her PhD. I thank Fredrik Seehusen for sharing his extensive knowledge in system modelling during the work on the SECURIS project and for the good times we had together with Mass Soldal Lund in Canada at FAST 2006. I would also like to thank Mass Soldal Lund for valuable contributions and comments on some of the chapters and for help with typesetting of the thesis in \LaTeX. I thank Bjørnar Solhaug for extensive comments and very helpful suggestions for improvements on the work on the framework for component-based risk analysis. Bjørnar Solhaug also gave useful comments with regard to formulating some of the rules for dependent risk graphs and for motivating the assumption-guarantee style for risk analysis. I thank Heidi Dahl and Iselin Engan who participated in the early work on Dependent CORAS which is a forerunner for dependent risk graphs. Heidi Dahl also defined the structured semantics of the CORAS language of which the formal semantics for dependent risk graphs presented in the thesis is based, and participated in defining the CORAS calculus for computing likelihoods of vertices and relations in threat diagrams. Thanks also to the other former and present colleagues in my group at SINTEF, Folker den Braber, Fredrik Vraalsen, Tom Lysemose, Aida Omerovic, Olav
Ligaarden, Tormod Håvaldsrud and Gencer Erdogan for interesting discussions and for being good colleagues. I would also like to thank Tormod Håvaldsrud, Jan Øyvind Aagedal and Øystein Haugen for advice on modelling in the UML.

Thanks to my husband Bjarte M. Østvold for holding the fort at home during the many Sundays and holidays that went into completing this thesis after the funding ran out. Thanks also to Bjarte for his unflattering good mood and for keeping the spirits up for us both when chances of completing the thesis seemed bleak. Bjarte has also read through and commented on several versions of the papers constituting this thesis. Finally, to speed the proof reading up in the end, Bjarte created lpchk: a proof analyser for structured proofs that checks consistency of step labelling and parentheses matchings. Thanks to my two children, Stål and Ylva, who have had a PhD-working mother for most and all of their lives. You insist on bringing focus back to the basic and important things in life by your persistent presence.

I thank my father in law Ketil Østvold and his wife Grethe Nygård who housed me and my family during the crazy Easter of 2010 where I sat in the basement finishing the paper on modular risk modelling, and only surfaced thrice daily to be fed. Thanks also to friends, neighbours, other members of my family and in particular my dad Asbjørn Brændeland for his continuous encouragement and support. Finally I would like to thank my running team Kvikke Kvinnv for all inspiring runs and all the fun.

The research on which this thesis reports has been financed by the research project COMA 160317 (Component-oriented model-based security analysis). COMA was funded by the Research Council of Norway as an individual PhD grant. During my period as a doctoral student I have been connected to the University of Oslo and then to SINTEF, where I also have worked as a part-time researcher.

---

1The Norwegian word kvikk may be understood as both clever and fast, and the word kvinnv is colloquial for the Norwegian word for women.
List of original publications

Component-based risk analysis:


Modular risk modelling:


Formal foundation for modular risk modelling:


Formal component model with a notion of risk:

# Contents

Abstract iii  
Acknowledgements v  
List of original publications vii  
Contents ix  

## I Overview

### 1 Introduction  
1.1 Risk analysis, its strengths and limitations  
1.1.1 Strengths  
1.1.2 Limitations  
1.2 Objective  
1.3 Contribution  
1.4 Organisation

## 2 Problem characterisation  
2.1 Motivation  
2.1.1 The need for component-based risk analysis  
2.2 Requirements  
2.2.1 Framework for component-based risk analysis  
2.2.2 Modular risk modelling  
2.2.3 Formal foundation for modular risk modelling  
2.2.4 Formal component model with a notion of risk

## 3 Research method  
3.1 Scientific method in computer science  
3.2 An iterative research process  
3.3 Strategies for evaluation  
3.3.1 Considerations for choice of evaluation strategies  
3.4 How we applied the method  
3.4.1 Framework for component-based risk analysis  
3.4.2 Modular risk modelling  
3.4.3 Formal foundation for modular risk modelling  
3.4.4 Formal component model with a notion of risk
4 State of the art 25
4.1 Framework for component-based risk analysis 25
4.1.1 Security requirements engineering 25
4.1.2 Risk analysis in system development 26
4.1.3 Measuring risks in component-based systems 27
4.2 Modular risk modelling 28
4.3 Formal foundation for modular risk modelling 28
4.4 Formal component model with a notion of risk 31
4.4.1 Component development techniques 31
4.4.2 Component models 32
4.4.3 Probabilistic component modelling 34

5 Overview of contributions 37
5.1 The overall picture 37
5.2 Contribution 1: Framework for component-based risk analysis 39
5.2.1 Risk analysis 40
5.2.2 Component-based development 40
5.2.3 Component-based risk analysis 41
5.2.4 Adapted CORAS method 42
5.3 Contribution 2: Modular risk modelling 45
5.3.1 CORAS language 45
5.3.2 Dependent threat diagrams 46
5.4 Contribution 3: Formal foundation for modular risk modelling 49
5.4.1 Semantics of risk graphs 49
5.4.2 Calculus 50
5.5 Contribution 4: Formal component model with a notion of risk 56
5.5.1 Denotational representation of interfaces 57
5.5.2 Denotational representation of components 58
5.5.3 Denotational representation of hiding 59
5.5.4 Component composition 59

6 Overview of research papers 61
6.1 Paper A: Using model-driven risk analysis in component-based development 61
6.2 Paper B: The dependent CORAS language 61
6.3 Paper C: Modular analysis and modelling of risk scenarios with dependencies 62
6.4 Paper D: A denotational model for component-based risk analysis 62

7 Discussion 65
7.1 Fulfilment of the success criteria 65
7.1.1 Framework for component-based risk analysis 65
7.1.2 Modular risk modelling 67
7.1.3 Formal foundation for modular risk modelling 68
7.1.4 Formal component model with a notion of risk 68
7.2 How our approach relates to state of the art 72
7.2.1 Framework for component-based risk analysis 72
7.2.2 Modular risk modelling 72
Part I

Overview
Chapter 1

Introduction

The contribution of this thesis is an approach to component-based risk analysis to facilitate the integration of risk analysis into component-based development. The approach consists of: a framework for component-based risk analysis; a modular approach to risk modelling; a formal foundation for modular risk modelling; and a formal component model integrating the notion of risk.

The topic of the thesis lies in the intersection of risk analysis and component-based development. The idea behind component-based development is that software systems should be built from reusable components in a modular manner, rather than programmed from scratch, as illustrated in Figure 1.1. A benefit of modularity is that a problem can be decomposed into smaller parts that can be handled independently of each other. Components interact through provided and required interfaces, illustrated by the ball and socket notation [120] in Figure 1.1.

Figure 1.1: Component-based systems are built from components

The flexibility offered by component-based software, and the potential for reducing production costs through reuse, has lead to an increased preference for component-based development techniques, such as Sun’s Enterprise JavaBeans (EJB) [118] and Microsoft’s .NET [112]. An important question for users and developers of component technology is whether to trust a new component to be integrated into a system. This is especially true for software handling safety- and security-critical tasks such as flight-control systems, or accounting [90, 32]. Risk analysis is an important tool for developers to determine the risk level of systems in general and to establish the appropriate protection level for systems.

In the following sections we discuss the strengths and limitations of existing risk analysis methods with regard to analysing risks of component-based systems. We also state the objective of our work and give an overview of the main contributions.
1 Introduction

1.1 Risk analysis, its strengths and limitations

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, which are most relevant for component-based development, 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. The purpose of defensive risk analysis is to gather sufficient knowledge about vulnerabilities, threats, consequences and probability, in order to establish the appropriate protection level for existing assets related to the system, service or business to be analysed.

An unwanted incident (accident) in the safety domain is an undesired event that leads to harm on life or property, such as discharge of toxic chemicals or nuclear reactor melt down. A security incident may be confidentiality breach, e.g. due to theft or a human blunder, compromised integrity of information of the system itself or loss of service availability. The combination of the probability and consequence of an unwanted incident constitutes a risk. If the risk level towards assets is found to be too high, it is necessary to identify countermeasures. Examples of countermeasures to safety or security risks are improving work processes related to safety-critical tasks, technical upgrades (e.g. installing seat belts in cars), encrypting confidential information, installing intrusion detection systems to counter flooding attacks, installing firewalls, etc. The identified countermeasures must be considered carefully with regard to their potential for introducing new risks into the system, costs, and trade-offs with regard to for example usability [126]. The risk analysis process should deliver a protection specification, describing the selected countermeasures and documenting the estimated risk level of the treated system.

1.1.1 Strengths

The technologies to achieve risk protection of software systems are fairly well known. The strength of a successful risk analysis lies in its ability to provide a basis for choosing an appropriate level of protection for the application or service in question. If the protection level is too low, the cost from risks will be too high. On the other hand, if the protection level is too high, it may result in employees bypassing security or safety policies or rendering a service too expensive or inconvenient for users. It is therefore important that the level of protection match the value of the assets to be protected. This is particularly important for the security domain where usability may be an asset in itself, which can be compromised by strict protection measures.

An automated scanner finds bugs in the implementation of the deployed system. A risk analysis, however, can find flaws in the design. Furthermore, a risk analysis may target non-technical aspects of a system, such as the humans using it and the organisation or enterprise within which the system exists. These are important features of risk analysis methods, as it has been documented [90, 100] that design flaws and insufficient requirement definitions are responsible for a large part of security and safety problems.
1.1.2 Limitations

In order for a risk analysis method to be effective, the target of the analysis must be clearly defined and assumptions about the environment and the time-frame of the analysis must be made clear. In existing risk analysis methods [3, 30, 142], the environment and the time-frame are viewed as constants. Furthermore, existing risk analysis methods are monolithic in the sense that systems in general are analysed as a whole. To our knowledge, no existing risk analysis methods tackle the problem of deducing the effect of composition with regard to risks. Hence, if the environment of the system changes or it is upgraded with a new component its risks may have to be analysed anew.

These features of existing risk analysis methods make them poorly suited to analyse modern component-based systems. Such systems may be upgraded several times during their lifetime and vulnerabilities and threats vary with the environment in which the systems exist [150]. Developers lack tools for establishing the appropriate protection level of component-based systems in the form of risk analysis methods tailored towards such systems [150, 100]. In order to overcome this problem risk analysis methods for component-based systems must be based on the same principles of modularity and composition as component-based development.

1.2 Objective

The purpose of this thesis is to provide an approach to component-based risk analysis that facilitates the integration of risk analysis into component-based development. By component-based we mean that it should be possible to identify, analyse and document risks at the component level. Furthermore the risk analysis process should be modular in the sense that analysis results can be composed into composite risk analyses. The objective of component-based risk analysis is to:

1. Support development of components that are both trustworthy and user friendly by aiding developers in selecting appropriate protection levels for component assets and developing components in accordance with the selected protection level.

2. Improving component robustness by supporting the integration of risk analysis into all steps of component development and thereby aid developers to identify design flaws.

3. Support reuse of completed component risk analyses to compose risk analyses of complex systems more efficiently than analysing them from scratch.

4. Provide a standardised format for documentation of risk analysis results to support maintenance of such results in combination with component upgrades.

1.3 Contribution

The main contributions of this thesis are: (1) a framework for component-based risk analysis; (2) a modular approach to risk modelling; (3) a formal foundation for modular risk modelling; (4) and a formal component model integrating the notion of risk.
(1) The framework for component-based risk analysis is meant to serve as a guide for component-developers who wish to integrate risk analysis into their development process. It uses the modular risk modelling approach to analyse and document component risks. Furthermore, the deduction rules provided by the formal foundation for modular risk analysis is used to reason about component risks and deduce an overall risk level from the risk analysis results of the component parts. As part of the evaluation of the framework we investigate how a risk analysis conducted according to the framework can be integrated step by step into a component-based development process.

(2) The modular risk modelling approach consists of a graphical risk modelling language with support for documenting risk analysis assumptions. We refer to this extended language as dependent CORAS since we use it to document dependencies on assumptions. 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. The approach is built on top of the CORAS risk modelling language [94]. A risk modelling language is meant to be used by persons involved in a risk analysis process, to aid the identification and analysis of risks.

(3) The formal foundation for modular risk modelling provides a set of deduction rules to reason about threat scenarios with dependencies. The rules of the calculus characterise conditions under which: the analysis of complex scenarios can be decomposed into separate analyses that can be carried out independently; the dependencies between scenarios can be resolved distinguishing bad dependencies (i.e., circular dependencies) from good dependencies (i.e., non-circular dependencies); and the risk analyses of separate system parts can be put together to provide a risk analysis for system as a whole.

(4) The formal component model with a notion of risk formalises the informal understanding of component-based risk analysis which forms the basis for the framework of contribution (1). It serves as a basis for applied methods for component-based risk analysis and for its integration into component-based development. The target group of the formal component model is method developers. The component model allows developers to prove pragmatic principles and rules for component-based risk analysis, and to convey the real meaning of risk analysis documentation with respect to an underlying component implementation.

1.4 Organisation

The Faculty of Mathematics and Natural Sciences at the University of Oslo recommends that a dissertation is presented either as a monograph or as a collection of research papers. We have chosen the latter, but in order to take advantage of some of the benefits offered by a monograph, we accompany the collected research paper with an extensive introductory part. The purpose of the introductory part is to explain the overall context of the artefacts presented in the research papers and to explain how they fit together.

The dissertation is structured into two main parts, where Part I constitutes the introductory part and Part II contains the collection of research papers. Each paper
in Part II is meant to be self-contained. The papers therefore overlap to some extent with regard to explanations and definitions of the basic terminology. The contents of the related work sections are covered by Chapter 4 in Part I on state of the art, and the discussion of related work in Chapter 7. We have structured the introductory part (Part I) into 8 chapters:

**Chapter 1 - Introduction** contextualises our thesis, introduces the most important concepts and explains the contribution and structure of the thesis.

**Chapter 2 - Problem characterisation** motivates our goal and refines it into a set of success criteria that the contributions should fulfil.

**Chapter 3 - Research method** explains the research method applied in the work on this thesis.

**Chapter 4 - State of the art** gives an overview of state of the art of relevance for the contributions presented in this thesis.

**Chapter 5 - Overview of contributions** gives an overview of our main contributions and explains how they relate to each other.

**Chapter 6 - Overview of research papers** gives an overview of the contents and contribution of each research paper.

**Chapter 7 - Discussion** discusses the extent to which our contributions satisfy the success criteria. We also discuss how our work relates to state of the art.

**Chapter 8 - Conclusion** summarises our contributions and discusses future work.
1 Introduction
Chapter 2

Problem characterisation

Our goal is to provide an approach to component-based risk analysis that supports the integration of risk analysis into component-based development. Such an approach should be based on a sound formal foundation and provide practical tools and guides for risk analysts. In this chapter we motivate our goal and refine it into a set of success criteria that we have aimed to fulfil.

2.1 Motivation

Component-based software engineering enables reuse of software parts and reduces production costs. With strict time-to-market requirements for software technology, products such as cars, smart phones and mobile devices in general are increasingly sold with upgradeable software. The flexibility offered by component-based software facilitates rapid development, but causes challenges for risk analysis that are not addressed by current methods.

The Integrated Risk Reduction of Information-based Infrastructure Systems (IRRIIS) project [34] has addressed the need for risk analysis methods targeting mutual dependent systems. IRRIIS have identified lack of appropriate risk analysis models as one of the key challenges in protecting critical infrastructures.

It has been documented [90, 100] that design flaws and insufficient requirement definitions are responsible for a large part of security and safety problems. Yet, in practise, there is little interaction between the requirement engineers on the one hand and safety and security analysts on the other [91, 137, 32]. For component developers the problem is that there is no risk analysis methods tailored towards such systems. In a survey on the use of risk analysis in software design, Verdon and McGraw points out that output from traditional risk-analysis methods is difficult to apply to modern software design [150, 100]. They are also concerned that few traditional risk analysis methods address the contextual variability of risks towards component-based systems, given changes in the environment of the systems.

2.1.1 The need for component-based risk analysis

Figures 2.1 and 2.2 illustrate the difference between a monolithic and a component-based approach to risk analysis of component-based systems ($PS:A+B$ stands for protection specification of system $A+B$).
In a monolithic approach, a component-based system is analysed from scratch. Composition of analysis results is not possible. Hence, if the environment of the system changes or it is upgraded with a new component its risks may have to be analysed anew. When component $B$ in Figure 2.1 is modified to become component $B'$, the risks of the upgraded system $A+B'$ have to be analysed anew. This is how risk analysis of component-based systems is conducted today. Since a risk analysis is time-consuming and requires a lot of resources, this situation is far from ideal.

In order to properly target risks related to component-based systems, risk analysis methods must be based on the same principles of encapsulation and modularity as component-based development. Ideally, a new component bought off the shelf to upgrade a system should include a protection specification. The protection specification should include all the information required to update the protection specification of a system upgraded with the new component.

In Figure 2.2 component $B'$ is accompanied by a protection specification and the risk analysis process is modular, supporting composition of component protection specifications. Hence, we can deduce the protection specification of the upgraded system from the protection specifications of its constituents, instead of analysing it from scratch.

### 2.2 Requirements

Above we outlined the problem area that motivates the work of this thesis and argued that there is a need for a component-based risk analysis method. We mean method in the broad sense; including an overall process, and methods and tools for perform-
ing each step of the process. As already explained in Section 1.2 the objective of component-based risk analysis is to:

1. Support development of components that are both trustworthy and user friendly by aiding developers to select appropriate protection levels for component assets and develop components in accordance with the selected protection level, in an incremental manner.

2. Improve component robustness by supporting the integration of risk analysis into all steps of component development and thereby aid developers to identify design flaws.

3. Support reuse of completed component risk analyses to compose risk analyses of complex systems more efficiently than analysing them from scratch.

4. Provide a standardised format for documentation of risk analysis results to support maintenance of such results in combination with component upgrades.

Developing a complete risk analysis method is an extensive task and lies beyond the scope of this thesis. We have four different contributions that each is necessary in developing a method for component-based risk analysis: (1) a framework for component-based risk analysis; (2) a modular approach to risk modelling; (3) a formal foundation for modular risk modelling; (4) and a formal component model integrating the notion of risk. The relation between (1), (2), (3) and (4), and their particular contribution to the overall objective of a component-based risk analysis method, is summarised in Chapter 5. The main hypothesis for each part is that it manages to fulfil its intended purpose, and that it is feasible to use by its intended users. As explained in Section 1.3, the different contributions targets different user groups:

- The target group of the framework for component-based risk analysis is component developers.

- The target group of the modular risk modelling approach is risk analysts, that is, persons responsible for conducting risks analyses.

- The target group for the formal foundation for modular risk analysis and the formal component model with a notion of risk is developers of risk analysis methods.

Below we have identified a set of success criteria that the artefacts should fulfil in order to satisfy the overall hypotheses.

### 2.2.1 Framework for component-based risk analysis

As already mentioned 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. For such a method to be usable in practise requires a workable process based on a clear conceptualisation of components and component risks. The purpose of the framework for component-based risk analysis is to fulfil this function. In particular the framework aims to fulfil the two first objectives: support development of components that are
both trustworthy and user friendly by aiding developers to select appropriate pro-
tection levels for component assets and develop components in accordance with the
selected protection level; and improving component robustness by supporting the in-
tegration of risk analysis into all steps of component development and thereby aid
developers to identify design flaws. This motivates the following success criteria:

1. The framework for component-based risk analysis should adhere to the same prin-
ciples of encapsulation and modularity as component-based development meth-
ods, without compromising the feasibility of the approach or the common un-
derstanding of risk. It must therefore be based on a clear conceptualisation of
component risks.

2. To provide knowledge about the impact of changes in sub-component upon system
risks, the framework should allow risk related concepts such as assets, incident,
consequence and incident probability, to be documented as an integrated part of
component behaviour.

3. To support robust component-development in practice it should be easy to com-
bine risk analysis results into a risk analysis for the component as a whole.

2.2.2 Modular risk modelling

The purpose of the modular risk modelling approach is to aid risk analysts in identifying
and analysing risks in a modular fashion. The modular risk modelling language should
be easy to understand and employ by the users. That the method is easy to understand
implies that the graphical diagrams are easy to read and understand by non-technical
personnel. That it is easy to employ entails that it should be possible to employ for
risk analysts who do not have a theoretical background, that is, it should be possible
to employ the method without understanding the underlying formalism. This can
for example be ensured by tool-support that allow automated diagram translations
and consistency checking, and hide the underlying computations. This motivates the
following success criteria:

1. The modelling approach should be based on well established and tested techniques
for risk modelling and analysis.

2. To facilitate modular risk analysis the modelling approach should allow the mod-
elling of threat scenarios with dependencies.

2.2.3 Formal foundation for modular risk modelling

The purpose of the formal foundation for modular risk modelling is to provide the
necessary formal basis for reasoning about threat scenarios described in a modular
risk modelling language. The composition and reuse of risk models requires a formal
semantics for risk scenarios and rules for reasoning about them. In order to be of
relevance for a broad audience, the formal foundation should explain the notion of
modular risk analysis at a general level and not depend on any specific modelling
language. The formal foundation should allow analysts to reuse threat models to
compose overall threat models more efficiently than analysing a system from scratch.
This motivates the following success criteria:
2.2 Requirements

1. To facilitate modular risk analysis the formal foundation should provide a formal calculus that characterises conditions under which:

   (a) the dependencies between scenarios can be resolved distinguishing bad dependencies (i.e., circular dependencies) from good dependencies (i.e., non-circular dependencies);

   (b) risk analyses of separate system parts can be put together to provide a risk analysis for system as a whole.

2. The formal foundation should allow the risk analysis of complex systems to be decomposed into separate parts that can be carried out independently.

2.2.4 Formal component model with a notion of risk

The objective to develop a method that allows incremental risk analysis and facilitates reuse, maintenance and composition of risk analysis results requires a sound formal basis. The formal foundation consists of a denotational component model that explains the notion of risk at the component level and a set of deduction rules that facilitates reasoning about component risks.

The purpose of the denotational component model is to serve as a formal basis for applicable methods and tools for component-based risk analysis. To be of practical use for its intended users the component model should be based on a commonsensical understanding of what a component is.

To integrate risk analysis into incremental component development requires that tools for building composite systems include rules for combining risk analysis results, so that risk analyses for component-based systems can be built in the same manner as composite systems. A prerequisite for decomposition and composition of risk analysis results is a better understanding of risk at the level of components. Documenting risks in a standardised format as an integrated part of a component contract will also facilitate reuse and maintenance of risk analysis results. Ideally, a new component bought off the shelf to upgrade a system should include documentation about its level of protection. The documentation should include all the information required to update the documentation on the protection level of a system upgraded with the new component.

To ensure compatibility with existing methods for development and risk analysis the extension of our model with risk notions should be in accordance with international standards for risk analysis terminology.

The purpose of risk analysis is to ensure that the risk level of a component does not violate asset protection requirements. The component model must therefore allow tool builders to convey the real meaning of risk analysis documentation with respect to an underlying component implementation. In particular it should be possible to check whether a given component satisfies the requirements to protection. As composition and decomposition is at the very core of a component-based method for risk analysis, the component model must allow tool builders to prove pragmatic rules for composition of risk analysis results. This motivates the following success criteria:

1. The component model should be defined in accordance with the conventional notion of a component.
2 Problem characterisation

2. The component model should define the precise meaning of risk notions at the component level. In particular it should define the meaning of asset, incident, consequence, probability and risk, with regard to a component. The definitions should be in accordance with existing standards for risk analysis terminology.

3. The component model should include rules for composition of components defined according to a component model with risk notions.

4. The component model should characterise what it means for a component to satisfy a requirements to protection definition.
Chapter 3

Research method

In this chapter we explain the research method applied for the work of this thesis, both in terms of the research process and in terms of the strategies chosen for testing our hypotheses. In Section 3.1 we compare different views upon the role of scientific method in computer science compared to other sciences, in particular with regard to what should count as a valid experiment. In Section 3.2 we present the research process that we apply in the thesis. In Section 3.3 we give an overview of strategies for hypotheses testing and in Section 3.4 we explain how we applied the described method in the work on our thesis and discuss the strategies chosen to evaluate the identified requirements.

3.1 Scientific method in computer science

Computer science is in a unique position compared to the natural sciences by the close connection between pure thought and practical application. While theories in physics concentrate on the what, many advances in the history of computer sciences have been driven by theories explaining how something can be achieved through the interaction between theory and the technology that realises it. The interweaving of pure theory and engineering in computer science has caused some identity problems for practitioners of the discipline. Are we toolsmiths or scientists or both? A closer look at the ongoing debate about the nature of the field reveals that computer scientists do not disagree so much about the applicability of a general scientific method, but rather what counts as experiments and valid tests of hypotheses and predictions in computer science.

In the ongoing debate about the nature of our discipline we can discern at least three distinct views:

*The demonstrator:* Computer science represents a totally new type of science which is neither purely mathematics nor natural science. We should therefore not be disappointed when computer science fails to conform to the expected experimental paradigms of the physical sciences. Dramatic demonstrations rather than critical experiments are the driving forces of progress in computer science [44, 45].

Similar views have been expressed by e.g. Milner [103] and Newell and Simon [107].

*The engineer:* The only science in computer science is the purely theoretical part, i.e. the part of computer science that belongs to mathematics. All the rest is engineering and hence the quest for a scientific method is irrelevant.
Brooks Jr., one of the exponents of the engineer view, goes even further than the demonstrator in distinguishing computer science from the classical experimental scientific paradigm. He urges computer scientists to take pride in being engineers, focus on the users of the tools and evaluate progress by their success, rather than striving to be something they are not [14, 15].

The empiricist: The only sound scientific method is the experimental paradigm of the physical sciences. Computer scientists should create models, formulate hypotheses, and test them using experiments [144].

Basili [7] and Tichy [144, 145] are some of the exponents of what we term the empiricist view.

With regard to what we term the demonstrator and the experimentalist view, we see that they do not disagree upon whether computer science is a science. The real disagreement concerns what should count as a valid experiment. Tichy et al. [145, 144] rule out case studies, demonstrations and so called proof-of-concept methods. Only benchmarking and simulations, under certain conditions, are OK. Benchmarking is a quantitative method for evaluating performance related features. In a quantitative study of 400 research papers within computer science Tichy et al. [145] found that 40 percent of the papers with claims that needed empirical support had none at all. An evaluation of 612 research papers by Zelkowitz and Wallace [153] found similar results.

We agree with Tichy that the engineering view upon computer science is too narrow, but we disagree with his view upon what should count as valid experiments. Empirical strategies, such as benchmarking, have the benefits of being controllable and are therefore easier to replicate than more qualitative methods. However as pointed out by Zelkowitz and Wallace [153], they are weak on generality and realism. Perhaps it is not the lack of experimentation that is the problem, but the criteria for testing hypotheses. In Section 3.3 we discuss the different strengths and weaknesses of different strategies. If we want tests that scores high on precision, generality and realism, Tichy et al.’s requirements to what counts as valid experiments become too narrow. We need to use several different strategies in parallel that have different strengths.

We opt for the middle road put forward by e.g. Dodig-Crnkovic [27] and Solheim and Stølen [138]. They argue for the applicability of a general scientific method to computer science, with a broader view upon what counts as valid strategies for hypothesis testing, than the one held by Tichy et al.

3.2 An iterative research process

The work on the thesis has been driven by an iterative process after a schema proposed by Solheim and Stølen [138]. They claim that technology research has a lot in common with classical research and should be conducted in accordance with the main principles of the hypothetico-deductive method of classical research: (1) recognition of a theoretical problem; (2) proposal of conjectured solution to the problem (hypothesis); (3) testing of the hypothesis (4) conclusion: retain hypothesis if test is positive, otherwise reject hypothesis as falsified and possibly devise new hypotheses/problems [81].

While the classical researcher seeks new knowledge about reality, the technology researcher is interested in how to produce a new and improved artefact [138]. The starting point for both is a need: for the classical researcher it is the need for a new
theory and for the technology researcher it is the need for a new artefact. Figure 3.1 shows the main steps of technology research:

![Diagram of technology research steps]

1. **Problem analysis** – The researcher surveys a potential need for a new and improved artefact by interacting with potential users and other interested parties. During this stage the researcher identifies a set of requirements to the artefact.

2. **Innovation** – The researcher attempts to construct an artefact that satisfies the potential need. The overall hypothesis is that the artefact satisfies the need. In order to evaluate the overall hypothesis, the researcher has to formulate sub-hypotheses about the properties of the artefact.

3. **Evaluation** – Predictions about the artefact are made based on the identified need. The researcher then evaluates the validity of the predictions. If the evaluation results are positive, the researcher may argue that the artefact satisfies the need.

As in classical research the process is iterative: if the results are negative, the researcher must adjust the requirements, or possibly build a new artefact and evaluate that.

### 3.3 Strategies for evaluation

As discussed in Section 3.1 the optimum strategy for hypothesis testing is to use several different strategies in parallel that have different strengths. According to McGrath [99], when you gather a batch of research evidence, you are always trying to maximise three things:

1. The *generality* of the evidence.
2. The *precision* of measurement.
3. The *realism* of the situation.

The best would be to choose a strategy that scores high on generality, precision and realism, but according to McGrath that is not possible. He divides research strategies into four categories. Strategies within the different categories have different strengths and weaknesses with regard to the criteria listed above, as illustrated in Figure 3.2.
The figure shows that the three properties are far from each other on the circle. Thus, strategies in the middle of the dark grey area score high on precision, but low on generality and realism. In order to obtain both generality, realism and precision, it is necessary to choose several strategies that complement each other. We see that a given category may contain strategies with different strengths and weaknesses. For example setting-independent strategies may be either strong on generality or strong on precision depending on their depth/width.

There have been proposals for classifying testing strategies within the field of computer science [153, 7, 79, 35] similar to those proposed by McGrath. We summarise the most common strategies below, and discuss their strengths and weaknesses with regard to the criteria listed by McGrath.

1. Strategies for evaluation in artificial settings.

   *Laboratory experiment.* A laboratory experiment [99, 35] (controlled experiment [7], synthetic environment experiment [153]) gives the researcher a large amount of control as he can isolate the variables to be examined. It scores high on precision but lacks realism and generality.

   *Benchmarking.* Benchmarking [79, 153] is a quantitative method for evaluating performance related features. Benchmarking has the benefit of being controllable and therefore easy to replicate, but is weak on generality and realism.

2. Strategies for evaluation in natural settings.

   *Field study.* A field study [99, 153] refers to direct observations of natural systems with minimum intervention from the researcher. It is strong on realism but lacks precision and generality as it is difficult to replicate.

   *Field experiment.* A field experiment [99, 35] (replicated experiment [153]) is an experiment conducted in a natural setting where the researcher intervenes in an attempt to manipulate certain features. It is more realistic than laboratory experiments and simulations, but lacks their precision and also lacks generality.

---

1McGrath [99] discuss methods for studies of groups, but we believe his observations are also relevant for computer science.
3.3 Strategies for evaluation

**Case study.** A case study [79, 7, 35, 153] involves an in-depth examination of a single instance or event: a case. Kitchenham [79] refers to a case study as an evaluation of a method or tool that has been used in practise on a real project. A case study can be seen as a variant of a field experiment and has the same strengths and weaknesses.

The term case study is sometimes used to mean evaluation of a tool or method in a fictitious case, but such evaluations does not count as case studies from a methodological point of view. We refer to such fictitious case studies as case-based evaluations. As a case-based evaluation is not conducted in a natural setting, we discuss this further under point 3.

**Action research.** Action research [35] is a mixture of a field experiment and a case study where the researcher takes part in the case under study. It is a method often used to improve processes or systems within organisations. The involvement of the researcher may give an even deeper understanding of the case or situation, but with the sacrifice of generality and the risk of observations being subjective [92].

3. Strategies for evaluation which are independent of the setting.

**Survey.** A survey [35, 79] (sample survey [99]) refers to the gathering of information from a broad sample of respondents through the use of questionnaires or interviews. A survey is less controlled than an experiment and therefore lack precision. Furthermore there is a likelihood of bias on the part of the respondents, which weakens the realism of a survey. The upside of surveys is that they allow the investigation of a greater number of variables than laboratory experiments. Furthermore, since they are independent of the setting they have a high degree of generality.

**Literature search.** A literature search [153] (review [35]) examines previously published studies. They are often used to identify the current state of, and important results within, the domain of the research project [93]. A literature review has the benefit of a large available database, but may be bias in the selection of published works.

**Qualitative screening.** Qualitative screening [79] (subjective/argumentative [35], assertion [153]) is a feature-based evaluation performed by a single individual who determines the features to be assessed. This approach can also be called proof-of-concept, which demonstrates that a particular configuration of ideas or an approach achieves its objectives. It may be difficult to evaluate the greater worth of artefacts serving a proof-of-concept role, because the advancement may be qualitative, as in increased functionality. In this context, what is considered better depends on subjective human judgements.

A case-based evaluation may be seen as a variant of qualitative screening, which is organised as a case study, but where the case is fictitious or the evaluation is conducted a posteriori, based on data from a real project. There may be practical reasons for using a fictitious case instead of a real one to evaluate a method or a set of hypotheses. There are a number of considerations a researcher needs to make in order to select an appropriate evaluation strategy, such as time and cost limitations (see Section 3.3.1). In practise it may be difficult to get industrial partners in a research project committed to participate in case studies that require
a lot of time and resources from the involved parties. An alternative to studying a “real case”, may therefore be to construct a fictitious one or to use experiences from a real project to evaluate a method a posteriori.

Subjective evaluation methods, such as qualitative screening and case-based evaluations, can be useful in the early phase of a research process, in the creation of ideas and insights. The potential for biased interpretations weakens their realism, however. They are located approximately at the border between generality and precision, with less control than experiments and less generality than surveys.

4. Strategies for evaluation which are independent of empirical evidence.

Formal analysis. Formal analysis (formal theory [99], formal theorem [35]) refers to a non-empirical argument based on logical reasoning. It scores high on generality, but low on realism and precision, as it is not empirical.

Simulation. A simulation operates on a model of a given system. They are low on precision for the same reason as formal analysis. Since they are system-specific, they score higher on realism than formal analysis, but lower on generality.

3.3.1 Considerations for choice of evaluation strategies

Not all strategies for evaluation may be applicable to a given research project. In order to select appropriate evaluation strategies, there are several points to consider [79, 80, 138], such as:

- The nature of the predictions, that is, do they concern quantitative features, qualitative features or formal properties of an artefact.

Predictions concerning quantitative features of an artefact, e.g. solving some task faster than any other artefact, can be tested using quantitative methods. Predictions concerning qualitative features (proof-of-concept), should be tested using qualitative methods, such as for example a case study or qualitative screening. Predictions about formal properties must necessarily be validated by a formal argument.

- The maturity of the artefact.

If the artefact is still being developed, there might not be sufficient information to warrant a survey. In the early phases of a research project, the most feasible approach might be a case study or the subjective/argumentative approach.

- Is the strategy realisable?

Time and cost are two important limitations when choosing a strategy for testing. There is also the question of whether the people that should participate in the evaluation are available. Field experiments require thorough planning and usually involve several people, and are therefore costly. At the other hand, we have computer simulations, which do not involve humans. They are therefore quick and easy to perform, if it is possible and relevant.
3.4 How we applied the method

As already mentioned the work on this thesis has followed an iterative process, illustrated in Figure 3.1. Based on an initial problem analysis, we identified the need for a component-based method for risk analysis. Our overall goal is to contribute to the development of such a method. We also identified a set of tasks necessary to fulfil the overall goal, such as defining a formal semantics, and a graphical syntax.

In order to investigate the main hypothesis we refined the goal into a set of success criteria for the identified tasks. The final version of the problem analysis, including the success criteria, is documented in Chapter 2. The success criteria include requirements to practical features of our contribution and requirements to formal properties of the component model.

The main innovation effort is described in Chapter 5. We have developed a framework for component-based risk analysis, a modular risk modelling approach, a formal foundation for modular risk modelling and a formal component model integrating a notion of risk, that each contribute to a component-based risk analysis method. The above research method was applied in the development of each contribution. Each step of the process has been conducted several times based on output from successive evaluations.

In Sections 3.4.1 to 3.4.4 we describe in more detail how each part was evaluated. The results of the evaluations are discussed in Chapter 7. With regard to the desired properties of evaluation strategies discussed in Section 3.3, we lack strategies that perform well with regard to realism. As discussed above, factors such as the nature of our predictions, the maturity of the artefacts and the realisability of evaluation strategies limited our choice of evaluation strategies. In Section 8.2 we discuss the possibility of developing a prototype tool implementing core parts of the method. Such a tool should be thoroughly tested and evaluated empirically, for example through surveys targeting relevant users, such as system developers and risk analysts.

3.4.1 Framework for component-based risk analysis

In order to decide which risk concepts to include at the component level, without compromising the requirements to encapsulation and modularity, we identified a set of risk concepts necessary to deduce valid analysis results and related these to the concept of a component in a conceptual model. To check the feasibility of the conceptual model as a basis for component-based risk analysis, we developed a framework for a component-based risk analysis method and applied it to a fictitious case involving an instant messaging component for smart phones, presented in Chapter 9.

Our overall requirement to the framework is that it should adhere to the same principles of encapsulation and modularity as component-based development methods, without compromising the feasibility of the approach or the common understanding of risk. We used the instant messaging case to evaluate the extent to which the framework fulfils the overall requirement of encapsulation and modularity. We also used 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. As part of the evaluation of the framework we investigated how a risk analysis conducted according to the framework can be integrated step by step into
3 Research method

3.4.2 Modular risk modelling

The goal of the risk modelling approach is to be able to decompose analyses into smaller parts and compose already completed analyses into an overall risk picture of a system as a whole. In order to achieve this we extended the graphical CORAS risk modelling language with an assumption-guarantee approach for the specification of dependencies. The usefulness of the assumption-guarantee approach was checked in an example involving the risk analysis of the power systems in the southern parts of Sweden and Norway, described in Chapter 11. Due to the strong mutual dependency between these systems, their risks are also mutually dependent.

We also applied dependent CORAS for the purpose of identifying, analysing and documenting risk of the instant messaging component, described in Chapter 9. In this example we followed the convention established in the component-based risk analysis framework that the components themselves are the target of analysis. Hence, this case also serves the purpose of evaluating the feasibility of dependent CORAS for modelling risks in a component-based setting. We also applied dependent CORAS to another example case from the domain of IT security, described in Chapter 10.

3.4.3 Formal foundation for modular risk modelling

The purpose of the formal foundation for modular risk modelling is to provide the necessary formal basis for reasoning about threat scenarios described in a modular risk modelling language. We require that the formal foundation for modular risk modelling explains the notion of modularity at a general level in order to be applicable for other modelling languages beyond dependent CORAS. In order to fulfil this requirement we introduced the general notion of a dependent risk graph that we can use to capture assumptions in a risk analysis. We argue that this is a common abstraction of graph and tree-based risk modelling techniques and discuss the possibility of instantiating various risk modelling techniques in our formal foundation in Chapter 11. We have defined a semantics for risk graphs and a calculus which is independent of the CORAS language. The rules of the calculus are proved to be the sound. We used the case involving the power systems in the southern parts of Sweden and Norway, described in Chapter 11, to check the applicability of the formal foundation to support modular risk analysis in a real case.

3.4.4 Formal component model with a notion of risk

The goal of the component model is to formalise risk notions in a component model, in order to be able to reason about components that include such notions. That is, we aim to formalise a part of reality, namely, the part of reality involving risk analysis of components, in order to reason about it.

To represent risk notions as part of component behaviour, the component model formalises the risk concepts that were identified in the previously described conceptual model. It is not possible to check the formal correctness of such mathematical definitions. Instead, we must check whether they match the intuition, seem reasonable and
are feasible to apply in a realistic setting, that is, a component-based development process in our case. The formalisation process functioned as a first test of the usefulness of conceptual model, which was revised accordingly. We evaluated the usefulness of the mathematical definitions in the same case we used for evaluating the component-based risk analysis framework, that is, in the development and analysis of an instant messaging component for smart phones. The evaluation addressed the feasibility of representing the selected risk notions at the component level.

One of the success criteria for the component model is that components that include risk notions are composable. In order to prove this property we first described informally how components interact in an operational model, described in Chapter 12. In the operational model a component is a set of interfaces. By construction, interfaces interact through a number of independent probabilistic choices. The behaviour of a component is completely determined by the behaviour of its constituents. We then gave a formal proof that the semantic component model is defined in accordance with the operational model. Hence, we can compose two or more interfaces and, since they by construction behave probabilistically independent, we can compute the probability of the behaviour of a component from the probabilities of the behaviours of its constituents.

All proofs are written in Lamport’s style for writing proofs [85]. This is a style for structuring formal proofs in a hierarchical manner in LaTeX [83], similar to that of natural deduction [146, 106]. As observed by Lamport the systematic structuring of proofs is essential to getting the results right in complex domains which it is difficult to have good intuitions about. We had several iterations of formulating operators for component composition, attempting to prove them correct and then reformulating them when the structured proof style uncovered inconsistencies. These iterations where repeated until the definitions where proven to be correct.
3 Research method
Chapter 4

State of the art

In this chapter, we give an overview of state of the art of relevance for the contributions presented in this thesis. There are four topics corresponding to our four contributions.

4.1 Framework for component-based risk analysis

By component-based risk analysis we mean risk analysis conducted in adherence to the basic principles of component-based development such as encapsulation of control and modularity of specifications. The motivation behind component-based risk analysis is to support the integration of risk analysis into component-based development.

The need to integrate security and safety issues into system development is widely recognised in the security and safety communities, and several approaches to this end have been proposed [100, 39, 20, 134, 135, 98, 97]. The proposals encompass the integration of security requirements into the requirements phase and the carrying out of risk analysis at design time.

4.1.1 Security requirements engineering

A security requirement is a requirement to the protection of information security in terms of confidentiality, integrity, availability, nonrepudiation, accountability and authenticity of information [64]. Security requirements engineering refers to the process of identifying, analysing, specifying and verifying system security requirements.

Khan and Han [77] have proposed a method for characterising security properties (i.e., confidentiality, integrity, availability, nonrepudiation, accountability and authenticity of information) of composite systems, based on a security characterisation framework for basic components [78, 75, 76]. They define a compositional security contract CsC for two components, which is based on the compatibility between their required and ensured security properties. They also give a guideline for constructing system level contracts, based on several CsCs.

There are several proposals to model-based approaches for the purpose of security requirements engineering. McDermott and Fox [98, 97] first proposed to use specialised use-cases for the purpose of threat identification. A similar approach by Sindre and Opdahl [134, 135] explains how to extend use cases with misuse cases as a means to elicit security requirements. SecureUML [91] 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 [72] is an extension to UML that enables the modelling of security-related features such as confidentiality and access control.

4.1.2 Risk analysis in system development

A risk analysis may be used at design time to check whether a detailed system specification fulfils the security requirements. Risk analysis is the systematic process to understand the nature – and to deduce the level – of risk. A risk is the chance of something happening that will have an impact on objectives (assets) [140]. The impact of a risk can be both positive and negative [63].

Risk analysis methods can be classified according to various criteria, such as domain (i.e., safety versus security), perspective (i.e., asset-based versus threat- and vulnerability-based) and whether they are commercial or standards-based. Risk analysis applied to the domain of information and communication technology (ICT) security is referred to as security risk analysis or risk analysis methods for software [100]. ICT security includes all aspects related to defining, achieving and maintaining confidentiality, integrity, availability, non-repudiation, accountability, authenticity and reliability of ICT [64]. In this section we give an overview of some existing asset-based, security risk analysis methods, both commercial and standards-based. The overview is partly based on two surveys of state-of-art within security risk analysis [25, 54].

OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation) is a framework for identifying and managing information risks [3]. OCTAVE is conducted in three phases: (1) identify information assets and their values, threats to those assets, and security requirements; (2) identify the vulnerabilities that expose the assets to threats, and (3) identify risks and estimate risk consequence and likelihood, prioritise risk and develop a protection strategy. OCTAVE comes with predefined templates for documenting information.

CRAMM (CCTA Risk Analysis and Management Method)\(^1\) in its original form was adopted as a standard by the U.K. government organisation CCTA (Central Computer and Telecommunications Agency), now renamed the Office of Government Commerce. It has been upgraded with computerised tool-support by Siemens/Insight who manages the current version of CRAMM. As OCTAVE, the CRAMM method is structured into three phases: (1) asset identification and valuation; (2) threat and vulnerability assessment; and (3) countermeasure selection and recommendation. Each phase is supported by the CRAMM tool [30]. CRAMM has no support for graphical risk modelling.

Microsoft has developed its own risk analysis method termed Threat modeling [58, 142, 59, 50]. Threat modeling is an integral part of Microsoft’s process for secure development, Security Development Lifecycle [59]. The method provides Data flow diagrams to represent the target of analysis graphically. Based on a graphical representation, the target is decomposed into relevant components that are analysed for susceptibility to threats. The threats are then ranked according to their risk level and mitigated. This process is repeated until the remaining threats have an acceptable risk level. The method provides so called threat threes, similar to attack trees [125], for modelling and analysing how threats may be accomplished through attack paths. The method is supported by a tool to document risk relevant information.

\(^1\)www.cramm.com
4.1 Framework for component-based risk analysis

Threat modelling have been criticised for relying too heavily on checklists [100], as it is based on the STRIDE model of computer security threats, by Howard and LeBlanc [58]. STRIDE is an acronym for Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and Elevation of privilege.

The CORAS model-based method for risk analysis [94] consists of a risk modelling language; the CORAS method which is 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.

The CORAS risk analysis method is structured into eight steps: (1) preparations for the analysis, (2) customer presentation of the target (3) refining the target using asset diagrams (4) approval of the target description (5) risk identification using threat diagrams (6) risk estimation using threat diagrams (7) risk evaluation using risk diagrams and (8) risk treatment using treatment diagrams.

The CORAS risk modelling language has been designed to support communication, documentation and analysis of security threat and risk scenarios. 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.

Jüri Jüriens and Houmb have proposed an approach to risk-driven development of security-critical systems using UMLsec [73]. 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 [72].

Other examples of risk analysis methods are Sun’s ACM/SAR (Adaptive Countermeasure Selection Mechanism/Security Adequacy Review) [40], Cigital’s architectural risk analysis process [100] and the standards-based COBIT (Control objectives for Information and Related Technology) from Information System Audit and Control Association (ISACA)².

4.1.3 Measuring risks in component-based systems

Few current risk analysis methods address component-based systems. As pointed out by Verdon and McGraw it is difficult to apply output from traditional risk-analysis methods to modern software design [150, 100]. The problem is that few traditional risk analysis methods address the contextual variability of risks towards component-based systems, given changes in the environment of the systems.

There are however some approaches to hazard analysis that address the propagation of failures in component-based systems. Several of these approaches describe system failure propagation by matching ingoing and outgoing failures of individual components. Giese et al. [38, 37] define a method for compositional hazard analysis of restricted UML component diagrams and deployment diagrams. They employ fault tree analysis to describe hazards and the combination of component failures that can cause them. For each component they describe a set of incoming failures, outgoing failures, local failures (events) and the dependencies between incoming and outgoing failures. Failure information of components can be composed by combining their failure dependencies. Papadopoulos et al. [111] apply a version of Failure Modes and Effects Analysis (FMEA) [10] that focuses on component interfaces, to describe the causes of output failures as logical combinations of internal component malfunctions or

²http://www.isaca.org/cobit/
deviations of the component inputs. They describe propagation of faults in a system by synthesising fault trees from the individual component results. Kaiser et al. [74] propose a method for compositional fault tree analysis. Component failures are described by specialised component fault trees that can be combined into system fault trees via input and output ports. Fenton and Neil [31] address the problem of predicting risks related to introducing a new component into a system, by applying Bayesian networks to analyse failure probabilities of components. They combine quantitative and qualitative evidence concerning the reliability of a component and use Bayesian networks to calculate the overall failure probability.

The model-driven performance risk analysis method by Cortelessa et al. [20] 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, developed by Goseva [39] et al.

4.2 Modular risk modelling

To our knowledge no existing risk modelling technique offers the possibility to document assumptions about external dependencies as part of the risk modelling. This is, however, a well known practise within system development [16]. The idea is that components are designed to work properly only in environments that satisfy certain assumptions. Assumption-guarantee specifications [68, 104] are a common approach to capture the assumptions about the environment on which the well-behavedness of a component relies. In the assumption-guarantee approach specifications consist of two parts, an assumption and a guarantee:

- The assumption specifies the assumed environment for the specified system part.
- The guarantee specifies how the system part is guaranteed to behave when executed in an environment that satisfies the assumption.

Assumption-guarantee specifications, sometimes also referred to as contractual specifications, are useful for specifying open systems, i.e., systems that interact with and depend on an environment. Several variations of such specifications have been suggested for different contexts. For example Meyer [102] introduces contracts in software design, with the design by contract principle. This paradigm is inspired by Hoare who first introduced a kind of contract specification in formal methods with his pre/postcondition style [52]. The contractual approach to specifications was later extended by others to also handle concurrency [104, 69].

Assumption-guarantee specifications facilitate modular system development through mechanisms for decomposition and composition of specifications. In particular, the separation of concerns in assumption-guarantee specifications makes it possible to resolve non-circular dependencies between system parts.

4.3 Formal foundation for modular risk modelling

There exist a number of modelling techniques that are used both to aid the structuring of threats and unwanted incidents (qualitative analysis) and to compute probabilities
of unwanted incidents (quantitative analysis). We refer to such techniques as threat modelling and analysis techniques. Robinson et al. [117] distinguish between three types of modelling techniques: trees, blocks, and integrated presentation diagrams. An integrated presentation diagram puts more emphasis on understandable graphic appearance than the two other and is generally less strict with regard to how the elements are organised. As opposed to trees and block modelling techniques, not all integrated presentation techniques have a formal semantics. We discuss some types of trees and integrated presentation diagrams below, as these are the two categories most commonly used within risk analysis.

Fault tree analysis [60] is a technique for the study of how low level failures may propagate to unwanted incidents. The technique was formalised by Bell Telephone Laboratories in 1962 for the study of a missile launch control system [62] and has later become the most widely used within technique within reliability and safety analysis [113]. A fault tree structures events in a logical binary tree, leading from the various failure points to an unwanted incident. A fault tree can be understood as a logical expression where each basic event is an atomic formula. Every fault tree can therefore be transformed to a tree on disjunctive normal form, that is, a disjunction of conjuncts, where each conjunct is a minimal cut set [110, 51]. A minimal cut set is a minimal set of basic events that is sufficient for the top event to occur. For example, the fault tree of Figure 4.1 has two minimal cut sets: \{generator fails, switch fails\} and \{generator fails, battery fails\}. By structuring the basic events into minimal cut sets

![Fault tree example](image)

Figure 4.1: Fault tree example

a fault tree can be used to identify which components contribute the most to a top event.

Fault trees are also used to determine the probability of an unwanted incident. If all the basic events in a fault tree are statistically independent, the probability of the undesired top event can be computed from the minimal cut sets according to the laws of probability [67]. Fault tree analysis requires that probabilities of basic events are given as exact values. This is a problem with regard to the applicability of conventional fault tree analysis to systems with insufficient historical data available. In systems like nuclear power plants there can be cases of extremely hazardous events that happen so seldom that it is difficult to estimate their probability [65, 105, 141]. Imprecision of data at the basic level propagates through the fault tree causing uncertain probability values for the top events. To address this problem, several approaches have been suggested to handle imprecise input data in fault trees based on the use of fuzzy set
4 State of the art

A fuzzy set is a set that allows for grades of membership. The grade of membership in a set $A$ is given by a function $f_A$, that given an element $e$, yields a number between zero and 1. Thus, the nearer the value of $f_A(e)$ is to 1, the higher the grade of membership of $e$ in $A$ [152]. A set whose membership function yields only zero or 1 is called a crisp set.

Misra and Weber [105] propose to represent probabilities of basic events using so-called fuzzy real numbers. Probabilities of top events are calculated using arithmetic operators for fuzzy numbers [28]. A fuzzy real number is a fuzzy subset of the real numbers which (1) is normal, that is, its membership function yields 1 for at least one element; (2) is convex and (3) the core consists of one value only. The core of a fuzzy set $A$ is the crisp sub-set of $A$ whose degree of membership in $A$ equals 1 [24]. A fuzzy set fulfilling these restrictions can be used to represent approximate numerical data. For example the set of numbers that are “nearly” 0.5 can be represented by a set whose membership function yields zero for numbers higher than 0.5, 1 for 0.5 and decreases for numbers approaching zero [148].

As mentioned a problem with traditional fault tree analysis, is that uncertain input propagates to the top event. To deal with this problem Suresh et al [141] have proposed a method using a so-called fuzzy uncertainty importance measure. This measure is used to identify the sources of uncertainty having greatest impact in the uncertainty of the top event.

Carreras and Walker have proposed to use intervals, instead of fuzzy numbers, to represent uncertainty of input data in fault trees. They use interval arithmetic to propagate the uncertain data through fault trees and generate output distributions reflecting the uncertainty in the input data [17].

An Event tree [61] starts with component failures and follows possible further system events through a series of final consequences. Event trees are developed through success/failure gates for each defence mechanism that is activated.

An attack tree [125] is basically a fault tree with a security-oriented terminology. Attack trees aim to provide a formal and methodical way of describing the security of a system based on the attacks it may be exposed to. The notation uses a tree structure similar to fault trees, with the attack goal as the top vertex and different ways of achieving the goal as leaf vertices.

A cause-consequence diagram [117, 96] (also called cause and effect diagram [114]) combines the features of both fault tree and event tree. When constructing a cause-consequence diagram one starts with an unwanted incident and develops the diagram backwards to find its causes (fault tree) and forwards to find its consequences (event tree) [54]. A cause-consequence diagram is, however, less structured than a tree and does not have the same binary restrictions. Cause-consequence diagrams are qualitative and cannot be used as a basis for quantitative analysis [114].

A Bayesian network (also called Bayesian belief network) [18] can be used as an alternative to fault trees and cause-consequence diagrams to illustrate the relationships between a system failure or an accident and its causes and contributing factors. A Bayesian network is more general than a fault tree since the causes do not have to be binary events and causes do not have to be connected through a specified logical gate. In this respect they are similar to cause-consequence diagrams. As opposed to cause-consequence diagrams, however, Bayesian networks can be used as a basis for quantitative analysis [114]. A Bayesian network is used to specify a joint probability
distribution for a set of variables [18]. It is a directed acyclic graph consisting of vertices that represent random variables and directed edges that specify dependence assumptions that must hold between the variables.

A CORAS threat diagram is used during the risk identification and estimation phases of the CORAS risk analysis process. It describes how different threats exploit vulnerabilities to initiate threat scenarios and unwanted incidents, and which assets the unwanted incidents affect. CORAS threat diagrams are meant to be used during brainstorming sessions where discussions are documented along the way. A CORAS diagram offers the same flexibility as cause-consequence diagrams and Bayesian networks with regard to structuring of diagram elements. It is organised as a directed acyclic graph consisting of vertices (threats, threat scenarios, unwanted incidents and affected assets) and directed edges between the vertices. CORAS diagrams were originally not designed for quantitative analysis. Likelihood values are meant to be assigned directly by workshop participants. Likelihoods may be assigned to both vertices and relations. However, the CORAS method provide rules for computing likelihood values of vertices given their parent vertices and relations leading to them, that are meant for checking the consistencies of assigned likelihood values. Those rules can also be used to extract the likelihood of an unwanted incident from likelihoods of threat scenarios leading to it.

4.4 Formal component model with a notion of risk

State of the art of relevance for understanding our formal component model includes component models, and component development techniques, in particular probabilistic component development techniques.

4.4.1 Component development techniques

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.

Intuitively a component is a standardised “thing” that can be mass-fabricated and reused in various constellations. According to the classic definition by Szyperski a software component ...

... is a unit of composition with contractually specified interfaces and explicit dependencies only. A software component can be deployed independently and is subject to composition by third parties [143].

That a component is a unit of independent deployment means that it needs to be well separated from its environment and other components. This distinction is provided by a clear specification of the component interfaces and by encapsulating the component implementation [21].

A development technique consists of a syntax for specifying component behaviour and system architectures, and rules (if formal) or guidelines (if informal/semi-formal) for incremental development of systems from specifications. We distinguish between
State of the art

Semiformal and formal development techniques, in accordance with existing literature [16, 25]. Development techniques are mostly used in the early stages of component development and usually do not provide mechanisms for obtaining the actual component implementations. A large number of development techniques, both semiformal and formal, are not necessarily directed towards component-based development. However, they can be used for component-based development in combination with a component model, see Section 4.4.2.

Semiformal techniques

Semiformal techniques are based on semiformal description and modelling languages, that seemingly are formal but lack a precisely defined syntax, or contain constructs with an unclear semantics [16]. UML [109] is a semiformal description and modelling language, which has become the de facto industry standard for system specification, including component-based system specification. There is a large number of semiformal development techniques built up around UML, such as for example RUP (Rational Unified Process) [84]. UML provide component diagrams to describe the structure of component-based systems and interaction diagrams for describing component interactions. Cheesman and Daniels have proposed a UML-based method specially tailored towards the specification and development of component-based systems [19].

Formal techniques

Formal techniques (also referred to as formal methods) differ from semiformal methods in that they are based on description and modelling languages with a formally defined syntax. Formal methods also provide a mathematical notion of refinement and a calculus for the verification of refinement steps. Examples of formal methods for system development are VDM (Vienna Development Method) [70], Z [49], B [2], TLA (Temporal Logic of Actions) [86], FOCUS [16], and STAIRS [47, 121, 115].

VDM, B and Z are specialised towards sequential code and abstract data types [16], whereas TLA, FOCUS and STAIRS emphasise on concurrency and interaction, which is most relevant for component-based development.

TLA is a logic for reasoning about concurrent systems. Components are specified using logical formulas. The component semantics is state-based. In FOCUS component behaviour is described as input/output relations. System components are connected by directed communication channels, i.e., channels that transmit messages in one direction. FOCUS uses streams to model the communication histories of channels. STAIRS [47, 121, 115] is a formal approach to compositional development of UML 2.x interactions, such as sequence diagrams or interaction overview diagrams. In addition to defining a denotational trace semantics for the main aspects of interactions, STAIRS focuses on how interactions may be developed incrementally based on a formal approach to refinement.

4.4.2 Component models

Lau and Wang [88] have surveyed current component models and classified them into a taxonomy based on commonly accepted criteria for successful component-based development [88]. According to the criteria components should be preexisting reusable software units which developers can reuse to compose software for different applications.
4.4 Formal component model with a notion of risk

Furthermore components should be composable into composite components which, in turn, can be composed with (composite) components into even larger composites, and so on. These criteria necessitate the use of a repository in the design phase. It must be possible to deposit and retrieve composites from a repository, just like any components.

Lau and Wang [88] divide current component models into four categories based on their facilities for composition during the various phases of a component life cycle. According to their evaluation, no current component models provide mechanisms for composition in all phases. They propose a component model with explicitly defined component connectors, to ensure encapsulation of control, thereby facilitating compositionality during all phases of development. Below is a brief overview of current software component models, based on the review by Lau and Wang [88].

**Acme** is a generic architecture description language [36]. In Acme and other Acme-like architecture description languages, a component is an architectural unit that represents a primary computational element and data store of a system. The interface of a component is defined by a set of ports, through which the component’s functionalities are exposed. Each port identifies a point of interaction between the component and its environment. A port can have different roles, such as sink (receive) and source (send). Components are composed of connectors (lines) that mediate the communication and coordination activities among components. In Acme and other architecture description languages, composition is possible in the design phase, but components must be constructed from scratch, since there is no repository. There is no deployment phase. The composition of the component instances (in the runtime phase) is the same as that of the components in the design phase [88].

In **UML 2.0** [109, 19], a component is a modular unit of a system with well-defined interfaces that is replaceable within its environment. A component defines its behaviour by one or more requires and provides interfaces which implement its required and provided services. Every required service is represented by a socket and every provided service by a lollipop. UML 2.0 components are combined using UML connectors in the design phase. Like Acme, UML has no repository. Hence, components must be constructed from scratch and composition is only possible in the design phase.

In **Enterprise JavaBeans (EJB)** [118]³, a component is an enterprise bean, which is a Java class that is hosted and managed by an EJB container. There are three types of enterprise beans: entity beans that model business data; session beans that model business processes; and message-driven beans that represent business processes. A message-driven bean can only be triggered by receiving messages from other beans. Beans can be deposited into the container and composed in the design phase, but there is no support for storage or retrieval of composite components as identifiable units. The reason is that the container is also the execution environment for beans [88]. EJB has no support for so called connection-oriented programming [143], where pre-fabricated components can be connected at runtime by third parties.

Microsoft's **.NET** [112], and OMG's **CCM (Corba Component Model)**[108] are other examples of component models where components can be deployed into and composed in repositories, but not retrieved from them [88].

In **JavaBeans**⁴, a component is a bean, which is just any Java class that has methods, events, and properties. As opposed to EJB, JavaBeans supports connection-

---

oriented programming [143]. Beans can define both event sources and event listeners that can be connected by third parties during deployment. Individual JavaBeans are constructed in a Java programming environment and deposited in the ToolBox of the BDK (Beans Development Kit), which is the repository for JavaBeans. Although the ToolBox acts like a repository, it does not support composition of beans. Thus, the only composition possible in JavaBeans is the composition of beans in the deployment phase [88].

In KobrA [5], a component is a UML stereotype. KobrA components can be constructed in a visual builder tool and deposited into a file system. The file system is the repository of KobrA components. In the design phase, KobrA components are composed by direct method calls. In the deployment phase, component implementations can be refined from their specifications. No new composition of component instances is possible.

4.4.3 Probabilistic component modelling

Several approaches have been proposed to extend component development techniques with the ability to express probabilistic behaviours. Some examples include probabilistic process algebra [132], probabilistic automata [131, 130], probabilistic action systems [133, 147, 42], probabilistic state transition systems [23], and probabilistic sequence diagrams [115].

Probabilistic system modelling is used for different purposes. One application domain is the modelling of probabilistic algorithms and games. Another is the specification of soft requirements when undesired behaviour of a system for practical reasons cannot be ruled out entirely [133, 115]. The following is an example of a soft requirement: ‘After a request has been sent, the probability of receiving the reply within 10 seconds should be at least 0.9’ [115].

Depending on their application domain, the different approaches vary with regard to how they deal with nondeterminism, and in particular how probability relates to nondeterminism.

Different types of nondeterminism

Different development techniques provide different types of nondeterminism reflecting different usage areas. During the early stages of an incremental development process nondeterminism is often used for the purpose of abstract specifications, where an implementer may choose between different alternatives that are equivalent with regard to fulfilling the overall requirements. This type of nondeterminism represents underspecification, which may be removed in the subsequent refinement process.

Nondeterminism is also used to model the arbitrary behaviour of an external environment. This notion of nondeterminism is useful for the specification of open reactive systems that interact with an environment they cannot control. For example CSP [53, 119, 6] distinguishes between internal and external nondeterministic choice. With internal nondeterminism, the system is free to choose whether it should offer all alternatives or only one (or some) of them. The choice may be performed at run-time, making the system nondeterministic, but the choice may also be made by the implementer, resulting in a deterministic system. For external nondeterminism (also called environmental choice), the behaviour is determined by the environment.
Action systems use a similar distinction between so-called angelic and demonic choice [6]. With angelic choice, the choice between the alternatives is made by the system with the goal of establishing a given post-condition. Demonic choice is seen as the choice the system cannot control and is assumed to be resolved by an environment with another goal. Hence, the system may only guarantee the given postcondition if that condition may be established for all of the demonic alternatives.

STAIRS, which is a formal approach to incremental system development with UML 2.x sequence diagrams, introduces another type of nondeterminism called inherent nondeterminism or mandatory choice [116, 121, 115]. Mandatory choice means that all alternatives must be possible and such choices may therefore not be reduced by refinement. It is useful in specification of system that needs to fulfil secure information flow properties [66, 127]. Secure information flow concerns what a user can deduce about a system from her own interaction with the system and her knowledge of the system’s capabilities [66]. In particular a user within one security domain should not be able to deduce whether a certain behaviour associated with a security domain at a higher level has occurred, based on observations. This kind of property requires a certain unpredictability which is ensured by requiring that for each trace of a system there is at least one other trace which is equivalent from the point of view of the low level user. This is a so called trace-set property, since it can only be falsified on sets of traces [101].

Trace-set properties that are specified by the operator for mandatory choice in STAIRS can be preserved under refinement. In a development technique that does not distinguish between these two types of non-determinism refinement of trace set properties is problematic, since refinement typically reduces underspecification and thereby nondeterminism [66, 119, 127, 128].

Extensions to probabilistic specifications

In order to model systems that are both reactive and probabilistic the external non-determinism caused by an arbitrarily behaving environment must be resolved. A common way to resolve external nondeterminism in reactive systems is the use of a scheduler [26, 89, 149, 33, 43, 131, 9]. A scheduler specifies how to choose between nondeterministic alternatives.

Segala and Lynch [131, 130] use a randomised scheduler to model input from an external environment and resolve the nondeterminism of a probabilistic I/O automaton. They define a probability space [29] for each probabilistic execution of an automaton, given a scheduler.

Alfaro et al. [23] present a probabilistic model for variable-based systems with trace semantics similar to that of Segala and Lynch. They define a trace as a sequence of states, and a state as an assignment of values to a set of variables. Each component has a set of controlled variables and a set of external variables. Alfaro et al. represent a system by a set of probability distributions on traces, called bundles. They use schedulers to choose the initial and updated values for variables. Unlike the model of Segala and Lynch, theirs allows multiple schedulers to resolve the nondeterminism of each component. The key idea is to have separate schedulers for the controlled and external variables to ensure that variable behaviours are probabilistically independent. According to Alfaro et al. this ensures so called deep compositionality of their system model.
In a system model with deep compositionality the semantics of a composite system can be obtained from the semantics of its constituents. In contrast, shallow compositionality provides the means to specify composite components syntactically [23]. The semantics of a composite specification is obtained from the syntactic composite specification, but the semantics of this composition is not directly related to that of its constituents.

Seidel uses a similar approach in her extension of CSP with probabilities [132]. Internal nondeterministic choice is replaced by probabilistic choice. A process is represented by a conditional probability measure that, given a trace produced by the environment, returns a probability distribution over traces.

An alternative approach to handle external nondeterminism in probabilistic, reactive systems is to treat the assignment of probabilities of alternative choices as a refinement. This approach is used for example in probabilistic action systems [133, 147], where nondeterministic choices are replaced by probabilistic choices. A nondeterministic action system is transformed to a (deterministic) probabilistic system through the distribution of probabilistic information over alternative behaviours.

Probabilistic STAIRS (pSTAIRS) [115] introduces a generalisation of the operator for mandatory choice with an operator for probabilistic choice, which is meant to describe the probabilistic choice between two or more alternative operands whose joint probability should add up to one. pSTAIRS supports underspecification with respect to probabilities through the assignment of an interval of probability values to an alternative behaviour, rather than a single probability. In pSTAIRS it is possible to combine nondeterminism in the form of underspecification with mandatory choice and probabilistic choice. Underspecification may be reduced through refinement, but mandatory behaviour may not.

STAIRS does not focus on component-based systems. In STAIRS all choices (nondeterministic, mandatory and probabilistic) are global, that is, the different types of choices may only be specified for closed systems, and there is no nondeterminism stemming from external input.
Chapter 5

Overview of contributions

In this chapter we give an overview of our main contributions and explain how they relate to each other supported by a small running example. We also refer to the relevant chapters in Part II for definitions and further explanations.

5.1 The overall picture

As already explained our overall goal is to provide an approach to component-based risk analysis to support the integration of risk analysis into component-based development. Our goal is motivated by the challenge for security and safety caused by component-based systems, as upgraded sub-components may interact with a system in unforeseen ways. We believe that the integration of risk analysis into component-based development will make it easier to predict the effects on component risks caused by upgrading or substituting sub-parts.

Our approach to component-based risk analysis consists of: (1) a framework for component-based risk analysis; (2) a modular approach to risk modelling; (3) a formal foundation for modular risk modelling; (4) and a formal component model integrating the notion of risk. Figure 5.1 illustrates how the contributions relate to each other and to component-based risk analysis. The grey boxes represent our contributions.

Figure 5.1: Our four contributions
5 Overview of contributions

Contribution (1), the framework for component-based risk analysis, uses contribution (2) the modular risk modelling approach, for the purpose of identifying, analysing and documenting component risks. The framework for component-based risk analysis is based on the CORAS method for model-driven risk analysis [94]. We propose to adapt CORAS to make it suitable for component-based risk analysis. We also propose a stepwise integration of the adapted component-based risk analysis method into a component-based development process. Figure 5.2 illustrates how the process of component-based development relates to component-based risk analysis:

- While the component requirements capture the quality of service and functional requirements, the component protection requirements specify the acceptable level of risk, that is, what may be tolerated with respect to risks.

- The component protection specification describes how the system should be redesigned to fulfil the protection requirements. It corresponds to the component specification.

- The mitigated component corresponds to the implemented component.

- The use of modular risk modelling approach, for the purpose of identifying, analysing and documenting component risk interactions corresponds to the use of system modelling techniques such as for example the UML [109], for the purpose of specifying component interactions.

Contribution (2), the modular risk modelling approach, uses a custom made assumption-guarantee style to describe threat scenarios with external dependencies. Assumption-guarantee reasoning has been suggested as a means to facilitate modular system development [68, 104, 1]. The idea is that a system is guaranteed to provide a certain functionality, if the environment fulfils certain assumptions. The assumption-guarantee
5.2 Contribution 1: Framework for component-based risk analysis

style is useful for reasoning about risks of systems that interact with an environment. One of the main challenges towards component-based risk analysis is that component risks depend on external threats which may vary depending on the platform the component exists on. We use assumption-guarantee specifications of risk scenarios to make explicit what is assumed about external threats without including the environment into the target of analysis. We call this approach dependent risk analysis. Dependent risk analysis allows us to reason about the different parts of a system independently and to combine risk analysis results for separate parts into a risk analysis of the system as a whole.

Contribution (3), the formal foundation for modular risk modelling, provides a set of deduction rules to reason about threat scenarios expressed in the assumption-guarantee style.

Contribution (4), the formal component model formalises the informal understanding of component-based risk analysis which forms the basis for the framework of contribution (1). It provides a formal understanding on the top of which one may build practical methods and tools for component-based risk analysis.

A component in our model consists of a set of interfaces, which each has a set of assets. Interfaces interact by transmitting and consuming messages. An event is either the transmission or consumption of a message. An event that harms an asset is an incident with regard to that asset. In order to model the probabilistic aspect of risk, we represent the behaviour of a component interface by a probability distribution over communication histories. The formal component model use queue histories that serve as schedulers to address the challenge of representing open components. We define composition at the semantic level. The semantics of a composite component, including risk behaviour, is obtained from the semantics of its constituent components.

Our component model is purely semantic. It can be used to represent component implementations. At this point we have no compliance operator to check whether a given component implementation complies with a component specification. The definition of a compliance relation between specifications in a suitable specification language, such as for example probabilistic STAIRS [115] is a task for future work.

5.2 Contribution 1: Framework for component-based risk analysis

A method for component-based risk analysis should be based on the same principles of encapsulation and modularity as component-based development methods. In conventional risk analysis the parts of the environment that are relevant for estimating the risk-level are often included as part of the target of analysis. In a component-based setting, however, we cannot expect to have knowledge about the environment of the component as that may change depending on the platform it is deployed in. In order to provide an approach to component-based risk analysis, we identify which risk concepts to include at the component level, without compromising the requirements to encapsulation and modularity. In this section we present our understanding of risks in a component setting using a conceptual model. We first explain the concepts of risk analysis in Section 5.2.1 and then explain our conceptual component model in Section 5.2.2. In Section 5.2.3 we explain how the two conceptual models relate to each other and present the integrated conceptual model for component-based risk analysis.
5.2.1 Risk analysis

Risk analysis is the systematic process to understand the nature of and to deduce the level of risk [140]. We explain the concepts of risk analysis and how they are related to each other through the conceptual model, captured by a UML class diagram [120] in Figure 5.3. The risk concepts are adapted from international standards for risk analysis terminology [139, 64]. The associations between the elements 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.

We explain the conceptual model as follows: Stakeholders are those people and organisations who are affected by a decision or activity. 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 event refers to the occurrence of a particular circumstance. An event which reduces the value of at least one asset is referred to as an incident. A consequence is the reduction in value caused by an incident to an asset. It can be measured qualitatively by linguistic expressions such as “minor”, “moderate”, “major”, or quantitatively, such as a monetary value. 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, i.e., an attacker, or unintentional, i.e., someone causing an incident by fault. Probability is the extent to which an incident will occur. Conceptually, as illustrated by the UML class diagram in Figure 5.3, 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.

5.2.2 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
5.2 Contribution 1: Framework for component-based risk analysis

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 5.4. An interface is a contract describing both the provided operations and the services required to provide the specified operations. A component is a collection of interfaces some of which may interact with each other. Interfaces interact by the transmission and consumption of messages. We refer to the transmission and consumption of messages as events.

5.2.3 Component-based risk analysis

Figure 5.5 shows how the conceptual model of risk analysis relates to the conceptual component model. We identify assets on behalf of component interfaces. Each 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\(^1\). The set of component assets is the union of the assets of its interfaces. An event that harms an asset is an incident with regard to that asset. 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.

\(^1\)Note that there may be interfaces with no assets; in this case the stakeholder corresponding to the interface has nothing to protect.
5.2.4 Adapted CORAS method

As already mentioned, the framework for component-based risk analysis is based on adapting the CORAS method to make it suitable for component-based risk analysis. Our contribution consists of the suggested adjustments of CORAS, as well as the extension of the CORAS language with dependent CORAS described in Section 5.3. Our overall requirement to the adjusted method is that it should adhere to the same principles of encapsulation and modularity as component-based development methods, without compromising the feasibility of the approach or the common understanding of risk.

The CORAS method consists of a risk modelling language; the CORAS method which is 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. For a full presentation of the CORAS method we refer to the book *Model-Driven Risk Analysis: The CORAS Approach* by Soldal Lund et al. [94].

As illustrated by Figure 5.6, the CORAS process is divided into eight steps. The first four are introductory in the sense that they establish a common understanding of the target of analysis and make the target description that will serve as a basis for the subsequent steps. The remaining four steps are devoted to the actual analysis. This includes identifying concrete risks and their risk level, as well as identifying and assessing potential treatments.

The full adapted method is presented and evaluated in Chapter 9. In the following we summarise the most important steps we have taken to adjust CORAS into a method for component-based risk analysis:

1. We do not allow any concepts that are external to a component to be part of the target of a component-based risk analysis. This requirement is ensured through the following adjustments:
   i. The target of analysis is the component or component interface being analysed.
   ii. We identify stakeholders as interfaces of the component that is the target of analysis.
   iii. External threats are not part of the target of analysis. We use so called dependent threat diagrams [11] to analyse component risk without including threats in the target. Dependent threat diagrams allow us to document assumptions about the environment and parameterise likelihood values arising from external threats.

2. Assets are initially identified at the component level. When we decompose the component into interfaces we must decide for each asset which of the component interfaces it belongs to. This is necessary because we identify and analyse risks at the interface level and later combine the interface risks into a risk picture for the component as a whole. The assignment of assets to interfaces is not part of the original CORAS method.

3. We specify component risks as an integrated part of the component specification, using the same type of specification techniques. In particular we use sequence
Figure 5.6: The eight steps of the CORAS method
diagrams to specify risk interactions based on risk analysis documentation. The idea is that we should be able to update our knowledge about the risk behaviour when a component-based system is upgraded with a new component. The relation between normal interactions and risk interactions is illustrated in Example 1.

**Example 1 (Specifying risk interactions)** Consider a media player whose behaviour is illustrated by the sequence diagrams in Figure 5.7. The sequence diagram to the left specifies the normal behaviour of a media player:

When the operation `play` is called with a music file as argument, the media player should either open the music file or do nothing. The conditions under which alternative may be chosen is left unspecified at this point. In the final implementation we would expect the music file to be played if the file format is found to be correct and nothing to happen otherwise. Such a constraint may be imposed through use of guards. We have not treated the use of guards in this thesis. See Runde et al. [122] for a discussion on the use of guards in sequence diagrams. The `xalt`-operator [115] specifies that an implementation must be able to handle both the alternatives: to play the music file or do nothing.

![Sequence Diagrams](image)

**Figure 5.7: Normal interaction versus risk interaction**

The sequence diagram to the right specifies the risk interactions of the media player. The risk interactions capture how the normal interactions can be mis-used in order to cause incidents that harm the interface assets. In this case the incident is arbitrary behaviour caused by someone attempting to open a crafted music file designed to exploit possible buffer overflow vulnerabilities of the media player. 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.

As already explained 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, and hence that

---

2The example is illustrated using sequence diagrams in STAIRS [48, 122, 121]. STAIRS is an approach to the compositional development of UML 2.0 interactions. A sequence diagram shows messages passed between system parts (interfaces, in our case), arranged in time sequence. An interface is shown as a lifeline, that is, a vertical line that represents it through the interaction.
5.3 Contribution 2: Modular risk modelling

A crafted file is opened. We estimate the probability of this alternative to be in the interval \((0.1, 0.25)\). The alternatives that a particular event happens or not happens are mutually exclusive. We use the \texttt{expalt} operator of probabilistic STAIRS \cite{115} to specify mutually exclusive probabilistic choice in a sequence diagram.

The possibility of a buffer overflow is a known vulnerability of older versions of Winamp \cite{22}. 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 \cite{95, 123}. 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.

5.3 Contribution 2: Modular risk modelling

In order to support modular risk modelling we have extended the CORAS risk modelling language with facilities for documenting and reasoning about risk analysis assumptions \cite{11}. We refer to this extended language as dependent CORAS since we use it to document dependencies on assumptions. The assumptions are mainly of relevance in relation to threat scenarios and unwanted incidents that document the potential impact of events. Dependent CORAS is therefore only concerned with the two kinds of CORAS diagrams that can express these, namely threat diagrams and treatment diagrams. In the following we only show examples of dependent threat diagrams. Dependent treatment diagrams are presented in Chapter 11.

5.3.1 CORAS language

The CORAS language has been designed to document and facilitate risk analysis, and to communicate risk-relevant information throughout the various phases of a risk analysis process. To facilitate communication between participants of diverse backgrounds, the language employs simple icons and relations that are easy to read. In particular, the CORAS language is meant to be used during brainstorming sessions where discussions are documented on-the-fly in CORAS threat diagrams.

CORAS threat diagrams are used to aid the identification and analysis of risks and to document risk analysis results. Threat diagrams describe how different threats exploit vulnerabilities to initiate threat scenarios and incidents, and which assets the incidents affect. The basic building blocks of threat diagrams are: threats (deliberate, accidental and non-human), vulnerabilities, threat scenarios, incidents and assets. Figure 5.8 presents the icons representing the basic building blocks.

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, unwanted incidents, and assets. The relations are of three kinds: initiate, leads-to, and impacts. We explain the construction of a CORAS threat diagram through a simple example.

Example 2 (Threat diagram for the media player) Figure 5.9 shows a threat diagram for the media player from Example 1. When drawing a threat diagram, we start by placing the assets to the far right, and potential threats to the far left. The

\footnote{We use \((a, b)\) to denote the open interval \(\{x \mid a < x < b\}\).}
5 Overview of contributions

media player has one asset: *Media player integrity*. We identify one accidental human threat *User*.

Next we place incidents to the left of the asset. As already indicated in Example 1, there is a possibility of arbitrary behaviour if someone attempts to play a file with the wrong file type. From a risk perspective we are worried about exploits taking advantage of this vulnerability to cause the unwanted incident *Arbitrary code execution*, as indicated in the diagram. The incidents represent events which have a negative impact on one or more of the identified assets. This impacts relation is represented by drawing an arrow from the unwanted incident to the relevant asset.

The next step consists in determining the different ways a threat may initiate an incident. We do this by placing threat scenarios, each describing a series of events, between the threats and unwanted incidents and connecting them all with initiate relations and leads-to relations. An initiate relation originates in a threat and terminates in a threat scenario or an incident. For example, the accidental threat *User* causes the unwanted incident *Arbitrary code execution*, by unknowingly opening a crafted file. This is captured by an initiate relation going from *User* to the threat scenario *MP user plays crafted file*.

### 5.3.2 Dependent threat diagrams

Assume we want to integrate the media player from Example 1 with a peer-to-peer instant messaging service running on a mobile platform. The instant messaging service, the media player, and the mobile platform have all undergone separate risk analyses and we would like to know the overall risks of the combined system. The instant messaging service depends on the media player for playing files sent to it from other instant messaging services, and both the instant messaging service and the media player depend on the mobile platform for their performance. Since the system parts depend on each other, their risks also depend on each other. In order to state what we assume about external threats we use a dependent threat diagram.
5.3 Contribution 2: Modular risk modelling

In the following examples we follow the convention established in the component-based risk analysis framework that the components themselves are the target of analysis. This is however, not a necessary requirement of dependent CORAS. In general, dependent threat diagrams are not linked directly to system components, as the target of an analysis may be restricted to an aspect or particular feature of a system. The modularity of dependent threat diagrams is achieved by the assumption-guarantee structure, not by the underlying component structure and composition is performed on risk analysis results, not components. In Chapter 11 we use dependent threat diagrams in an analysis which does not follow the conventions of component-based risk analysis and where the target of analysis is not a component.

Example 3 (Dependent threat diagrams) Figure 5.10 shows a dependent threat diagram for the media player. 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.

![Figure 5.10: Dependent threat diagram for the media player](image)

External systems are not part of the target, because the risks of the media player is analysed without knowledge of which platform it will run on or which programs it will interact with. Everything inside the border line belongs to the target; everything on the border line, like the leads-to relations from the unwanted incident _IM service opens crafted file_, and from the threat scenario _MP user plays crafted file_, to the threat scenario _Buffer overflow in media player_ belongs to the target. We state our assumptions about factors in the environment that may affect the risk level of the media player, by placing them outside the border line. Hence, everything completely outside the border line like the unwanted incident _IM service opens crafted file_ and the threat scenario _MP user plays crafted file_, are the assumptions that we make for the sake of the analysis. The relations from the assumption to the target illustrate in what way the environment is assumed to affect the target.

Figure 5.11 shows the dependent CORAS diagram for the instant messaging service. The instant messaging service is a component consisting of several interfaces that each has its own asset: _Availability_, _Media file_ and _Efficiency_. The target scenario for the instant messaging service depends on the unwanted incident _Mobile platform corrupted_ and on two external threat scenarios initiated by external threats.

The dependent diagrams of Figures 5.11 and 5.10 illustrate that assumptions in one analysis can be part of the target in another analysis. The assumption _IM service_
5 Overview of contributions

Figure 5.11: Dependent threat diagram for the instant messaging service

*opens crafted file*, for example, is in Figure 5.10 an assumption about the potential causes of a buffer overflow in the media player. In Figure 5.11 the same threat scenario is part of the target.

Figure 5.12 shows the dependent CORAS diagram for the mobile platform. We have identified one asset *Platform integrity* of the mobile platform, that may be harmed by the unwanted incident *Mobile platform corrupted*. The target scenario of the mobile platform depends on the unwanted incident *Arbitrary code execution*, which is part of the target in the diagram in Figure 5.10, and the threat scenario *Malicious code downloaded on platform*.

The next step is to combine the risk analysis results for the instant messaging service, the media player, and the mobile platform to obtain the overall risks of the combined system. In order to do this we need a set of rules to reason about diagrams. These rules are part of the formal foundation that we present in the following section.
5.4 Contribution 3: Formal foundation for modular risk modelling

The formal foundation on the top of which we have defined rules for reasoning about CORAS diagrams is based on the notion of a risk graph. A risk graph is used to structure the events leading to incidents and to estimate likelihoods of incidents. It may be seen as a common abstraction of several risk modelling techniques, such as tree-based diagrams (e.g. Fault Tree Analysis (FTA) [60], Event Tree Analysis (ETA) [61], Attack trees [125]) or graph-based diagrams [12] (e.g. Cause-consequence diagrams [117, 96], Bayesian networks [18], CORAS threat diagrams [94]).

5.4.1 Semantics of risk graphs

We distinguish between two types of risk graphs: basic risk graphs and dependent risk graphs. A basic risk graph is a finite set of vertices and relations between the vertices. A vertex is denoted by \( v_i \), while a relation from \( v_i \) to \( v_j \) is denoted by \( v_i \rightarrow v_j \). Each vertex represents a scenario. A relation \( v_1 \rightarrow v_2 \) from \( v_1 \) to \( v_2 \) means that \( v_1 \) may lead to \( v_2 \), possibly via other intermediate vertices.

Vertices and relations can be assigned probability intervals. Letting \( P \) denote a probability interval, we write \( v_i(P) \) to indicate that the probability interval \( P \) is assigned to \( v_i \). Similarly, we write \( v_i \rightarrow v_j(P) \) to indicate that the probability interval \( P \) is assigned to the relation from \( v_i \) to \( v_j \). If no probability interval is explicitly assigned, we assume that the interval is \([0, 1] \).

A dependent risk graph is similar to a basic risk graph, except that the set of vertices and relations is divided into two disjoint sets representing the assumptions and the target. We write \( A \triangleright T \) 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.

A risk graph can be viewed as a description of the part of the world that is relevant for our analysis. As our main concern is analysis of scenarios, incidents, and their probability, we assume that the relevant part of the world is represented by a probability space [29] on traces. A trace is a finite or infinite sequence of events. We let \( \mathcal{H} \) denote the set of all traces. A probability space is a triple consisting of the sample space, i.e., the set of possible outcomes (here: the set of all traces \( \mathcal{H} \)), a set \( \mathcal{F} \) of measurable subsets of the sample space and a measure \( \mu \) that assigns a probability to each element in \( \mathcal{F} \). The semantics of a risk graph is a set of statements about the probability of trace sets representing vertices or combinations of vertices. This means that the semantics consists of statements about \( \mu \). We assume that \( \mathcal{F} \) is sufficiently rich to contain all relevant trace sets. For example, we may require that \( \mathcal{F} \) is the cone-\( \sigma \)-field of \( \mathcal{H} \) [130]. For a full definition of the semantics of risk graphs, see Chapter 11.

For combinations of vertices we let \( v_1 \sqcap v_2 \) denote the occurrence of both \( v_1 \) and \( v_2 \) where \( v_1 \) occurs before \( v_2 \) (but not necessarily immediately before), while \( v_1 \sqcup v_2 \) denotes the occurrence of at least one of \( v_1 \) or \( v_2 \). We say that a vertex is atomic if it is not of the form \( v_1 \sqcap v_2 \) or \( v_1 \sqcup v_2 \). For every atomic vertex \( v_i \) we assume that a set of finite traces \( V_i \) representing the vertex has been identified.

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:

$$i(D, D') \overset{\text{def}}{=} \{ v \in D : \exists v' \in D' : v \rightarrow v' \in D \cup D' \} \cup \{ v \rightarrow v' \in D : v' \in D' \}$$

Let for example

$$D = \{ IM \}$$
$$D' = \{ BO, IM \rightarrow BO \}$$
$$D'' = \{ UM, UM \rightarrow BO \}$$

represent different sub-graphs of the dependent threat diagram in Figure 5.10, based on the abbreviations in Table 5.1. Then $i(D, D') = \{ IM \}$ and $i(D, D'') = \emptyset$.

**Table 5.1: Abbreviations of vertices**

- $IM = IM$ service opens crafted file
- $UM = User$ initiates $MP$ user plays crafted file with likelihood Likely
- $BO = Buffer$ overflow in media player

A dependent risk graph on the form $A \triangleright T$ 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 $A \triangleright T$ is defined by:

$$(5.1) \quad \left[ A \triangleright T \right] \overset{\text{def}}{=} \forall T' \subseteq T : \left[ i(A \cup T \setminus T', T') \right] \Rightarrow \left[ T' \right]$$

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

### 5.4.2 Calculus

We have defined a set of deduction rules for computing likelihood values of vertices in threat diagrams and a set of rules for reasoning about dependencies. We show that the calculus is sound in Chapter 11. By soundness we mean that all statements that can be derived using the rules of the calculus are valid with respect to the semantics of risk graphs.

The rules are of the following form:

$$\frac{R_1 \quad R_2 \quad \ldots \quad R_i}{C}$$

We refer to $R_1, \ldots, R_i$ as the premises and to $C$ as the conclusion. The interpretation is as follows: if the premises are valid so is the conclusion.

Since a risk graph has only one type of vertex; a threat scenario, we must transform the relations and vertices of a CORAS diagram to fit with the semantics of risk graphs. For example 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 $t_j$. Since threats are not events but rather persons or things, we do not assign likelihood values to threats in CORAS, but to the initiate relation leading from the threat instead. We therefore combine a threat $t_j$, initiate relation $i_j$ with likelihood $P_j$ and sub-vertex $s_j$. 

5.4 Contribution 3: Formal foundation for modular risk modelling

into a new threat scenario vertex: \(\text{Threat } t_j \text{ initiates } s_j \text{ with likelihood } P_j\). For the full procedure for instantiating the risk graphs in CORAS see Chapter 11.

In the following we use dependent threat diagrams to exemplify a subset of the rules. We only include the rules that are used in the examples. The full calculus is defined in Chapter 11.

**Calculating likelihoods**

In the risk estimation phase the CORAS diagrams are annotated with likelihood and consequence values. Threat scenarios, incidents, initiate relations, and leads-to relations may be annotated with likelihood values. The rules presented here may be used to compute likelihood values of vertices given their parent vertices and relations leading to the vertices, and to check the consistency of the assigned likelihood values.

The relation rule formalises the conditional probability semantics embedded in a relation. The likelihood of \(v_1 \sqsupseteq v_2\) is equal to the likelihood of \(v_1\) multiplied by the conditional likelihood of \(v_2\) given that \(v_1\) has occurred. The new vertex \(v_1 \sqsupseteq v_2\) may be seen as a decomposition of the vertex \(v_2\) representing the cases where \(v_2\) occurs after \(v_1\).

**Rule 1 (Relation)** If there is a direct relation going from vertex \(v_1\) to \(v_2\), we have:

\[
\frac{v_1(P_1)}{(v_1 \sqcap v_2)(P_1 \cdot P_2)}
\]

where multiplication of probability intervals is defined as follows:

\[
[min_1, max_1] \cdot [min_2, max_2] \overset{\text{Def.}}{=} [min_1 \cdot min_2, max_1 \cdot max_2]
\]

If two vertices are statistically independent the likelihood of their union is equal to the sum of their individual likelihoods minus the likelihood of their intersection.

**Rule 2 (Independent vertices)** If the vertices \(v_1\) and \(v_2\) are statistically independent, we have:

\[
\frac{v_1(P_1)}{(v_1 \sqcup v_2)(P_1 + P_2 - P_1 \cdot P_2)}
\]

where subtraction of probability intervals is obtained by replacing \(\cdot\) with \(-\) in Definition (5.2).

**Example 4 (Likelihood computation)** Since the unwanted incident *Arbitrary code execution* depends on an incident and a threat scenario in the assumption, the likelihood of this incident depends on the likelihood of the assumed events. In the dependent threat diagram in Figure 5.13 we have instantiated the likelihood of the assumed unwanted incident *IM service opens crafted file* with *Rare* and the likelihood of the assumed threat scenario with *Likely*. The linguistic likelihood terms are mapped to probability intervals in Table 5.2.

For simplicity we have assigned exact probabilities to the leads-to relations, but we could have assigned intervals to them as well. For example we estimate that the likelihood of the unwanted incident *IM service opens crafted file* leading to the unwanted
5 Overview of contributions

<table>
<thead>
<tr>
<th>Likelihood</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rare</td>
<td>([0, 0.1])</td>
</tr>
<tr>
<td>Unlikely</td>
<td>([0.1, 0.25])</td>
</tr>
<tr>
<td>Possible</td>
<td>([0.25, 0.5])</td>
</tr>
<tr>
<td>Likely</td>
<td>([0.5, 0.8])</td>
</tr>
<tr>
<td>Certain</td>
<td>([0.8, 1.0])</td>
</tr>
</tbody>
</table>

Table 5.2: Likelihood scale

![Diagram](image)

Figure 5.13: Calculating likelihood values for the threat diagram of the Media player incident *Arbitrary code execution* to be 0.2, and likewise for the threat scenario *MP user plays crafted file* leading to the unwanted incident *Arbitrary code execution*.

If a diagram is incomplete, we can deduce only the lower bounds of the probabilities. For the purpose of the example we assume that the diagram is complete in the sense that no other threats, threat scenarios, or unwanted incidents than the ones explicitly shown lead to the threat scenario or unwanted incident in the diagram. Furthermore, we assume that the unwanted incident *IM service opens crafted file* and the threat scenario *MP user plays crafted file*, as well as the leads-to relations going from these to the threat scenario *Buffer overflow in media player*, are statistically independent.

For sake of readability we use the shorthand notations for the elements listed in Table 5.1, and \(AC\) for *Arbitrary code execution*. In order to calculate the probability for \(BO\), we first calculate the probabilities for \(IM \sqcap BO\) and \(UM \sqcap BO\). Applying Rule 1 we obtain the probability interval \([0, 0.1]\) for \(IM \sqcap BO\) and the probability interval \([0.1, 0.16]\) for \(UM \sqcap BO\). Since \(IM \sqcap BO\) and \(UM \sqcap BO\) are statistically independent, we may apply Rule 2 in order to calculate the probability of \((IM \sqcap BO) \sqcup (UM \sqcap BO)\). This gives the probability \([0.1, 0.6] + [0, 0.02] - [0.1, 0.6] \cdot [0, 0.02] \approx [0.1, 0.18]\) which lies within the interval defined as *Unlikely*. Applying Rule 1 again, we obtain the likelihood *Unlikely* for the vertex \(AC\).

**Reasoning about dependencies**

In order to reason about dependencies we first explain what is meant by dependency in the formal semantics. The relation \(D \nmid \nmid D'\) 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 3 (Independence)

\[ D \nparallel D' \iff D \cap D' = \emptyset \land i(D, D') = \emptyset \]

Note that \( D \nparallel D' \) does not imply \( D' \nparallel D \).

When we combine two or more interfaces or components into a component the result is a new open component, that is, a component that interacts with an environment. The combined threat diagrams of several interfaces will therefore often be a new dependent threat diagram with assumptions about risks stemming from the environment.

In order to support the sequential composition of several dependent threat diagrams into a new dependent diagram we have introduced an additional rule in Chapter 10 which is not part of the basic set of rules defined in Chapter 11. The rule states that if we have two dependent diagrams \( A_1 \triangleright T_1 \) and \( A_2 \triangleright T_2 \) 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 \( A_1 \cup A_2 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1 \cup T_2 \).

Rule 4 (Sequential composition)

\[
\begin{align*}
A_1 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1 & \quad A_2 \cup \{v\} \triangleright T_2 & \quad (A_1 \cup T_1) \cap (A_2 \cup T_2) = \emptyset
\end{align*}
\]

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 Chapter 10.

In the following examples we illustrate how to deduce the validity of the combined threat diagram for the instant messaging service, the media player, and the mobile platform shown in Figure 5.16.

Example 5 (Combining dependent diagrams) Let \( A_1 \triangleright T_1 \) represent the dependent threat diagram in Figure 5.11 and \( A_2 \triangleright T_2 \) represent the dependent threat diagram in Figure 5.10. We use the shorthand notations from Table 5.1, and \( HA \) for \textit{Hacker sends crafted file} initiates \textit{IM service opens crafted file}, for the elements that are active in the application of the rules. Let

\[
A_1' = A_1 \setminus \{HA\} \\
T_1' = T_1 \setminus \{IM, HA \rightarrow IM\} \\
A_2' = A_2 \setminus \{IM\}
\]

We assume that the dependent diagrams in Figures 5.11 and 5.10 are correct, that is: we assume the validity of

\[
A_1' \cup \{HA\} \triangleright \{HA \rightarrow IM, IM\} \cup T_1', \quad A_2' \cup \{IM\} \triangleright T_2
\]

Since \( (A_1' \cup T_1') \cap (A_2' \cup T_2) = \emptyset \), we can apply Rule 4 to the equations above and deduce

\[
A_1' \cup A_2' \cup \{HA\} \triangleright \{HA \rightarrow IM, IM\} \cup T_1' \cup T_2
\]

which corresponds to the combined diagram in Figure 5.14. \( \square \)

\[ ^4 \text{Normally dependent diagrams are combined after vertices and leads-to relations have been assigned likelihoods and impacts relations have been assigned consequence values. The assignment of likelihood and consequence values are, however, not necessary for the purpose of understanding dependency rules, and we leave them out of the example for the sake of simplicity.} \]
Overview of contributions

Figure 5.14: Sequential composition of dependent threat diagrams

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 5 (Target simplification)**

\[
\frac{A \triangleright T \cup T' \quad T' \triangleright T}{A \triangleright T}
\]

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

**Rule 6 (Assumption simplification)**

\[
\frac{A \cup A' \triangleright T \quad A \triangleright (A' \cup T)}{A' \triangleright T}
\]

**Example 6 (Target and assumption simplification)** Let \( A_3 \triangleright T_3 \) represent the dependent threat diagram in Figure 5.14. We decompose the assumption \( A_3 \) into \( A'_3 \) and \( A''_3 \) as illustrated by the two stippled borders around the assumption in Figure 5.14. We also decompose the target \( T_3 \) into \( T'_3 \) and \( T''_3 \) as illustrated by the two stippled borders around the target. Since \( A'_3 \cup A''_3 \cup \{HA\} \triangleright \{HA \rightarrow IM, IM\} \cup T'_1 \cup T_2 = A'_3 \cup A''_3 \triangleright T'_3 \cup T''_3 \), we have

(5.5) \[ A'_3 \cup A''_3 \triangleright T'_3 \cup T''_3 \]

by 5.4. From the dependent diagram in Figure 5.14, we see that \( T'_3 \) is independent of \( T''_3 \), that is, we have \( T'_3 \triangleright T''_3 \) according to Rule 3. Hence, by applying Rule 5 we can deduce

(5.6) \[ A'_3 \cup A''_3 \triangleright T'_3 \]
5.4 Contribution 3: Formal foundation for modular risk modelling

Since we also have \( A_3'' \uplus A_3' \cup T_3' \), we can apply Rule 6 and deduce
\[
(5.7) \quad A_3' \triangleright T_3'
\]
Using the same procedure we can also deduce the validity of
\[
(5.8) \quad A_3'' \triangleright T_3''
\]

In the next example we show how we can obtain the overall risk picture for the whole system, while keeping the assumptions about the external environment.

**Example 7 (Obtaining the overall risk picture)** Let \( A_4 \triangleright T_4 \) represent the dependent threat diagram in Figure 5.12. We use \( MC \) as a shorthand notation for *Mobile platform corrupted*. Since \((A_4 \cup T_4 \setminus \{MC\}) \cap (A_3' \setminus \{MC\}) = \emptyset\), we can apply Rule 4 to the dependent diagrams \( A_4 \triangleright T_4 \) and \( A_3' \triangleright T_3' \) and deduce
\[
(5.9) \quad A_4 \triangleright T_4 \cup T_3'
\]
which corresponds to the combined threat diagram in Figure 5.15.

Figure 5.15: Combined threat diagram

Let \( A_5 \triangleright T_5 \) represent the dependent threat diagram in Figure 5.15. Since \( A_4 \triangleright T_4 \cup T_3' = A_5 \triangleright T_5 \) we have
\[
(5.10) \quad A_5 \triangleright T_5
\]
by 5.9. We see that there are two vertices in \( A_5'' \) that leads to the unwanted incident *Arbitrary code execution*, which is in the assumption of the dependent diagram \( A_5 \triangleright T_5 \), but both of them go via other vertices. In order to apply Rule 4 to \( A_3'' \triangleright T_3'' \) and \( A_5 \triangleright T_5 \), we therefore interpret \( \{BO, BO \rightarrow AC, AC\} \) as a new vertex:

\[
BOA = Buffer\ overflow\ in\ media\ player\ leads\ to\ Arbitrary\ code\ execution\ with\ likelihood\ 1.0
\]

Let
\[
\begin{align*}
A_3''' &= A_3'' \setminus \{UM\} \\
T_3''' &= T_3'' \setminus \{UM \rightarrow BOA, BOA\} \\
A_5' &= A_5 \setminus \{BOA\}
\end{align*}
\]
By 5.8 and 5.10 we have

\[(5.11) \quad A_3'' \cup \{UM\} \triangleright T_3'' \cup \{UM \rightarrow BOA, BOA\}, \quad A_5' \cup \{BOA\} \triangleright T_5\]

Since \((A_3'' \cup T_3'') \cap (A_5' \cup T_5) = \emptyset\), we can apply rule 4 to the equations above and deduce

\[(5.12) \quad A_3'' \cup A_5' \cup \{UM\} \triangleright \{UM \rightarrow BOA, BOA\} \cup T_3'' \cup T_5\]

which corresponds to the combined diagram in Figure 5.16.

Figure 5.16: Combined threat diagram for the overall system

5.5 Contribution 4: Formal component model with a notion of risk

In order to develop a method for component-based risk analysis we need a solid formal basis. To provide such a formal basis, we have defined a denotational semantics for component behaviour that integrates the notion of risk. The denotational component model is meant to serve as a basis for method developers to build pragmatic principles and rules for component-based risk analysis. It gives a precise meaning to risk notions such as assets and unwanted incidents at the component level and provides rules for composing components with such notions. It also constitutes a framework for specifying components with risk notions, although it is not tied to any specific specification language. The full denotational model is defined in Chapter 12.
The component model does not say anything about how to obtain risk analysis results for components, such as the cause of an unwanted incident or its consequence and probability. In order to obtain information about component risks we must apply a risk analysis method, such as for example the modular approach described in Sections 5.2 and 5.3. Dependent CORAS supports analysis of open components by allowing the representation of threat diagrams with external dependencies. The component model is, however, defined independently of any risk analysis method used. The denotational component model formalises our conceptual model of component-based risk (Figure 5.5), in a trace-based semantics that defines:

- The denotational representation of an interface as a probabilistic process. Interface risks are incorporated as a part of the interface behaviour.
- The denotational representation of a component as a collection of interfaces, some of which may interact with each other.
- The denotational representation of hiding.
- Component composition.

### 5.5.1 Denotational representation of interfaces

In order to represent the behavioural aspects of risk, such as the probability of unwanted incidents, we make use of an asynchronous communication paradigm. We represent the behaviour of a component interface by a probability distribution over communication histories\(^5\). In order to resolve the external non-determinism caused by an arbitrarily behaving environment, we represent interfaces as functions of their queue histories. Incoming messages to an interface are stored in a queue and are consumed by the interface in the order they are received.

In addition to resolving external non-determinism the usage of queues has the effect of decoupling interactions between causally dependent interfaces. If the choices made by two distinct interfaces are statistically dependent it must be because they are physically related in some way, because they update the same variable, or that they observe the behaviour of each other through the exchange of messages, or that they observe the same behaviour of a third party. Using queues, we encapsulate all external behaviour affecting the decisions of an interface, which later allows us to reason about statistical dependencies of interface behaviour. The semantics of an interface is the set of probability spaces given all possible queue histories of the interface.

If the set of possible traces of an interface is infinite, the probability of a single trace may be zero. To obtain the probability that a certain sequence of events occurs up to a particular point in time, we can look at the probability of the set of all extensions of that sequence in a given trace set. Thus, instead of talking of the probability of a single trace, we are concerned with the probability of a set of traces with common prefix, called a cone.

In accordance with the conceptual model (Figure 5.5), an incident with regard to an asset is an event that harms the asset. The consequence of an event with regard to an asset is represented by a positive number indicating its level of seriousness with

\(^5\)Technically, an execution is represented by a special kind of probability space, as explained in Chapter 12
5 Overview of contributions

regard to the asset in question. Formally, an event is an incident with regard to an asset if it has a consequence value larger than zero, with regard to that asset. The same event may be an incident with respect to more than one assets; moreover, an event that is not an incident with respect to one asset, may be an incident with respect to another.

Our component model provides all the means necessary to calculate the risks associated with component interfaces. For example, to obtain the probability that an incident $e$ occurs during an interface execution we look at the union of the cones of all traces $t$ where $e$ occurs. We assume that events in a trace are totally ordered by time. This means that a given incident occurs only once in each trace. Furthermore, we assume that time-stamps are rational numbers which implies that an interface has a countable number of events. Since the set of finite sequences formed from a countable set is countable [82], the union of cones where $e$ occurs in $t$ is countable. This is a prerequisite for being able to measure the total probability of $e$ occurring.

By incorporating risks into the interface and component behaviour, the component model may convey the real meaning of risk analysis documentation with respect to an underlying implementation. For example, each of the identified risks in Example 3 represents events that are part of the behaviour of the specified components. They can therefore be represented in the formal representation of the components in question, according to the denotational model described here.

5.5.2 Denotational representation of components

In accordance with the conceptual component model (Figure 5.4), a component is a collection of interfaces some of which may interact. Figure 5.17 shows two different ways in which two interfaces $n_1$ and $n_2$ with queues $q_1$ and $q_2$, and sets of assets $a_1$ and $a_2$, can be combined into a component. We may think of the arrows as directed channels.

- In Figure 5.17.1 there is no communication between the interfaces of the component, that is, the queue of each interface only contains messages from external interfaces.

- In Figure 5.17.2 the interface $n_1$ transmits to $n_2$ which again transmits to the environment. Moreover, only $n_1$ consumes messages from the environment.
5.5.3 Denotational representation of hiding

In most component-based approaches one separates between the external and purely internal interaction. Purely internal interaction is hidden when the component is viewed as a black-box. When we bring in the notion of risk we would like to be able to observe all interactions that harm externally observable assets, even if it is internal. Hence, we require that incidents that affect assets belonging to externally visible interfaces are externally visible, even if they constitute purely internal interaction.

5.5.4 Component composition

We have also defined composition for probabilistic components. The behaviour of a composite component is obtained from the behaviours of its sub-components in a basic mathematical way. Since we integrate the notion of risk in component behaviour this means that we obtain the combined risks for two components $A$ and $B$, by looking at the risks of the composite component $A \otimes B$. 
5 Overview of contributions
Chapter 6
Overview of research papers

The main results of the work presented in this thesis are documented in the papers in Part II. In this chapter we give an overview of the contribution of each paper and list the publication details. We also indicate how much of the results are credited the author of this thesis.

6.1 Paper A: Using model-driven risk analysis in component-based development

Authors: Gyrd Brøndeland and Ketil Stølen.


My contribution: I was the main author, responsible for about 90% of the work.

Main topics: The paper presents a framework for component-based risk analysis. It gives an overview of existing system specification and risk analysis techniques that may be used to carry out the various tasks involved in component-based risk analysis and identifies areas where further research is needed in order to obtain a full method for component-based risk analysis.

6.2 Paper B: The dependent CORAS language

Authors: Gyrd Brøndeland, Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen.


6 Overview of research papers

My contribution: I was the main author, responsible for about 80% of the work.

Main topics: The paper presents an extension of the CORAS risk modelling language with facilities for documenting risk analysis assumptions. We refer to this extended language as dependent CORAS since we use it to document dependencies on 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.

6.3 Paper C: Modular analysis and modelling of risk scenarios with dependencies

Author: Gyrd Brændeland, Atle Refsdal, and Ketil Stølen.


My contribution: I was one of two main authors, responsible for about 45% of the work.

Main topics: The paper introduces the notion of a risk graph, as a common abstraction over risk modelling techniques. A risk graph is meant to be used during the risk estimation phase of a risk analysis to aid the estimation of likelihood values. The paper also introduces the notion of a dependent risk graph as a means to document assumptions of a risk analysis. A dependent risk graph is divided into two parts; an assumption that describes the assumptions on which the risk estimates depend, and a target. The paper includes a formal semantics and a calculus for risk graphs.

6.4 Paper D: A denotational model for component-based risk analysis

Author: Gyrd Brændeland, Atle Refsdal and Ketil Stølen.

Publication status: Technical report 363, University of Oslo, Department of Informatics, 2011.


My contribution: I was the main author, responsible for about 90% of the work.

Main topics: The paper presents a denotational trace semantics for component behaviour that defines: (1) the denotational representation of an interface as a probabilistic process; (2) the denotational representation of a component as a collection of interfaces, some of which may interact with each other; (3) the denotational representation of hiding; and (4) component composition. The paper also gives a precise meaning to risk notions such as assets and unwanted incidents as a part of the component behaviour. The denotational component model
is meant to serve as a basis for method developers to build pragmatic principles and rules for component-based risk analysis.
Chapter 7

Discussion

In this chapter we discuss and evaluate the contributions of this thesis. Due to the preliminary state of the presented contributions, we have used case-based examples and formal proofs for the evaluation of the success criteria. As discussed in Chapter 3, case-based evaluations can be useful in the early phase of a research process, in the creation of ideas and insights [35]. In order to empirically verify the feasibility of our artefacts and their influence on component quality and software development progress, further evaluations are necessary. In Section 7.1 we evaluate the contributions against the success criteria formulated in Chapter 2, and in Section 7.2 we discuss how our contributions relate to state of the art.

7.1 Fulfilment of the success criteria

In Chapter 2 we formulated our overall research goal: to provide an approach to component-based risk analysis that supports the integration of risk analysis into component-based development. Since the development of a complete risk analysis method is too extensive for a PhD thesis, we identified four sub-tasks as a first step towards a component-based asset driven risk analysis method, namely to develop: (1) a framework for component-based risk analysis; (2) a modular approach to risk modelling; (3) a formal foundation for modular risk modelling; (4) and a formal component model integrating the notion of risk. The overall goal was refined into a set of success criteria that each contribution should fulfil. In the following sub-sections we discuss to what extent the success criteria have been fulfilled for each contribution.

7.1.1 Framework for component-based risk analysis

1. The framework 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 the common understanding of risk. It must therefore be based on a clear conceptualisation of component risks.

The framework is based on a conceptual model of component-based risk analysis that was defined to clarify which risk concepts to include at the component level, without compromising the requirements to encapsulation and modularity. The risk concepts
in the conceptual model are adapted from international standards for risk analysis terminology [139, 64].

In Chapter 9 we apply the framework to a fictitious case involving an instant messaging component for smart phones and use this to evaluate this success criterion. We also discuss some problems for the feasibility of the approach caused by adhering to the principle of encapsulation and modularity. This principle implies that we do not allow any concepts that are external to a component to be part of the component-based risk analysis framework. This requirement is ensured through: (1) letting the target of analysis be the component or component interface being analysed; (2) identifying stakeholders as interfaces of the component that is the target of analysis; (3) and using dependent threat diagrams [11] to analyse component risk without including external threats or systems as part of the target.

Since we identify a stakeholder as a component interface it means that we identify assets on behalf of a component and its interfaces. A component asset may for example be confidentiality of information handled by a component interface. Limiting the understanding of a stakeholder in this way may be problematic from a risk analysis perspective, because 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. As a solution to this problem we propose 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 usage. An indirect asset is an asset that is harmed only through harm to other assets.

To conclude the evaluation of this criterion, we believe our framework adheres to the same principles of encapsulation and modularity as component-based development, but we have not yet solved all the problems this causes for the feasibility of applying the framework to real cases. We have identified some problems and identified areas where further research is needed, to obtain a full method for component-based risk analysis.

2. To provide knowledge about the impact of changes in sub-component upon system risks, the framework should allow risk related concepts such as assets, incident, consequence and incident probability, to be documented as an integrated part of component behaviour.

The framework for component-based risk analysis and the case-based evaluation of its application, presented in Chapter 9, provides guidelines for identifying and documenting assets at the component level. We document incidents and risks as an integrated part of the component behaviour and use the same type of specification techniques to specify normal interactions and 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. In order to specify risk interactions we first identify and estimate risk using threat diagrams. These steps are part of the original CORAS process, on which the presented framework is based, but should adhere to 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.
7.1 Fulfilment of the success criteria

We use sequence diagrams in pSTAIRS [115] to specify risk interactions based on risk analysis documentation. The idea is that we should be able to update our knowledge about the risk behaviour when a component-based system is upgraded with a new component. However, the current version of pSTAIRS has no facilities for documenting assumptions about the environment. Furthermore, the formal semantics of STAIRS (which is the basis for pSTAIRS) as defined by [48] and [46] 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. See future work in Section 8.2 for a suggested solution to this problem.

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

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 sub-components in the example described in Section 5.3, we only need to analyse how the changes affect that particular sub-component. The risk level for the overall system can be obtained using the described operations for composition.

7.1.2 Modular risk modelling

1. The modelling approach should be based on well established and tested techniques for risk modelling and analysis.

The modelling approach is an extension of the CORAS modelling language. 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 [55, 56, 57]. The CORAS method was compared to six other risk analysis methods in a test performed by the Swedish Defence Research Agency in 2009 [8]. 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.

2. To facilitate modular risk analysis the modelling approach should allow the modelling of threat scenarios with dependencies.

This success criterion is fulfilled through extending the CORAS risk modelling language with facilities for documenting and reasoning about risk analysis assumptions [11]. We refer to this extended language as dependent CORAS since we use it to document dependencies on assumptions. A dependent CORAS diagram consists of an assumption and a target. The target corresponds to a CORAS threat diagram, describing how different threats exploit vulnerabilities to initiate threat scenarios and unwanted incidents, and which assets the unwanted incidents affect. The assumption describes external threats and unwanted incidents that may lead to threat scenarios and unwanted incidents within the target. Graphically we use a rectangular borderline to
separate the assumption from the target. Everything within the borderline, crossing relations and vertices on the border belongs to the target. The rest belong to the assumption on which the analysis results depend.

7.1.3 Formal foundation for modular risk modelling

1. To facilitate modular risk analysis the formal foundation should provide a formal calculus that characterises conditions under which:

   (a) the dependencies between scenarios can be resolved distinguishing bad dependencies (i.e., circular dependencies) from good dependencies (i.e., non-circular dependencies);

   (b) risk analyses of separate system parts can be put together to provide a risk analysis for system as a whole.

The formal foundation for modular risk modelling provides a formal semantics for risk graphs, which are more general than dependent CORAS diagrams and can be seen as a common abstraction for several risk modelling techniques. It also provides a calculus for reasoning about risk graphs. In Chapter 11 we show how the calculus can be used to reason about dependent threat diagrams instantiated in the semantics of risk graphs that describe mutually dependent systems. We apply the calculus in an example involving the power systems in the southern parts of Sweden and Norway. We show that in this example we can resolve dependencies joining diagrams in a borderlike pattern. In general, our approach is able to handle arbitrary long chains of dependencies, as long as they are not circular. The calculus also contains rules for deducing when a risk graph is a valid composition of two or more dependent risk graphs. Hence, success criteria 1 (a) and 1 (b) are fulfilled.

2. The formal foundation should allow the risk analysis of complex systems to be decomposed into separate parts that can be carried out independently.

This success criterion is implicitly fulfilled through the inclusion of rules for reasoning about risk graphs with dependencies. We can decompose the target of a risk analysis into several targets that are analysed independently, but where the analysis of each sub-target state assumptions about dependencies with regard to other sub-targets. If two sub-targets depend on each other this means that assumptions in one analysis will be part of the target in another analysis. It may be necessary to revise some of the assumptions when two or more mutually dependent risk graphs shall be put together. Given that the conditions for combining risk graphs are fulfilled, the individual risk graphs can be combined to obtain the risk picture for the whole system.

7.1.4 Formal component model with a notion of risk

1. The component model should be defined in accordance with the conventional notion of a component.

In order to discuss the extent to which our approach fulfils this criterion we begin with a brief discussion of what the conventional notion of a component is.
The definitions of components vary depending on what aspects of component-based development they focus on: different phases in the development (design, implementation, run time); business aspects (business components, service components etc.); modular development techniques (semiformal techniques such as UML [109], formal techniques such as Temporal Logic of Actions [86], FOCUS [16], and STAIRS [47, 121, 115]). However, the basic features of a component introduced in the classic definition by Szyperski, have been widely adopted in later definitions:

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties [143].

That a component is a unit of independent deployment means that it needs to be distinguished from its environment and other components. This distinction is provided by a clear specification of the component interfaces and by encapsulating the component implementation [21]. The interface specification should describe both the services provided by the component and the service required by the environment in order to provide these services. Thus, at a general level, a component definition should include the following notions:

- provided services;
- required services;
- composition.

Szyperski’s definition has been criticised for its lack of reference to a component model and lack of a definition of composition [87, 88]. Lau and Wang [88] maintain that a component model is necessary to provide the underlying semantic framework, defining:

- the syntax of components, i.e., how they are constructed and represented;
- the semantics of components, i.e., what components are meant to be;
- the composition of components, i.e., how they are composed or assembled.

They have surveyed different component models with regard to their facilities for composition during the various phases of a component life cycle. According to their evaluation, no current component models provide mechanisms for composition in all phases.

In Szyperski’s defence it may be said that it is difficult to explain how components are composed at a general level, as what is meant by composition varies in the same way the meaning of a component does. Thus, that a component is a unit of composition means different things in different settings.

Our component model formalises the informal understanding of component-based risk analysis in the conceptual model described in Section 5.2.3. In order to discuss whether our component model properly defines the notion of composition, we must look at what compositionality means in a formal setting.

In formal development techniques compositionality can mean two things: compositional development and compositionality of components. Compositional development implies that refinement is closed under composition: if a component $C_2$ refines $C_1$
then the result of the parallel composition $C_2 || C$ of $C_2$ with any component $C$ refines $C_1 || C$ [71, 130]. Compositionality of components implies that the semantics (i.e., the behavioural representation) of a component can be obtained from its constituents [23, 115].

Since we aim to measure component risks, we need a probabilistic understanding of component behaviour. To facilitate modularity of probabilistic behaviour we made certain assumptions about how components interact. Our two basic strategies involve decoupling of interface interaction through the use of queues and decomposition of probabilistic choices to obtain a degree of granularity where a single choice involves only one interface. These strategies are explained in greater detail in Chapter 12.

We define behavioural representation of a component interface as a function of a complete queue history. Interface behaviour is a set of probability distributions over traces, one for each queue history. We could have chosen to represent only closed components by always including the assumed environment. That would simplify our semantic model, as closed components can be represented by only one probability distribution, instead of a set of probability distributions. If we restrict the model to closed components, however, we would only manage to define component composition at the syntactic level. We define composition at the semantic level, and obtain the semantics of a composite component from the semantics of its constituents. One advantage of this is the possibility to define compositional or modular refinement at a later stage.

Our formalisation of the component model is purely semantic. It can be used to represent component implementations. We have at this point not defined a syntax for specifying components.

To keep our component model simple and general we do not distinguish between usage and realisation contracts. We simply have interfaces, which are contracts describing both the provided services and the services required to provide the specified operations. A component is a collection of interfaces some of which may interact with each other. Hence, at the conceptual level our component model implicitly contains both provided and required services.

To summarise, we argue that our component model is defined in accordance with the commonsensical understanding of a component, with regard to the notions of provided and required services. Since our component model is purely semantic, we do not fulfil the requirement by Lau and Wang that component models should also define the syntax of components. With regard to compositionality, we cannot claim that our model is compositional in the sense used by e.g. Jonsson [71] and Segala [130], since this notion is related to refinement, something which we do not cover. We do however define a composition operator for components in Chapter 12 and the semantics of a composed component can be obtained from the semantics of its constituents.

2. The component model should define the precise meaning of risk notions at the component level. In particular it should define the meaning of asset, incident, consequence, probability and risk, with regard to a component. The definitions should be in accordance with existing standards for risk analysis terminology.

In Chapter 12 we give a precise meaning to the concept of asset, incident, incident consequence and incident probability with regard to a component execution represented in a denotational semantics. We define an asset as a physical or conceptual entity which is of value for an interface. An event that harms an asset is an incident with regard to that asset. An event is either the consumption or the transmission of a message
by an interface. A consequence is a measure on the level of seriousness of an incident with regard to an asset. As explained in Section 5.5 we obtain the probability that an incident \( e \) occurs during an interface execution by looking at the union of all behaviours where \( e \) occurs. The risk value of an incident \( e \) in an execution \( H \), with regard to an asset \( a \), is the combination of the probability of \( e \) in \( H \) and its consequence value. The risk concepts formalises the informal understanding of risk in the conceptual model described in Section 5.2.3. The risk concepts in the conceptual model are adapted from international standards for risk analysis terminology [139, 64]. Hence, success criterion 2 is fulfilled.

3. The component model should include rules for composition of components defined according to a component model with risk notions.

In Chapter 12 we define rules for composition of probabilistic components with a notion of risk. The behaviour of a composite component is obtained from the behaviours of its sub-components in a basic mathematical way. Since we integrate the notion of risk in component behaviour this means that we obtain the combined risks for two components \( A \) and \( B \), by looking at the risks of the composite component \( A \otimes B \). Hence, success criterion 3 is fulfilled.

4. The component model should characterise what it means for a component to satisfy a requirements to protection definition.

In order to evaluate this success criterion we first discuss what type of property a requirements to protection definition describes. McLean [101] distinguishes between property falsification on the basis of a single trace and property falsification on the basis of a set of traces. Properties of traces encompass safety and liveness properties, and are the kind of properties originally discussed by Alpern and Schneider [124, 4]. Properties of traces can be falsified by only one trace. Properties of sets of traces are referred to as “possibilistic properties” [101]. They encompass information security flow properties, and can only be falsified by looking at the complete behaviour of a system.

A requirements to protection definition establishes the accepted level of risk towards asset. It may be on the form: “The risk value for asset \( a \) of component \( C \) should be no higher than \( n \).” A risk value is the combination of the consequence and probability of an incident. As explained in Section 5.5 our component model provides all the means necessary to calculate the risks associated with components and component interfaces. Since, as explained in Chapter 12 there will in general be several different behaviours that can lead to the same incident, the overall probability of a risk can only be calculated by looking at the set of behaviours leading to the incident in question. This means that a requirements to protection definition describes a property of sets of traces and can therefore only be falsified by looking at the complete behaviour of a system.

We do not discuss explicitly how to check that a requirements to protection definition is satisfied in the presentation of the formal component model. Since we have all the means necessary to calculate component risks it is, however, straightforward to describe what it means for a component to satisfy a requirements to protection definition \( P \): A given component execution \( E \), satisfies a requirements to protection definition for an asset \( a \), if the total risk value in \( E \) with regard to \( a \) does not violate \( P \). A component \( C \) satisfies \( P \) if every possible execution of \( E \) satisfies \( P \).
7 Discussion

7.2 How our approach relates to state of the art

In this section, we discuss how our work relates to state of the art. We structure the discussion into four sub-sections; one for each topic discussed in Chapter 4.

7.2.1 Framework for component-based risk analysis

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.

The approaches to component-based hazard analysis described in Section 4.1.3 [38, 37, 111, 10, 74, 31] are limited to hazard analysis targeting hazards caused by software or hardware failures. Our approach allows documentation not only of system failures, but also assumptions about the threats that may cause them, and the consequences they may lead to in terms of loss of asset value. Furthermore, our focus is not on failures as such, but rather on mis-use of normal interactions that may cause incidents harming component assets.

The approach to risk-driven development of security-critical systems proposed by Jürjens and Houmb [73] is perhaps the most similar to ours 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 [73] have no particular focus on component-oriented specification.

7.2.2 Modular risk modelling

To our knowledge no existing risk modelling technique offers the possibility to document assumptions about external dependencies as part of the risk modelling. In Section 4.2 we discussed approaches to assumption-guarantee reasoning in modular system development [68, 69, 104, 102, 52]. The presented modular risk modelling approach transfers the assumption-guarantee style to threat modelling to support documentation of environment assumptions. Similar to the assumption-guarantee approach it is divided into two parts, an assumption and a target.

1. The assumption describes the assumptions on which the risk estimates depend.

2. The target documents the estimated risk level for the system being analysed.

For example, if we analyse risks towards a database storing valuable information, we may assume that force majeure events such as flooding or fire do not occur. We may document such assumptions explicitly in a dependent threat diagram by assigning the probability 0 to these events. We only guarantee the documented risk level of the target, given that events such as flooding or fire do not occur.

Another similarity between dependent threat diagrams and existing assumption-guarantee approaches is the support for modular reasoning. The modular risk modelling approach is built on top of a formal semantics which characterise conditions...
under which risk analysis results can be composed and how non-circular dependencies can be resolved.

7.2.3 Formal foundation for modular risk modelling

A risk graph can be seen as a common abstraction of the modelling techniques described in Section 4.3 [60, 61, 125, 96, 114, 18]. A risk graph combines the features of both fault tree and event tree, but does not require that causes are connected through a specified logical gate. A risk graph may have more than one root vertex. Moreover, in risk graphs 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 risk graph 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 because it may be used to uncover inconsistencies. Uncovering inconsistencies helps to clarify misunderstandings and pinpoint aspects of the analysis that must be considered more carefully. For a discussion on how to instantiate fault trees and CORAS threat diagrams in risk graphs, see Chapter 11.

Risk graphs allow underspecification of risks through the assignment of sets of probability values to vertices and relations. This is important with regard to usability of the method, because in many practical situations it is difficult to find exact likelihood values. As discussed in Section 4.3, conventional fault tree analysis does not handle uncertain input data, but there are several elaborations of fault tree analysis that do [136, 105, 151, 141, 17].

7.2.4 Formal component model with a notion of risk

We have chosen to use a denotational semantics for our formal component model, rather than an operational semantics. Different semantic models were considered in the initial phase of the work on the thesis, such as the operational trace semantics of CSP [53] and the stream based semantic model of FOCUS [16]. A benefit of choosing a denotational semantics rather than an operational semantics for our component model is that it is particularly suitable for mathematical reasoning and abstract requirements. Furthermore, a requirement to the semantic model is that it should be simple, providing only the necessary expressive power required for our purposes. This requirement was motivated by the fact that it is easier to introduce and explore new concepts in a simple model than in a complex one.

To keep things simple we have also made a number of assumptions in the formal component model. For example, we assume that communication between component interfaces is asynchronous and that events are totally ordered by time. These assumptions do not restrict the expressive power of our denotational semantics, as discussed in Chapter 12, since a more general understanding can be simulated in our model.

Our component model is purely semantic. It can be used to represent component implementations. We have at this point not defined syntax for specifying components. The purpose of the presented component model is to form the necessary basis for building applied tools and methods for component-based risk analysis. The approaches to specifying probabilistic components discussed in Section 4.4.3 [131, 130, 23] can be used as a basis for a specification language needed in such a method.
Discussion

Since we wish to represent the behaviour of a component independently of its environment we cannot use global choice operators of the type used in STAIRS [115]. We define probabilistic choice at the level of individual component interfaces and use queue histories to resolve external nondeterminism. The idea to use queue histories to resolve the external nondeterminism of probabilistic components is inspired by the use of schedulers described above.
Chapter 8

Conclusion

In this chapter we summarise what has been achieved, and discuss possibilities for future work.

8.1 What has been achieved

We have argued that risk analysis methods for component-based systems should be based on the same principles of modularity and composition as component-based development. Our goal has been to contribute to a component-based risk analysis method that supports the integration of risk analysis into component-based development. In order to achieve this goal we have developed (1) a framework for component-based risk analysis; (2) a modular approach to risk modelling; (3) a formal foundation for modular risk modelling; and (4) a formal component model integrating the notion of risk. In the following we explain the achievements of each contribution in more detail.

8.1.1 Framework for component-based risk analysis

The framework for component-based risk analysis provides a process for conducting risk analysis at the component level and guidelines for conducting each task of the process. The presented framework is based on the CORAS method for model-driven risk analysis [94]. Our contribution consists of the suggested adjustments of the CORAS process, as well as the extension of the CORAS language with dependent CORAS described in Section 5.3. The process provided by the framework mirrors that of component-based development, thereby facilitating the stepwise integration of risk analysis into component-based development. We propose to identify and document risks in parallel to component specification and integrating the identified risks in the description of the component behaviour.

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. The risk level of a composite system can be obtained using our calculus for modular risk analysis. Hence, there is no need to analyse a system from scratch when a sub-part (component) is modified. We only need to analyse how the changes affect the risk level of the modified sub-component.

The framework for component-based risk analysis ties together the component model, the modular risk modelling approach and the formal foundation for modu-
lar risk modelling in the following way: (1) components are specified according to the component model; (2) risks are modelled and analysed using the modular risk modelling approach; and (3) analysis results are combined using the rules of the formal foundation for modular risk modelling.

8.1.2 Modular risk modelling

To support the risk identification and analysis phase of a component-based risk analysis process (see Section 4.1), we have proposed an extension of the graphical risk modelling language CORAS [94]. We have argued that a risk analysis method for component-based systems should be based on the same principles as modular component development. According to Abadi and Lamport modular development involves [1]:

- decomposing a system into smaller parts (i.e. components);
- composing components to form a larger system.

As pointed out by Abadi and Lamport, when reasoning about a given component in a decomposed system, it is necessary to state the assumptions made about the component’s environment and then prove that this assumption is satisfied by the other components. When specifying a reusable component, without knowing precisely where it will be used, it must be made explicit what the component assumes of its environment. This motivates a division of component specifications into an assumption and a guarantee.

Dependent CORAS transfers the assumption-guarantee style to threat modelling. The idea is simple: Instead of stating assumptions that the environment is required to fulfil in order for a system to provide its desired behaviour, we state assumptions about how the environment behaves in order to cause undesired behaviour of the target of analysis, in terms of risks. We achieve this by extending CORAS threat diagrams with facilities for modelling dependencies. Such extended CORAS diagrams are referred to as dependent CORAS diagrams. Dependent CORAS was motivated by the need to deal with mutual dependencies in risk analysis of systems of systems [13, 12]. A modular approach to risk modelling and analysis is, however, useful for risk analysis of composite systems in general. A benefit of documenting explicitly the assumptions on which risk analysis results depend is the possibility to compare component risks independent of context, which is a prerequisite for creating a market for components with documented risk properties.

8.1.3 Formal foundation for modular risk modelling

The formal foundation for modular risk modelling provides a formal semantics and rules for reasoning about threat scenarios described in a modular risk modelling language. It introduces the general notion of a risk graph, which may be seen as a common abstraction of several risk modelling techniques, such as tree-based diagrams (e.g. Fault Tree Analysis (FTA) [60], Event Tree Analysis (ETA) [61], Attack trees [125]) or graph-based diagrams [12] (e.g. Cause-consequence diagrams [117, 96], Bayesian networks [18], CORAS threat diagrams [94]). The rules of the calculus may be used to verify that a risk graph constitutes a correct composition of two or more risk graphs. We have also provided guidelines for interpreting a CORAS threat diagram as a risk graph in
order to reason about it using the calculus of risk graphs. The rules of the calculus are proved to be the sound.

### 8.1.4 Formal component model with a notion of risk

The formal component model allows us to represent risk notions in relation to component behaviour, such as asset, unwanted incident, consequence and probability. The component model is inspired by the idea behind misuse cases and abuse cases [98, 97, 134, 135]. Namely that component misuse can be identified in a similar manner to component use. Misuse cases are used to represent actions that a system should prevent, as a means to elicit security requirements. Risk analysis takes a different approach. Rather than preventing all undesired behaviour it aims to establish the appropriate level of risk and select the protection mechanisms required to ensure that this risk level is not exceeded. The rationale behind this approach is that there are risks it may be too costly or technically impossible to prevent. Hence, risk behaviour is part of the behaviour that is implicitly allowed within the specified component behaviour but not necessarily intended. We have developed a formal component model that allows the explicit representation of risk behaviour as part of the component behaviour.

The component model provides a formal basis for applied methods for component-based risk analysis. By representing risks as part of the component behaviour, the component model conveys the real meaning of risk analysis documentation with respect to an underlying component implementation. The composition of risk behaviour corresponds to ordinary component composition.

The component model does not say anything about how to obtain risk analysis results for components, such as the cause of an unwanted incident or its consequence and probability. In order to obtain information about component risks we must apply a risk analysis method, such as for example the modular approach described above.

### 8.2 Future work

As already mentioned the overall goal of the thesis is to contribute to a component-based asset driven risk analysis method. The framework for component-based risk analysis presented in Chapter 9 describes a process for such a method. As explained above, the framework uses the modular risk modelling approach (contribution 2) to model and analyse component risks and the formal foundation for modular risk analysis (contribution 3) to reason about component risks. The formal component model (contribution 4) may be used to represent components developed according to an integrated risk analysis and development approach.

In Chapter 9 we identify tasks for further work which is needed in order to obtain a full method for component-based risk analysis. We discuss some of these tasks in further detail below. We also discuss improvements to the artefacts that have already been developed.

In order to be able to check that a component implementation fulfils a requirements to protection specification we would like to define formally (1) how sequence diagrams in probabilistic STAIRS (pSTAIRS) relates to probabilistic interface executions and (2) how dependent risk graphs relate to sequence diagrams.

Refsdal et al. [115] have defined a compliance relation which explains the relation
between a component specification in pSTAIRS and a component implementation. However, this compliance relation in pSTAIRS is not directly applicable in our case. In pSTAIRS all choices (nondeterministic, mandatory and probabilistic) are global, that is, the different types of choices may only be specified for closed systems, and there is no nondeterminism stemming from external input. The behaviour of a component in pSTAIRS is therefore represented by a single probability space [115]. This differs from our approach where a component implementation is represented as a function of a queue history. The semantics of a component is the set of probability spaces given all possible queue histories. This is due to our choice to allow open components.

We would also like to define rules for refinement of component specifications that preserves requirements to protection. The requirements to protection specify what may be tolerated with regard to risks towards assets. The probability of an incident can only be calculated by looking at the set of behaviours leading to the incident, since, in general, several different behaviours may lead to the same incident. This implies that a protection requirement defines a property of a set of traces, similar to information flow properties. Conventional refinement techniques do not apply to information flow security properties [66, 41, 119]. Seehusen et al. [129, 128] have, however, shown that information flow properties are preserved by limited refinement in STAIRS. It should be possible to show similar results for requirements to protection.

With regard to the formal foundation for modular risk analysis, the carrying out of a proof that a composition is correct is quite cumbersome. In order to make the use of dependent threat diagrams feasible in a real risk analysis it should be supported by a tool that could perform the derivations automatically or semi-automatically, such as the interactive theorem prover Isabelle\textsuperscript{1}.

Another point for improvement is the rules for computing likelihood values of threat scenarios and incidents in threat diagrams. The calculation of likelihood values propagating through a diagram are only possible on the condition that events from separate branches leading to a vertex are statistically independent or mutually exclusive. However, there is no direct way to deduce from a diagram the statistical dependencies between events represented by vertices and relations. A solution might be to encode such dependencies into the syntax, in a similar fashion to Bayesian networks [18]. Encoding dependencies into the syntax has computational advantages, but on the other hand it may complicate the possibility to create CORAS threat diagrams on-the-fly during brainstorming sessions, which is one of the current strengths of the CORAS language.

We would also like to investigate in more detail how our approach to modular risk analysis applies to other risk modelling techniques such as fault trees and Bayesian belief networks.

A method for modular risk analysis should provide guidelines for how to compare two scenarios based on an understanding of what they describe, rather than their syntactic content. In the presented examples of dependent CORAS diagrams we need an exact syntactic match between the assumptions of one dependent diagram and the target of another in order to verify their correct composition. In practice two risk analyses performed independently of each other will not provide exact syntactic matches of the described threat scenarios. One possible approach is to combine the presented approach with the idea of so-called high-level diagrams that describe threat

\textsuperscript{1}http://isabelle.in.tum.de/
scenarios at a higher abstraction level. For a description of high-level CORAS see Lund et al. [94]. Finally, we would like to extend the CORAS tool to handle dependent CORAS.
8 Conclusion
Bibliography


Bibliography


Part II

Research papers
Chapter 9

Paper A: Using model-driven risk analysis in component-based development
9 Paper A: Using model-driven risk analysis in component-based development
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

February 28, 2011

Abstract

Modular system development causes challenges for security and safety as upgraded sub-components 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** 5  
   1.1 A framework for component-based risk analysis ................. 5  
   1.2 Outline .................................. 6  

2 **Background** 7  
   2.1 Risk analysis ................................ 7  
      2.1.1 Model-driven risk analysis ..................... 8  
   2.2 Component-based development ............................ 8  
      2.2.1 Model-driven component development .............. 10  

3 **Component-based risk analysis and development** 11  
   3.1 Component-based risk analysis concepts ....................... 11  
   3.2 Integrating risk analysis into component-based development ...... 12  

4 **Requirements** 13  
   4.1 Requirements definition .................................. 15  
      4.1.1 Use cases .................................... 15  
      4.1.2 Business concept model ............................ 16  
   4.2 Requirements to protection definition ....................... 17  
      4.2.1 Identify component assets ........................ 17  
      4.2.2 Establish the level of protection .................. 18  

5 **Interfaces and their assets** 19  
   5.1 Interface identification ................................... 19  
      5.1.1 Identifying system interfaces and operations ........... 21  
      5.1.2 Identifying business interfaces ....................... 21  
      5.1.3 Interface dependencies .............................. 22  
      5.2 Interface asset identification ........................... 23  

6 **Interactions** 25  
   6.1 Interface interactions ..................................... 25  
   6.2 Interface risk interactions .................................. 30  
      6.2.1 Identify interface risks ............................. 30  
      6.2.2 Estimate likelihoods ................................. 34  
      6.2.3 Estimate consequences ................................. 37  
      6.2.4 Specifying risk interactions ......................... 39  

7 **Specification** 44  
   7.1 Component specification .................................... 44  
   7.2 Component protection specification .......................... 46  
      7.2.1 Reasoning about dependent diagrams .................. 46  
      7.2.2 Combining interface risks into component risks ......... 48  
      7.2.3 Evaluating component risks ........................... 50  

8 **Related work** 54  

9 **Conclusion and discussion** 56
<table>
<thead>
<tr>
<th>References</th>
<th>58</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong> Proofs</td>
<td>62</td>
</tr>
<tr>
<td><strong>B</strong> Key terms and definitions</td>
<td>67</td>
</tr>
</tbody>
</table>
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](image)

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.

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 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
Figure 2: The eight steps of the CORAS method

Figure 3: Conceptual component model
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 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 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

\(^1\)Note that there may be interfaces with no assets; in this case the stakeholder corresponding to the interface has nothing to protect.
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, we first conduct the development step and then conduct the corresponding risk analysis step.

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.

## 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.
Figure 7: Adapting CORAS into a component-based risk analysis process
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](image)

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 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 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.
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.
### Likelihood Description

<table>
<thead>
<tr>
<th>Likelihood</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unlikely</td>
<td>((0.00, 0.25])²</td>
</tr>
<tr>
<td>Possible</td>
<td>((0.25, 0.50])</td>
</tr>
<tr>
<td>Likely</td>
<td>((0.50, 0.70])</td>
</tr>
<tr>
<td>Almost certain</td>
<td>((0.70, 0.99])</td>
</tr>
<tr>
<td>Certain</td>
<td>({1.00})</td>
</tr>
</tbody>
</table>

Table 1: Likelihood scale for probabilities

The risk values decide the requirements to protection: risks with a low risk value are acceptable, while risks 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.

<table>
<thead>
<tr>
<th>Consequence</th>
<th>Unlikely</th>
<th>Possible</th>
<th>Likely</th>
<th>Almost certain</th>
<th>Certain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moderate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Major</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Protection criteria for UserID

<table>
<thead>
<tr>
<th>Consequence</th>
<th>Unlikely</th>
<th>Possible</th>
<th>Likely</th>
<th>Almost certain</th>
<th>Certain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moderate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Major</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

²We use \((a, b]\) to denote the open interval \(\{x \mid a < x < b\}\).
<table>
<thead>
<tr>
<th>Consequence</th>
<th>Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unlikely</td>
</tr>
<tr>
<td>Minor</td>
<td></td>
</tr>
<tr>
<td>Moderate</td>
<td></td>
</tr>
<tr>
<td>Major</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Protection criteria for *Availability*

<table>
<thead>
<tr>
<th>Consequence</th>
<th>Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unlikely</td>
</tr>
<tr>
<td>Minor</td>
<td></td>
</tr>
<tr>
<td>Moderate</td>
<td></td>
</tr>
<tr>
<td>Major</td>
<td></td>
</tr>
</tbody>
</table>

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
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.

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](image)

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](image)

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 point through which other interfaces may employ the services of the interface. For example, the required interface Channel indicated by the

![Figure 14: Interface dependencies](image)
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 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: 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 UserMgr 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.

Figure 15: Asset identification for the Chat interface

Figure 16: Asset identification for the FileTransfer and UserMgr interfaces

Figure 17: Asset identification for the Channel interface
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 misused 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.

In addition to defining semantics for existing UML operators, STAIRS also in-
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 fulfills 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 `⟨!s, ?s, !r, ?r⟩` where `?` denotes consumption and `!` denotes transmission of a message.

Formally we let `H` denote the set of all well-formed traces over the set of events `E`. A trace is well-formed if, for each message, the send event is ordered before the corresponding consume 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 `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 `H \ (p, n)`. The inconclusive traces result from the incompleteness of interactions, representing traces that are not described as positive or negative by

---

Figure 18: Example of a sequence diagram

```
+sd Send|
|Chat : |
|id:Channel |
|send(id, msg) ➔ receive(msg)
```

A
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 \( H \setminus n \) 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 \( H \setminus n \) 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 \( p\)-STAIRS 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 \textit{send} and \textit{login} operations of the Chat interface, respectively. When a user wants to chat she invokes the \textit{send} operation of the Chat interface with the ID of her buddy and message as parameters. The Chat interface then calls the operation \textit{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 \textit{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 \texttt{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 \textit{ok(id)} or \textit{fail(id)} messages as alternative replies to the call \textit{validate(id,pwd)}. Instead we specify these as two separate invocations of an \textit{ok} and a \textit{fail} operation. In reality the interface invoking the \textit{ok} or \textit{fail} operations will be the same as the one who consumed the \textit{validate} operation. Due to our convention that only the specified interface can transmit messages, however, the transmitter of the \textit{ok} and \textit{fail} messages are not included in the sequence diagram.

Example 10 (UserMgr interactions) The UserMgr interface handles user information. When it receives the message \textit{validate(id, pwd)}, it should either send the message \textit{ok(id)} or the message \textit{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 \textit{ok(id)} if the password is correct and \textit{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.

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

Example 11 (FileTransfer interactions) Figure 22 specifies the \textit{sendFile} operation of the FileTransfer interface. If a user wants to send a music file to one of her buddies she calls the operation \textit{sendFile} of the FileTransfer interface, with the buddy ID and the music file as parameters. The FileTransfer interface must call an operation \textit{receiveFile} of a Channel interface with the required buddy id, in order to implement this operation.
Figure 21: The validate operation of the UserMgr interface

Figure 22: The sendFile operation of the FileTransfer interface

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.

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 vulnerabilities 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.

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 initi-
A leads-to relation originates in a threat scenario or an incident. It originates in 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 crafted 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.

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.

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 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) database 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.

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 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 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, i 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](image)

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.

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.

<table>
<thead>
<tr>
<th>Source scenario</th>
<th>Leads-to</th>
<th>Target scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨0.25, 0.50⟩</td>
<td>⟨0.00, 0.25⟩</td>
<td>⟨0.25 × 0.00, 0.50 × 0.25⟩ = ⟨0.00, 0.13⟩</td>
</tr>
</tbody>
</table>

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.

![Diagram showing likelihood estimation for login related threats](image)

Figure 33: Likelihood estimation for login related threats

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 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 $Y$.

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 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.

\hfill \Box

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

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.

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 workflow. 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.

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.

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.

![Figure 40: Consequence estimation for modified query success](image)

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

![Figure 41: Consequence estimation for reception of malicious code](image)

![Figure 42: Consequence estimation for spimming and flooding](image)

### 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 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 \( (0.50, 0.70) \) 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 \texttt{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 \texttt{palt} or an \texttt{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 \texttt{expalt} operator in the corresponding sequence diagram. The first operand of the \texttt{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 \texttt{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.

![Sequence Diagram](image)

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 (0.50, 0.70).

41
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 \( [0.50, 0.7] \) 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 \( [0.50, 0.7] \) 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 \( [0.50, 0.7] \) 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 \( [0, 0.28] \) 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: \( [0, 0.4] \times [0.5, 0.7] = [0 \times 0.50, 0.4 \times 0.7] = [0, 0.28] \)

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 \( [0, 0.2] \), 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 \texttt{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 modified query. This alternative is therefore represented by the generic message \textit{otherwise}, sent from the thief to itself.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure45}
\caption{Thief attempts modified query}
\end{figure}

\begin{example}[\textit{Channel risk interactions}]
The sequence diagram in Figure 46 specifies the sequence of events involved in the target in Figure 41.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Figure46}
\caption{The Channel interface receives a crafted mp3 file}
\end{figure}

The traces obtained by the first operand of the \texttt{expalt} operator represent the case that constitutes the threat scenario: that a crafted music file is sent to the \texttt{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.

### 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

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.

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](image)

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 $[IM\_Login]$ denote the sequence diagram to the left in Figure 47. All events in $[IM\_Login]$ 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 $[IM\_Login]$.

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 representation of \textit{IM\_Login} is the same as the denotational representation of \textit{User login} with \(j_1\) and \(j_2\) substituted by \(i_1\) and \(i_2\). The denotation of the sequence diagram \textit{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 \textit{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 \textit{Chat} interface and the \textit{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 \( D \):

\[
v \rightarrow v' \in D \Rightarrow v \in D \land v' \in D
\]

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 \( A \triangleright T \) 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 \( A \triangleright T \) to be well-formed we have the following requirements:

\[
\begin{align*}
v \rightarrow v' & \in A \land v' \in A \cup T \tag{2} \\
v \rightarrow v' & \in T \land v \in A \cup T \tag{3} \\
v \rightarrow v' & \in A \cup T \land v \in A \cup T \tag{4} \\
A \cap T &= \emptyset \tag{5}
\end{align*}
\]

Note that (4) is implied by (2) and (3). This means that if \( A \triangleright T \) 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 \( v \) 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: \textit{Threat} \( t_j \) \textit{initiates} \( s_j \) \textit{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 \( v \) 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: \textit{Incident} \( u_j \) \textit{harms asset} \( a \) \textit{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).

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:

\[
i(D, D') \overset{\text{def}}{=} \{ v \in D \mid \exists v' \in D' : v \rightarrow v' \in D \cup D' \} \cup \{ v \rightarrow v' \in D \mid v' \in D' \}
\]

Let for example

\[
\begin{align*}
A'_1 &= \{ \text{\textit{Tmq}} \} \\
T'_1 &= \{ \text{\textit{Tmq}} \xrightarrow{\text{Certain}} \text{\textit{Ma}}, \text{\textit{Ma(X)}}, \text{\textit{Ma}} \rightarrow \text{\textit{MahC}}, \text{\textit{MahC}} \} \\
A''_1 &= \{ \text{\textit{Ms(Y)}} \} \\
T''_1 &= \{ \text{\textit{Ms}} \xrightarrow{\text{Certain}} \text{\textit{Ul}}, \text{\textit{Ul(Y)}}, \text{\textit{Ul}} \xrightarrow{\text{UlhC}}, \text{\textit{UlhC}} \}
\end{align*}
\]
Figure 49: Representing the threat diagram in Figure 38 as a dependent risk graph

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 $i(T'_1, T''_1) = \emptyset$.

$T_{mq}$ = Thief initiates Thief attempts modified query with likelihood $X$.
$M_a$ = Modified query attempt
$M_{ahC}$ = Modified query attempt harms Chat UserId with consequence Minor
$M_s$ = Modified query successful
$U_l$ = Unauthorised login
$U_{lhC}$ = 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 $A \succ T$ 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 $A \succ T$ is defined by:

$$\left[ A \succ T \right] \overset{\text{def}}{=} \forall T' \subseteq T : \left[ i(A \cup T \setminus T', T'') \right] \Rightarrow \left[ T' \right] \quad (7)$$

Note that if the assumption of a dependent graph $A \succ T$ is empty (i.e. $A = \emptyset$) it means that we have the graph $T$, that is the semantics of $\emptyset \succ T$ 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 $D \nmid D'$ means that $D'$ does not depend on any vertex or relation in $D$. 

A
This means that $D$ does not have any interface towards $D'$ and that $D$ and $D'$ have no common elements:

**Definition 1 (Independence)**

$$D \nless D' \Leftrightarrow D \cap D' = \emptyset \land i(D, D') = \emptyset$$

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

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)**

$$A \ntriangleright T \cup T' \quad T' \nless T \quad \overline{A \ntriangleright T}$$

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

**Rule 3 (Assumption simplification)**

$$A \cup A' \ntriangleright T \quad A \nless A' \cup T \quad \overline{A' \ntriangleright T}$$

**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 $A'_1 \cup A''_1 \ntriangleright T'_1 \cup T''_1$ represent the diagram in Figure 49. We assume that the dependent diagram in Figure 49 is correct, that is: we assume the validity of

$$A'_1 \cup A''_1 \ntriangleright T'_1 \cup T''_1$$

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

$$A'_1 \cup A''_1 \ntriangleright T'_1 \quad (8)$$

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

$$A'_1 \ntriangleright T'_1 \quad (9)$$

Using the same procedure we can also deduce the validity of

$$A''_1 \ntriangleright T''_1 \quad (10)$$

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 $A_1 \ntriangleright T_1$ and $A_2 \ntriangleright T_2$ 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 $A_1 \cup A_2 \cup \{v\} \ntriangleright \{v \rightarrow v', v'\} \cup T_1 \cup T_2.$
Rule 4 (Sequential composition)

\[ A_1 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1 \quad A_2 \cup \{v'\} \triangleright T_2 \quad (A_1 \cup T_1) \cap (A_2 \cup T_2) = \emptyset \]

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.

![Modified query attempt](X) \[\text{Likely} \times [X]\]

By (9), (13) and the fact that \((A_1' \setminus \{Tmq\} \cup T_1' \setminus \{M \cdot ([X])\}) \cap (A_2 \setminus \{M \cdot ([X])\} \cup T_2 \cup T'''_1) = \emptyset\), 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 User-Mgr 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 abso-
lute 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 we could say, given that the likelihood of the assumed
threat scenario Thief attempts modified query is Likely, that the likelihood of the inci-
dent 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 compo-
nent 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 dia-
gram 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.

\begin{tabular}{|c|c|}
\hline
Consequence & Likelihood \\
\hline
Unlikely & Possible & Likely & Almost certain & Certain \\
\hline
Assuming $Thief$ initiates $Thief$ attempts modified query with likelihood $Likely$ & \\
\hline
Minor & $UI1$ & & & \\
\hline
Moderate & $UI2$ & & & \\
\hline
Major & $UI3$ & & & \\
\hline
\end{tabular}

Table 11: Risk evaluation for $User Id$

\begin{tabular}{|c|c|}
\hline
Consequence & Likelihood \\
\hline
Unlikely & Possible & Likely & Almost certain & Certain \\
\hline
Assuming $Impersonator$ initiates $Impersonator$ poses as buddy with likelihood $Likely$ & \\
\hline
Minor & $UM1$ & & & \\
\hline
Moderate & & & & \\
\hline
Major & & & & \\
\hline
\end{tabular}

Table 12: Risk evaluation for $Message$

\begin{tabular}{|c|c|}
\hline
Consequence & Likelihood \\
\hline
Unlikely & Possible & Likely & Almost certain & Certain \\
\hline
Assuming $Spimmer$ initiates $Spimming$ with likelihood $Likely$ and $Adversary$ initiates $Flooding$ attack with likelihood $Likely$ & \\
\hline
Minor & $UE1$ & $UE2$ & & \\
\hline
Moderate & & & & \\
\hline
Major & & & & \\
\hline
\end{tabular}

Table 13: Risk evaluation for $Availability$

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 component specification in this example.
Table 14: Risk evaluation for Media file
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 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,
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 misuse 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 facilitates 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 Isabelle³.

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. 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

³http://isabelle.in.tum.de/
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


59


**A Proofs**

**Theorem 1**

\[(A_1 \cup T_1) \cap (A_2 \cup T_2) = \emptyset \land \forall v \notin A_1 \land v' \notin A_2 \land (\{v \rightarrow v', v'\} \cap T_1 = \emptyset)\]

\[\land \left[ A_1 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1 \right] \land \left[ A_2 \cup \{v'\} \triangleright T_2 \right] \Rightarrow\]

\[\left[ A_1 \cup A_2 \cup \{v\} \triangleright \{v \rightarrow v', v'\} \cup T_1 \cup T_2 \right]\]

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:**

(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: \[ A_1 \cup A_2 \cup \{ v \} \triangleright \langle v \rightarrow v', v' \rangle \cup T_1 \cup T_2 \]

(2.1) \( \forall T \subseteq \{ v \rightarrow v', v' \} \cup T_1 \cup T_2 \):

\[ i(A_1 \cup A_2 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1 \cup T_2) \setminus (T \cup T) \Rightarrow \top \]

(3.1) \textbf{ASSUME:} \( T \subseteq \{ v \rightarrow v', v' \} \cup T_1 \cup T_2 \)

\[ i(A_1 \cup A_2 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1 \cup T_2) \setminus (T \cup T) \Rightarrow \top \]

(4.1) \textbf{ASSUME:} \[ i(A_1 \cup A_2 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1 \cup T_2) \setminus (T \cup T) \]

\textbf{PROOF:} \[ T \]

(5.1) \textbf{LET:} \( T = T' \cup T'' \)

\[ (T' \subseteq \{ v \rightarrow v', v' \} \cup T_1) \land (T'' \subseteq T_2) \]

\textbf{PROOF:} By (3.1).

(5.2) \[ i(A_1 \cup A_2 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1 \cup T_2) \setminus (T' \cup T'') \]

\textbf{PROOF:} By assumption (4.1), (5.1) 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 \rightarrow v', v' \} \cup T_1 \setminus T', T' \) \Rightarrow \top \]

\textbf{PROOF:} By assumption (1.1), (5.1) and definition (7).

(7.2) \[ i(A_1 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1 \setminus T', T' \]

(8.1) \[ i(A_1 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1 \setminus T', T' \]

\textbf{PROOF:} By assumptions (1.1.1, (1.1.2, (1.1.3, (1.1.4 and (5.1).

(9.2) \textbf{ASSUME:} \( V \in i(A_1 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1) \setminus T', T' \)

\textbf{PROOF:} \( V \in i(A_1 \cup A_2 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1 \cup T_2) \setminus (T' \cup T'' \cup T'') \)

(10.1) \textbf{CASE:} \( V = v_1 \), that is \( V \) is a vertex.

(11.1) \( v_1 \in A_1 \cup A_2 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1 \cup T_2 \setminus (T' \cup T'') \)

(12.1) \( v_1 \in A_1 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1 \setminus T' \)

\textbf{PROOF:} By assumption (9.2), assumption (10.1) and definition (6).

(12.2) \( v_1 \in (A_1 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1) \setminus T' \)

\textbf{PROOF:} By (12) and elementary set theory.

(12.3) \textbf{Q.E.D.}

\textbf{PROOF:} By (9.1), (12) and the rule of replacement (Troelstra and Schwichtenberg, 2000).

(11.2) \( \exists v_2 \in T' \cup T'' : \{ v_1 \rightarrow v_2 \} \in (A_1 \cup A_2 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1 \cup T_2) \setminus (T' \cup T'') \cup (T' \cup T'') \]

(12.1) \( \exists v_2 \in T' : \{ v_1 \rightarrow v_2 \} \in (A_1 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1) \setminus T' \)

\textbf{PROOF:} By assumption (9.2), assumption (10.1) and definition (6).

(12.2) \textbf{LET:} \( v_2 \in T' \) such that \( v_1 \rightarrow v_2 \in (A_1 \cup \{ v \} \cup \{ v \rightarrow v', v' \} \cup T_1) \setminus T' \cup T' \)

\textbf{PROOF:} By (12)
Proof: By (12)2 and elementary set theory.

\((12)4. \ v_1 \rightarrow v_2 \in A_1 \cup A_2 \cup \{v\} \cup\)
\(\{(v \rightarrow v', v') \cup T_1 \cup T_2) \setminus (T' \cup T'')\)

\((13)1. \ v_1 \rightarrow v_2 \in (A_1 \cup \{v\} \cup \{v \rightarrow v', v'\} \cup T_1) \setminus T') \cup T' \cup\)
\((A_2 \cup T_2 \setminus T'')\)

Proof: By (12)2 and elementary set theory

\((13)2. \ \text{Q.E.D.})\)

Proof: By (9)1, (13)1 and the rule of replacement (Troelstra and Schwichtenberg, 2000).

\((12)5. \ \text{Q.E.D.})\)

Proof: By (12)3, (12)4 and \(\exists\) introduction.

\((11)3. \ \text{Q.E.D.})\)

Proof: By (11)1, (11)2 and definition (6).

\((10)2. \ \text{Case: } V = v_1 \rightarrow v_2, \text{ that is } V \text{ is a relation.})\)

\((11)1. \ v_1 \rightarrow v_2 \in A_1 \cup \{v\} \cup \{(v \rightarrow v', v') \cup T_1) \setminus T' \wedge v_2 \in T'\)

Proof: By assumption (9)2, assumption (10)2 and definition (6).

\((11)2. \ v_1 \rightarrow v_2 \in A_1 \cup A_2 \cup \{v\} \cup \{(v \rightarrow v', v') \cup T_1 \cup T_2) \setminus (T' \cup T'')\)

\((12)1. \ v_1 \rightarrow v_2 \in (A_1 \cup \{v\} \cup \{v \rightarrow v', v'\} \cup T_1) \setminus T') \cup (A_2 \cup T_2 \setminus T'')\)

Proof: By (11)1, \(\land\) elimination and elementary set theory.

\((12)2. \ \text{Q.E.D.})\)

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, \(\land\) elimination and elementary set theory.

\((11)4. \ \text{Q.E.D.})\)

Proof: By (11)2, (11)3 and definition (6).

\((10)3. \ \text{Q.E.D.})\)

Proof: The cases (10)1 and (10)2 are exhaustive.

\((9)3. \ \text{Q.E.D.})\)

Proof: \(\subseteq\)-rule.

\((8)2. \ \text{Q.E.D.})\)

Proof: By (5)2 and (8)1.

\((7)3. \ \text{Q.E.D.})\)

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).

\((7)2. \ [\ i(A_2 \cup \{v'\} \cup T_2 \setminus T'', T''') \ ]\)

\((8)1. \ [\ i(A_1 \cup A_2 \cup \{v\} \cup \{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'') \cup (T'' \cup T''') \ ] \Rightarrow [\ T' \ ]\)

Proof: By (5)2, (6)1 and \(\land\) introduction.

\((8)2. \ i(A_2 \cup \{v'\} \cup T_2 \setminus T'', T'') \subseteq\)
\(i(A_1 \cup A_2 \cup \{v\} \cup \{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'', T'' \cup T''') \cup T'\)

\((9)1. \ \text{Assume: } V \in i(A_2 \cup \{v'\} \cup T_2 \setminus T'', T'')\)

Prove: \(V \in i(A_1 \cup A_2 \cup \{v\} \cup (\{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'', T'' \cup T''') \cup T'\)

\((10)1. \ \text{Assume: } V \notin i(A_1 \cup A_2 \cup \{v\} \cup (\{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'', T'' \cup T''') \cup T'\)
\[(\{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'') \cup T'\]

**Prove:** $\perp$

\[\langle 11 \rangle 1. \quad V \notin (A_1 \cup A_2 \cup \{v\} \cup \{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'') \cup T'\]

**Proof:** By assumption \langle 10 \rangle 1, definition (6) and elementary set theory.

\[\langle 11 \rangle 2. \quad V \in (A_1 \cup A_2 \cup \{v\} \cup \{v \rightarrow v', v'\} \cup T_1 \cup T_2) \setminus (T' \cup T'') \cup T'\]

**Proof:** By assumption \langle 9 \rangle 1 and definition (6).

\[\langle 12 \rangle 2. \quad v' \notin T_2\]

**Proof:** By assumption \langle 12 \rangle 1, assumption \langle 12 \rangle 3 and elementary set theory.

\[\langle 12 \rangle 3. \quad \text{Case: } V \in A_2\]

\[\langle 13 \rangle 1. \quad \text{Q.E.D.}\]

**Proof:** By \langle 12 \rangle 1, assumption \langle 12 \rangle 3 and elementary set theory.

\[\langle 12 \rangle 4. \quad \text{Case: } V = v'\]

\[\langle 13 \rangle 1. \quad \text{Case: } v' \in T'\]

\[\langle 14 \rangle 1. \quad \text{Q.E.D.}\]

**Proof:** By assumption \langle 13 \rangle 1 and elementary set theory.

\[\langle 13 \rangle 2. \quad \text{Case: } v' \notin T'\]

\[\langle 14 \rangle 1. \quad v' \notin T''\]

**Proof:** By \langle 12 \rangle 2, \langle 5 \rangle 1 and elementary set theory.

\[\langle 14 \rangle 2. \quad \text{Q.E.D.}\]

**Proof:** By assumption \langle 13 \rangle 2, \langle 14 \rangle 1, assumption \langle 12 \rangle 4 and elementary set theory.

\[\langle 13 \rangle 3. \quad \text{Q.E.D.}\]

**Proof:** The cases \langle 13 \rangle 1 and \langle 13 \rangle 2 are exhaustive.

\[\langle 12 \rangle 5. \quad \text{Case: } V \in T_2 \land V \notin T''\]

\[\langle 13 \rangle 1. \quad T_2 \cap T' = \emptyset\]

\[\langle 14 \rangle 1. \quad v \rightarrow v' \notin T_2\]

**Proof:** By \langle 12 \rangle 2, the assumption that $A_2 \cup \{v'\} \triangleright T_2$ is well-formed and requirement (3).

\[\langle 14 \rangle 2. \quad \text{Q.E.D.}\]

**Proof:** By assumption \langle 1 \rangle 1.1, \langle 12 \rangle 2, \langle 14 \rangle 1 and elementary set theory.

\[\langle 13 \rangle 2. \quad \text{Q.E.D.}\]

**Proof:** By assumption \langle 12 \rangle 5, \langle 13 \rangle 1 and elementary set theory.

\[\langle 12 \rangle 6. \quad \text{Q.E.D.}\]

**Proof:** The cases \langle 12 \rangle 3, \langle 12 \rangle 4 and \langle 12 \rangle 5 are exhaustive.

\[\langle 11 \rangle 3. \quad \text{Q.E.D.}\]

**Proof:** By \langle 11 \rangle 1, \langle 11 \rangle 2 and $\perp$ introduction.

\[\langle 10 \rangle 2. \quad \text{Q.E.D.}\]

**Proof:** Proof by contradiction.
Proof: \subseteq rule.

\langle 8 \rangle 3. Q.E.D.

Proof: By \langle 8 \rangle 1 and \langle 8 \rangle 2.

\langle 7 \rangle 3. Q.E.D.

Proof: By \langle 7 \rangle 1, \langle 7 \rangle 2 and \Rightarrow elimination.

\langle 6 \rangle 3. Q.E.D.

Proof: By \langle 6 \rangle 1 and \langle 6 \rangle 2.

\langle 5 \rangle 4. Q.E.D.

Proof: By \langle 5 \rangle 1, \langle 5 \rangle 3 and the rule of replacement.

\langle 4 \rangle 2. Q.E.D.

Proof: \Rightarrow introduction

\langle 3 \rangle 2. Q.E.D.

Proof: \forall introduction.

\langle 2 \rangle 2. Q.E.D.

Proof: By \langle 2 \rangle 1 and definition (7).

\langle 1 \rangle 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.
Chapter 10

Paper B: The dependent CORAS language
The Dependent CORAS Language*

Gyrd Brændeland  Mass Soldal Lund  Bjørnar Solhaug
Ketil Stølen

Abstract

The basic CORAS language offers no support for the explicit documentation of assumptions. This may be unfortunate since the validity of the diagrams we make during a risk analysis, and therefore the very validity of the risk analysis results, may depend on assumptions. This chapter presents dependent CORAS, which is a language extension to support the documentation of and reasoning about risk analysis assumptions. The reasoning about assumptions and dependencies is supported by four deduction rules.

*This chapter was originally published as a chapter in Model-Driven Risk Analysis: The CORAS Approach, pages 267-279. Springer, 2010 [2]. We have made some small modifications such as including a couple of tables and a figure from the book that was only referred to in the original chapter, in order to make it possible to read the chapter independently of the book.
## Contents

1  Introduction ........................................ 4

2  Modelling Dependencies Using the CORAS Language ................. 5
   2.1  Dependent CORAS Diagrams .................................. 5
   2.2  Representing Assumptions Using Dependent CORAS Diagrams .......... 7
   2.3  How to Schematically Translate Dependent CORAS Diagrams into English Prose .................................................. 9

3  Reasoning and Analysis Using Dependent CORAS Diagrams .............. 11
   3.1  Assumption Independence ..................................... 14
   3.2  Assumption Simplification ..................................... 14
   3.3  Target Simplification ......................................... 15
   3.4  Assumption Consequence ...................................... 17

4  Example Case in Dependent CORAS ................................ 18
   4.1  Creating Dependent Threat Diagrams .......................... 19
   4.2  Combining Dependent Threat Diagrams ........................... 21

5  Summary .............................................. 24
1 Introduction

The environment of the target is the surrounding things of relevance that may affect or interact with the target; in the most general case, the environment is the rest of the world. The explicit documentation of environment assumptions is important because the validity of the diagrams we make during a risk analysis, and thereby the validity of the risk analysis results, may depend on assumptions. If we only operate with informal descriptions of the assumptions, they may easily be forgotten with the effect that the conclusions that we draw from the risk analysis are believed to be more general than they really are.

Assumptions are important in relation to risk analysis for several reasons:

- In a risk analysis, we often make assumptions to simplify or focus the analysis. It may for example be that the customer has great trust in a particular aspect of the target, like the encryption algorithms or the safety system, and we are therefore asked to assume that there will be no incidents originating from that aspect, thus keeping it outside the target of the analysis. Alternatively, it may be that the customer is only worried about risks in relation to the daily management of a business process and we are asked to ignore situations caused by more extreme environmental events like fire or flooding.

- Most artefacts are only expected to work properly when used in their intended environment. There are, for example, all kinds of expectations and requirements to the behaviour of a car when driven on a motorway, but less so when it is dropped from a ship in the middle of the ocean. When doing a risk analysis, it is often useful to make assumptions about the environment in which artefacts of the target are used in order to avoid having to take into consideration risks that are of no real relevance for the target of analysis.

- Environment assumptions are also often useful in order to facilitate reuse of risk analysis results. We may, for example, be hired by a service provider to conduct a risk analysis of its service from the point of view of a potential service consumer. The service provider wants to use the risk analysis to convince potential service consumers that the service is trustworthy. Environment assumptions can then be used to characterise what we may assume about the infrastructure of these potential service consumers. In fact, the correct formulation of these assumptions may be essential for a successful outcome. If the environment assumptions are too weak, it may be that we identify risks that are irrelevant for the practical use of the service and thereby give the impression that the service is less trustworthy than it really is. If the environment assumptions are too strong, it may be that we frighten away potential service consumers because we without any real reason impose assumptions that their infrastructure does not fulfil.

- Explicit documentation of assumptions is also an important means to facilitate reasoning about risk analysis documentation. Assume, for example, that a potential service consumer already has completed a risk analysis with respect to its own infrastructure. When this service consumer is presented with the risk analysis, we have conducted on behalf of the service provider, it should in principle be possible from these two analyses to deduce an overall risk picture for the combined infrastructure. This requires, however, a careful inspection of the assumptions under which these two analyses have been carried out.
The assumptions of a risk analysis are what we take as granted or accept as true, although they actually may not be so. An assumption is obviously not something that comes out of the blue, but is usually something that we have strong evidence for or high confidence in, for example, that there are no malicious insiders within a particular organisation or that passwords fulfilling certain criteria are resistant to brute force attacks. We may furthermore use assumptions as a means to choose the desired or appropriate focus and scope of the analysis. For example, even though there is strong reason to believe that there are no malicious insiders, we may still assume the contrary in order to ensure an acceptable risk level even in case of disloyal servants. Or we may, for example, assume that power failure, fire and flood do not occur while knowing that this is not so, in order to understand the general risk picture and risk level when such incidents are ignored.

In this chapter, we extend the CORAS language to facilitate the documentation of and reasoning about risk analysis assumptions. We refer to this extended language as dependent CORAS since we use it to document dependencies on assumptions. The assumptions are mainly of relevance in relation to threat scenarios and unwanted incidents that document the potential impact of events. Dependent CORAS is therefore only concerned with the two kinds of CORAS diagrams that can express these, namely threat diagrams and treatment diagrams.

2 Modelling Dependencies Using the CORAS Language

In this section, we first introduce the dependent CORAS language and explain how we construct dependent CORAS diagrams. Thereafter, we explain how such diagrams should be interpreted by describing how to unambiguously translate any dependent CORAS diagram into English prose.

2.1 Dependent CORAS Diagrams

Figure 1 exemplifies a dependent treatment diagram. The only thing that is really new in this diagram is the border line separating the target from the assumptions. Everything inside the border line belongs to the target; everything on the border line, like the threat scenario Low hydro availability, and every relation crossing the border line, like the leads-to relation from Fire to Grid overload, also belong to the target. The remaining elements, that is everything completely outside the border line like the high-level threat scenario Fire, the threat Lack of rain, and the initiates relation from Lack of rain to Low hydro availability, are the assumptions that we make for the sake of the analysis.

The only difference between a dependent diagram on the one hand and an ordinary or high-level threat diagram or treatment diagram as defined by Soldal Lund et al. [2] on the other hand is the border line. Hence, if we remove the border line, the diagrams are required to be syntactically correct according to the definitions given in the book Model-Driven Risk Analysis: The CORAS Approach by Soldal Lund et al. [2].

With respect to the border line, we impose the following constraints for both kinds of dependent CORAS diagrams:

- Relations cannot cross the border line from the inside to the outside.
- Assets can occur only on the inside of the border line.
High load on transmission corridor

Lack of rain

Low hydro availability

High import of power

Assumption: High import of power critical

Low hydro availability

Increase power self-sufficiency rate

Grid overload [1:5 years]

Power production

Border line between target and assumptions

Assumption

Fire

[0.0]

Figure 1: Dependent treatment diagram
- Impacts relations can occur only on the inside of the border line.

- Threats cannot occur on the border line. They may, however, occur both on the inside and on the outside of the border line.

- Initiates relations cannot cross the border line. This means that initiates relations are either fully inside or fully outside the border line.

From the above, it follows that threat scenarios as well as unwanted incidents may occur on the border line. They may also occur on the outside and on the inside of the border line. Leads-to relations are the only relations that may cross the border line. They may also occur on the outside and on the inside of the border line. There are no special restrictions on the treats relation. Whether conditional likelihoods on leads-to relations that cross the border are placed inside or outside the border line is furthermore irrelevant. In Fig. 1, for example, the conditional likelihood on the leads-to relation from Fire to Grid overload is incidentally placed within the border line.

2.2 Representing Assumptions Using Dependent CORAS Diagrams

In the following, we present some small, illustrative examples in order to explain how assumptions can be represented and dealt with using dependent CORAS. The examples correspond to the motivation presented in the introduction of this chapter.

The dependent diagram of Fig. 1 exemplifies how we can explicitly utilise assumptions to simplify and focus a risk analysis. For example, by assuming that fires do not occur, and hence assigning the probability 0.0 to this threat scenario, this potential source of risk is ignored in the analysis. This assumption is of course generally not true, but if the customer of the risk analysis in question is not responsible for maintaining power supply and emergency planning in case of disasters such as fire, the assumption makes sense; given the assumption, the customer will get an evaluation of the risks that are within the customer's area of responsibility. If, on the other hand, the customer is required to maintain an acceptable risk level even when fires are considered, this particular assumption must be assigned a realistic likelihood estimate. In any case, dependent diagrams allow the explicit documentation of such assumptions, and thereby also the documentation of the assumptions under which the results of the risk analysis are valid.

Providers of software and services commonly specify hardware (HW) requirements such as processor (CPU) power, memory size and hard drive size. In order to guarantee, for example, a certain level of software dependability and service availability, the provider will then typically assume that the HW of the end-users fulfils the requirements.

The dependent diagram of Fig. 2 exemplifies a fragment of a risk analysis conducted by a service provider in order to ensure that the risk of loss of service availability for the end-users is acceptable. Since the service is expected to work properly only when the network connection is up and when the end-user's HW fulfils the requirements to CPU power, the fulfilment of these requirements are explicitly specified as assumptions. In particular, the likelihoods for the threats System failure and End-user's HW to initiate, respectively, the threat scenarios Loss of network connection and Data processing fails are set to 0.0. These likelihoods are assumptions that may not hold, but by making such assumptions the service provider can guarantee a certain level of service availability provided that the end-users deploy the service in the intended environment.
A risk analysis with explicit assumptions as illustrated in Fig. 2 can furthermore conveniently be reused for different purposes and in different analyses. For example, if different end-users have different needs, the service provisioning can be divided into different categories such as standard version and light version. If the different categories have different HW requirements, the analysis can be conducted separately for each category, and thereby for each kind of end-user. The dependent diagram of Fig. 2 could, for example, be reused in a setting in which availability is less critical for the end-user and where the HW requirements therefore can be eased. Such a change in the assumptions could be the replacement of the likelihood 0.0 on the initiates relation from the threat End-user’s HW by the likelihood 0.2. The likelihood of the unwanted incident must then be updated to reflect the new assumption.

The diagram of Fig. 3 illustrates the risk analysis of the same service provisioning. The difference from Fig. 2 is that the analysis is conducted for the service consumer, that is for the end-user, instead of for the service provider. From the likelihood 0.0 of the initiates relation pointing to the threat scenario Data processing fails of the assumptions, we see that also this analysis takes into account the HW requirements for deploying the service. Again, it may not be true that the end-user will always deploy the service using HW with sufficient CPU power. However, by making this assumption the analysis yields the risk level under this assumption.

The dependent diagrams of Figs. 2 and 3 furthermore show that assumptions in one analysis can be part of the target in another analysis, and vice versa. The assumption Loss of network connection, for example, is in Fig. 2 an assumption about the end-user made by the service provider. In Fig. 3, the same threat scenario is part of the target. The service consumer similarly makes the assumption Developer causes flaw in SW in Fig. 3, which is part of the target in Fig. 2.

The two separate analyses with different assumptions also show how separate analysis results can be combined into an overall risk picture by combining diagrams. For example, if the service consumer have conducted a risk analysis that is partly documented by the
dependent diagram of Fig. 3 and is then presented the analysis results of the service provider, as partly documented by the dependent diagram of Fig. 2, she can combine the diagrams into the dependent diagram shown in Fig. 4.

The initial assumption Developer causes flaw in SW in Fig. 3 must then be assigned the likelihood value 0.05 as estimated in the target of Fig. 2. This particular threat scenario then no longer represents an assumption in the combined picture. Similarly, the assumption Loss of network connection in Fig. 2 is assigned the likelihood value 0.1. Once the assumptions in the one diagram is harmonised with the target in the other diagram and vice versa, the diagrams can be combined into one dependent diagram. As shown in the resulting diagram of Fig. 4, the elements that are assumptions in both diagrams, as the threat scenario Data processing fails, remain assumptions in the combined diagram.

2.3 How to Schematically Translate Dependent CORAS Diagrams into English Prose

The CORAS risk modelling language has a formal syntax and a structured semantics, defined by a formal translation of any CORAS diagram into a paragraph in English [1, 2]. In the following, we describe the semantics of dependent CORAS in the same way as the semantics of basic CORAS diagrams and high-level CORAS diagrams is presented in Model-Driven Risk Analysis [2]. The reader is referred to Soldal Lund et al. [2] for a more formal description.

Using a shorthand notation, we denote any dependent CORAS diagram by $A \Rightarrow T$. $T$ denotes the part of the diagram that belongs to the target, which is everything inside the border line, every leads-to relation crossing the border line, as well as every threat scenario and unwanted incident on the border line. $A$ denotes the assumptions, which is the rest. In other words, $A$ denotes everything outside the border line including relations pointing at threat scenarios or unwanted incidents on the border line.

Both $T$ and $A$ are fragments of CORAS threat diagrams or treatment diagrams, possibly
The suffix “To the extent there are explicit dependencies” is significant. It implies that if there are no relations explicitly connecting \( A \) to \( T \), we do not gain anything from assuming \( A \).

**Example 1** Consider the dependent threat diagram in Fig. 5. In order to translate the diagram into English, we first determine for each element whether it is an assumption or belongs to the target. Since all elements inside or on the border line, as well as all relations that cross the border line, belong to the target, it is only the threat scenario *Fire* that constitutes an assumption. The rest belongs to the target.

The procedure for translate threat scenarios into English described in *Model-Driven Risk Analysis* [2] yields the following translation of the assumption:

Threat scenario *Fire* occurs with likelihood \([0.0, 0.1]\).

The part of the diagram that belongs to the target makes a threat diagram that is translated into the following:
Unwanted incident *Grid overload* occurs with likelihood $1:5\text{ years}$.

*Power production* is a direct asset.

*Fire* leads to *Grid overload* with conditional likelihood $1.0$.

*Grid overload* impacts *Power production* with consequence *critical*.

Using the above schematic procedure for translating the full dependent diagram of Fig. 5, we then get the following paragraph in English:

Unwanted incident *Grid overload* occurs with likelihood $1:5\text{ years}$.

*Power production* is a direct asset.

*Fire* leads to *Grid overload* with conditional likelihood $1.0$.

*Grid overload* impacts *Power production* with consequence *critical*.

Assuming:

Threat scenario *Fire* occurs with likelihood $[0.0, 0.1]$.

To the extent there are explicit dependencies.

### 3 Reasoning and Analysis Using Dependent CORAS Diagrams

In order to facilitate reasoning about and analysis of assumptions, we introduce four deduction rules. We start by defining some central concepts and helpful notation. For arbitrary diagrams, diagram elements and diagram relations of relevance for this chapter, we use the syntactic variables shown in Table 1.

**Example 2** The dependent diagram of Fig. 6 will serve as an example to illustrate the new concepts. The diagram is based on the AutoParts example described in *Model-Driven Risk Analysis* [2] and shows the explicit environmental assumption of flood and how it may lead to loss of availability of the online store.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Diagram construct</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>Threat diagram or treatment diagram, possibly high-level</td>
</tr>
<tr>
<td>$T$</td>
<td>Fragment of threat diagram or treatment diagram, possibly high-level, that constitutes the target part of a dependent diagram</td>
</tr>
<tr>
<td>$A$</td>
<td>Fragment of threat diagram or treatment diagram, possibly high-level, that constitutes the assumption part of a dependent diagram</td>
</tr>
<tr>
<td>$A \triangleright T$</td>
<td>Dependent diagram of assumptions $A$ and target $T$</td>
</tr>
<tr>
<td>$e$</td>
<td>Diagram element of threat, threat scenario, unwanted incident or asset</td>
</tr>
<tr>
<td>$e_1 \rightarrow e_2$</td>
<td>Initiates relation, leads-to relation or impacts relation from $e_1$ to $e_2$</td>
</tr>
<tr>
<td>$P$</td>
<td>Diagram path of connected relations</td>
</tr>
</tbody>
</table>

Table 1: Naming conventions

Figure 6: Diagram paths and dependencies
For sake of readability, we use the following shorthand notations for the various elements of the diagram:

\[
\begin{align*}
at & = \text{Attacker} \\
aa & = \text{Attacker initiates DoS attack} \\
oy & = \text{Online store down 1 minute to 1 day} \\
f1 & = \text{Flood} \\
oe & = \text{Online store down 1 day or more} \\
os & = \text{Online store}
\end{align*}
\]

When we are reasoning about assumptions we need to identify dependencies of target elements upon the assumptions. These dependencies are explicitly represented by diagram paths from the assumptions to the target. A path \( P \) is a set of connected, consecutive relations:

\[
P = \{ e_1 \rightarrow e_2, e_2 \rightarrow e_3, \ldots, e_{n-1} \rightarrow e_n \}
\]

We write \( e_1 \rightarrow P \) and \( P \rightarrow e_n \) to state that \( P \) is a path commencing in element \( e_1 \) and ending in element \( e_n \), respectively. We use \( e \in D \) to denote that \( e \) is an element in diagram \( D \), and \( P \subseteq D \) to state that \( P \) is a path in diagram \( D \).

**Example 3** When we are identifying dependencies, we can ignore the vulnerabilities. The diagram of Fig. 6 can therefore be represented by the set \( es \) of elements and the set \( er \) of relations as follows:

\[
\begin{align*}
es &= \{ \text{at, aa, oy, fl, oe, os} \} \\
er &= \{ \text{at} \rightarrow aa, \text{aa} \rightarrow oy, \text{oy} \rightarrow os, \text{fl} \rightarrow oe, oe \rightarrow os \}
\end{align*}
\]

We let the pair \( D = (es, rs) \) of elements \( es \) and relations \( rs \) denote the diagram. A path is a set of connected, consecutive relations, so the set \( \{ \text{at} \rightarrow aa, \text{aa} \rightarrow oy \} \) is an example of a path in \( D \).

Let \( D \) be a CORAS diagram, \( D' \) be a fragment of \( D \), and \( D'' \) the result of removing this fragment from \( D \). An element \( e \in D'' \) is independent of \( D' \) if for any path \( P \subseteq D \) and element \( e' \in D \)

\[
e' \rightarrow P \land P \rightarrow e \implies e' \notin D'
\]

Hence, \( e \) is independent of \( D' \) if there are no paths to \( e \) in \( D \) commencing from an element \( e' \) in \( D' \).

We say that \( D'' \) is independent of \( D' \) if each element in \( D'' \) is independent of \( D' \), in which case we write \( D' \upharpoonright D'' \). By \( D' \cup D'' \), we denote the diagram fragment resulting from conjoining \( D' \) and \( D'' \). Hence, \( D = D' \cup D'' \).

**Example 4** Let \( T \) denote the fragment of the dependent diagram \( D \) of Fig. 6 that consists of all the elements and relations inside the border, as well as the relation crossing the border. Using our shorthand notation, we can represent this fragment by the following pair of elements and relations:

\[
T = (\{ \text{at, aa, oy, oe, os} \}, \{ \text{at} \rightarrow aa, \text{aa} \rightarrow oy, \text{oy} \rightarrow os, \text{fl} \rightarrow oe, oe \rightarrow os \})
\]

Let, furthermore, \( A \) denote the result of removing \( T \) from \( D \). The diagram fragment \( A = (\{ \text{fl} \}, \emptyset) \) then represents the assumptions.

In order to check whether \( T \) is independent of \( A \), we need to check whether each of the elements of \( T \) is independent of \( A \) in \( D \). Clearly, each of the elements \( at, aa \) and \( oy \) of \( T \) is
independent of $A$ since there is no path in the diagram $D$ that leads from an element in $A$ to any of $at$, $aa$, and $oy$. The unwanted incident $oe$, however, is not independent of $A$ since there is a path commencing in $fl$ that ends in $oe$. Hence, the fragment $T$ of $D$ is not independent of $A$.

Notice that when we are reasoning about dependencies in CORAS diagrams we are considering how threat scenarios and unwanted incidents as documented in one part of a diagram depend on threat scenarios and unwanted incidents as documented in another part. Since vulnerabilities are annotations on relations, we do not need to take them into account when identifying and analysing dependencies. In the setting of dependency analysis, we can likewise ignore treatment scenarios and treats relations since they can be understood as mere annotations on the dependency relevant information that is conveyed by a threat diagram. Therefore, as shown by Table 1, we do not include vulnerabilities, treatment scenarios or treats relations in the shorthand notation for this chapter.

Having introduced the notion of dependency, we now turn to the rules for reasoning about dependencies. These rules assume dependent CORAS diagrams $D$ that are partitioned into the fragments $T$ and $A$ for target and assumptions, respectively, such that $T \cup A = D$. The rules are of the following form:

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

We refer to $P_1, \ldots, P_n$ as the premises and to $C$ as the conclusion. The interpretation is that if the premises are valid, so is the conclusion.

### 3.1 Assumption Independence

The following rule states that if we have deduced $T$ assuming $A$, and $T$ is independent of $A$, then we may deduce $T$.

**Rule 1 (Assumption independence)**

$$\frac{A \not\triangleright T \quad A \not\triangleright T}{\triangleright T}$$

From the second premise, it follows that there is no path from $A$ to an element in $T$. Since the first premise states $T$ assuming $A$ to the extent there are explicit dependencies, we may deduce $T$.

**Example 5** The dependent diagram of Fig. 7 is a simple example of assumption independence. If, for example, redundant power supply is ensured within the target, the risk of loss of service availability is independent of the public power supply. Since there is no path from the assumption to the target, the target is independent of this assumption. By Rule 1, the assumption can be removed when we reason about the risks in this particular diagram.

### 3.2 Assumption Simplification

The following rule allows us to remove a fragment of the assumptions that is not connected to the rest.
**Rule 2 (Assumption simplification)**

\[
A \cup A' \triangleright T \quad A \not\Rightarrow A' \cup T
\]

\[
A' \triangleright T
\]

The second premise implies that there are no paths from \(A\) to the rest of the diagram. Hence, the validity of the first premise does not depend upon \(A\) in which case the conclusion is also valid.

**Example 6** The diagram of Fig. 8 exemplifies assumption simplification by the threat scenario *Outage in public power supply*. Since the target as well as the remaining assumptions are independent of this assumption, it can by Rule 2 be removed along with the leads-to relation ending in it when we are reasoning about the dependencies of the target on the assumptions in this diagram.

### 3.3 Target Simplification

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

**Rule 3 (Target simplification)**

\[
A \triangleright T \cup T' \quad T' \not\Rightarrow T
\]

\[
A \triangleright T
\]

The second premise implies that there is no path from \(A\) to \(T\) via \(T'\). Hence, the validity of the first premise implies the validity of the conclusion.

**Example 7** The dependent diagram of Fig. 8 exemplifies target simplification by the unwanted incident *Hard drive crashes* and the asset *Availability of local data*. If we want to reason about risks in relation to the asset *Availability of service* only, we can remove the parts of the target on which this asset does not depend. Since there is no path from the assumptions
Figure 8: Assumption and target simplification
via *Hard drive crashes* or *Availability of local data* to any of the other target elements, the unwanted incident and asset in question can by Rule 3 be removed from the target.

### 3.4 Assumption Consequence

To make use of these rules when scenarios are composed, we also need a consequence rule.

#### Rule 4 (Assumption consequence)

\[
\frac{A \cup A' \triangleright T}{A' \triangleright T} \quad \triangleright A
\]

Hence, if \(T\) holds assuming \(A \cup A'\) to the extent there are explicit dependencies, and we can also show \(A\), then it follows that \(A' \triangleright T\).

Given a dependent diagram \(A \cup A' \triangleright T\) we can use Rule 4 to combine this diagram with a diagram \(D\) in which the validity of \(A\) has been shown. In particular, we can make a new diagram based on \(A \cup A' \triangleright T\) by moving the fragment \(A\) from the assumptions to the target as described in the following example.

**Example 8** Figure 9 shows a dependent diagram that documents the unwanted incident *Service provisioning interrupted* where the threat *Developer* and the threat scenario *Developer causes flaw in SW* are within the target. This is a separate analysis of elements that form part of the assumptions in Fig. 8, and we can therefore use this diagram as the second premise of Rule 4 to deduce the part of the diagram of Fig. 8 that depends on these assumptions.

The dependent diagram of Fig. 10 shows the result of applying Rule 4 with Figs. 8 and 9 as premises. In order to explicitly document that the elements that previously were mere assumptions are no longer so, as shown in the diagram of Fig. 9, these elements have in Fig. 10 been placed within the target.

The element of Fig. 8 that can be removed by Rule 2, namely *Outage in public power supply*, as well as the elements that can be removed by Rule 3, namely *Hard drive crashes* and *Availability of local data*, have also been removed in Fig. 10.

Notice that for the cases in which we can show the validity of all assumptions \(A\) in a dependent diagram \(A \triangleright T\), possibly after assumption and target simplification, we can by Rule 4 conclude \(T\) under no further assumptions. This is conveniently captured by a specialisation of Rule 4 in which \(A'\) is empty.
Remark If we can show the validity of all assumptions $A$ of the dependent diagram $A \triangleright T$, we can use the following specialisation of Rule 4:

$$\frac{A \triangleright T \triangleright A}{\triangleright T}$$

4 Example Case in Dependent CORAS

In order to illustrate the usage of dependent CORAS diagrams, we use the AutoParts example described in *Model-Driven Risk Analysis* [2]. AutoParts is originally a mail order company whose business is to sell spare parts and accessories for a wide range of car makes and vehicle models. It is distributing catalogues by mail that present its products and is usually shipping the goods to the customers by cash on delivery mail.

As described in AutoParts *Model-Driven Risk Analysis* [2] has decided to make a transition from the manual system to an automated online store and wants to do a risk analysis of the system before launching. Of particular concern for the management is that the web application is connected to both their customer database, their inventory database and their online store software. In addition the management fears that the new, and more complex, system will be less stable than the old, primitive one. Finally, the management is worried that the processing and storage of personal data submitted by the customers during registration to the system do not comply with data protection laws.

In the following, we use dependent CORAS diagrams to document assumptions about threat scenarios in the environment that may affect the risk level of the target. In this case, the documented assumptions serve two purposes: (1) Simplifying the analysis by allowing us to assume that certain events will not take place. (2) Facilitate reuse by describing the assumptions in a generic manner.
4.1 Creating Dependent Threat Diagrams

Figure 11 shows a dependent threat diagram for the planned online store of AutoParts. For the purpose of this example case, we assume that AutoParts has conducted a risk analysis of the old system a few years ago. The previous analysis focused on risks towards confidentiality of data stored in the inventory and customer databases, related to remote access. The CORAS diagrams documenting risks and treatments for the databases in *Model-Driven Risk Analysis* [2] document the previous analysis. AutoParts now wants to analyse any new risks that may be introduced in connection with the planned launch of the online store. The focus of this new analysis is the availability of the services provided by the online store to the customers and to the sub-suppliers. We use the consequence scale in Table 2 and the likelihood scale in Table 3, taken from Lund et al [2].

During the initial steps of the analysis, the target team of the risk analysis agrees that the risk of flood should be held outside of the analysis. If the system is hit by a flood, it...
may render the online store unavailable for many days. However, we simply assume that the chance of a flood is zero in order to evaluate the overall risk level when this scenario is not taken into account. We state our assumptions about factors in the environment that may affect the risk level of the online store by placing them outside the border of the target as shown in the dependent diagram of Fig. 11. The relation from the assumption Flood to the target illustrates in what way this assumption is assumed to affect the target.

Since the scenarios outside the border line of the dependent diagram describe our assumptions they might not be verified with respect to the target, and may therefore be wrong. They simply describe the assumptions upon which the validity of our analysis relies. We document the assumption that the chance of flood is zero by placing the referring threat scenario Flood, with likelihood zero, outside of the border.

The target team also agrees that the inventory and customer databases should be left out of the target of the analysis, as these have already been analysed in the previous study. We assume that some of the recommended treatments from the previous study have been implemented and that the threat diagrams have been updated to document the current risk level for the databases. The availability of the online store relies on the availability of customer information and the inventory database. Therefore, threats towards the availability of the databases may also pose threats towards the availability of the online store. The analysis team documents this dependency in the diagram by having a referring threat scenario Databases unavailable outside the target border that leads to the unwanted incident Online store down up to 1 minute.

For the purpose of identifying risks, the likelihood of this threat scenario is not important. We therefore parameterise the likelihood value of the assumption Databases unavailable. The possibility to generalise assumptions through parameterisation facilitates reuse of analysis documentation. The likelihood of the incident Online store down up to 1 minute depends upon the likelihood of all three threat scenarios leading to it, including the assumption Databases unavailable. Hence, we cannot compute the likelihood of this unwanted incident until we instantiate the likelihood of Databases unavailable with a specific value. However, we could of course characterise the likelihood of Online store down up to 1 minute by an expression referring to the parameter X.

To summarise, the referring threat scenarios Flood and Databases unavailable of Fig. 11
represent analysis assumptions, while everything else, including the leads-to relations from the assumptions, represents the target.

### 4.2 Combining Dependent Threat Diagrams

When the risks towards the online store have been identified and risk values have been estimated in workshops, it is up to the analysis team to combine the documentation from the old and the new analysis in order to obtain an overall risk picture of the system as a whole. To keep the combined analysis manageable, the analysis team first creates a high-level diagram from the dependent diagram in Fig. 11, the result of which is depicted in Fig. 12.

![Diagram](image)

**Figure 12: High-level dependent CORAS diagram for the online store**

Figure 13 shows the high-level threat diagram with likelihoods for unwanted incidents regarding the databases. There is no element in this diagram corresponding directly to the referring threat scenario *Databases unavailable* in the dependent diagram in Fig. 12. However, both of the referring threat scenarios *Hacker attack on inventory DB* and *Hacker attack on customer DB* may lead to unavailability of the databases. The analysis team therefore creates a new high-level diagram, shown in Fig. 14 with the referring threat scenario *Databases unavailable* that contains the referring threat scenarios *Hacker attack on customer DB* and *Hacker attack on inventory DB*. Figure 15 shows the referenced threat scenario *Databases unavailable*. The referenced threat scenario *Hacker attack on customer DB* is shown in Fig. 16, taken from Lund et al [2].
Figure 13: High-level threat diagram for the databases

Figure 14: High-level threat diagram for the databases combining threat scenarios
Hacker attack on customer DB

Databases unavailable

Figure 15: Referenced threat scenario *Databases unavailable*

Hacker attack on inventory DB

Figure 16: Referenced threat scenario *Hacker attack on customer DB*
The referenced threat scenario *Databases unavailable* has five gates, two in-gates and three out-gates. For presentation purposes we show in the corresponding referring threat scenarios only the gates that are relevant for the diagram in which these referring threat scenarios occur. For example, in the diagrams in Figs. 11 and 12 we only show the out-gate $o_3$, whereas in the diagram in Fig. 14 this gate is not shown. In the referenced threat scenario of Fig. 15, however, all gates are shown. The in-gates $i_1$ and $i_2$ as well as the out-gates $o_1$ and $o_2$ are shown also on the corresponding referring threat scenario of Fig. 14. The out-gate $o_3$ explains the relation from the assumption *Databases unavailable* in the dependent diagram of Fig. 11.

Figure 17 presents the combined high-level threat diagram for the databases and the online store. The threat diagram of Fig. 14 allows the assumption *Databases unavailable* of the dependent diagram in Fig. 12 to be moved to the target, while the assumption about zero probability of flooding remains an assumption.

The dependencies between the two diagrams in Figs. 12 and 14 go only one way, that is to say the former diagram depends on elements in the latter, but not vice versa. It is therefore straightforward to combine the two diagrams as illustrated in Fig. 17, when substituting $X$ with *unlikely* in Fig. 12. Initially, the unwanted incident *Online down up to 1 minute* was not assigned any likelihood because of the dependency on the assumption *Databases unavailable*. In the combined diagram, however, we use the additional information about this assumption to deduce the likelihood *possible* to this unwanted incident, as shown in Fig. 17.

In order to argue that the combined diagram in Fig. 17 follows from the diagrams in Figs. 12 and 14, we apply the rules introduced in Sect. 3.

Let

$$A_1 \triangleright T_1$$

denote the dependent diagram for the online store in Fig. 12 with *unlikely* substituted for $X$. The assumptions $A_1$ is naturally decomposed into $A'_1$ and $A''_1$ where $A'_1$ is the referring threat scenario *Databases unavailable* and $A''_1$ is the referring threat scenario *Flood*. Let

$$A_2 \triangleright T_2$$

denote the high-level threat diagram for the databases in Fig. 14. There is no border in the diagram of Fig. 14 which means that we can understand it as a dependent diagram with no assumptions. $A_2$ is therefore empty, whereas $T_2$ is the set of all elements and relations in the diagram.

We now wish to deduce

$$A''_1 \triangleright T_1 \cup T_2$$

which corresponds to the combined dependent diagram in Fig. 17.

Since $A_2$ is empty, we immediately have $\triangleright T_2$. As the threat scenario *Databases unavailable*, which equals $A'_1$, is a fragment of $T_2$, we can also immediately deduce $\triangleright A'_1$. From the premises $A'_1 \cup A''_1 \triangleright T_1$ and $\triangleright A'_1$, we can by Rule 4 deduce $A''_1 \triangleright T_1$. Hence, we have deduced $A''_1 \triangleright T_1$ and $\triangleright T_2$, which allows us to conclude $A''_1 \triangleright T_1 \cup T_2$ as desired.

5 Summary

In this chapter, we have introduced the dependent CORAS language, which facilitates the documentation of and reasoning about risk analysis assumptions. Assumptions are used
Figure 17: Combined threat diagram for the data bases and online store
in risk analysis to simplify the analysis, to avoid having to consider risks of no practical relevance and to support reuse. The only syntactic difference between a dependent CORAS diagram and an ordinary CORAS diagram is the border line that distinguishes the target from the assumptions. The reasoning about assumptions and dependencies as documented by dependent diagrams is supported by four deductions rules.

References


Chapter 11

Paper C: Modular analysis and modelling of risk scenarios with dependencies

This chapter was originally published in *Journal of Systems and Software*, Volume 83, Issue 10, October 2010, pages 1995-2013. The paper defines a calculus for dependent risk graphs and includes proofs of the soundness of the rules. For the purpose of readability of the proofs we have added an appendix with the proofs in a larger font than in the original paper. The appendix containing the proofs in the enlarged font is added after the paper.
A Proofs

In order to show the main theorem: that the calculus defined in Section 1 is sound, we first prove soundness of all the rules. The soundness of a rule on the form

\[
\frac{A, D_1, \ldots, D_n}{D}
\]

where \(A\) is a set of assumptions not formulated as part of a dependent risk graph is proved by showing that

\[
A \land \llbracket D_1 \rrbracket \land \ldots \land \llbracket D_n \rrbracket \Rightarrow \llbracket D \rrbracket
\]

That is, we show that the new constraints given in \(D_1\)\ldots\(D_n\) together with the assumption \(A\).

A.1 Theorems and proofs related to computing likelihoods

Theorem 9 (Leads-to)

\[
\llbracket v_1(P_1) \rrbracket \land \llbracket v_1 \xrightarrow{P_2} v_2 \rrbracket \Rightarrow \llbracket (v_1 \sqcap v_2)(P_1 \cdot P_2) \rrbracket
\]

Proof.

(1)1. Assume: 1. \(\llbracket v_1(P_1) \rrbracket\)

2. \(\llbracket v_1 \xrightarrow{P_2} v_2 \rrbracket\)

prove: \(\llbracket (v_1 \sqcap v_2)(P_1 \cdot P_2) \rrbracket\)

(2)1. \(\mu_c(tr(v_1)) \in P_1\)

proof: by assumption (1)1,1 and definition (9)

(2)2. \(\mu_c(tr(v_1 \sqcap v_2)) \in \mu_c(tr(v_1)) \cdot P_2\)

proof: by assumption (1)1,2 and definition (10)

(2)3. Q.E.D.

proof: by (2)1, (2)2 and definition (9)

(1)2. Q.E.D.

proof: \(\Rightarrow\)-rule: (1)1

\[\Box\]

Theorem 10 (Mutually exclusive vertices) We interpret mutual exclusivity of \(v_1\) and \(v_2\) as \(\mu_c(tr(v_1) \cap tr(v_1)) = 0\).

\[
\llbracket v_1(P_1) \rrbracket \land \llbracket v_2(P_2) \rrbracket \land \mu_c(tr(v_1) \cap tr(v_2)) = 0 \Rightarrow \llbracket (v_1 \sqcup v_2)(P_1 + P_2) \rrbracket
\]

Proof.

(1)1. Assume: 1. \(\llbracket v_1(P_1) \rrbracket \land \llbracket v_2(P_2) \rrbracket\)

2. \(\mu_c(tr(v_1) \cap tr(v_2)) = 0\)

prove: \(\llbracket (v_1 \sqcup v_2)(P_1 + P_2) \rrbracket\)

(2)1. \(\mu_c(tr(v_1 \sqcup v_2)) \in P_1 + P_2\)

(3)1. \(tr(v_1 \sqcup v_2) = tr(v_1) \cup tr(v_2)\)

proof: by definition (8)

(3)2. \(\mu_c(tr(v_1 \sqcup v_2)) = \mu_c(tr(v_1) \cup tr(v_2))\)
Proof: By (3)1
(3)3. \( \mu_c(tr(v_1) \cup tr(v_2)) = \mu_c(tr(v_1)) + \mu_c(tr(v_2)) - \mu_c(tr(v_1) \cap tr(v_2)) \)
    Proof: By the fact that \( \mu \) is a probability measure
(3)4. \( \mu_c(tr(v_1) \cup tr(v_2)) = \mu_c(tr(v_1)) + \mu_c(tr(v_2)) \)
    Proof: By (3)3 and assumption (1)1,2
(3)5. \( \mu_c(tr(v_1 \cup v_2)) = \mu_c(tr(v_1)) + \mu_c(tr(v_2)) \)
    Proof: By (3)2 and (3)4
(3)6. \( \mu_c(tr(v_1)) \in P_1 \land \mu_c(tr(v_2)) \in P_2 \)
    Proof: By assumption (1)1,1
(3)7. \( \mu_c(tr(v_1)) + \mu_c(tr(v_2)) \in P_1 + P_2 \)
    Proof: By (3)6
(3)8. Q.E.D.
    Proof: By (3)5 and (3)7
(2)2. Q.E.D.
    Proof: By (2)1 and Definition (9)
(1)2. Q.E.D.
    Proof: \( \Rightarrow \)-rule: (1)1

Theorem 11 (Statistically independent vertices) We interpret statistical independence between \( v_1 \) and \( v_2 \) as \( \mu_c(tr(v_1) \cap tr(v_1)) = \mu_c(tr(v_1)) \cdot \mu_c(tr(v_2)) \).

\[
[v_1(P_1)] \land [v_2(P_2)] \land \mu_c(tr(v_1) \cap tr(v_2)) = \mu_c(tr(v_1)) \cdot \mu_c(tr(v_2)) \Rightarrow 
[(v_1 \cup v_2)(P_1 + P_2 - P_1 \cdot P_2)]
\]

Proof.
(1)1. Assume: 1. \([v_1(P_1)] \land [v_2(P_2)]\]
                      2. \( \mu_c(tr(v_1) \cap tr(v_2)) = \mu_c(tr(v_1)) \cdot \mu_c(tr(v_2)) \)
    Prove: \([v_1 \cup v_2)(P_1 + P_2 - P_1 \cdot P_2)]\]
(2)1. \( \mu_c(tr(v_1 \cup v_2)) \in P_1 + P_2 - P_1 \cdot P_2 \)
(3)1. \( tr(v_1 \cup v_2) = tr(v_1) \cup tr(v_2) \)
    Proof: By Definition (8)
(3)2. \( \mu_c(tr(v_1 \cup v_2)) = \mu_c(tr(v_1) \cup tr(v_2)) \)
    Proof: By (3)1
(3)3. \( \mu_c(tr(v_1) \cup tr(v_2)) = \mu_c(tr(v_1)) + \mu_c(tr(v_2)) - \mu_c(tr(v_1) \cap tr(v_2)) \)
    Proof: By the fact that \( \mu \) is a probability measure
(3)4. \( \mu_c(tr(v_1) \cup tr(v_2)) = \mu_c(tr(v_1)) + \mu_c(tr(v_2)) - \mu_c(tr(v_1)) \cdot \mu_c(tr(v_2)) \)
    Proof: By (3)3 and assumption (1)1,2
(3)5. \( \mu_c(tr(v_1 \cup v_2)) = \mu_c(tr(v_1)) + \mu_c(tr(v_2)) - \mu_c(tr(v_1)) \cdot \mu_c(tr(v_2)) \)
    Proof: By (3)2 and (3)4
(3)6. \( \mu_c(tr(v_1)) \in P_1 \land \mu_c(tr(v_2)) \in P_2 \)
    Proof: By assumption (1)1,1
(3)7. \( \mu_c(tr(v_1)) + \mu_c(tr(v_2)) - \mu_c(tr(v_1)) \cdot \mu_c(tr(v_2)) \in P_1 + P_2 - P_1 \cdot P_2 \)
    Proof: By (3)6
(3)8. Q.E.D.
    Proof: By (3)5 and (3)7
(2)2. Q.E.D.
Proof: By ⟨2⟩1 and Definition (9)
(1)2. Q.E.D.
	Proof: ⇒-rule: ⟨1⟩1

A.2 Proofs related to dependent risk graphs

Lemma 12 (Empty assumption)

\[ [\emptyset \triangleright T] \iff [T] \]

Proof.

(1)1. \[ [\emptyset \triangleright T] \Rightarrow [T] \]
(2)1. Assume: \[ [\emptyset \triangleright T] \]
	Prove: \[ [T] \]
(3)1. \[ \forall T' \subseteq T: [i(T \setminus T', T')] \Rightarrow [T'] \]
	Proof: By assumption (2)1 and Definition (15)
(3)2. Let: \[ T' = T \]
(3)3. \[ [i(\emptyset, T)] \Rightarrow [T] \]
	Proof: By ⟨3⟩1 and ⟨3⟩2
(3)4. \[ i(\emptyset, T) = \emptyset \]
	Proof: By Definition (14)
(3)5. \[ T \Rightarrow [T] \]
	Proof: By ⟨3⟩3 and ⟨3⟩4
(3)6. Q.E.D.
	Proof: By ⟨3⟩5
(2)2. Q.E.D.
	Proof: ⇒-rule: ⟨2⟩1

(1)2. \[ [\emptyset \triangleright T] \Leftarrow [T] \]
(2)1. Assume: \[ [T] \]
	Prove: \[ [\emptyset \triangleright T] \]
(3)1. Q.E.D.
	Proof: By assumption (2)1 and Definition (15)
(2)2. Q.E.D.
	Proof: ⇒-rule: ⟨2⟩1
(1)3. Q.E.D.
	Proof: By (1)1 and (1)2

Lemma 13 (Union of interfaces)

\[ i(A \cup A', T) = i(A, T) \cup i(A', T) \cup \{v \in A \setminus A' | \exists v' \in T : v \rightarrow v' \in A' \setminus A\} \]
\[ \cup \{v \in A' \setminus A | \exists v' \in T : v \rightarrow v' \in A \setminus A'\} \]

Proof.

(1)1. Let: \[ S_1 = \{v \in A \setminus A' | \exists v' \in T : v \rightarrow v' \in A' \setminus A\} \]
\[ S_2 = \{v \in A' \setminus A | \exists v' \in T : v \rightarrow v' \in A \setminus A'\} \]
1. \( A \cup A' \subseteq i(A, T) \cup i(A', T) \cup S_1 \cup S_2 \)

2. Assume: \( e \in i(A \cup A', T) \)

Prove: \( e \in i(A, T) \cup i(A', T) \cup S_1 \cup S_2 \)

3. Case: \( e \) is a node, i.e. \( e = v \) for some node \( v \)

   1. \( v \in A \cup A' \land \exists v' \in T : v \rightarrow v' \in A \cup A' \cup T \)
      
      Proof: By assumption (2)1, assumption (3)1 and Definition (14)

   2. Let: \( v' \in T \) such that \( v \rightarrow v' \in A \cup A' \cup T \)
      
      Proof: By (4)1

3. Assume: \( v \notin i(A, T) \cup i(A', T) \cup S_1 \cup S_2 \)

   Proof: \( \bot \)

   1. \( v \notin A \backslash \ A \)
      
      Proof: By assumption (5)2 and (4)1 (\( v \in A \cup A' \))

   2. \( v \rightarrow v' \notin A' \cup T \)
      
      Proof: By (6)1, (4)2 (\( v' \in T \)) and assumption (4)3 (\( v \notin i(A', T) \))

   3. \( v \rightarrow v' \in A \backslash A' \)
      
      Proof: By (6)2 and (4)2

   4. \( v \in S_2 \)
      
      Proof: By (6)1 and (6)3 and (4)2 (\( v' \in T \))

   5. Q.E.D.
      
      Proof: By (6)4 and assumption (4)3

3. Case: \( v \rightarrow v' \notin A \cup T \)

   1. \( v \rightarrow v' \in A' \backslash A \)
      
      Proof: By assumption (5)3 and (4)2

   2. \( v \notin A \backslash A' \)
      
      Proof: By (6)1 and (4)2 (\( v' \in T \)) and assumption (4)3 (\( v \notin S_1 \))

   3. \( v \in A' \)
      
      Proof: By (6)2 and (4)1 (\( v \in A \cup A' \))

   4. \( v \in A' \land v \rightarrow v' \in A' \cup T \land v' \in T \)
      
      Proof: By (6)3 and (6)1 and (4)2

   5. \( v \in i(A', T) \)
      
      Proof: By (6)4 and Definition (14)

   6. Q.E.D.
      
      Proof: By (6)5 and assumption (4)3

4. Q.E.D.

   Proof: By (5)1, the cases (5)2 and (5)3 are exhaustive

4. Q.E.D.

   Proof: \( \bot \)-rule: (4)3

3. Case: \( e \) is a relation, i.e. \( e = v \rightarrow v' \) for some nodes \( v, v' \)

   1. \( v \rightarrow v' \in A \cup A' \land v' \in T \)
      
      Proof: By assumption (2)1 and assumption (3)2 and Definition (14)

   2. Assume: \( v \rightarrow v' \notin i(A, T) \cup i(A', T) \cup S_1 \cup S_2 \)
      
      Proof: \( \bot \)

   1. \( v \rightarrow v' \notin A \)
      
      Proof: By assumption (4)2 (\( v \rightarrow v' \notin i(A, T) \)), (4)1 (\( v' \in T \)) and Definition (14)
\(5\)2. \(v \rightarrow v' \notin A\)

**Proof:** By assumption \(4\)2 \((v \rightarrow v' \notin i(A', T))\), \(4\)1 \((v' \in T)\) and Definition (14)

\(5\)3. Q.E.D.

**Proof:** By \((5)1\), \((5)2\) and \(4\)1 \((v \rightarrow v' \in A \cup A')\)

\(4\)3. Q.E.D.

**Proof:** By assumption \((4)2\) the cases \((3)1\), \((3)2\), \((3)3\) and \((3)4\) are exhaustive

\(2\)2. Q.E.D.

**Proof:** By assumption \((2)1\) the cases \((3)1\) and \((3)2\) are exhaustive
Theorem 14 (Assumption independence)

\[ A \not\vdash T \land \llbracket A \rhd T \rrbracket \Rightarrow \llbracket T \rrbracket \]

Proof.

(1) Assume: 1. \( A \not\vdash T \)

2. \( \llbracket A \rhd T \rrbracket \)

Prove: \( \llbracket T \rrbracket \)

(2) Assume: \( \forall T' \subseteq T : \llbracket i(A \cup T \setminus T', T') \rrbracket \Rightarrow \llbracket T' \rrbracket \)

Proof: By assumption (1)1,2

(2) Let: \( T' = T \)

(2) Prove: \( \llbracket T \rrbracket \)

Proof: By (2)1 and (2)2

(2) i(\( A, T \)) = \emptyset

Proof: By assumption (1)1,1 and Definition (16)

(2) i(\( T \)) = \emptyset

Proof: By assumption (2)3 and (2)4

(2) Q.E.D.

Proof: \( \Rightarrow \)-rule: (1)1

(1) Q.E.D.

Theorem 15 (Assumption simplification)

\[ A \not\vdash A' \cup T \land \llbracket A \cup A' \rhd T \rrbracket \Rightarrow \llbracket A' \rhd T \rrbracket \]

Proof.

(1) Assume: 1. \( A \not\vdash A' \cup T \)

2. \( \llbracket A \cup A' \rhd T \rrbracket \)

Prove: \( \llbracket A' \rhd T \rrbracket \)

(2) Assume: \( \forall T' \subseteq T : \llbracket i(A' \cup T \setminus T', T') \rrbracket \Rightarrow \llbracket T' \rrbracket \)

Proof: By (1)1 and Lemma 13

(2) i(\( T \)) = \emptyset

Proof: By assumption (3)1 and Definition (14)

(5) i(\( A, T \)) = \emptyset

Proof: By assumption (5)1 and Lemma 13

(5)3. i(\( A, T \)) = \emptyset

Proof: By assumption (3)1 and Definition (14)

(5)4. i(\( A, T \)) = \emptyset

(6) i(\( A, A' \cup T \)) = \emptyset
\textbf{Proof:} By assumption \langle 1 \rangle, 1, 5.
\langle 6 \rangle. \ i(A, T) \subseteq i(A, A' \cup T)
\textbf{Proof:} By Definition \langle 4 \rangle.
\langle 6 \rangle. \ Q.E.D.
\textbf{Proof:} By \langle 6 \rangle and \langle 6 \rangle.
\langle 5 \rangle. \ i(A, T_1) = \emptyset
\textbf{Proof:} By \langle 5 \rangle and \langle 5 \rangle.
\langle 5 \rangle. \ S_1 = \emptyset
\langle 6 \rangle. \ Assume: \ v_1 \in S_1
\hspace{1em} \textbf{Prove:} \ \bot
\langle 7 \rangle. \ \exists v_2 \in T_1 : v_1 \rightarrow v_2 \in (A' \cup T \setminus T_1) \setminus A
\hspace{2em} \textbf{Proof:} By assumption \langle 6 \rangle and \langle 5 \rangle.
\langle 7 \rangle. \ \text{Let:} \ v_2 \in T_1 \text{ such that } v_1 \rightarrow v_2 \in (A' \cup T \setminus T_1) \setminus A
\hspace{2em} \textbf{Proof:} By \langle 7 \rangle.
\langle 7 \rangle. \ A \setminus (A' \cup T \setminus T_1) \subseteq A
\hspace{2em} \textbf{Proof:} By basic set theory
\langle 7 \rangle. \ v_1 \in A
\hspace{2em} \textbf{Proof:} By assumption \langle 6 \rangle, \langle 5 \rangle, \langle 7 \rangle, \langle 7 \rangle, \langle 7 \rangle.
\langle 7 \rangle. \ (A' \cup T \setminus T_1) \setminus A \subseteq A \cup A' \cup T
\hspace{2em} \textbf{Proof:} By basic set theory
\langle 7 \rangle. \ v_1 \rightarrow v_2 \in A \cup A' \cup T
\hspace{2em} \textbf{Proof:} By \langle 7 \rangle and \langle 7 \rangle.
\langle 7 \rangle. \ v_2 \in T
\hspace{2em} \textbf{Proof:} By \langle 7 \rangle (v_2 \in T_1) \text{ and assumption } \langle 3 \rangle.
\langle 7 \rangle. \ v_1 \in i(A, A' \cup T)
\hspace{2em} \textbf{Proof:} By \langle 7 \rangle, \langle 7 \rangle, \langle 7 \rangle.
\langle 7 \rangle. \ i(A, A' \cup T) = \emptyset
\hspace{2em} \textbf{Proof:} By assumption \langle 1 \rangle, \langle 1 \rangle \text{ and Definition } \langle 16 \rangle.
\langle 7 \rangle. \ Q.E.D.
\hspace{2em} \textbf{Proof:} By \langle 7 \rangle and \langle 7 \rangle.
\langle 6 \rangle. \ Q.E.D.
\hspace{2em} \textbf{Proof:} By \langle 6 \rangle.
\langle 5 \rangle. \ S_2 = \emptyset
\langle 6 \rangle. \ Assume: \ v_1 \in S_2
\hspace{1em} \textbf{Prove:} \ \bot
\langle 7 \rangle. \ \exists v_2 \in T_1 : v_1 \rightarrow v_2 \in A \setminus (A' \cup T \setminus T_1)
\hspace{2em} \textbf{Proof:} By assumption \langle 6 \rangle, \langle 5 \rangle.
\langle 7 \rangle. \ \text{Let:} \ v_2 \in T_1 \text{ such that } v_1 \rightarrow v_2 \in A \setminus (A' \cup T \setminus T_1)
\hspace{2em} \textbf{Proof:} By \langle 7 \rangle.
\langle 7 \rangle. \ A \setminus (A' \cup T \setminus T_1) \subseteq A
\hspace{2em} \textbf{Proof:} By basic set theory
\langle 7 \rangle. \ v_1 \rightarrow v_2 \in A
\hspace{2em} \textbf{Proof:} By \langle 7 \rangle and \langle 7 \rangle.
\langle 7 \rangle. \ T_1 \subseteq A' \cup T
\hspace{2em} \textbf{Proof:} By assumption \langle 3 \rangle, \langle 5 \rangle, \text{ and basic set theory}
\langle 7 \rangle. \ v_2 \in A' \cup T
\hspace{2em} \textbf{Proof:} By \langle 7 \rangle and \langle 7 \rangle.
\langle 7 \rangle. \ v_1 \rightarrow v_2 \in i(A, A' \cup T)
\hspace{2em} \textbf{Proof:} By \langle 7 \rangle.
**Theorem 16 (Target simplification)**

\[ T' \vdash T \land [A \triangleright T \cup T'] \Rightarrow [A \triangleright T] \]

Assume that \( A \) is closed under source nodes (i.e. that \( v \rightarrow v' \in A \Rightarrow v \in A \)) and \( A \cap (T' \cup T) = \emptyset \), which follows if \( A \triangleright T \cup T' \) is syntactically correct.

**Proof.**

\((1)\) **Assume:** 1. \( T' \vdash T \)
2. \( [A \triangleright T \cup T'] \)

**Prove:** \( [A \triangleright T] \)

\((2)\) \( \forall T'' \subseteq T : [i(A \cup T \setminus T'', T'')] \Rightarrow [T''] \)

\((3)\) **Assume:** \( T_1 \subseteq T \)

**Prove:** \( [i(A \cup T \setminus T_1, T_1)] \Rightarrow [T_1] \)

\((4)\) **Assume:** \( [i(A \cup T \setminus T_1, T_1)] \)

**Prove:** \( [T_1] \)

\((5)\) **Assume:** \( T_1 \subseteq T \cup T' \)

**Prove:** By assumption \( (3)\)1

\((5)\) **Assume:** \( [i(A \cup (T \cup T') \setminus T_1, T_1)] \Rightarrow [T_1] \)

**Prove:** By \( (5)\)1 and assumption \( (1)\)1,2

\((5)\) **Assume:** \( A \cup (T \cup T') \setminus T_1 = A \cup T \setminus T_1 \cup T' \)
PROOF: By assumption $\langle 3 \rangle$1 and assumption $\langle 1 \rangle$1,1 (which implies $T' \cap T = \emptyset$)

$\langle 5 \rangle$.4. $\llbracket i(A \cup T \setminus T_1 \cup T', T_1) \rrbracket \Rightarrow \llbracket T_1 \rrbracket$
PROOF: By $\langle 5 \rangle$.3 and $\langle 5 \rangle$.2

$\langle 5 \rangle$.5. $\llbracket i(A \cup T \setminus T_1 \cup T', T_1) \rrbracket$

$\langle 6 \rangle$.1. LET: $S_1 = \{ v \in (A \cup T \setminus T_1) \setminus T' \mid \exists v' \in T_1 : v \rightarrow v' \in (A \cup T \setminus T_1) \}$
$S_2 = \{ v \in T' \setminus (A \cup T \setminus T_1) \mid \exists v' \in T_1 : v \rightarrow v' \in (A \cup T \setminus T_1) \}$

$\langle 6 \rangle$.2. $i(A \cup T \setminus T_1 \cup T', T_1) = i(A \cup T \setminus T_1, T_1) \cup i(T', T_1) \cup S_1 \cup S_2$
PROOF: By $\langle 6 \rangle$.1 and Lemma 13

$\langle 6 \rangle$.3. $i(T', T_1) = \emptyset$
PROOF: By assumption $\langle 1 \rangle$1,1 and assumption $\langle 3 \rangle$1

$\langle 6 \rangle$.4. $S_1 = \emptyset$

$\langle 7 \rangle$.1. ASSUME: $v \in S_1$
PROOF: \perp

$\langle 8 \rangle$.1. $\exists v' \in T_1 : v \rightarrow v' \in T' \setminus (A \cup T \setminus T_1)$
PROOF: By assumption $\langle 7 \rangle$.1

$\langle 8 \rangle$.2. LET: $v' \in T_1$ such that $v \rightarrow v' \in T' \setminus (A \cup T \setminus T_1)$
PROOF: By $\langle 8 \rangle$.1

$\langle 8 \rangle$.3. $i(T', T) = \emptyset$
PROOF: By assumption $\langle 1 \rangle$1,1

$\langle 8 \rangle$.4. $\{ v_1 \rightarrow v_2 \in T' \mid v_2 \in T \} = \emptyset$
PROOF: By $\langle 8 \rangle$.3

$\langle 8 \rangle$.5. $v' \in T$
PROOF: By $\langle 8 \rangle$.2 ($v' \in T_1$) and assumption $\langle 3 \rangle$1

$\langle 8 \rangle$.6. $v \rightarrow v' \notin T''$
PROOF: By $\langle 8 \rangle$.4 and $\langle 8 \rangle$.5

$\langle 8 \rangle$.7. Q.E.D.
PROOF: By $\langle 8 \rangle$.6 and $\langle 8 \rangle$.2 ($v \rightarrow v' \in T' \setminus (A \cup T \setminus T_1)$)

$\langle 7 \rangle$.2. Q.E.D.
PROOF: By $\langle 7 \rangle$.1

$\langle 6 \rangle$.5. $S_2 = \emptyset$

$\langle 7 \rangle$.1. ASSUME: $v \in S_2$
PROOF: \perp

$\langle 8 \rangle$.1. $v \in T' \setminus (A \cup T \setminus T_1) \land \exists v' \in T_1 : v \rightarrow v' \in (A \cup T \setminus T_1) \setminus T'$
PROOF: By assumption $\langle 7 \rangle$.1

$\langle 8 \rangle$.2. LET: $v' \in T_1$ such that $v \rightarrow v' \in (A \cup T \setminus T_1) \setminus T'$
PROOF: By $\langle 8 \rangle$.1

$\langle 8 \rangle$.3. $i(T', T) = \emptyset$
PROOF: By assumption $\langle 1 \rangle$1,1

$\langle 8 \rangle$.4. $\{ v_1 \in T' \mid \exists v_2 \in T : v_1 \rightarrow v_2 \in T' \} = \emptyset$
PROOF: By $\langle 8 \rangle$.3

$\langle 8 \rangle$.5. $v' \in T \land v \in T'$
PROOF: By $\langle 8 \rangle$.2 ($v' \in T_1$), assumption $\langle 3 \rangle$1 and $\langle 8 \rangle$.1 ($v \in T' \setminus (A \cup T \setminus T_1)$)

$\langle 8 \rangle$.6. $v \rightarrow v' \notin T \cup T'$
PROOF: By $\langle 8 \rangle$.5 and $\langle 8 \rangle$.4

$\langle 8 \rangle$.7. $v \rightarrow v' \in A$
PROOF: By $\langle 8 \rangle$.6 and $\langle 8 \rangle$.2 ($v \rightarrow v' \in (A \cup T \setminus T_1) \setminus T'$)
\(8\). \( v \in A \)

**Proof:** By \(8\) and the assumption that \( A \) is closed under source-nodes (i.e. that \( v \to v' \in A \Rightarrow v \in A \))

\(9\). \( v \notin T' \cup T \)

**Proof:** By \(8\) and the assumption that \( A \cap (T' \cup T) = \emptyset \)

\(10\). Q.E.D.

**Proof:** By \(8\) \( (v \notin T') \) and \(8\) \((v \in T') \)

\(7\). Q.E.D.

**Proof:** By \(7\)

\(6\). \( i(A \cup T \setminus T_1 \cup T', T_1) = i(A \cup T \setminus T_1, T_1) \)

**Proof:** By \(6\) \(2\), \(6\) \(3\), \(6\) \(4\) and \(6\) \(5\)

\(7\). \( [i(A \cup T \setminus T_1 \cup T', T_1)] \iff [i(A \cup T \setminus T_1, T_1)] \)

**Proof:** By \(6\) \(6\)

\(6\). Q.E.D.

**Proof:** By \(6\) \(7\) and assumption \(4\) \(1\)

\(5\). Q.E.D.

**Proof:** By \(5\) \(4\) and \(5\) \(5\)

\(4\). Q.E.D.

**Proof:** \(\Rightarrow\)-rule: \(4\) \(1\)

\(3\). Q.E.D.

**Proof:** \(\forall\)-rule: \(3\) \(1\)

\(2\). Q.E.D.

**Proof:** By \(2\) \(1\)

\(1\). Q.E.D.

**Proof:** \(\Rightarrow\)-rule: \(1\) \(1\)

\[\square\]

**Theorem 17 (Assumption consequence)**

\[ [ A \cup A' \triangleright T ] \land [ \triangleright A ] \Rightarrow [ A' \triangleright T ] \]

Assume \( v \to v' \in A \Rightarrow v \in A \).

**Proof.**

\(1\). **Assume:** 1. \([ A \cup A' \triangleright T ] \]

2. \([ \triangleright A ] \]

**Prove:** \([ A' \triangleright T ] \)

\(2\). \(\forall T' \subseteq T : [ i(A' \cup T \setminus T', T') ] \Rightarrow [ T' ] \)

\(3\). **Assume:** \( T_1 \subseteq T \)

**Prove:** \([ i(A' \cup T_1 \cup T', T_1) ] \Rightarrow [ T_1 ] \)

\(4\). **Assume:** \([ i(A' \cup T_1 \cup T', T_1) ] \)

**Prove:** \([ T_1 ] \)

\(5\). **Let:** \( S_1 = \{ v \in A \setminus (A' \cup T \setminus T_1) \mid \exists v' \in T_1 : v \to v' \in (A' \cup T \setminus T_1) \setminus A \} \)

\( S_2 = \{ v \in (A' \cup T \setminus T_1) \setminus A \mid \exists v' \in T_1 : v \to v' \in A \setminus (A' \cup T \setminus T_1) \} \)

\(5\). \( i(A \cup A' \cup T \setminus T_1, T_1) = i(A, T_1) \cup i(A' \cup T \setminus T_1, T_1) \cup S_1 \cup S_2 \)

**Proof:** By Lemma 13

\(5\). \([ i(A \cup A' \cup T \setminus T_1, T_1) ] \iff [ i(A, T_1) ] \land [ i(A' \cup T \setminus T_1, T_1) ] \land [ S_1 ] \land [ S_2 ] \)

**Proof:** By \(5\) \(2\)
4. \[ [i(A, T_1)] \]
   (6) 1. \( i(A, T_1) \subseteq A \)
   **Proof:** By Definition (14)

4. \[ [A] \]
   (6) 2. **Proof:** By assumption (1)1,2 and Lemma 12

3. Q.E.D.
   **Proof:** By (6)1 and (6)2

5. \[ [S_1] \]
   (6) 1. \( S_1 \subseteq A \)
   **Proof:** By assumption (1)1,2 and Lemma 12

3. Q.E.D.
   **Proof:** By (6)1 and (6)2

5. \[ [S_2] \]
   (6) 1. \( S_2 = \emptyset \)
   (7) 1. **Assume:** \( v \in S_2 \)
      **Prove:** \( \perp \)
   (8) 1. \( v \in (A' \cup T \setminus T_1) \setminus A \)
      **Proof:** By assumption (7)1
   (8) 2. \( \exists v' \in T_1 : v \rightarrow v' \in A \setminus (A \cup T \setminus T_1) \)
      **Proof:** By assumption (7)1
   (8) 3. **Let:** \( v' \in T_1 \) such that \( v \rightarrow v' \in A \setminus (A \cup T \setminus T_1) \)
      **Proof:** By (8)2
   (8) 4. \( v \rightarrow v' \in A \)
      **Proof:** By (8)3
   (8) 5. \( v \in A \)
      **Proof:** By (8)4 and the assumption that \( v \rightarrow v' \in A \Rightarrow v \in A \)

6. Q.E.D.
   **Proof:** By (8)5 and (8)1

7. 2. Q.E.D.
   **Proof:** By (7)1

6. 2. Q.E.D.
   **Proof:** By (6)1

5. 7. \[ [i(A \cup A' \cup T \setminus T_1, T_1)] \]
   **Proof:** By (5)3, (5)4, assumption (4)1, (5)5 and (5)6

5. 8. \[ [i(A \cup A' \cup T \setminus T_1, T_1)] \Rightarrow [ T_1 ] \]
   **Proof:** By assumption (3)1 and assumption (1)1, 1

5. 9. Q.E.D.
   **Proof:** By (5)7 and (5)8

4. 2. Q.E.D.
   **Proof:** \( \Rightarrow \)-rule: (4)1

3. 2. Q.E.D.
   **Proof:** \( \forall \)-rule: (3)1

2. 2. Q.E.D.
   **Proof:** By (2)1

1. 2. Q.E.D.
   **Proof:** \( \Rightarrow \)-rule: (1)1
A.3 Main theorem

Theorem 8 (Soundness). The calculus for risk graphs is sound.

Proof.

(1) By Theorems 9 to 17.
Chapter 12

Paper D: A denotational model for component-based risk analysis
Paper D: A denotational model for component-based risk analysis
Abstract

Risk analysis is an important tool for developers to establish the appropriate protection level of a system. Unfortunately, the shifting environment of components and component-based systems is not adequately addressed by traditional risk analysis methods. This report addresses this problem from a theoretical perspective by proposing a denotational model for component-based risk analysis. In order to model the probabilistic aspect of risk, we represent the behaviour of a component by a probability distribution over communication histories. The overall goal is to provide a theoretical foundation facilitating an improved understanding of risk in relation to components and component-based system development.

Key words: Risk analysis, component-based development, denotational semantics
## Contents

1 Introduction .......................................................... 5
   1.1 Outline of report ................................................. 6

2 An informal explanation of component-based risk analysis .......... 7
   2.1 Risk analysis .................................................. 7
   2.2 Components and interfaces ..................................... 8
   2.3 Component-based risk analysis .................................. 9
   2.4 Behaviour and probability ....................................... 9
   2.5 Observable component behaviour ............................... 11

3 Denotational representation of interface behaviour .................. 12
   3.1 Sets .................................................................. 12
   3.2 Events ................................................................ 12
   3.3 Sequences .......................................................... 13
   3.4 Traces ................................................................ 15
   3.5 Probabilistic processes .......................................... 16
   3.6 Probabilistic interface execution ............................... 18
       3.6.1 Constraints on interface behaviour ..................... 19

4 Denotational representation of an interface with a notion of risk .... 20
   4.1 Assets .............................................................. 20
   4.2 Incidents and consequences ..................................... 20
   4.3 Incident probability .............................................. 21
   4.4 Risk function ..................................................... 21
   4.5 Interface with a notion of risk ................................... 22

5 Denotational representation of component behaviour ................. 22
   5.1 Probabilistic component execution ............................. 22
   5.2 Trace sets of a composite component ........................... 23
   5.3 Conditional probability measure of a composite component .... 24
   5.4 Composition of probabilistic component executions ........... 27

6 Denotational representation of a component with a notion of risk .... 27

7 Hiding .................................................................. 28

8 Related work ............................................................ 31
   8.1 Security modelling ............................................... 31
   8.2 Probabilistic components ........................................ 31
   8.3 Component models ................................................. 32

9 Conclusion ................................................................ 33

A Auxiliary definitions .................................................. 37
   A.1 Sets ................................................................. 37
   A.2 Logic ............................................................... 37
   A.3 Probability theory ................................................ 38
B Proofs

B.1 Measure theory ........................................... 40
B.2 Probabilistic component execution .................................. 40
B.3 Conditional probability measure of a composite component ........ 49
B.4 Hiding .................................................................. 92
1. Introduction

The flexibility offered by component-based development techniques, such as Sun’s Enterprise JavaBeans (EJB) [39] and Microsoft’s .NET [36], and the potential for reducing production costs through reuse, has lead to an increased preference for such techniques. With strict time-to-market requirements for software technology, products such as cars, laptops, smart phones and mobile devices in general are increasingly sold with upgradeable parts. The flexibility offered by component-based development facilitates rapid development and deployment, but causes challenges for risk analysis that are not addressed by current methods.

An important question for users and developers of component technology is whether to trust a new component to be integrated into a system. This is especially true for systems that handle safety and security-critical tasks such as flight-control systems, or accounting systems [30, 11]. Output from traditional risk-analysis methods is, however, difficult to apply to modern software design. Furthermore, few traditional risk analysis methods take into account that the risk level towards component-based systems may change, given changes in the environment of the systems [53, 33].

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 and safety or security, which are the most relevant for component-based development, risk analysis is concerned with protecting existing assets from harm. We focus upon the latter type of risk analysis, referred to in the following as asset-driven risk analysis. In asset-driven risk analysis, the analysis of threats, vulnerabilities and incidents are driven by the identified assets. An asset may be anything of value to the client of the risk analysis, such as information, software, hardware, services or human life. Assets may also be purely conceptual, such as for example the reputation of an organisation.

The purpose of asset-driven risk analysis is to gather sufficient knowledge about vulnerabilities, threats, consequences and probabilities, in order to establish the appropriate protection level for assets. It is important that the level of protection matches the value of the assets to be protected. If the protection level is too low, the cost from risks will be too high. If the protection level is too high, it may render a service inconvenient for users. A certain level of risk may be acceptable if the risk is considered to be too costly or technically impossible to rule out entirely. Hence, a risk is part of the behaviour of a system that is implicitly allowed but not necessarily intended. Based on this observation we define a component model that integrates the explicit representation of risks as part of the component behaviour and provides rules for composing component risks. We also explain how the notion of hiding can be understood in this component model. We define a hiding operator that allows partial hiding of internal interactions, to ensure that interactions affecting the component risk level are not hidden. We are not aware of other approaches where the concept of risk is integrated in a formal component semantics.

An advantage of representing risks as part of the component behaviour, is that the risk level of a composite component, as well as its behaviour, is obtained by composing the representations of its sub-components. That is, the composition of risks corresponds to ordinary component composition. The component model provides a foundation for component-based risk analysis, by conveying how risks manifests themselves in an
underlying component implementation. By component-based risk analysis we mean that risks are identified, analysed and documented at the component level, and that risk analysis results are composable. The objective of component-based risk analysis is to support development of components that are both trustworthy and user friendly by aiding developers in selecting appropriate protection levels for component assets and develop components in accordance with the selected protection level.

Understanding risks in a component-based setting is challenging because the concept of risk traditionally incorporates some knowledge about how the environment behaves. In order to define a formal foundation for component-based risk analysis, we must decide which risk concepts to include at the component level, without compromising the modularity of our components. In conventional risk analysis external threats are often included and their likelihoods are analysed as part of the overall analysis. The rationale is that the likelihood of an incident is determined from various factors, including the motivation and resources of a potential threat. In a component-based setting, however, we cannot expect to have knowledge about the environment of the component as that may change depending on the platform it is deployed in. Moreover, it is a widely adopted requirement to components that they are separated 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 [6]. In order to obtain a method for component-based risk analysis, current methods must be adapted to comply with the same principles of modularity and composition as component-based development.

1.1. Outline of report

The objective of Section 2 is to give an informal understanding of component-based risk analysis. Risk is the probability that an event affects an asset with a given consequence. In order to model component risks, we explain the concept of asset, asset value and consequence in a component setting. In order to represent the behavioural aspects of risk, such as the probability of unwanted incidents, we make use of an asynchronous communication paradigm. The selection of this paradigm is motivated as part of the informal explanation of component-based risk analysis. We also explain the notions of observable and unobservable behaviour in a component model with assets. The informal understanding introduced in Section 2 is thereafter formalised in a semantic model that defines:

- The denotational representation of interfaces as probabilistic processes (Section 3).

- The denotational representation of interface risks including the means to represent risk probabilities (Section 4). Interface risks are incorporated as a part of the interface behaviour.

- The denotational representation of a component as a collection of interfaces or sub-components, some of which may interact with each other (Section 5). We obtain the behaviour of a component from the probabilistic processes of its constituent interfaces or sub-components in a basic mathematical way.

- The denotational representation of component risks (Section 6).

- The denotational representation of hiding (Section 7).
We place our work in relation to ongoing research within related areas in Section 8. Finally we summarise our findings and discuss possibilities for future work in Section 9.

2. An informal explanation of component-based risk analysis

In this section we describe informally the notion of component-based risk analysis that we aim to formalise in the later sections of this report. In Section 2.1 we explain the concepts of risk analysis and how they relate in a conceptual model. In Section 2.2 we explain the conceptual component model, and in Section 2.3 we explain how the two conceptual models relate to each other. In Section 2.4 we motivate the selection of communication paradigm and explain the behaviour of probabilistic component interfaces. In Section 2.5 we explain which behaviour should be observable in a component with assets, and which should be hidden.

2.1. Risk analysis

Risk analysis is the systematic process to understand the nature of and to deduce the level of risk [48]. We explain the concepts of risk analysis and how they are related to each other through the conceptual model, captured by a UML class diagram [40] in Figure 1. The risk concepts are adapted from international standards for risk analysis terminology [48, 18, 17]. The associations between the elements 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.

![Figure 1: Conceptual model of risk analysis](image)

We explain the conceptual model as follows: Stakeholders are those people and organisations who are affected by a decision or activity. 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 event refers to the occurrence of a particular circumstance. An event which reduces the value of at least one asset is referred to as an incident. A consequence is the reduction in value caused by an incident to an asset. It can be measured qualitatively by linguistic expressions such as “minor”, “moderate”, “major”, or quantitatively, such as a monetary value. 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, i.e., an attacker, or unintentional, i.e., someone causing an incident by mistake. Probability is a measure of the chance of occurrence of an event, expressed as a number between 0 and 1. Conceptually, as illustrated by the UML class diagram in Figure 1, a risk consists of an incident, its probability, and its 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.

2.2. Components and interfaces

Intuitively a component is a standardised “artefact” that can be mass-fabricated and reused in various constellations. According to the classical definition by Szyperski, a software component

... is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties [49].

That a component is a unit of independent deployment means that it needs to be well separated from its environment and other components. A component, therefore, encapsulates its constituent parts. A third party is one that cannot be expected to have access to the construction details of the components involved. A component therefore needs to be sufficiently self-contained.

Components interact through interfaces. An interface is often seen as a contract, specifying what it will provide given that the environment fulfils certain conditions or assumptions. Cheesman and Daniels [4] distinguish between usage and realisation contracts. According to their component definition a component 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.

Our component model is illustrated in Figure 2. To keep the component model simple and general we do not distinguish between usage and realisation. A component is simply a collection of interfaces some of which may interact with each other. Interfaces interact by the transmission and consumption of messages. We refer to the transmission and consumption of messages as events.

Figure 2: Conceptual component model
2.3. Component-based risk analysis

Figure 3 shows how the conceptual model of risk analysis relates to the conceptual component model. To ensure modularity of our component model we represent a stakeholder by the component interface, and identify assets on behalf of component interfaces. Each 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\(^1\). A vulnerability may be understood as a property (or lack thereof) of an interface that makes it prone to a certain attack. It may therefore be argued that the vulnerability concept should be associated to the interface concept. However, from a risk perspective a vulnerability is relevant to the extent that it can be exploited to harm a specific asset, and we have therefore chosen to associate it with the asset concept. The concept of a threat is not part of the conceptual model, because a threat is something that belongs to the environment of a component. We cannot expect to have knowledge about the environment of the component as that may change depending on the where it is deployed. An event that harms an asset is an incident with regard to that asset. 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.

2.4. Behaviour and probability

A probabilistic understanding of component behaviour is required in order to measure risk. We adopt an asynchronous communication model. 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 [16].

An interface interacts with an environment whose behaviour it cannot control. From the point of view of the interface the choices made by the environment are non-deterministic. In order to resolve the external non-determinism caused by the

\(^1\)Note that there may be interfaces with no assets; in this case the stakeholder corresponding to the interface has nothing to protect.
environment we use queues that serve as schedulers. Incoming messages to an interface are stored in a queue and are consumed by the interface in the order they are received. The idea is that, for a given sequence of incoming messages to an interface, we know the probability with which the interface produces a certain behaviour. For simplicity we assume that an interface does not send messages to itself.

A component is a collection of interfaces some of which may interact. For a component consisting of two or more interfaces, a queue history not only resolves the external non-determinism, but also all internal non-determinism with regard to the interactions of its sub-components. The behaviour of a component is the set of probability distributions given all possible queue histories of the component.

Figure 4 shows two different ways in which two interfaces $n_1$ and $n_2$ with queues $q_1$ and $q_2$, and sets of assets $a_1$ and $a_2$, can be combined into a component. We may think of the arrows as directed channels.

- In Figure 4 (1) there is no direct communication between the interfaces of the component, that is, the queue of each interface only contains messages from external interfaces.

- In Figure 4 (2) the interface $n_1$ transmits to $n_2$ which again transmits to the environment. Moreover, only $n_1$ consumes messages from the environment.

Initially, the queue of each interface is empty; its set of assets is fixed throughout an execution. When initiated, an interface chooses probabilistically between a number of different actions (as described in Figure 5). An action consists of transmitting an arbitrary number of messages in some order. The number of transmission messages may be finite, including zero which corresponds to the behaviour of skip, or infinite. The storing of a transmitted message in a queue is instantaneous: a transmitted message is placed in the queue of the recipient, without time delay. There will always

![Figure 4: Two interface compositions](image)

```plaintext
while true do
    begin
        probabilistic_choice(action_1, ..., action_m);
        if done then break;
        blocking_consume(message);
    end
```

Figure 5: Pseudo-code for the input-output behaviour of an interface
be some delay between the transmission of a message and the consumption of that message. After transmitting messages the interface may choose to quit or to check its queue for messages. Messages are consumed in the order they arrive. If the queue is empty, an attempt to consume blocks the interface from any further action until a new message arrives. The consumption of a message gives rise to a new probabilistic choice. Thereafter, the interface may choose to quit without checking the queue again, and so on.

A probabilistic choice over actions never involves more than one interface. This can always be ensured by decomposing probabilistic choices until they have the granularity required. Suppose we have three interfaces; die, player1 and player2 involved in a game of Monopoly. The state of the game is decided by the position of the players' pieces on the board. The transition from one state to another is decided by a probabilistic choice “Throw die and move piece”, involving both the die and one of the players. We may however, split this choice into two separate choices: “Throw die” and “Move piece”. By applying this simple strategy for all probabilistic choices we ensure that a probabilistic choice is a local event of an interface.

The probability distribution over a set of actions, resulting from a probabilistic choice, may change over time during an execution. Hence, our probabilistic model is more general than for example a Markov process [54, 34], where the probability of a future state given the present is conditionally independent of the past. This level of generality is needed to be able to capture all types of probabilistic behaviour relevant in a risk analysis setting, including human behaviour.

The behaviour of a component is completely determined by the behaviour of its constituent interfaces. We obtain the behaviour of a component by starting all the interfaces simultaneously, in their initial state.

2.5. Observable component behaviour

In most component-based approaches there is a clear separation between external and purely internal interaction. External interaction is the interaction between the component and its environment; while purely internal interaction is the interaction within the components, in our case, the interaction between the interfaces of which the component consists. Contrary to the external, purely internal interaction is hidden when the component is viewed as a black-box.

When we bring in the notion of risk, this distinction between what should be externally and only internally visible is no longer clear cut. After all, if we blindly hide all internal interaction we are in danger of hiding (without treating) risks of relevance for assets belonging to externally observable interfaces. Hence, purely internal interaction should be externally visible if it may affect assets belonging to externally visible interfaces. Consider for example the component pictured in Figure 6. In a conventional component-oriented approach, the channels $i_2, i_3, o_2$ and $o_3$ would not be externally observable from a black-box point of view. From a risk analysis perspective it seems more natural to restrict the black-box perspective to the right hand side of the vertical line. The assets belonging to the interface $n_1$ are externally observable since the environment interacts with $n_1$. The assets belonging to the interfaces $n_2$ and $n_3$ are on the other hand hidden since $n_2$ and $n_3$ are purely internal interfaces. Hence, the channels $i_3$ and $o_3$ are also hidden since they can only impact the assets belonging to $n_1$ indirectly via $i_2$ and $o_2$. The channels $i_2$ and $o_2$ are however only partly hidden since
the transmission events of \( i_2 \) and the consumption events of \( o_2 \) may include incidents having an impact on the assets belonging to \( n_1 \).

3. Denotational representation of interface behaviour

In this section we explain the formal representation of interface behaviour in our denotational semantics. We represent interface behaviour by sequences of events that fulfil certain well-formedness constraints. Sequences fulfilling these constraints are called traces. We represent probabilistic interface behaviour as probability distributions over sets of traces.

3.1. Sets

We use standard set notation, such as union \( A \cup B \), intersection \( A \cap B \), set difference \( A \setminus B \), cardinality \( \#A \) and element of \( e \in A \) in the definitions of our basic concepts and operators. We write \( \{e_1, e_2, e_3, \ldots, e_n\} \) to denote the set consisting of \( n \) elements \( e_1, e_2, e_3, \ldots, e_n \). Sometimes we also use \([i..n]\) to denote a totally ordered set of numbers between \( i \) and \( n \). We introduce the special symbol \( \mathbb{N} \) to denote the set of natural numbers:

\[
\mathbb{N} \overset{\text{def}}{=} \{0, 1, 2, 3, \ldots, n, n+1, \ldots\}
\]

and \( \mathbb{N}_+ \) to denote the set of strictly positive natural numbers:

\[
\mathbb{N}_+ = \mathbb{N} \setminus \{0\}
\]

3.2. Events

There are two kinds of events: transmission events tagged by \( ! \) and consumption events tagged by \( ? \). \( \mathcal{K} \) denotes the set of kinds \{\( !, ? \}\}. An event is a pair of a kind and a message. A message is a quadruple \( \langle s, tr, co, q \rangle \) consisting of a signal \( s \), a transmitter \( tr \), a consumer \( co \) and a time-stamp \( q \), which is a rational number. The consumer in the message of a transmission event coincides with the addressee, that is, the party intended to eventually consume the message.

The active party in an event is the one performing the action denoted by its kind. That is, the transmitter of the message is the active party of a transmission event and the consumer of the message is the active party of a consumption event.

We let \( \mathcal{S} \) denote the set of all signals, \( \mathcal{P} \) denote the set of all parties (consumers and transmitters), \( \mathcal{Q} \) denote the set of all time-stamps, \( \mathcal{M} \) denote the set of all messages and \( \mathcal{E} \) denote the set of all events. Formally we have that:
We define the functions
\[ k_. \in \mathcal{E} \rightarrow \mathcal{K} \quad tr_. \in \mathcal{E} \rightarrow \mathcal{P} \quad co_. \in \mathcal{E} \rightarrow \mathcal{P} \quad q_. \in \mathcal{E} \rightarrow \mathcal{Q} \]
to yield the kind, transmitter, consumer and time-stamp of an event. For any party \( p \in \mathcal{P} \), we use \( \mathcal{E}_p \) to denote the set of all events in which \( p \) is the active part. Formally
\[
\mathcal{E}_p \overset{\text{def}}{=} \{ e \in \mathcal{E} | (k.e = ! \land tr.e = p) \lor (k.e = ? \land co.e = p) \}
\]
(1)

For a given party \( p \), we assume that the number of signals assigned to \( p \) is a most countable. That is, the number of signals occurring in messages consumed by or transmitted to \( p \) is at most countable.
We use \( \mathcal{E}_p^\downarrow \) to denote the set of transmission events with \( p \) as consumer. Formally
\[
\mathcal{E}_p^\downarrow \overset{\text{def}}{=} \{ e \in \mathcal{E} | k.e = ! \land co.e = p \}
\]

3.3. Sequences
For any set of elements \( A \), we let \( A^\omega, A^\infty, A^* \) and \( A^n \) denote the set of all sequences, the set of all infinite sequences, the set of all finite sequences, and the set of all sequences of length \( n \) over \( A \). We use \( \langle \rangle \) to denote the empty sequence of length zero and \( \langle 1, 2, 3, 4 \rangle \) to denote the sequence of the numbers from 1 to 4. A sequence over a set of elements \( A \) can be viewed as a function mapping positive natural numbers to elements in the set \( A \). We define the functions
\[
#_\cdot \in A^\omega \rightarrow \mathbb{N} \cup \{\infty\} \quad \subseteq \cdot \in A^\omega \times A^\omega \rightarrow \mathbb{B}
\]
to yield the length, the \( n \)th element of a sequence and the prefix ordering on sequences\(^2\). Hence, \( #s \) yields the number of elements in \( s \), \( s[n] \) yields \( s \)'s \( n \)th element if \( n \leq #s \), and \( s_1 \subseteq s_2 \) evaluates to true if \( s_1 \) is an initial segment of \( s_2 \) or if \( s_1 = s_2 \).
For any \( 0 \leq i \leq #s \) we define \( s|_i \) to denote the prefix of \( s \) of length \( i \). Formally:
\[
\downarrow_\cdot \in A^\omega \times \mathbb{N} \rightarrow A^\omega \\
s|_i \overset{\text{def}}{=} \begin{cases} 
  s' \text{ if } 0 \leq i \leq #s, \text{ where } #s' = i \land s' \subseteq s \\
  s \text{ if } i > #s 
\end{cases}
\]
(3)
Due to the functional interpretation of sequences, we may talk about the range of a sequence:
\[
\text{rng}_\cdot \in A^\omega \rightarrow \mathcal{P}(A)
\]
(4)

\(^2\)The operator \( \times \) binds stronger than \( \rightarrow \) and we therefore omit the parentheses around the argument types in the signature definitions.
For example if $s \in A^\infty$, we have that:

$$\text{rng}.s = \{s[n] | n \in \mathbb{N}_+ \}$$

We define an operator for obtaining the sets of events of a set of sequences, in terms of their ranges:

$$\text{ev}. \in \mathcal{P}(A^\omega) \rightarrow \mathcal{P}(A)$$

$$\text{ev}.S \overset{\text{def}}{=} \bigcup_{s \in S} \text{rng}.s$$

We also define an operator for concatenating two sequences:

$$\otimes \in A^\omega \times A^\omega \rightarrow A^\omega$$

$$s_1 \otimes s_2[n] \overset{\text{def}}{=} \begin{cases} s_1[n] & \text{if } 1 \leq n \leq \#s_1 \\ s_2[n - \#s_1] & \text{if } \#s_1 < n \leq \#s_1 + \#s_2 \end{cases}$$

Concatenating two sequences implies gluing them together. Hence $s_1 \otimes s_2$ denotes a sequence of length $\#s_1 + \#s_2$ that equals $s_1$ if $s_1$ is infinite and is prefixed by $s_1$ and suffixed by $s_2$, otherwise.

The filtering function $\circledast$ is used to filter away elements. By $B \circledast s$ we denote the sequence obtained from the sequence $s$ by removing all elements in $s$ that are not in the set of elements $B$. For example, we have that

$$\{1, 3\} \circledast \langle 1, 1, 2, 1, 3, 2 \rangle = \langle 1, 1, 3 \rangle$$

We define the filtering operator formally as follows:

$$\circledast \in \mathcal{P}(A) \times \mathcal{P}(A^\omega) \rightarrow \mathcal{P}(A^\omega)$$

$$B \circledast \langle \rangle \overset{\text{def}}{=} \langle \rangle$$

$$B \circledast (\langle e \rangle \otimes s) \overset{\text{def}}{=} \begin{cases} \langle e \rangle \otimes B \circledast s & \text{if } e \in B \\ B \circledast s & \text{if } e \notin B \end{cases}$$

For an infinite sequence $s$ we need the additional constraint:

$$(B \cap \text{rng}.s) = \emptyset \Rightarrow B \circledast s = \langle \rangle$$

We overload $\circledast$ to filtering elements from sets of sequences as follows:

$$\circledast \in \mathcal{P}(A) \times \mathcal{P}(A^\omega) \rightarrow \mathcal{P}(A^\omega)$$

$$B \circledast S \overset{\text{def}}{=} \{B \circledast s | s \in S\}$$

We also need a projection operator $\Pi_i.s$ that returns the $i$th element of an $n$-tuple $s$ understood as a sequence of length $n$. We define the projection operator formally as:

$$\Pi_i. \in \{1 \ldots n\} \times A^n \rightarrow A$$

$$\Pi_i[.] \in A^\omega \times \mathbb{N}_+ \rightarrow A$$
The projection operator is overloaded to sets of index values as follows.

\[ \Pi_{I,s} \in \mathbb{P}(\{1 \ldots n\}) \setminus \emptyset \times A^n \to \bigcup_{1 \leq k \leq n} A^k \]

\[ \Pi_{I,s} \overset{\text{def}}{=} s' \]

where \( \forall j \in I : \Pi_{j,s} = \Pi_{\#(i \in I | i \leq j)} . s' \land \#s' = \#I \)

For example we have that:

\[ \Pi_{\{1,2\}}.⟨p, q, r⟩ = ⟨p, q⟩ \]

For a sequence of tuples \( s \), \( \Pi_{I,s} \) denotes the sequence of \( k \)-tuples obtained from \( s \), by projecting each element in \( s \) with respect to the index values in \( I \). For example we have that

\[ \Pi_{\{1,2\}} . (⟨a, r, p⟩, ⟨b, r, p⟩) = (\Pi_{\{1,2\}} . ⟨a, r, p⟩) \mapsto (\Pi_{\{1,2\}} . ⟨b, r, p⟩) = (⟨a⟩, ⟨b⟩) \]

We define the projection operator on a sequence of \( n \)-tuples formally as follows:

\[ \Pi_{I,s} \in \mathbb{P}(\{1 \ldots n\}) \setminus \emptyset \times (A^n)^\omega \to \bigcup_{1 \leq k \leq n} (A^k)^\omega \]

\[ \Pi_{I,s} \overset{\text{def}}{=} s' \]

where

\[ \forall j \in \{1 \ldots \#s\} : \Pi_{I,s}[j] = s'[j] \land \#s = \#s' \]

If we want to restrict the view of a sequence of events to only the signals of the events, we may apply the projection operator twice, as follows:

\[ \Pi_1. (\Pi_2. (⟨a, r, p⟩, ⟨b, r, p⟩)) = (⟨a⟩, ⟨b⟩) \]

Restricting a sequence of events, that is, pairs of kinds and messages, to the second elements of the events yields a sequence of messages. Applying the projection operator a second time with the subscript 1 yields a sequence of signals.

### 3.4. Traces

A trace \( t \) is a sequence of events that fulfils certain well-formedness constraints reflecting the behaviour of the informal model presented in Section 2. We use traces to represent communication histories of components and their interfaces. Hence, the transmitters and consumers in a trace are interfaces. We first formulate two constraints on the timing of events in a trace. The first makes sure that events are ordered by time while the second is needed to avoid Zeno-behaviour. Formally:

\[ \forall i, j \in [1..\#t] : i < j \Rightarrow q.t[i] < q.t[j] \]

\[ \#t = \infty \Rightarrow \forall k \in Q : \exists i \in \mathbb{N} : q.t[i] > k \]

For simplicity, we require that two events in a trace never have the same time-stamp. We impose this requirement by assigning each interface a set of time-stamps disjoint from the set of time-stamps assigned to every other interface. Every event of an
interface is assigned a unique time-stamp from the set of time-stamps assigned to the interface in question.

The first constraint makes sure that events are totally ordered according to when they take place. The second constraint states that time in an infinite trace always eventually progress beyond any fixed point in time. This implies that time never halts and Zeno-behaviour is therefore not possible. To lift the assumption that two events never happen at the same time, we could replace the current notion of a trace as a sequence of events, to a notion of a trace as a sequence of sets of events where the messages in each set have the same time-stamp.

We also impose a constraint on the ordering of transmission and consumption events in a trace $t$. According to the operational model a message can be transmitted without being consumed, but it cannot be consumed without having been transmitted. Furthermore, the consumption of messages transmitted to the same party must happen in the same order as transmission. However, since a trace may include consumption events with external transmitters, we can constrain only the consumption of a message from a party which is itself active in the trace. That is, the ordering requirements on $t$ only apply to the communication between the internal parties. This motivates the following formalisation of the ordering constraint:

\begin{equation}
\text{let } N = \{ n \in \mathcal{P} \mid \text{rng}.t \cap \mathcal{E}_n \neq \emptyset \} \\
\text{in } \forall n, m \in N: \\
\text{let } i = (\{?\} \times (\mathcal{S} \times n \times m \times \mathcal{Q})) \odot t \\
\text{let } o = (\{!\} \times (\mathcal{S} \times n \times m \times \mathcal{Q})) \odot t \\
\text{in } \Pi_{\{1,2,3\}}.(\Pi_{\{2\}}.i) \sqsubseteq \Pi_{\{1,2,3\}}.(\Pi_{\{2\}}.o) \land \forall j \in \{1..\#i\} : q.o[j] < q.i[j]
\end{equation}

The first conjunct of constraint (10) requires that the sequence of consumed messages sent from an internal party $n$ to another internal party $m$, is a prefix of the sequence of transmitted messages from $n$ to $m$, when disregarding time. We abstract away the timing of events in a trace by applying the projection operator twice. Thus, we ensure that messages communicated between internal parties are consumed in the order they are transmitted. The second conjunct of constraint 10 ensures that for any single message, transmission happens before consumption when both the transmitter and consumer are internal. We let $\mathcal{H}$ denote the set of all traces $t$ that are well-formed with regard to constraints (8), (9) and (10).

### 3.5. Probabilistic processes

As explained in Section 2.4, we understand the behaviour of an interface as a probabilistic process. The basic mathematical object for representing probabilistic processes is a probability space \cite{14, 47}. A probability space is a triple $(\Omega, \mathcal{F}, f)$, where $\Omega$ is a sample space, that is, a non-empty set of possible outcomes, $\mathcal{F}$ is a non-empty set of subsets of $\Omega$, and $f$ is a function from $\mathcal{F}$ to $[0, 1]$ that assigns a probability to each element in $\mathcal{F}$.

The set $\mathcal{F}$, and the function $f$ have to fulfil the following constraints: The set $\mathcal{F}$ must be a $\sigma$-field over $\Omega$, that is, $\mathcal{F}$ must be not be empty, it must contain $\Omega$ and be closed under complement and countable union. The function $f$ must be a probability.

\footnote{Note that this is the relative complement with respect to $\Omega$, that is if $A \in \mathcal{F}$, then $\Omega \setminus A \in \mathcal{F}$.}
measure on $\mathcal{F}$, that is, a function from $\mathcal{F}$ to $[0, 1]$ such that $f(\emptyset) = 0$, $f(\Omega) = 1$, and for every sequence $\omega$ of disjoint sets in $\mathcal{F}$, the following holds: $f(\bigcup_{i=1}^{\#\omega} \omega[i]) = \sum_{i=1}^{\#\omega} f(\omega[i])$ [12]. The last property is referred to as countably additive, or $\sigma$-additive.

We represent a probabilistic execution $H$ by a probability space with the set of traces of $H$ as its sample space. If the set of possible traces in an execution is infinite, the probability of a single trace may be zero. To obtain the probability that a certain sequence of events occurs up to a particular point in time, we can look at the probability of the set of all extensions of that sequence in a given trace set. Thus, instead of talking of the probability of a single trace, we are concerned with the probability of a set of traces with common prefix, called a cone. By $c(t, D)$ we denote the set of all continuations of $t$ in $D$. For example we have that

$$
c(\langle a \rangle, \{\langle a, a, b, b \rangle, \langle a, a, c, c \rangle \}) = \{\langle a, a, b, b \rangle, \langle a, a, c, c \rangle \}
c(\langle a, a, b \rangle, \{\langle a, a, b, b \rangle, \langle a, a, c, c \rangle \}) = \{\langle a, a, b \rangle \}
c(\langle b \rangle, \{\langle a, a, b, b \rangle, \langle a, a, c, c \rangle \}) = \emptyset
$$

We define the cone of a finite trace $t$ in a trace set $D$ formally as:

**Definition 3.1 (Cone).** Let $D$ be a set of traces. The cone of a finite trace $t$, with regard to $D$, is the set of all traces in $D$ with $t$ as a prefix:

$$c_\in \mathcal{H} \times \mathbb{P}(\mathcal{H}) \rightarrow \mathbb{P}(\mathcal{H})$$

$$c(t, D) \overset{\text{def}}{=} \{t' \in D \mid t \sqsubseteq t'\}$$

We define the cone set with regard to a set of traces as:

**Definition 3.2 (Cone set).** The cone set of a set of traces $D$ consists of the cones with regard to $D$ of each finite trace that is a prefix of a trace in $D$:

$$C_\in \mathbb{P}(\mathcal{H}) \rightarrow \mathbb{P}(\mathbb{P}(\mathcal{H}))$$

$$C(D) \overset{\text{def}}{=} \{c(t, D) \mid \#t \in \mathbb{N} \land \exists t' \in D : t \sqsubseteq t'\}$$

We understand each trace in the trace set representing a probabilistic process $H$ as a complete history of $H$. We therefore want to be able to distinguish the state where an execution stops after a given sequence and the state where an execution may continue with different alternatives after the sequence. We say that a finite trace $t$ is complete with regard to a set of traces $D$ if $t \in D$. Let $D$ be a set of set of traces. We define the complete extension of the cone set of $D$ as follows:

**Definition 3.3 (Complete extended cone set).** The complete extended cone set of a set of traces $D$ is the union of the cone set of $D$ and the set of singleton sets containing the finite traces in $D$:

$$C_{E_\in} \in \mathbb{P}(\mathcal{H}) \rightarrow \mathbb{P}(\mathbb{P}(\mathcal{H}))$$

$$C_{E}(D) \overset{\text{def}}{=} C(D) \cup \{\{t\} \subseteq D \mid \#t \in \mathbb{N}\}$$

We define a probabilistic execution $H$ formally as:
Definition 3.4 (Probabilistic execution). A probabilistic execution \( H \) is a probability space:

\[
P(\mathcal{H}) \times P(P(\mathcal{H})) \times (P(\mathcal{H}) \rightarrow [0, 1])
\]

whose elements we refer to as \( D_H, \mathcal{F}_H \) and \( f_H \) where \( D_H \) is the set of traces of \( H \), \( \mathcal{F}_H \) is the \( \sigma \)-field generated by \( C_E(D_H) \), that is the intersection of all \( \sigma \)-fields including \( C_E(D_H) \), called the cone-\( \sigma \)-field of \( D_H \), and \( f_H \) is a probability measure on \( \mathcal{F}_H \).

If \( D_H \) is countable then \( P(D_H) \) (the power set of \( D_H \)) is the largest \( \sigma \)-field that can be generated from \( D_H \) and it is common to define \( \mathcal{F}_H \) as \( P(D_H) \). If \( D_H \) is uncountable, then, assuming the continuum hypothesis, which states that there is no set whose cardinality is strictly between that of the integers and that of the real numbers, the cardinality of \( D_H \) equals the cardinality of the real numbers, and hence of \([0, 1]\). This implies that there are subsets of \( P(D_H) \) which are not measurable, and \( \mathcal{F}_H \) is therefore usually a proper subset of \( P(D_H) \) [9]. A simple example of a process with uncountable sample space, is the process that throws a fair coin an infinite number of times [37, 10]. Each execution of this process can be represented by an infinite sequence of zeroes and ones, where 0 represents “head” and 1 represents “tail”. The set of infinite sequences of zeroes and ones is uncountable, which can be shown by a diagonalisation argument [5].

3.6. Probabilistic interface execution

We define the set of traces of an interface \( n \) as any well-formed trace consisting solely of events where \( n \) is the active party. Formally:

\[
\mathcal{H}_n \overset{\text{def}}{=} \mathcal{H} \cap E_n \omega
\]

We define the behavioural representation of an interface \( n \) as a function of its queue history. A queue history of an interface \( n \) is a well-formed trace consisting solely of transmission events

\[
\langle !m_1, \ldots, !m_k \rangle
\]

with \( n \) as consumer. That a queue history is well formed implies that the events in the queue history are totally ordered by time. We let \( B_n \) denote the set of queue histories of an interface \( n \). Formally:

\[
B_n \overset{\text{def}}{=} \mathcal{H} \cap E_n^{\omega}
\]

A queue history serves as a scheduler for an interface, thereby uniquely determining its behaviour [44, 7]. Hence, a queue history gives rise to a probabilistic execution of an interface. That is, the probabilistic behaviour of an interface \( n \) is represented by a function of complete queue histories for \( n \). A complete queue history for an interface \( n \) records the messages transmitted to \( n \) for the whole execution of \( n \), as opposed to a partial queue history that records the messages transmitted to \( n \) until some (finite) point in time. We define a probabilistic interface execution formally as:

Definition 3.5 (Probabilistic interface execution). A probabilistic execution of an interface \( n \) is a function that for every complete queue history of \( n \) returns a probabilistic execution:

\[
I_{n-} \in B_n \rightarrow P(\mathcal{H}_n) \times P(P(\mathcal{H}_n)) \times (P(\mathcal{H}_n) \rightarrow [0, 1])^4
\]
Hence, $I_n(\alpha)$ denotes the probabilistic execution of $n$ given the complete queue history $\alpha$. We let $D_n(\alpha), F_n(\alpha)$ and $f_n(\alpha)$ denote the projections on the three elements of the probabilistic execution of $n$ given queue history $\alpha$. i.e. $I_n(\alpha) = (D_n(\alpha), F_n(\alpha), f_n(\alpha))$.

In Section 2 we described how an interface may choose to do nothing. In the denotational trace semantics we represent doing nothing by the empty trace. Hence, given an interface $n$ and a complete queue history $\alpha$, $D_n(\alpha)$ may consist of only the empty trace, but it may never be empty.

### 3.6.1. Constraints on interface behaviour

The queue history of an interface represents the input to it from other interfaces. In Section 2.4 we described informally our assumptions about how interfaces interact through queues. In particular, we emphasised that an interface can only consume messages already in its queue, and the same message can be consumed only once. We also assumed that an interface does not send messages to itself. Hence, we require that any $t \in D_n(\alpha)$ fulfils the following constraints:

(11) \[
\begin{align*}
\text{let } i &= \{\{?\} \times \mathcal{M}\} \otimes t \\
\quad \text{in } \Pi_{\{1,2\}}(\Pi_{\{2\}}, i) &\subseteq \Pi_{\{1,2\}}(\Pi_{\{2\}}, \alpha) \land \forall j \in \{1..#i\}: q.\alpha[j] < q.i[j]
\end{align*}
\]

The first conjunct of constraint (11) states that the sequence of consumed messages in $t$ is a prefix of the messages in $\alpha$, when disregarding time. Thus, we ensure that $n$ only consumes messages it has received in its queue and that they are consumed in the order they arrived. The second conjunct of constraint (11) ensures that messages are only consumed from the queue after they have arrived and with a non-zero delay. Constraint (12) ensures that an interface does not send messages to itself.

A complete queue history of an interface uniquely determines its behaviour. However, we are only interested in capturing time causal behaviour in the sense that the behaviour of an interface at a given point in time should depend only on its input up to and including that point in time and be independent of the content of its queue at any later point.

In order to formalise this constraint, we first define an operator for truncating a trace at a certain point in time. By $t \downarrow_k$ we denote the timed truncation of $t$, that is, the prefix of $t$ including all events in $t$ with a time-stamp lower than or equal to $k$. For example we have that:

\[
\langle \langle c, q, r, 1 \rangle, \langle a, r, p, 3 \rangle, \langle b, r, p, 5 \rangle \rangle_4 = \langle \langle c, q, r, 1 \rangle, \langle a, r, p, 3 \rangle \rangle_4
\]

\[
\langle \langle c, q, r, 1 \rangle, \langle a, r, p, 3 \rangle, \langle b, r, p, 5 \rangle \rangle_8 = \langle \langle c, q, r, 1 \rangle, \langle a, r, p, 3 \rangle, \langle b, r, p, 5 \rangle \rangle_8
\]

\[
\langle \langle c, q, r, \frac{1}{2} \rangle, \langle a, r, p, \frac{1}{2} \rangle, \langle b, r, p, \frac{3}{2} \rangle \rangle_\frac{3}{2} = \langle \langle c, q, r, \frac{1}{2} \rangle, \langle a, r, p, \frac{1}{2} \rangle \rangle_\frac{3}{2}
\]

The function $\downarrow_k$ is defined formally as follows:

(13) \[
\downarrow_k \in \mathcal{H} \times Q \rightarrow \mathcal{H}
\]

\[
t\downarrow_k \overset{\text{def}}{=} \left\{ \begin{array}{ll}
\emptyset & \text{if } t = \emptyset \lor q.t[1] > k \\
r & \text{otherwise where } r \subseteq t \land q.r[\#r] \leq k \\
\quad \land (\#r < \#t \Rightarrow q.t[\#r + 1] > k)
\end{array} \right.
\]

\[\text{Note that the type of } I_n \text{ ensures that for any } \alpha \in B_n : \text{rng} \alpha \cap evD_n(\alpha) = \emptyset\]
We overload the timed truncation operator to sets of traces as follows:

\[
\downarrow_s \in \mathbb{P}(\mathcal{H}) \times \mathcal{Q} \rightarrow \mathbb{P}(\mathcal{H})
\]

\[
S|_k \overset{\text{def}}{=} \{ t|_k \mid t \in S \}
\]

We may then formalise the time causality as follows:

\[
∀\alpha, \beta \in \mathcal{B}_n : ∀q \in \mathcal{Q} : \alpha|_q = \beta|_q \Rightarrow (D_n(\alpha)|_q = D_n(\beta)|_q) \land
\]

\[
((∀t_1 \in D_n(\alpha) : ∀t_2 \in D_n(\beta)) : t_1|_q = t_2|_q) \Rightarrow
\]

\[
(f_n(\alpha)(c(t_1|_q, D_n(\alpha))) = f_n(\beta)(c(t_2|_q, D_n(\beta))))
\]

The first conjunct states that for all queue histories \(\alpha, \beta\) of an interface \(n\), and for all points in time \(q\), if \(\alpha\) and \(\beta\) are equal until time \(q\), then the trace sets \(D_n(\alpha)\) and \(D_n(\beta)\) are also equal until time \(q\). The second conjunct states that if \(\alpha\) and \(\beta\) are equal until time \(q\), and we have two traces in \(D_n(\alpha)\) and \(D_n(\beta)\) that are equal until time \(q\), then the likelihoods of the cones of the two traces truncated at time \(q\) in their respective trace sets are equal. Thus, the constraint ensures that the behaviour of an interface at a given point in time depends on its queue history up to and including that point in time, and is independent of the content of its queue history at any later point.

4. Denotational representation of an interface with a notion of risk

Having introduced the underlying semantic model, the next step is to extend it with concepts from risk analysis according to the conceptual model in Figure 3. As already explained, the purpose of extending the semantic model with risk analysis concepts is to represent risks as an integrated part of interface and component behaviour.

4.1. Assets

An asset is a physical or conceptual entity which is of value for a stakeholder, that is, for an interface (see Section 2.1) and which the stakeholder wants to protect. We let \(\mathcal{A}\) denote the set of all assets and \(\mathcal{A}_n\) denote the set of assets of interface \(n\). Note that \(\mathcal{A}_n\) may be empty. We require:

\[
∀n, n' \in \mathcal{P} : n \neq n' \Rightarrow \mathcal{A}_n \cap \mathcal{A}_{n'} = \emptyset
\]

Hence, assets are not shared between interfaces.

4.2. Incidents and consequences

As explained in Section 2.3 an incident is an event that reduces the value of one or more assets. This is a general notion of incident, and of course, an asset may be harmed in different ways, depending on the type of asset. Some examples are reception of corrupted data, transmission of classified data to an unauthorised user, or slow response to a request. We provide a formal model for representing events that harm assets. For a discussion of how to obtain further risk analysis results for components, such as the cause of an unwanted incident, its consequence and probability we refer to [2].

In order to represent incidents formally we need a way to measure harm inflicted upon an asset by an event. We represent the consequence of an incident by a positive
integer indicating its level of seriousness with regard to the asset in question. For example, if the reception of corrupted data is considered to be more serious for a given asset than the transmission of classified data to an unauthorised user, the former has a greater consequence than the latter with regard to this asset. We introduce a function

\[ cv_n(e, a) \in \mathcal{E}_n \times \mathcal{A}_n \rightarrow \mathbb{N} \]  

that for an event \( e \) and asset \( a \) of an interface \( n \), yields the consequence of \( e \) to \( a \) if \( e \) is an incident, and 0 otherwise. Hence, an event with consequence larger than zero for a given asset is an incident with regard to that asset. Note that the same event may be an incident with respect to more than one asset; moreover, an event that is not an incident with respect to one asset, may be an incident with respect to another.

4.3. Incident probability

The probability that an incident \( e \) occurs during an execution corresponds to the probability of the set of traces in which \( e \) occurs. Since the events in each trace are totally ordered by time, and all events include a time-stamp, each event in a trace is unique. This means that a given incident occurs only once in each trace.

We can express the set describing the occurrence of an incident \( e \), in a probabilistic execution \( H \), as \( occ(e, D_H) \) where the function \( occ \) is formally defined as follows:

\[ occ \in \mathcal{E} \times \mathbb{P}(\mathcal{H}) \rightarrow \mathbb{P}(\mathcal{H}) \]

\[ occ(e, D) \overset{\text{def}}{=} \{ t \in D \mid e \in \text{rng} \cdot t \} \]

The set \( occ(e, D_H) \) corresponds to the union of all cones \( c(t, D_H) \) where \( e \) occurs in \( t \) (see Section 3.5). Any union of cones can be described as a disjoint set of cones [43]. As described in Section 3, we assume that an interface is assigned at most a countable number of signals and we assume that time-stamps are rational numbers. Hence, it follows that an interface has a countable number of events. Since the set of finite sequences formed from a countable set is countable [25], the union of cones where \( e \) occurs in \( t \) is countable. Since by definition, the cone-\( \sigma \)-field of an execution \( H \), is closed under countable union, the occurrence of an incident can be represented as a countable union of disjoint cones, that is, it is an element in the cone-\( \sigma \)-field of \( H \) and thereby has a measure.

4.4. Risk function

The risk function of an interface \( n \) takes a consequence, a probability and an asset as arguments and yields a risk value represented by a positive integer. Formally:

\[ rf_n(e, p, a) \in \mathbb{N} \times [0, 1] \times \mathcal{A}_n \rightarrow \mathbb{N} \]

The risk value associated with an incident \( e \) in an execution \( H \), with regard to an asset \( a \), depends on the probability of \( e \) in \( H \) and its consequence value. We require that

\[ rf_n(e, p, a) = 0 \Leftrightarrow c = 0 \lor p = 0 \]

Hence, only incidents have a positive risk value, and any incident has a positive risk value.
4.5. Interface with a notion of risk

Putting everything together we end up with the following representation of an interface:

**Definition 4.1 (Semantics of an interface).** An interface $n$ is represented by a quadruple

$$(I_n, A_n, cv_n, rf_n)$$

consisting of its probabilistic interface execution, assets, consequence function and risk function as explained above.

Given such a quadruple we have the necessary means to calculate the risks associated with an interface for a given queue history. A risk is a pair of an incident and its risk value. Hence, for the queue history $\alpha \in \mathcal{B}_n$ and asset $a \in \mathcal{A}_n$ the associated risks are

$$\{rv | rv = rf_n(cv(e, a), f_n(occ(e, D_n(\alpha))), a) \land rv > 0 \land e \in \mathcal{E}_n\}$$

5. Denotational representation of component behaviour

A component is a collection of interfaces, some of which may interact. We may view a single interface as a basic component. A composite component is a component containing at least two interfaces (or basic components). In this section we lift the notion of probabilistic execution from interfaces to components. Furthermore, we explain how we obtain the behaviour of a component from the behaviours of its sub-components. In this section we do not consider the issue of hiding; this is the topic of Section 7.

In Section 5.1 we introduce the notion of conditional probability measure, conditional probabilistic execution and probabilistic component execution. In Section 5.2 we characterise the trace set of a composite component from the trace sets of its sub-components. The cone-$\sigma$-field of a probabilistic component execution is generated straightforwardly from that. In Section 5.3 we explain how to define the conditional probability measure for the cone-$\sigma$-field of a composite component from the conditional probability measures of its sub-components. Finally, in Section 5.4, we define a probabilistic component execution of a composite component in terms of the probabilistic component executions of its sub-components. We sketch the proof strategies for the lemmas and theorems in this section and refer to Appendix B for the full proofs.

5.1. Probabilistic component execution

The behaviour of a component is completely determined by the set of interfaces it consists of. We identify a component by the set of names of its interfaces. Hence, the behaviour of the component $\{n\}$ consisting of only one interface $n$, is identical to the behaviour of the interface $n$. For any set of interfaces $N$ we define:

$$\mathcal{E}_N \stackrel{\text{def}}{=} \bigcup_{n \in N} \mathcal{E}_n$$

$$\mathcal{E}_N^\downarrow \stackrel{\text{def}}{=} \bigcup_{n \in N} \mathcal{E}_n^\downarrow$$

$$\mathcal{H}_N \stackrel{\text{def}}{=} \mathcal{H} \cap \mathcal{E}_N^\omega$$

$$\mathcal{B}_N \stackrel{\text{def}}{=} \mathcal{H} \cap \mathcal{E}_N^\omega$$
Just as for interfaces, we define the behavioural representation of a component \( N \) as a function of its queue history. For a single interface a queue history \( \alpha \) resolves the external nondeterminism caused by the environment. Since we assume that an interface does not send messages to itself there is no internal non-determinism to resolve. The function representing an interface returns a probabilistic execution which is a probability space. Given an interface \( n \) it follows from the definition of a probabilistic execution, that for any queue history \( \alpha \in \mathcal{B}_n \), we have \( f_n(\alpha)(D_n(\alpha)) = 1 \).

For a component \( N \) consisting of two or more sub-components, a queue history \( \alpha \) must resolve both external and internal non-determinism. For a given queue history \( \alpha \) the behaviour of \( N \), is obtained from the behaviours of the sub-components of \( N \) that are possible with regard to \( \alpha \). That is, all internal choices concerning interactions between the sub-components of \( N \) are fixed by \( \alpha \). This means that the probability of the set of traces of \( N \) given a queue history \( \alpha \) may be lower than 1, violating the requirement of a probability measure. In order to formally represent the behaviour of a component we therefore introduce the notion of a conditional probability measure.

**Definition 5.1 (Conditional probability measure).** Let \( D \) be a non-empty set and \( \mathcal{F} \) be a \( \sigma \)-field over \( D \). A conditional probability measure \( f \) on \( \mathcal{F} \) is a function that assigns a value in \([0,1]\) to each element of \( \mathcal{F} \) such that; either \( f(A) = 0 \) for all \( A \) in \( \mathcal{F} \), or there exists a constant \( c \in (0,1) \)\(^5\) such that the function \( f' \) defined by \( f'(A) = f(A)/c \) is a probability measure on \( \mathcal{F} \).

We define a conditional probabilistic execution \( H \) formally as:

**Definition 5.2 (Conditional probabilistic execution).** A conditional probabilistic execution \( H \) is a measure space \([14]\):

\[
\mathbb{P}(\mathcal{H}) \times \mathbb{P}(\mathbb{P}(\mathcal{H})) \times (\mathbb{P}(\mathcal{H}) \to [0,1])
\]

whose elements we refer to as \( D_H \), \( \mathcal{F}_H \) and \( f_H \) where \( D_H \) is the set of traces of \( H \), \( \mathcal{F}_H \) is the cone-\(\sigma\)-field of \( D_H \), and \( f_H \) is a conditional probability measure on \( \mathcal{F}_H \).

We define a probabilistic component execution formally as:

**Definition 5.3 (Probabilistic component execution).** A probabilistic execution of a component \( N \) is a function \( I_N \) that for every complete queue history of \( N \) returns a conditional probabilistic execution:

\[
I_{N \leftarrow} : \mathcal{B}_N \to \mathbb{P}(\mathcal{H}_N) \times \mathbb{P}(\mathbb{P}(\mathcal{H}_N)) \times (\mathbb{P}(\mathcal{H}_N) \to [0,1])
\]

Hence, \( I_N(\alpha) \) denotes the probabilistic execution of \( N \) given the complete queue history \( \alpha \). We let \( D_N(\alpha), \mathcal{F}_N(\alpha) \) and \( f_N(\alpha) \) denote the canonical projections of the probabilistic component execution on its elements.

### 5.2. Trace sets of a composite component

For a given queue history \( \alpha \), the combined trace sets \( D_{N_1}(\mathcal{E}_{N_1}^1 \oplus \alpha) \) and \( D_{N_2}(\mathcal{E}_{N_2}^1 \oplus \alpha) \) such that all the transmission events from \( N_1 \) to \( N_2 \) are in \( \alpha \) and the other way around,

\(^5\)We use \( \langle a, b \rangle \) to denote the open interval \( \{ x \mid a < x < b \} \).
constitute the legal set of traces of the composition of \( N_1 \) and \( N_2 \). Given two probabilistic component executions \( I_{N_1} \) and \( I_{N_2} \) such that \( N_1 \cap N_2 = \emptyset \), for each \( \alpha \in \mathcal{B}_{N_1 \cup N_2} \) we define their composite trace set formally as:

\[
(D_{N_1} \otimes D_{N_2} \cdot = \mathcal{B}_{N_1 \cup N_2} \rightarrow \mathbb{P}(\mathcal{H}_{N_1 \cup N_2}))
\]

\[
D_{N_1} \otimes D_{N_2}(\alpha) \overset{\text{def}}{=} \{ t \in \mathcal{H}_{N_1 \cup N_2} | \mathcal{E}_{N_1} \otimes t \in D_{N_1}(\mathcal{E}_{N_1}^1 \otimes \alpha) \land \mathcal{E}_{N_2} \otimes t \in D_{N_2}(\mathcal{E}_{N_2}^1 \otimes \alpha) \land
(\{!\} \times \mathcal{S} \times N_2 \times N_1 \times \mathcal{Q}) \otimes t \sqsubseteq (\{!\} \times \mathcal{S} \times N_2 \times N_1 \times \mathcal{Q}) \otimes \alpha \land
(\{!\} \times \mathcal{S} \times N_1 \times N_2 \times \mathcal{Q}) \otimes t \sqsubseteq (\{!\} \times \mathcal{S} \times N_1 \times N_2 \times \mathcal{Q}) \otimes \alpha\}
\]

The definition ensures that the messages from \( N_2 \) consumed by \( N_1 \) are in the queue history of \( N_1 \) and vice versa. The operator \( \otimes \) is obviously commutative and also associative since the sets of interfaces of each component are disjoint.

For each \( \alpha \in \mathcal{B}_{N_1 \cup N_2} \), the cone-\( \sigma \)-field is generated as before. Hence, we define the cone-\( \sigma \)-field of a composite component as follows:

\[
(22) \quad \mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha) \overset{\text{def}}{=} \sigma(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))
\]

where \( \sigma(D) \) denotes the \( \sigma \)-field generated by the set \( D \). We refer to \( C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \) as the composite extended cone set of \( N_1 \cup N_2 \).

5.3. Conditional probability measure of a composite component

Consider two components \( C \) and \( O \) such that \( C \cap O = \emptyset \). As described in Section 2, it is possible to decompose a probabilistic choice over actions in such a way that it never involves more than one interface. We may therefore assume that for a given queue history \( \alpha \in \mathcal{B}_{C \cup O} \), the behaviour represented by \( D_C(\mathcal{E}_C^1 \otimes \alpha) \) is independent of the behaviour represented by \( D_O(\mathcal{E}_O^1 \otimes \alpha) \). Given this assumption the probability of a certain behaviour of the composed component equals the product of the probabilities of the corresponding behaviours of \( C \) and \( O \), by the law of statistical independence. As explained in Section 3.5, to obtain the probability that a certain sequence of events \( t \) occurs up to a particular point in time in a set of traces \( D \), we can look at the cone of \( t \) in \( D \). For a given cone \( c \in C_E(D_C \otimes D_O(\alpha)) \) we obtain the corresponding behaviours of \( C \) and \( O \) by filtering \( c \) on the events of \( C \) and \( O \), respectively.

The above observation with regard to cones does not necessarily hold for all elements of \( \mathcal{F}_C \otimes \mathcal{F}_O(\alpha) \). The following simple example illustrates that the probability of an element in \( \mathcal{F}_C \otimes \mathcal{F}_O(\alpha) \), which is not a cone, is not necessarily the product of the corresponding elements in \( \mathcal{F}_C(\mathcal{E}_C^1 \otimes \alpha) \) and \( \mathcal{F}_O(\mathcal{E}_O^1 \otimes \alpha) \). Assume that the component \( C \) tosses a fair coin and that the component \( O \) tosses an Othello piece (a disk with a light and a dark face). We assign the singleton time-stamp set \( \{1\} \) to \( C \) and the singleton time-stamp set \( \{2\} \) to \( O \). Hence, the traces of each may only contain one event. For the purpose of readability we represent in the following the events by their signals. The assigned time-stamps ensure that the coin toss represented by the events \( \{h, t\} \) comes
before the Othello piece toss. We have:

\[
\begin{align*}
D_C(\langle \rangle) &= \{\langle h \rangle, \langle t \rangle \} \\
F_C(\langle \rangle) &= \emptyset, \{\langle h \rangle\}, \{\langle t \rangle\}, \{\langle h \rangle, \langle t \rangle\} \\
f_C(\langle \rangle)(\{\langle h \rangle\}) &= 0.5 \\
f_C(\langle \rangle)(\{\langle t \rangle\}) &= 0.5
\end{align*}
\]

and

\[
\begin{align*}
D_O(\langle \rangle) &= \{\langle b \rangle, \langle w \rangle\} \\
F_O(\langle \rangle) &= \emptyset, \{\langle b \rangle\}, \{\langle w \rangle\}, \{\langle b \rangle, \langle w \rangle\} \\
f_O(\langle \rangle)(\{\langle b \rangle\}) &= 0.5 \\
f_O(\langle \rangle)(\{\langle w \rangle\}) &= 0.5
\end{align*}
\]

Let \( D_{CO} = D_C \otimes D_O \). The components interacts only with the environment, not with each other. We have:

\[
D_{CO}(\langle \rangle) = \{\langle h, b \rangle, \langle h, w \rangle, \langle t, b \rangle, \langle t, w \rangle\}
\]

We assume that each element in the sample space (trace set) of the composite component has the same probability. Since the sample space is finite, the probabilities are given by discrete uniform distribution, that is each trace in \( D_{CO}(\langle \rangle) \) has a probability of 0.25. Since the traces are mutually exclusive, it follows by the laws of probability that the probability of \( \{\langle h, b \rangle\} \cup \{\langle t, w \rangle\} \) is the sum of the probabilities of \( \{\langle h, b \rangle\} \) and \( \{\langle t, w \rangle\} \), that is 0.5. But this is not the same as \( f_C(\langle \rangle)(\{\langle h \rangle, \langle t \rangle\}) \cdot f_O(\langle \rangle)(\{\langle b \rangle, \langle w \rangle\}) \overset{6}{=} 1 \).

Since there is no internal communication between \( C \) and \( O \), there is no internal non-determinism to be resolved. If we replace the component \( O \) with the component \( R \), which simply consumes whatever \( C \) transmits, a complete queue history of the composite component reflects only one possible interaction between \( C \) and \( R \). Let \( D_{CR} = D_C \otimes D_R \). To make visible the compatibility between the trace set and the queue history we include the whole events in the trace sets of the composite component. We have:

\[
\begin{align*}
D_{CR}(\langle \langle h, C, R, 1 \rangle \rangle) &= \{\langle \langle h, C, R, 1 \rangle, ?\langle h, C, R, 2 \rangle \rangle\} \\
D_{CR}(\langle \langle t, C, R, 1 \rangle \rangle) &= \{\langle \langle t, C, R, 1 \rangle, ?\langle t, C, R, 2 \rangle \rangle\}
\end{align*}
\]

For a given queue history \( \alpha \), the set \( E_C \otimes D_{CR}(\alpha) \) is a subset of the trace set \( D_C(E_C^1 \otimes \alpha) \) that is possible with regard to \( \alpha \) (that \( E_C \otimes D_{CR}(\alpha) \) is a subset of \( D_C(E_C^1 \otimes \alpha) \) follows from Lemma B.21 which is shown in Appendix B). We call the set of traces of \( C \) that are possible with regard to a given queue history \( \alpha \) and component \( R \) for \( CT_{C-R}(\alpha) \), which is short for conditional traces.

Given two components \( N_1 \) and \( N_2 \) and a complete queue history \( \alpha \in B_{N_1 \cup N_2} \), we define the set of conditional traces of \( N_1 \) with regard to \( \alpha \) and \( N_2 \) formally as:

\[
(24) \quad CT_{N_1-N_2}(\alpha) \overset{\text{def}}{=} \{ t \in D_{N_1}(E_{N_1}^1 \otimes \alpha) \mid ([l] \times S \times N_1 \times N_2 \times Q) \otimes t \subseteq ([l] \times S \times N_1 \times N_2 \times Q) \otimes \alpha \}
\]

\[\overset{6}{=}\text{We use} \cdot \text{to denote normal multiplication.}\]
Lemma 5.4. Let \( I_{N_1} \) and \( I_{N_2} \) be two probabilistic component executions such that \( N_1 \cap N_2 = \emptyset \) and let \( \alpha \) be a queue history in \( B_{N_1 \cup N_2} \). Then

\[
CT_{N_1-N_2}(\alpha) \in F_{N_1}(E_{N_1}^1 \otimes \alpha) \wedge CT_{N_2-N_1}(\alpha) \in F_{N_2}(E_{N_2}^1 \otimes \alpha)
\]

Proof Sketch: The set \( CT_{N_1-N_2}(\alpha) \) includes all traces in \( D_{N_1}(E_{N_1}^1 \otimes \alpha) \) that are compatible with \( \alpha \), i.e., traces that are prefixes of \( \alpha \) when filtered on the transmission events from \( N_1 \) to \( N_2 \). The key is to show that this set can be constructed as an element in \( F_{N_1}(E_{N_1}^1 \otimes \alpha) \). If \( \alpha \) is infinite, this set corresponds to the union of (1) all finite traces in \( D_{N_1}(E_{N_1}^1 \otimes \alpha) \) that are compatible with \( \alpha \) and (2) the set obtained by constructing countable unions of cones of traces that are compatible with finite prefixes of \( \alpha |_{i} \) for all \( i \in \mathbb{N} \) (where \( \alpha |_{i} \) denotes the prefix of \( \alpha \) of length \( i \)) and then construct the countable intersection of all such countable unions of cones. If \( \alpha \) is finite the proof is simpler, and we do not get into the details here. The same procedure may be followed to show that \( CT_{N_2-N_1}(\alpha) \in F_{N_2}(E_{N_2}^1 \otimes \alpha) \).

As illustrated by the example above, we cannot obtain a measure on a composite cone-\( \sigma \)-field in the same manner as for a composite extended cone set. In order to define a conditional probability measure on a composite cone-\( \sigma \)-field, we first define a measure on the composite extended cone set it is generated from. We then show that this measure can be uniquely extended to a conditional probability measure on the generated cone-\( \sigma \)-field. Given two probabilistic component executions \( I_{N_1} \) and \( I_{N_2} \) such that \( N_1 \cap N_2 = \emptyset \), for each \( \alpha \in B_{N_1 \cup N_2} \) we define a measure \( \mu_{N_1} \otimes \mu_{N_2}(\alpha) \) on \( C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \) formally as follows:

\[
\begin{align*}
(25) & \quad \mu_{N_1} \otimes \mu_{N_2} \in B_{N_1 \cup N_2} \rightarrow (C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \rightarrow [0, 1]) \\
\mu_{N_1} \otimes \mu_{N_2}(\alpha)(c) & \overset{\text{def}}{=} f_{N_1}(E_{N_1}^1 \otimes \alpha)(E_{N_1} \otimes c) \cdot f_{N_2}(E_{N_2}^1 \otimes \alpha)(E_{N_2} \otimes c)
\end{align*}
\]

Theorem 5.5. The function \( \mu_{N_1} \otimes \mu_{N_2}(\alpha) \) is well defined.

Proof Sketch: For any \( c \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \) we must show that \( (E_{N_1} \otimes c) \in F_{N_1}(E_{N_1}^1 \otimes \alpha) \) and \( (E_{N_2} \otimes c) \in F_{N_2}(E_{N_2}^1 \otimes \alpha) \). If \( c \) is a singleton (containing exactly one trace) the proof follows from the fact that (1): if \( (D, F, f) \) is a conditional probabilistic execution and \( t \) is a trace in \( D \), then \( \{t\} \in F \) [37], and (2): that we can show \( E_{N_1} \otimes t \in D_{N_1}(E_{N_1}^1 \otimes \alpha) \wedge E_{N_2} \otimes t \in D_{N_2}(E_{N_2}^1 \otimes \alpha) \) from Definition 3.3 and definition (22).

If \( c \) is a cone \( c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \) in \( C(D_{N_1} \otimes D_{N_2}(\alpha)) \), we show that \( CT_{N_1-N_2}(\alpha) \), intersected with \( c(E_{N_1} \otimes t, D_{N_1}(E_{N_1}^1 \otimes \alpha)) \) and the traces in \( D_{N_1}(E_{N_1}^1 \otimes \alpha) \) that are compatible with \( t \) with regard to the timing of events, is an element of \( F_{N_1}(E_{N_1}^1 \otimes \alpha) \) that equals \( (E_{N_1} \otimes c) \). We follow the same procedure to show that \( (E_{N_2} \otimes c) \in F_{N_2}(E_{N_2}^1 \otimes \alpha) \).

Lemma 5.6. Let \( I_{N_1} \) and \( I_{N_2} \) be two probabilistic component executions such that \( N_1 \cap N_2 = \emptyset \) and let \( \mu_{N_1} \otimes \mu_{N_2} \) be a measure on the extended cones set of \( D_{N_1} \otimes D_{N_2} \) as defined by (25). Then, for all complete queue histories \( \alpha \in B_{N_1 \cup N_2} \)

1. \( \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\emptyset) = 0 \)
2. \( \mu_{N_1} \otimes \mu_{N_2}(\alpha) \) is \( \sigma \)-additive
3. \( \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \leq 1 \)
Proof sketch: We sketch the proof strategy for point 2 of Lemma 5.6. The proofs of point 1 and 3 are simpler, and we do not go into the details here. Assume $\phi$ is a sequence of disjoint sets in $C_E(D_{N_1} \otimes D_{N_2}(\alpha))$. We construct a sequence $\psi$ of length $\#\phi$ such that $\forall i \in [1, \#\phi]: \psi[i] = \{(E_{N_1} \otimes t, E_{N_2} \otimes t) | t \in \phi[i]\}$ and show that $\bigcup_{i=1}^{\#\phi} \psi[i] = E_{N_1} \otimes \bigcup_{i=1}^{\#\phi} \phi[i] \times E_{N_2} \otimes \bigcup_{i=1}^{\#\phi} \phi[i]$. It follows by Theorem 5.5 that $(E_{N_1} \otimes \bigcup_{i=1}^{\#\phi} \phi[i]) \times (E_{N_2} \otimes \bigcup_{i=1}^{\#\phi} \phi[i])$ is a measurable rectangle [14] in $F_{N_1}(E_{N_1} \otimes \alpha) \times F_{N_2}(E_{N_2} \otimes \alpha)$. From the above, and the product measure theorem [14] it can be shown that $F_{N_1}(E_{N_1} \otimes \alpha)(E_{N_2} \otimes \bigcup_{i=1}^{\#\phi} \phi[i]) \cdot f_{N_2}(E_{N_2} \otimes \alpha)(E_{N_2} \otimes \phi[i]) = \sum_{i=1}^{\#\phi} f_{N_1}(E_{N_1} \otimes \alpha)(E_{N_1} \otimes \phi[i]) \cdot f_{N_2}(E_{N_2} \otimes \alpha)(E_{N_2} \otimes \phi[i])$.

Theorem 5.7. There exists a unique extension of $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ to the cone-$\sigma$-field $F_{N_1} \otimes F_{N_2}(\alpha)$.

Proof sketch: We extend $C_E(D_{N_1} \otimes D_{N_2}(\alpha))$ in a stepwise manner to a set obtained by first adding all complements of the elements in $C_E(D_{N_1} \otimes D_{N_2}(\alpha))$, then adding the finite intersections of the new elements and finally adding finite unions of disjoint elements. For each step we extend $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ and show that the extension is $\sigma$-additive. We end up with a finite measure on the field generated by $C_E(D_{N_1} \otimes D_{N_2}(\alpha))$. By the extension theorem [14] it follows that this measure can be uniquely extended to a measure on $F_{N_1} \otimes F_{N_2}(\alpha)$.

Corollary 5.8. Let $f_{N_1} \otimes f_{N_2}(\alpha)$ be the unique extension of $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ to the cone-$\sigma$-field $F_{N_1} \otimes F_{N_2}(\alpha)$. Then $f_{N_1} \otimes f_{N_2}(\alpha)$ is a conditional probability measure on $F_{N_1} \otimes F_{N_2}(\alpha)$.

Proof sketch: We first show that $f_{N_1} \otimes f_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \leq 1$. When $f_{N_1} \otimes f_{N_2}(\alpha)$ is a measure on $F_{N_1} \otimes F_{N_2}(\alpha)$ such that $f_{N_1} \otimes f_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \leq 1$ we can show that $f_{N_1} \otimes f_{N_2}(\alpha)$ is a conditional probability measure on $F_{N_1} \otimes F_{N_2}(\alpha)$.

5.4. Composition of probabilistic component executions

We may now lift the $\otimes$-operator to probabilistic component executions. Let $I_{N_1}$ and $I_{N_2}$ be probabilistic component executions such that $N_1 \cap N_2 = \emptyset$. For any $\alpha \in B_{N_1 \cup N_2}$ we define:

(26) $I_{N_1} \otimes I_{N_2}(\alpha) \overset{\text{def}}{=} (D_{N_1} \otimes D_{N_2}(\alpha), F_{N_1} \otimes F_{N_2}(\alpha), f_{N_1} \otimes f_{N_2}(\alpha))$

where $f_{N_1} \otimes f_{N_2}(\alpha)$ is defined to be the unique extension of $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ to $F_{N_1} \otimes F_{N_2}(\alpha)$.

Theorem 5.9. $I_{N_1} \otimes I_{N_2}$ is a probabilistic component execution of $N_1 \cup N_2$.

Proof sketch: This can be shown from definitions (22) and (23) and Corollary 5.8.

6. Denotational representation of a component with a notion of risk

For any disjoint set of interfaces $N$ we define:

$A_N \overset{\text{def}}{=} \bigcup_{n \in N} A_n$

$c_{N} \overset{\text{def}}{=} \bigcup_{n \in N} c_{n}$

$r_{N} \overset{\text{def}}{=} \bigcup_{n \in N} r_{n}$

27
The reason why we can take the union of functions with disjoint domains is that we understand a function as a set of maplets. A maplet is a pair of two elements corresponding to the argument and the result of a function. For example the following set of three maplets

\{(e_1 \mapsto f(e_1)), (e_2 \mapsto f(e_2)), (e_2 \mapsto f(e_2))\}

characterises the function \(f \in \{e_1, e_2, e_3\} \to S\) uniquely. The arrow \(\mapsto\) indicates that the function yields the element to the right when applied to the element to the left [3].

We define the semantic representation of a component analogous to that of an interface, except that we now have a set of interfaces \(N\), instead of a single interface \(n\):

**Definition 6.1 (Semantics of a component).** A component is represented by a quadruple

\[(I_N, A_N, cv_N, rf_N)\]

consisting of its probabilistic component execution, its assets, consequence function and risk function, as explained above.

We define composition of components formally as:

**Definition 6.2 (Composition of components).** Given two components \(N_1\) and \(N_2\) such that \(N_1 \cap N_2 = \emptyset\). We define their composition \(N_1 \otimes N_2\) by

\[(I_{N_1} \otimes I_{N_2}, A_{N_1} \cup A_{N_2}, cv_{N_1} \cup cv_{N_2}, rf_{N_1} \cup rf_{N_2})\]

### 7. Hiding

In this section we explain how to formally represent hiding in a denotational semantics with risk. As explained in Section 2.5 we must take care not to hide incidents that affect assets belonging to externally observable interfaces, when we hide internal interactions. An interface is externally observable if it interacts with interfaces in the environment. We define operators for hiding assets and interface names from a component name and from the semantic representation of a component. The operators are defined in such a way that partial hiding of internal interaction is allowed. Thus internal events that affect assets belonging to externally observable interfaces may remain observable after hiding. Note that hiding of assets and interface names is optional. The operators defined below simply makes it possible to hide e.g. all assets belonging to a certain interface \(n\), as well as all events in an execution where \(n\) is the active party. We sketch the proof strategies for the lemmas and theorems in this section and refer to Appendix B for the full proofs.

Until now we have identified a component by the set of names of its interfaces. This has been possible because an interface is uniquely determined by its name, and the operator for composition is both associative and commutative. Hence, until now it has not mattered in which order the interfaces and resulting components have been composed. When we in the following introduce two hiding operators this becomes however an issue. For example, consider a component identified by

\[N \overset{\text{def}}{=} \{c_1, c_2, c_3\}\]
Then we need to distinguish the component \( \delta c_2 : N \), obtained from \( N \) by hiding interface \( c_2 \), from the component \( \{c_1, c_3\} \).

To do that we build the hiding information into the name of a component obtained with the use of hiding operators. A component name is from now one either

(a) a set of interface names,
(b) of the form \( \delta n : N \) where \( N \) is a component name and \( n \) is an interface name,
(c) of the form \( \sigma a : N \) where \( N \) is a component name and \( a \) is an asset, or
(d) of the form \( N_1 \oplus N_2 \) where \( N_1 \) and \( N_2 \) are component names and at least one of \( N_1 \) or \( N_2 \) contains a hiding operator.

Since we now allow hiding operators in component names we need to take this into consideration when combining them. We define a new operator for combining two component names \( N_1 \) and \( N_2 \) as follows:

\[
N_1 \oplus N_2 \overset{\text{def}}{=} \begin{cases} N_1 \cup N_2 & \text{if neither } N_1 \text{ nor } N_2 \text{ contain hiding operators} \\ N_1 \oplus N_2 & \text{otherwise} \end{cases}
\]

By \( \text{in}(N) \) we denote the set of all hidden and not hidden interface names occurring in the component name \( N \). We generalise definitions (18) to (21) to component names with hidden assets and interface names as follows:

\[
\begin{align*}
E_{\sigma a} : N & \overset{\text{def}}{=} E_N \\
E_{\delta n} : N & \overset{\text{def}}{=} \text{in}(N) \setminus \{n\} \\
E_{\delta n}^\dagger : N & \overset{\text{def}}{=} E_{\delta n}^\dagger \\
E_{\delta n}^\dagger : N & \overset{\text{def}}{=} E_{\delta n}^\dagger \setminus \{n\} \\
H_{\sigma a} : N & \overset{\text{def}}{=} H \cap E_{\sigma a} : N \overset{\omega}{=} H \cap E_{\delta n} : N \overset{\omega}{=} \\
H_{\delta n} : N & \overset{\text{def}}{=} H \cap \text{in}(N) \\
B_{\sigma a} : N & \overset{\text{def}}{=} B_N \\
B_{\delta n} : N & \overset{\text{def}}{=} (E_{\delta n} \setminus \{n\} \cup \text{in}(N)) \otimes B_N \\
\end{align*}
\]

**Definition 7.1 (Hiding of interface in a probabilistic component execution).**

Given an interface name \( n \) and a probabilistic component execution \( I_N \) we define:

\[
\begin{align*}
\delta n : I_N(\alpha) & \overset{\text{def}}{=} (D_{\delta n} : N(\alpha), F_{\delta n} : N(\alpha), f_{\delta n} : N(\alpha)) \\
where \quad D_{\delta n} : N(\alpha) & \overset{\text{def}}{=} \{E_{\delta n} : N \otimes t \mid t \in D_N(\delta n : \alpha)\} \\
F_{\delta n} : N(\alpha) & \overset{\text{def}}{=} \sigma(C_E(D_{\delta n} : N(\alpha))) \text{ i.e., the cone-\(\sigma\)-field of } D_{\delta n} : N(\alpha) \\
f_{\delta n} : N(\alpha)(c) & \overset{\text{def}}{=} f_N(\delta n : \alpha)(\{t \in D_N(\delta n : \alpha) \mid E_{\delta n} : N \otimes t \in c\}) \\
\delta n : \alpha & \overset{\text{def}}{=} (E_{\delta n} \setminus \{n\} \cup \text{in}(N)) \otimes \alpha \\
\end{align*}
\]

When hiding an interface name \( n \) from a queue history \( \alpha \), as defined in the last line of Definition 7.1, we filter away the external input to \( n \) but keep all internal transmissions, including those sent to \( n \). This is because we still need the information about the internal interactions involving the hidden interface to compute the probability of interactions it is involved in, after the interface is hidden from the outside.
Lemma 7.2. If $I_N$ is a probabilistic component execution and $n$ is an interface name, then $\delta n : I_N$ is a probabilistic component execution.

Proof sketch: We must show that: (1) $D_{\delta n : N}(\alpha)$ is a set of well-formed traces; (2) $\mathcal{F}_{\delta n : N}(\alpha)$ is the cone-$\sigma$-field of $D_{\delta n : N}(\alpha)$; and (3) $f_{\delta n : N}(\alpha)$ is a conditional probability measure on $\mathcal{F}_{\delta n : N}(\alpha)$. (1) If a trace is well-formed it remains well-formed after filtering away events with the hiding operator, since hiding interface names in a trace does not affect the ordering of events. The proof of (2) follows straightforwardly from Definition 7.1.

In order to show (3), we first show that $f_{\delta n : N}(\alpha)$ is a measure on $\mathcal{F}_{\delta n : N}(\alpha)$. In order to show this, we first show that the function $f_{\delta n : N}$ is well defined. I.e., for any $c \in \mathcal{F}_{\delta n : N}(\alpha)$ we show that $\{t \in D_{N}(\delta n : \alpha) | E_{\delta n : N} \otimes t \in c\} \in \mathcal{F}_{N}(\delta n : \alpha)$. We then show that $f_{N}(\delta n : \alpha)(\emptyset) = 0$ and that $f_{N}(\delta n : \alpha)$ is $\sigma$-additive. Secondly, we show that $f_{\delta n : N}(\alpha)(D_{\delta n : N}(\alpha)) \leq 1$. When $f_{\delta n : N}(\alpha)$ is a measure on $\mathcal{F}_{\delta n : N}(\alpha)$ such that $f_{\delta n : N}(\alpha)(D_{\delta n : N}(\alpha)) \leq 1$ we can show that $f_{\delta n : N}(\alpha)$ is a conditional probability measure on $\mathcal{F}_{\delta n : N}(\alpha)$.

Definition 7.3 (Hiding of component asset). Given an asset $a$ and a component $(I_N, A_N, cv_N, rf_N)$ we define:

$$\sigma a : (I_N, A_N, cv_N, rf_N) \overset{\text{def}}{=} (I_N, \sigma a : A_N, \sigma a : cv_N, \sigma a : rf_N)$$

where

$$\sigma a : A_N \overset{\text{def}}{=} A_N \setminus \{a\}$$

$$\sigma a : cv_N \overset{\text{def}}{=} cv_N \setminus \{(e, a) \mapsto c | e \in \mathcal{E} \land c \in \mathbb{N}\}$$

$$\sigma a : rf_N \overset{\text{def}}{=} rf_N \setminus \{(c, p, a) \mapsto r | c, r \in \mathbb{N} \land p \in [0, 1]\}$$

As explained in Section 6 we see a function as a set of maplets. Hence, the consequence and risk function of a component with asset $a$ hidden is the set-difference between the original functions and the set of maplets that has $a$ as one of the parameters of its first element.

Theorem 7.4. If $N$ is a component and $a$ is an asset, then $\sigma a : N$ is a component.

Proof sketch: This can be shown from Definition 7.3 and Definition 6.1.

We generalise the operators for hiding interface names and assets to the hiding of sets of interface names and sets of assets in the obvious manner.

Definition 7.5 (Hiding of component interface). Given an interface name $n$ and a component $(I_N, A_N, cv_N, rf_N)$ we define:

$$\delta n : (I_N, A_N, cv_N, rf_N) \overset{\text{def}}{=} (\delta n : I_N, \sigma A_n : A_N, \sigma A_n : cv_N, \sigma A_n : rf_N)$$

Theorem 7.6. If $N$ is a component and $n$ is an interface name, then $\delta n : N$ is a component.

Proof sketch: This can be show from Lemma 7.2 and Theorem 7.4.

Since, as we have shown above, components are closed under hiding of assets and interface names, the operators for composition of components, defined in Section 5, are not affected by the introduction of hiding operators. We impose the restriction that two components can only be composed by $\otimes$ if their sets of interface names are disjoint, independent of whether they are hidden or not.
8. Related work

In this section we place our work in relation to ongoing research within related areas such as security modelling and approaches to representing probabilistic components. We also relate our component model to a taxonomy of component models [28].

8.1. Security modelling

There are a number of proposals to integrate security requirements into the requirements specification, such as SecureUML and UMLsec. SecureUML [32] 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 specifies security requirements for well-behaved applications in predictable environments. UMLsec [19] is an extension to UML that enables the modelling of security-related features such as confidentiality and access control. Neither of these two approaches have particular focus on component-oriented specification.

Khan and Han [22] characterise security properties of composite systems, based on a security characterisation framework for basic components [23, 20, 21]. They define a compositional security contract CsC for two components, which is based on the compatibility between their required and ensured security properties. They also give a guideline for constructing system level contracts, based on several CsCs. This approach has been designed to capture security properties, while our focus is on integrating risks into the semantic representation of components.

8.2. Probabilistic components

In order to model systems that are both reactive and probabilistic the external nondeterminism caused by the environment must be resolved. Our idea to use queue histories to resolve the external nondeterminism of probabilistic components is inspired by the use of schedulers, also known as adversaries, which is a common way to resolve external nondeterminism in reactive systems [8, 44, 7]. A scheduler specifies how to choose between nondeterministic alternatives.

Segala and Lynch [44, 43] use a randomised scheduler to model input from an external environment and resolve the nondeterminism of a probabilistic I/O automaton. They define a probability space [9] for each probabilistic execution of an automaton, given a scheduler.

Alfaro et al. [7] present a probabilistic model for variable-based systems with trace semantics similar to that of Segala and Lynch. They define a trace as a sequence of states, and a state as an assignment of values to a set of variables. Each component has a set of controlled variables and a set of external variables. Alfaro et al. represent a system by a set of probability distributions on traces, called bundles. They use schedulers to choose the initial and updated values for variables. Unlike the model of Segala and Lynch, their allows multiple schedulers to resolve the nondeterminism of each component. The key idea is to have separate schedulers for the controlled and external variables to ensure that variable behaviours are probabilistically independent. According to Alfaro et al. this ensures so called deep compositionality of their system model.

In a system model with deep compositionality the semantics of a composite system can be obtained from the semantics of its constituents. In contrast, shallow compositionality provides the means to specify composite components syntactically [7].
semantics of a composite specification is obtained from the syntactic composite specification, but the semantics of this composition is not directly related to that of its constituents.

Seidel uses a similar approach in her extension of CSP with probabilities [45]. Internal nondeterministic choice is replaced by probabilistic choice. A process is represented by a conditional probability measure that, given a trace produced by the environment, returns a probability distribution over traces.

An alternative approach to handle external nondeterminism in probabilistic, reactive systems is to treat the assignment of probabilities of alternative choices as a refinement. This approach is used for example in probabilistic action systems [46, 52], where nondeterministic choices are replaced by probabilistic choices. A nondeterministic action system is transformed to a (deterministic) probabilistic system through the distribution of probabilistic information over alternative behaviours.

Our decision to use a cone-based probability space to represent probabilistic systems is inspired by the work on probabilistic I/O automata [44, 43] by Segala and Lynch and probabilistic sequence diagrams (pSTAIRS) [38, 37] by Refsdal. Segala uses probability spaces whose $\sigma$-fields are cone-$\sigma$-fields to represent fully probabilistic automata, that is, automata with probabilistic choice but without non-determinism. In pSTAIRS [37] the ideas of Segala and Lynch is applied to the trace-based semantics of STAIRS [15, 42, 41]. A probabilistic system is represented as a probability space where the $\sigma$-field is generated from a set of cones of traces describing component interactions. In pSTAIRS all choices (nondeterministic, mandatory and probabilistic) are global, that is, the different types of choices may only be specified for closed systems, and there is no nondeterminism stemming from external input.

Since we wish to represent the behaviour of a component independently of its environment we cannot use global choice operators of the type used in pSTAIRS. We build upon the work of pSTAIRS and extend its probabilistic model to open, reactive components. We define probabilistic choice at the level of individual component interfaces and use queue histories to resolve external nondeterminism. Hence, we represent a probabilistic component execution as a function of queue histories, instead of by a single probability space.

8.3. Component models

Lau and Wang [28] have surveyed current component models and classified them into a taxonomy based on commonly accepted criteria for successful component-based development [28]. According to the criteria components should be pre-existing reusable software units which developers can reuse to compose software for different applications. Furthermore components should be composable into composite components which, in turn, can be composed with (composite) components into even larger composites, and so on. These criteria necessitate the use of a repository in the design phase. It must be possible to deposit and retrieve composites from a repository, just like any components.

Lau and Wang [28] divide current component models into four categories based on their facilities for composition during the various phases of a component life cycle. According to their evaluation, no current component model provides mechanisms for composition in all phases. They propose a component model with explicitly defined component connectors, to ensure encapsulation of control, thereby facilitating compositionality during all phases of development.
Our component model is purely semantic. It can be used to represent component implementations. We have at this point not defined a syntax for specifying components. The purpose of the presented component model is to form the necessary basis for building applied tools and methods for component-based risk analysis. Current approaches to specifying probabilistic components, discussed in Section 8.2, can be used as a basis for a specification language needed in such a method.

9. Conclusion

We have presented a component model that integrates component risks as part of the component behaviour. The component model is meant to serve as a formal basis for component-based risk analysis. To ensure modularity of our component model we represent a stakeholder by the component interface, and identify assets on behalf of component interfaces. Thus we avoid referring to concepts that are external to a component in the component model.

In order to model the probabilistic aspect of risk, we represent the behaviour of a component by a probability distribution over traces. We use queue histories to resolve both internal and external non-determinism. The semantics of a component is the set of probability spaces given all possible queue histories of the component.

We define composition in a fully compositional manner: The semantics of a composite component is completely determined by the semantics of its constituents. Since we integrate the notion of risk into component behaviour, we obtain the risks of a composite component by composing the behavioural representations of its sub-components.

The component model provides a foundation for component-based risk analysis, by conveying how risks manifest themselves in an underlying component implementation. By component-based risk analysis we mean that risks are identified, analysed and documented at the component level, and that risk analysis results are composable.

Our semantic model is not tied to any specific syntax or specification technique. At this point we have no compliance operator to check whether a given component implementation complies with a component specification. In order to be able to check that a component implementation fulfils a requirement to protection specification we would like to define a compliance relation between specifications in STAIRS, or another suitable specification language, and components represented in our semantic model.

We believe that a method for component-based risk analysis will facilitate the integration of risk analysis into component-based development, and thereby make it easier to predict the effects on component risks caused by upgrading or substituting sub-parts.

Acknowledgements

The research presented in this report has been partly funded by the Research Council of Norway through the research projects COMA 160317 (Component-oriented model-based security analysis) and DIGIT 180052/S10 (Digital interoperability with trust), and partly by the EU 7th Research Framework Programme through the Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS).

We would like to thank Bjarte M. Østvold for creating lpchk: a proof analyser for proofs written in Lamport’s style [27] that checks consistency of step labelling and performs parentheses matching, and also for proof reading and useful comments.
References


A. Auxiliary definitions

Here is a summary of the definitions we use to prove the results in Appendix B.

A.1. Sets

We use \( \mathbb{N} \) to denote the set of \textit{natural numbers}:

\[
\mathbb{N} \overset{\text{def}}{=} \{0, 1, 2, 3, \ldots, n, n+1, \ldots\}
\]

and \( \mathbb{N}_+ \) to denote the set of strictly positive natural numbers:

\[
\mathbb{N}_+ = \mathbb{N} \setminus \{0\}
\]

The cross product of two sets \( A \) and \( B \), denoted \( A \times B \), is the set of all pairs where the first element is in \( A \) and the second element is in \( B \). Formally,

\[
A \times B \overset{\text{def}}{=} \{(a, b) \mid a \in A, b \in B\}
\]

A.2. Logic

We sometimes use \texttt{let} statements in order to make substitution in logical formulas more readable. Any \texttt{let} statement is on the following form

\[
\text{let } v_1 = e_1 \\
\vdots \\
\text{let } v_n = e_n \\
\text{in } P
\]

where \( v_1, \ldots, v_n \) are logical variables, \( e_1, \ldots, e_n \) are expressions, and \( P \) is a formula. We require that the variables are distinct

\[ j \neq k \Rightarrow v_j \neq v_k \]
and that $v_j$ is not free in the expression $e_k$ if $k \leq j$. The let statement can be understood as a shorthand for the formula

$$\exists v_1, \ldots, v_n : v_1 = e_1 \land \cdots \land v_n = e_n \land P$$

We often use where to introduce auxiliary identifiers $v_1, \ldots, v_n$. The where statement is of the form

$$P_1 \ \text{where} \ v_1, \ldots, v_n \ \text{so that} \ P_2$$

where $v_1, \ldots, v_n$ are logical variables and $P_1, P_2$ are formulas. It can be understood as a shorthand for the formula

$$\exists v_1, \ldots, v_n : P_1 \land P_2$$

### A.3. Probability theory

We introduce some basic concepts from measure theory [14, 12, 9, 31] that we use to define probabilistic executions.

**Definition A.1 (Countable set).** A set is countable if its elements can be arranged into a finite or infinite sequence [25].

**Definition A.2 (Countably additive function).** A function $f$ on a set $D$ is countably additive (also referred to as $\sigma$-additive) if for every sequence $\omega$ of disjoint sets in $D$ whose union is also in $D$ we have

$$f(\bigcup_{i=1}^{\#\omega} \omega[i]) = \sum_{i=1}^{\#\omega} f(\omega[i])$$

**Definition A.3 (Field).** Given a set $D$, a collection $\mathcal{F} \subset \mathcal{P}(D)$ is called a field if and only if $\emptyset \in \mathcal{F}$, $D \in \mathcal{F}$ and for all $A$ and $B$ in $\mathcal{F}$, we have $A \cup B \in \mathcal{F}$ and $B \setminus A \in \mathcal{F}$. A field generated by a set of subsets $C$ of $D$, denoted by $\mathcal{F}(C)$, is the smallest field containing $C$, that is, the intersection of all fields containing $C$.

**Definition A.4 (Sigma field ($\sigma$-field)).** A field $\mathcal{F}$ with regard to a set $D$ is called a $\sigma$-field if for any sequence $\omega$ of sets in $\mathcal{F}$ we have $\bigcup_{i=1}^{\#\omega} \omega[i] \in \mathcal{F}$.

A $\sigma$-field generated by a set $C$ of subsets of $D$ is denoted by $\sigma(C)$.

**Definition A.5 (Measurable space).** Let $D$ be a non-empty set, and $\mathcal{F}$ a $\sigma$-field over $D$. Then $(D, \mathcal{F})$ is called a measurable space, and the sets in $\mathcal{F}$ are called measurable sets.

**Definition A.6 (Measure).** Let $D$ be a non-empty set and $\mathcal{F}$ be a $\sigma$-field over $D$. A measure $\mu$ on $\mathcal{F}$ is a function that assigns a non-negative real value (possibly $\infty$) to each element of $\mathcal{F}$ such that

1. $\mu(\emptyset) = 0$
2. $\mu$ is $\sigma$-additive

The measure $\mu$ is finite if $\mu(D) < \infty$. It is $\sigma$-finite if and only if $D$ can be written as $\bigcup_{i=1}^{\#\phi} \phi[i]$, where $\phi[i] \in \mathcal{F}$ and $\mu(\phi[i]) < \infty$ for all $i$.  

38
Definition A.7 (Probability measure). Let $D$ be a non-empty set and $\mathcal{F}$ be a $\sigma$-field over $D$. A probability measure $\mu$ is a measure on $\mathcal{F}$ such that

$$\mu(D) = 1$$

Definition A.8 (Measure space). A measure space is a triple $(D, \mathcal{F}, f)$, where $(D, \mathcal{F})$ is a measurable space, and $f$ is a measure on $(D, \mathcal{F})$.

Definition A.9 (Probability space). A probability space is a triple $(D, \mathcal{F}, f)$ where $(D, \mathcal{F})$ is a measurable space, and $f$ is a probability measure on $\mathcal{F}$.

Definition A.10 (Measurable rectangle). Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be $\sigma$-fields over $D_1$ and $D_2$. Let $D$ be the cartesian product of $D_1$ and $D_2$; $D_1 \times D_2$. A measurable rectangle in $D$ is a set $A = A_1 \times A_2$, such that $A_1 \in \mathcal{F}_1$ and $A_2 \in \mathcal{F}_2$. The smallest $\sigma$-field containing all measurable rectangles of $D$ is called the product $\sigma$-field, denoted by $\mathcal{F}_1 \times \mathcal{F}_2$.

Definition A.11 (Extensions of a set). Let $C$ be a set of subsets of a non-empty set $D$. We define a stepwise extension of $C$ as follows:

1. $F_1(C) \overset{\text{def}}{=} C \cup \{\emptyset\} \cup \{D \setminus A \mid A \in C\}
2. F_2(C) \overset{\text{def}}{=} \\bigcap_{i=1}^{n} A_i \mid \forall i \in [1..n]: A_i \in F_1(C)\}
3. F_3(C) \overset{\text{def}}{=} \{\bigcup_{i=1}^{n} A_i \mid \forall i \in [1..n]: A_i \in F_2(C) \land \\
\forall j, m \in [1..n]: j \neq m \Rightarrow A_j \cap A_m = \emptyset\}$

B. Proofs

In this section we state all the lemmas and theorems and provide proofs for the ones that are not directly based on other sources. All proofs are written in Lamport’s style for writing proofs [27]. This is a style for structuring formal proofs in a hierarchical manner in LaTeX [26], similar to that of natural deduction [51, 35]. As observed by Lamport the systematic structuring of proofs is essential to getting the results right in complex domains which it is difficult to have good intuitions about. We have had several iterations of formulating operators for component composition, attempting to prove them correct and then reformulating them when the structured proof style uncovered inconsistencies. These iterations where repeated until the definitions where proven to be correct.

The following tables give the page number for each theorem, lemma and corollary. If a theorem, lemma or corollary is used in proofs of other results we also include references to the results using it.

---

7The product $\sigma$-field of $\mathcal{F}_1$ and $\mathcal{F}_2$ is commonly denoted by $\mathcal{F}_1 \times \mathcal{F}_2$, but we use $\mathcal{F}_1 \times \mathcal{F}_2$ to avoid confusion with the cross product of $\mathcal{F}_1$ and $\mathcal{F}_2$.  

---

39
**B.1. Measure theory**

In the following we present some basic results from measure theory that we use in the later proofs. These are taken from other sources [9, 14, 24, 31, 1], and the proofs can be found there.

**Proposition B.1.** Let \( C \) be a set of subsets of a non-empty set \( D \) and extend \( C \) to a set \( F_3(C) \) as defined in Definition A.11. Then \( C \subseteq \text{FOCUS}_1(C) \subseteq F_2(C) \subseteq F_3(C) \) and \( F(C) = F_3(C) \), that is \( F_3(C) \) is the field generated by \( C \).

**Lemma B.2.** Let \( C \) be a set of subsets of a non-empty set \( D \). Then \( \sigma(C) = \sigma(F(C)) \).

**Theorem B.3 (Extension theorem).** A finite measure \( \mu \) on a field \( F \) has a unique extension to the \( \sigma \)-field generated by \( F \). That is, there exists a unique measure \( \mu' \) on \( \sigma(F) \) such that for each element \( C \) of \( F \), \( \mu'(C) = \mu(C) \).

**Theorem B.4 (Product Measure Theorem).** Let \((D_1, \mathcal{F}_1, \mu_1)\) and \((D_2, \mathcal{F}_2, \mu_2)\) be two measure spaces where \( \mu_1 \) and \( \mu_2 \) are \( \sigma \)-finite. Let \( D = D_1 \times D_2 \) and \( \mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2 \). Then there is a unique measure \( \mu \) on \( \mathcal{F} \), such that

\[
\mu(A_1 \times A_2) = \mu_1(A_1) \cdot \mu_2(A_2)
\]

for every measurable rectangle \( A_1 \times A_2 \in \mathcal{F} \). The measure \( \mu \) is called the product of \( \mu_1, \mu_2 \) and the triplet \((D, \mathcal{F}, \mu)\) is called the product measure space.

**B.2. Probabilistic component execution**

In the following we state and prove some lemmas that we use to prove the main result in Section B.3; namely that we can construct a conditional probability measure on the cone-\( \sigma \)-field generated by the cone set obtained from the parallel execution of the trace sets of two probabilistic component executions.
<table>
<thead>
<tr>
<th>Result</th>
<th>Page</th>
<th>Used in proof of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lemma B.15</td>
<td>Page 49</td>
<td>L B.16</td>
</tr>
<tr>
<td>Lemma B.16</td>
<td>Page 49</td>
<td>L 5.5</td>
</tr>
<tr>
<td>Lemma B.17</td>
<td>Page 50</td>
<td>C B.18</td>
</tr>
<tr>
<td>Corollary B.18</td>
<td>Page 50</td>
<td>L B.26,L B.35,L B.36</td>
</tr>
<tr>
<td>Observation B.19</td>
<td>Page 50</td>
<td>L 5.4</td>
</tr>
<tr>
<td>Lemma B.20</td>
<td>Page 52</td>
<td>L 5.4</td>
</tr>
<tr>
<td>Lemma 5.4</td>
<td>Page 53</td>
<td>L B.26</td>
</tr>
<tr>
<td>Lemma B.21</td>
<td>Page 60</td>
<td>L B.24,L B.26</td>
</tr>
<tr>
<td>Lemma B.22</td>
<td>Page 61</td>
<td>L B.28,L B.23</td>
</tr>
<tr>
<td>Lemma B.23</td>
<td>Page 64</td>
<td>L B.24</td>
</tr>
<tr>
<td>Lemma B.24</td>
<td>Page 67</td>
<td>L B.26</td>
</tr>
<tr>
<td>Lemma B.25</td>
<td>Page 69</td>
<td>L B.26, L 5.6,L 5.6</td>
</tr>
<tr>
<td>Lemma B.26</td>
<td>Page 70</td>
<td>T 5.5</td>
</tr>
<tr>
<td>Theorem 5.5</td>
<td>Page 71</td>
<td>L B.28,L 5.6</td>
</tr>
<tr>
<td>Lemma B.27</td>
<td>Page 72</td>
<td>L B.28</td>
</tr>
<tr>
<td>Lemma B.28</td>
<td>Page 73</td>
<td>L 5.6,L B.29</td>
</tr>
<tr>
<td>Lemma 5.6</td>
<td>Page 76</td>
<td>T 5.7,C 5.8,L B.29</td>
</tr>
<tr>
<td>Lemma B.29</td>
<td>Page 78</td>
<td>L B.32,T 5.7</td>
</tr>
<tr>
<td>Lemma B.30</td>
<td>Page 81</td>
<td>C B.31</td>
</tr>
<tr>
<td>Corollary B.31</td>
<td>Page 82</td>
<td>L B.32</td>
</tr>
<tr>
<td>Lemma B.32</td>
<td>Page 83</td>
<td>L B.33,T 5.7</td>
</tr>
<tr>
<td>Lemma B.33</td>
<td>Page 88</td>
<td>T 5.7</td>
</tr>
<tr>
<td>Theorem 5.7</td>
<td>Page 89</td>
<td>C 5.8</td>
</tr>
<tr>
<td>Lemma B.34</td>
<td>Page 89</td>
<td>C 5.8,L B.39</td>
</tr>
<tr>
<td>Corollary 5.8</td>
<td>Page 91</td>
<td>T 5.9</td>
</tr>
<tr>
<td>Theorem 5.9</td>
<td>Page 92</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: List of results in Section B.3

<table>
<thead>
<tr>
<th>Result</th>
<th>Page</th>
<th>Used in proof of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lemma B.35</td>
<td>Page 92</td>
<td>C B.37</td>
</tr>
<tr>
<td>Lemma B.36</td>
<td>Page 94</td>
<td>C B.37</td>
</tr>
<tr>
<td>Corollary B.37</td>
<td>Page 96</td>
<td>L B.38</td>
</tr>
<tr>
<td>Lemma B.38</td>
<td>Page 96</td>
<td>L 7.2</td>
</tr>
<tr>
<td>Lemma 7.2</td>
<td>Page 100</td>
<td>T 7.6</td>
</tr>
<tr>
<td>Theorem 7.4</td>
<td>Page 100</td>
<td>T 7.6</td>
</tr>
<tr>
<td>Theorem 7.6</td>
<td>Page 101</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: List of results in Section B.4.
Lemma B.5. (Adapted from Lemma 4.2.4 in Segala [43, p. 54]). Let $I_N$ be a probabilistic component execution as defined in Definition 5.3 and let $\alpha$ be a complete queue history $\alpha \in B_N$. Then

\begin{align*}
(1) & \forall t_1, t_2 \in D_N(\alpha) : t_1 \sqsubseteq t_2 \Rightarrow c(t_2, D_N(\alpha)) \subseteq c(t_1, D_N(\alpha)) \\
(2) & \forall t_1, t_2 \in D_N(\alpha) : t_1 \not\sqsubseteq t_2 \land t_2 \not\sqsubseteq t_1 \Rightarrow c(t_2, D_N(\alpha)) \cap c(t_1, D_N(\alpha)) = \emptyset
\end{align*}

Proof. Follows from Definition 3.1. \qed

Corollary B.6. Let $I_N$ be a probabilistic component execution as defined in Definition 5.3 and let $\alpha$ be a complete queue history $\alpha \in B_N$. Then

\[ \forall c_1, c_2 \in C_E(D_N(\alpha)) : c_1 \cap c_2 \neq \emptyset \Rightarrow c_1 \subseteq c_2 \lor c_2 \subseteq c_1 \]

Proof:
\begin{enumerate}
\item Assume: $c_1 \subseteq C_E(D_N(\alpha)) \land c_2 \subseteq C_E(D_N(\alpha))$
\item Prove: $c_1 \cap c_2 \neq \emptyset$ \Rightarrow $c_1 \subseteq c_2 \lor c_2 \subseteq c_1$
\item Assume: $c_1 \cap c_2 \neq \emptyset$
\item Prove: $c_1 \subseteq c_2 \lor c_2 \subseteq c_1$
\end{enumerate}

\begin{enumerate}
\item Case: $c_1 \subseteq C_E(D_N(\alpha)) \land c_2 \subseteq C_E(D_N(\alpha))$
\item Proof: By assumption (3)1 and Definition 3.3 it follows that at least one of $c_1$ or $c_2$ contains only one trace. Since it is also the case, by assumption (2)1, that $c_1$ and $c_2$ shares at least one element, the required result follows from elementary set theory.
\item Case: $c_1 \subseteq C(D_N(\alpha)) \land c_2 \subseteq C(D_N(\alpha))$
\item Proof: By assumption (4)1.
\item Let: $t_1 \in H, t_2 \in H$ such that $c_1 = c(t_1, D_N(\alpha)) \land c_2 = c(t_2, D_N(\alpha))$
\item Proof: By (4)2.
\item Proof: By assumption (2)1, (4)2 and Lemma B.5 (2).
\item Case: $t_2 \sqsubseteq t_1$
\item Proof: By assumption (5)2 and Lemma B.5 (1) $(c(t_1, D_N(\alpha)) \subseteq c(t_2, D_N(\alpha)))$ and $\lor$-introduction.
\item Case: $t_1 \sqsubseteq t_2$
\item Proof: By assumption (5)3 and Lemma B.5 (1) $(c(t_2, D_N(\alpha)) \subseteq c(t_1, D_N(\alpha)))$ and $\lor$-introduction.
\item Proof: By (5)1, (5)2, (5)3 and $\lor$ elimination.
\item Proof: By (4)1, (4)2 and the rule of replacement [51].
\item Proof: By assumption (1)1, the cases (3)1 and (3)2 are exhaustive.
\item Proof: $\Rightarrow$-introduction.
\end{enumerate}
Corollary B.7. Let $I_N$ be a probabilistic component execution as defined in Definition 5.3 and let $\alpha$ be a complete queue history $\alpha \in B_N$. Then any union of elements in $C_E(D_N(\alpha))$ can be described as a disjoint union of elements in $C_E(D_N(\alpha))$.

**Proof.** Follows from Corollary B.6.

Lemma B.8. Let $I_N$ be a probabilistic component execution as defined in Definition 5.3 and let $\alpha$ be a complete queue history $\alpha \in B_N$. Then

\[ \forall A, B \in F_N(\alpha) : A \subseteq B \Rightarrow f_N(\alpha)(A) \leq f_N(\alpha)(B) \]

**Proof.** Since $A \subseteq B$ we have $B = A \cup (B \cap (D_N(\alpha) \setminus A))$ where $A$ and $B \cap (D_N(\alpha) \setminus A)$ are disjoint. Therefore $f_N(\alpha)(B) = f_N(\alpha)(A) + f_N(\alpha)(B \cap (D_N(\alpha) \setminus A)) \geq f_N(\alpha)(A)$.

Lemma B.9. Let $I_N$ be a probabilistic component execution as defined in Definition 5.3, let $\alpha$ be a complete queue history $\alpha \in B_N$, and let $S$ be a non-empty set of finite prefixes of traces in $D_N(\alpha)$. Then

\[ \bigcup_{t \in S} c(t, D_N(\alpha)) \text{ is a countable union of elements in } C(D_N(\alpha)) \]

**Proof:**

1. **Assume:** $S \neq \emptyset \land \forall t \in S : \exists t' \in D_N(\alpha) : t \subseteq t' \land \#t \in \mathbb{N}$
   **Prove:** $\bigcup_{t \in S} c(t, D_N(\alpha))$ is a countable union of elements in $C(D_N(\alpha))$.

   1. **Assume:** $S \neq \emptyset \land \forall t \in S : \exists t' \in D_N(\alpha) : t \subseteq t' \land \#t \in \mathbb{N}$
      **Prove:** $\bigcup_{t \in S} c(t, D_N(\alpha))$ is a countable union of elements in $C(D_N(\alpha))$.

   1. **Assume:** $S \neq \emptyset \land \forall t \in S : \exists t' \in D_N(\alpha) : t \subseteq t' \land \#t \in \mathbb{N}$
      **Prove:** $\bigcup_{t \in S} c(t, D_N(\alpha))$ is a countable union of elements in $C(D_N(\alpha))$.

2. **Proof:** By assumption (1)1 and Definition 3.3.

3. **Proof:** By assumption (1)1, since time-stamps are rational numbers and we assume that interfaces are assigned a countable number of signals, we have a countable number of events, and the set of finite sequences formed from a countable set is countable [25].

4. **Proof:** By (3)2, Q.E.D.

Corollary B.10. Let $I_N$ be a probabilistic component execution as defined in Definition 5.3, let $\alpha$ be a complete queue history $\alpha \in B_N$, and let $S$ be a (possibly empty) set of finite prefixes of traces in $D_N(\alpha)$. Then

\[ \bigcup_{t \in S} c(t, D_N(\alpha)) \in F_N(\alpha) \]

**Proof:**
Lemma B.11. Let $I_N$ be a probabilistic component execution as defined in Definition 5.3 and let $\alpha$ be a complete queue history $\alpha \in B_N$. Then

$\forall t_1 \in (\mathcal{H} \cap \mathcal{E}^*) : c(t_1, D_N(\alpha)) \in C(D_N(\alpha)) \Rightarrow
D_N(\alpha) \setminus c(t_1, D_N(\alpha)) \text{ is a countable union of elements in } C_E(D_N(\alpha))$.

Proof:

(1.1) Assume: $t_1 \in (\mathcal{H} \cap \mathcal{E}^*)$
Prove: $c(t_1, D_N(\alpha)) \in C(D_N(\alpha)) \Rightarrow
D_N(\alpha) \setminus c(t_1, D_N(\alpha)) \text{ is a countable union of elements in } C_E(D_N(\alpha))$.

(2.1) Assume: $c(t_1, D_N(\alpha)) \in C(D_N(\alpha))$
Prove: $D_N(\alpha) \setminus c(t_1, D_N(\alpha)) \text{ is a countable union of elements in } C_E(D_N(\alpha))$.

(3.1) Let: $S = \{ t \in \mathcal{H} | \exists t_2 \in D_N(\alpha) : t \subseteq t_2 \land \#t \leq \#t_1 \land t \neq t_1 \}$

(3.2) Case: $S = \emptyset$

(4.1) $D_N(\alpha) \setminus c(t_1, D_N(\alpha)) = \emptyset$

(5.1) Assume: $D_N(\alpha) \setminus c(t_1, D_N(\alpha)) \neq \emptyset$
Prove: \top

(6.1) $\exists t \in D_N(\alpha) : t \notin c(t_1, D_N(\alpha))$
Prove: By assumption (5.1).

(6.2) Let: $t \in D_N(\alpha)$ such that $t \notin c(t_1, D_N(\alpha))$
Prove: By (6.1).

(6.3) $t_1 \not\subseteq t$
Prove: By (6.2) and Definition 3.1.

(6.4) $\exists t' \in \mathcal{H} : t' \subseteq t \land \#t' \leq \#t_1 \land t' \neq t_1$
Prove: By (6.3) and definition (2).

(6.5) Let: $t' \in \mathcal{H}$ such that $t' \subseteq t \land \#t' \leq \#t_1 \land t' \neq t_1$
Prove: By (6.4).

(6.6) $t' \in S$
Proof: By \( \langle 3 \rangle 1 \), \( \langle 6 \rangle 2 \) and \( \langle 6 \rangle 5 \).
\( \langle 6 \rangle 7 \). \( S \neq \emptyset \)
Proof: By \( \langle 6 \rangle 5 \) and \( \langle 6 \rangle 6 \).
\( \langle 6 \rangle 8 \). Q.E.D.
Proof: By assumption \( \langle 3 \rangle 2 \), \( \langle 6 \rangle 7 \) and \( \bot \) introduction.
\( \langle 5 \rangle 2 \). Q.E.D.
Proof: Proof by contradiction.
\( \langle 4 \rangle 2 \). \( D_N(\alpha) = c(t_1, D_N(\alpha)) \)
Proof: By \( \langle 4 \rangle 1 \) and elementary set theory.
\( \langle 4 \rangle 3 \). Q.E.D.
Proof: By assumption \( \langle 2 \rangle 1 \), \( \langle 4 \rangle 2 \) and the rule of replacement [51].
\( \langle 3 \rangle 3 \). Case: \( S \neq \emptyset \)
\( \langle 4 \rangle 1 \). \( D_N(\alpha) \setminus c(t_1, D_N(\alpha)) = \bigcup_{t \in S} c(t, D_N(\alpha)) \)
\( \langle 5 \rangle 1 \). \( \bigcup_{t \in S} c(t, D_N(\alpha)) \subseteq D_N(\alpha) \setminus c(t_1, D_N(\alpha)) \)
\( \langle 6 \rangle 1 \). Assume: \( t_2 \in \bigcup_{t \in S} c(t, D_N(\alpha)) \)
Prove: \( t_2 \in D_N(\alpha) \setminus c(t_1, D_N(\alpha)) \)
\( \langle 7 \rangle 1 \). \( t_2 \in D_N(\alpha) \land t_2 \notin c(t_1, D_N(\alpha)) \)
\( \langle 8 \rangle 1 \). \( t_2 \in D_N(\alpha) \)
Proof: By assumption \( \langle 6 \rangle 1 \), \( \langle 3 \rangle 1 \) and Definition 3.1.
\( \langle 8 \rangle 2 \). \( t_2 \notin c(t_1, D_N(\alpha)) \)
\( \langle 9 \rangle 1 \). \( t_1 \not\subseteq t_2 \)
\( \langle 10 \rangle 1 \). \( \exists t \in \mathcal{H} : t \subseteq t_2 \land \#t \leq \#t_1 \land t \neq t_2 \)
Proof: By assumption \( \langle 6 \rangle 1 \) and \( \langle 3 \rangle 1 \).
\( \langle 10 \rangle 2 \). Q.E.D.
Proof: By \( \langle 10 \rangle 1 \) and definition (2).
\( \langle 9 \rangle 2 \). Q.E.D.
Proof: By \( \langle 9 \rangle 1 \) and Definition 3.1.
\( \langle 8 \rangle 3 \). Q.E.D.
Proof: By \( \langle 8 \rangle 1 \) and \( \langle 8 \rangle 2 \) and \( \land \)-introduction.
\( \langle 7 \rangle 2 \). Q.E.D.
Proof: By \( \langle 7 \rangle 1 \) and elementary set theory.
\( \langle 6 \rangle 2 \). Q.E.D.
Proof: By \( \langle 6 \rangle 1 \) and \( \subseteq \)-rule [29]
\( \langle 5 \rangle 2 \). \( D_N(\alpha) \setminus c(t_1, D_N(\alpha)) \subseteq \bigcup_{t \in S} c(t, D_N(\alpha)) \)
\( \langle 6 \rangle 1 \). Assume: \( \exists t_2 \in \mathcal{H} : c(t_1, D_N(\alpha)) \land t_2 \notin \bigcup_{t \in S} c(t, D_N(\alpha)) \)
Prove: \( \bot \)
\( \langle 7 \rangle 1 \). Let: \( t_2 \) be a trace such that \( t_2 \in D_N(\alpha) \setminus c(t_1, D_N(\alpha)) \land t_2 \notin \bigcup_{t \in S} c(t, D_N(\alpha)) \)
Proof: By \( \langle 6 \rangle 1 \).
\( \langle 7 \rangle 2 \). \( t_1 \not\subseteq t_2 \)
Proof: By Definition 3.1 and the first conjunct of \( \langle 7 \rangle 1 \) which implies \( t_2 \notin c(t_1, D_N(\alpha)) \)
\( \langle 7 \rangle 3 \). \( \exists t \in \mathcal{H} : \#t \leq \#t_1 \land t \not\subseteq t_2 \land t \neq t_1 \)
Proof: By \( \langle 7 \rangle 2 \) and definition (2).
\( \langle 7 \rangle 4 \). \( t_2 \in \bigcup_{t \in S} c(t, D_N(\alpha)) \)
Proof: By \( \langle 7 \rangle 3 \) and \( \langle 3 \rangle 1 \).
\( \langle 7 \rangle 5 \). Q.E.D.
Proof: By \( \langle 7 \rangle 1 \), \( \langle 7 \rangle 4 \) and \( \bot \)-introduction.
\langle6\rangle 2. Q.E.D.  
Proof: Proof by contradiction.

\langle5\rangle 3. Q.E.D.  
Proof: By \langle5\rangle 1, \langle5\rangle 2 and \Rightarrow\text{-rule for sets} \[29].

\langle4\rangle 2. \bigcup_{t \in S} c(t, D_N(\alpha)) is a countable union of elements in \( C_E(D_N(\alpha)) \).

\langle5\rangle 1. \bigcup_{t \in S} c(t, D_N(\alpha)) is a countable union of elements in \( C(D_N(\alpha)) \).
Proof: By assumption \langle1\rangle 1, \langle3\rangle 1, assumption \langle3\rangle 3 and Lemma B.9.

\langle5\rangle 2. Q.E.D.  
Proof: By \langle5\rangle 1, since \( C(D_N(\alpha)) \subseteq C_E(D_N(\alpha)) \).

\langle4\rangle 3. Q.E.D.  
Proof: By \langle4\rangle 1, \langle4\rangle 2 and the rule of replacement \[51].

\langle3\rangle 4. Q.E.D.  
Proof: The cases \langle3\rangle 2 and \langle3\rangle 3 are exhaustive.

\langle2\rangle 2. Q.E.D.  
Proof: By =\text{-introduction}

\langle1\rangle 2. Q.E.D.  
Proof: By \forall\text{-introduction}

**Lemma B.12.** Let \( I_N \) be a probabilistic component execution as defined in Definition 5.3 and let \( \alpha \) be a complete queue history \( \alpha \in \mathcal{B}_N \). Then

\[ \forall t_1 \in (\mathcal{H} \cap \mathcal{E}^*) : \{t_1\} \in C_E(D_N(\alpha)) \setminus C(D_N(\alpha)) \Rightarrow \] \[ D_N(\alpha) \setminus \{t_1\} \text{ is a countable union of elements in } C_E(D_N(\alpha)) \]

Proof:

\langle1\rangle 1. Assume: \( t_1 \in (\mathcal{H} \cap \mathcal{E}^*) \)
Proof: \( \{t_1\} \in C_E(D_N(\alpha)) \setminus C(D_N(\alpha)) \Rightarrow \) \[ D_N(\alpha) \setminus \{t_1\} \text{ is a countable union of elements in } C_E(D_N(\alpha)) \]

\langle2\rangle 1. Assume: \( \{t_1\} \in C_E(D_N(\alpha)) \setminus C(D_N(\alpha)) \)
Proof: \[ D_N(\alpha) \setminus \{t_1\} \text{ is a countable union of elements in } C_E(D_N(\alpha)) \]

\langle3\rangle 1. \#t_1 \in \mathbb{N}
Proof: By assumption \langle2\rangle 1 and Definition 3.3.

\langle3\rangle 2. \( t_1 \in D_N(\alpha) \)
Proof: By assumption \langle2\rangle 1 and Definition 3.3.

\langle3\rangle 3. \( D_N(\alpha) \setminus c(t_1, D_N(\alpha)) \) is a countable union of elements in \( C_E(D_N(\alpha)) \).
Proof: By \langle3\rangle 1 and Lemma B.11.

\langle3\rangle 4. Let: \( S = \{t \in \mathcal{H} | \#t = \#t_1 + 1 \land \exists t' \in c(t_1, D_N(\alpha)) : t \sqsubseteq t' \} \)

\langle3\rangle 5. Case: \( S = \emptyset \)
\langle4\rangle 1. \( c(t_1, D_N(\alpha)) = \{t_1\} \)
Proof: By \langle3\rangle 4 and assumption \langle3\rangle 5.

\langle4\rangle 2. Q.E.D.
Proof: By \langle4\rangle 1, \langle3\rangle 3 and the rule of replacement \[51].

\langle3\rangle 6. Case: \( S \neq \emptyset \)
\langle4\rangle 1. \( (D_N(\alpha) \setminus c(t_1, D_N(\alpha))) \cup (\bigcup_{t \in S} c(t, D_N(\alpha))) \) is a countable union of elements in \( C_E(D_N(\alpha)) \).
\langle5\rangle 1. \( \bigcup_{t \in S} c(t, D_N(\alpha)) \) is a countable union of elements in \( C_E(D_N(\alpha)) \)
\langle6\rangle 1. \( \bigcup_{t \in S} c(t, D_N(\alpha)) \) is a countable union of elements in \( C(D_N(\alpha)) \).
Proof: By assumption \langle1\rangle 1, assumption \langle2\rangle 1, \langle3\rangle 4, assumption \langle3\rangle 6 and Lemma B.9.
\( \langle 6 \rangle 2. \text{Q.E.D.} \)

**Proof:** By \( \langle 6 \rangle 1. \) since \( C(D_N(\alpha)) \subseteq C_E(D_N(\alpha)) \).

\( \langle 5 \rangle 2. \text{Q.E.D.} \)

**Proof:** By \( \langle 5 \rangle 1. \), \( \langle 3 \rangle 3 \) and elementary set theory.

\( \langle 4 \rangle 2. \) \( D_N(\alpha) \setminus \{t_1\} = (D_N(\alpha) \setminus c(t_1, D_N(\alpha))) \cup \bigcup_{t \in S} c(t, D_N(\alpha)) \)

\( \langle 5 \rangle 1. \) \( c(t_1, D_N(\alpha)) \setminus \{t_1\} = \bigcup_{t \in S} c(t, D_N(\alpha)) \)

\( \langle 6 \rangle 1. \) \( c(t_1, D_N(\alpha)) \setminus \{t_1\} \subseteq \bigcup_{t \in S} c(t, D_N(\alpha)) \)

\( \langle 7 \rangle 1. \) Assume \( t' \in c(t_1, D_N(\alpha)) \setminus \{t_1\} \)

**Prove:** \( t' \in \bigcup_{t \in S} c(t, D_N(\alpha)) \)

\( \langle 8 \rangle 1. \) \( t' \in D_N(\alpha) \)

**Proof:** By assumption \( \langle 7 \rangle 1 \) and Definition 3.1.

\( \langle 8 \rangle 2. \exists t'' \in S : t'' \subseteq t' \)

\( \langle 9 \rangle 1. \) \( t' \in c(t_1, D_N(\alpha)) \)

**Proof:** By \( \langle 7 \rangle 1 \).

\( \langle 9 \rangle 2. \) \( t_1 \not\subseteq t' \)

**Proof:** By \( \langle 9 \rangle 1 \) and Definition 3.1.

\( \langle 9 \rangle 3. \) \( t_1 \neq t' \)

**Proof:** By assumption \( \langle 7 \rangle 1 \).

\( \langle 9 \rangle 4. \) \( \#t' > \#t_1 \)

**Proof:** By \( \langle 9 \rangle 2 \) and \( \langle 9 \rangle 3 \).

\( \langle 9 \rangle 5. \exists t'' \in \mathcal{H} : \#t'' = \#t_1 + 1 \wedge t'' \subseteq t' \)

**Proof:** By \( \langle 9 \rangle 4, \langle 9 \rangle 1 \) and Definition 3.1.

\( \langle 9 \rangle 6. \) Let: \( t'' \) be a trace such that \( \#t'' = \#t_1 + 1 \wedge t'' \subseteq t' \)

**Proof:** By \( \langle 9 \rangle 5 \).

\( \langle 9 \rangle 7. \) \( t'' \in S \)

**Proof:** By \( \langle 3 \rangle 4, \langle 9 \rangle 6 \) and \( \langle 9 \rangle 1 \).

\( \langle 9 \rangle 8. \text{Q.E.D.} \)

**Proof:** By \( \langle 9 \rangle 6, \langle 9 \rangle 7 \) and \( \exists \)-introduction.

\( \langle 8 \rangle 3. \text{Q.E.D.} \)

**Proof:** By \( \langle 8 \rangle 1, \langle 8 \rangle 2 \) and Definition 3.1.

\( \langle 7 \rangle 2. \text{Q.E.D.} \)

**Proof:** \( \subseteq \)-rule [29].

\( \langle 6 \rangle 2. \) \( \bigcup_{t \in S} c(t, D_N(\alpha)) \subseteq c(t_1, D_N(\alpha)) \setminus \{t_1\} \)

\( \langle 7 \rangle 1. \) \( \bigcup_{t \in S} c(t, D_N(\alpha)) \subseteq c(t_1, D_N(\alpha)) \)

**Proof:** By \( \langle 3 \rangle 4 \), Lemma B.5 and elementary set theory.

\( \langle 7 \rangle 2. \) \( t_1 \not\in \bigcup_{t \in S} c(t, D_N(\alpha)) \)

**Proof:** By \( \langle 3 \rangle 4 \) and Definition 3.1.

\( \langle 7 \rangle 3. \text{Q.E.D.} \)

**Proof:** By \( \langle 7 \rangle 1 \) and \( \langle 7 \rangle 2 \).

\( \langle 6 \rangle 3. \text{Q.E.D.} \)

**Proof:** By \( \langle 6 \rangle 1, \langle 6 \rangle 2 \) and \( = \)-rule for sets [29].

\( \langle 5 \rangle 2. \) \( D_N(\alpha) \setminus \{t_1\} = (D_N(\alpha) \setminus c(t_1, D_N(\alpha))) \cup (c(t_1, D_N(\alpha)) \setminus \{t_1\}) \)

\( \langle 6 \rangle 1. \) \( \{t_1\} \subseteq c(t_1, D_N(\alpha)) \)

**Proof:** By \( \langle 3 \rangle 2 \) and Definition 3.1.

\( \langle 6 \rangle 2. \) \( c(t_1, D_N(\alpha)) \subseteq D_N(\alpha) \)

**Proof:** By Definition 3.1.

\( \langle 6 \rangle 3. \text{Q.E.D.} \)

**Proof:** By \( \langle 6 \rangle 1, \langle 6 \rangle 2 \) and elementary set theory.
\textbf{Corollary B.13.} Let $I_N$ be a probabilistic component execution as defined in Definition 5.3 and let $\alpha$ be a complete queue history $\alpha \in B_N$. Then

$$\forall c \in C_E(D_N(\alpha)) : D_N(\alpha) \setminus c \text{ is a countable union of elements in } C_E(D_N(\alpha)).$$

\textbf{Proof.} Follows from Lemma B.11 and Lemma B.12.

\textbf{Lemma B.14.} Let $I_N$ be a probabilistic component execution as defined in Definition 5.3 and let $\alpha$ be a complete queue history $\alpha \in B_N$. Then

$$\forall A \in F_N(\alpha) : A \text{ is a countable union of elements in } C_E(D_N(\alpha)).$$

\textbf{Proof:}

1. \textbf{Assume:} $A \in F_N(\alpha)$.
   \textbf{Prove:} $A$ is a countable union of elements in $C_E(D_N(\alpha))$.
   \textbf{Proof Sketch:} By induction on the construction of $A$.
   \begin{enumerate}
   \item \textbf{Case:} $A \in C_E(D_N(\alpha))$ (Induction basis)
   \item \textbf{Q.E.D.}
   \end{enumerate}
2. \textbf{Case:} $A = D_N(\alpha) \setminus B$ (induction step)
   \begin{enumerate}
   \item \textbf{Assume:} $B$ is a countable union of elements in $C_E(D_N(\alpha))$.
     \textbf{Prove:} $D_N(\alpha) \setminus B$ is a countable union of elements in $C_E(D_N(\alpha))$.
   \item \textbf{Let:} $\phi$ be a sequence of elements in $C_E(D_N(\alpha))$ such that $B = \bigcup_{i=1}^{\#\phi} \phi[i]$
     \textbf{Proof:} By (\ref{assumption})1 and Definition A.1.
   \item $D_N(\alpha) \setminus \bigcup_{i=1}^{\#\phi} \phi[i] = \bigcap_{i=1}^{\#\phi} (D_N(\alpha) \setminus \phi[i])$
     \textbf{Proof:} By the infinite version of De Morgan’s laws (2.7) for sets [13].
   \item $\bigcap_{i=1}^{\#\phi} (D_N(\alpha) \setminus \phi[i])$ is a countable union of elements in $C_E(D_N(\alpha))$.
     \textbf{Proof:} By (\ref{let})1 and Corollary B.13.
   \item \textbf{Q.E.D.}
     \textbf{Proof:} By (\ref{assumption})1 and (\ref{let})2, since the subset of a countable set is countable.
   \end{enumerate}
3. \textbf{Assume:} $D_N(\alpha) \setminus \phi[i]$ is a countable union of elements in $C_E(D_N(\alpha))$
   \textbf{Prove:} By (\ref{let})1 and (\ref{assume})3 and the rule of replacement [51].
   \item \textbf{Q.E.D.}
   \textbf{Proof:} Induction step.
B.3. Conditional probability measure of a composite component

Lemma B.15. Let \( I_N \) be a probabilistic component execution as defined in Definition 5.3 and let \( \alpha \) be a complete queue history \( \alpha \in \mathcal{B}_N \). Then

\[
\forall t \in D_N(\alpha) : \{t\} \in \mathcal{F}_N(\alpha)
\]

Lemma B.16. Let \( I_{N_1} \) and \( I_{N_2} \) be two probabilistic component executions such that \( N_1 \cap N_2 = \emptyset \), let \( \alpha \) be a queue history in \( \mathcal{B}_{N_1 \cup N_2} \) and let \( \mu_{N_1} \otimes \mu_{N_2}(\alpha) \) be a measure on \( C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \) as defined in (25). Then

\[
\forall t_1 \in \mathcal{H} \cap \mathcal{E}^* : \{t_1\} \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \setminus C(D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow
\]

\[
(\mathcal{E}_{N_1} \otimes \{t_1\}) \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^i \otimes \alpha) \land (\mathcal{E}_{N_2} \otimes \{t_1\}) \in \mathcal{F}_{N_2}(\mathcal{E}_{N_2}^i \otimes \alpha)
\]

Proof:

Assume: \( t_1 \in \mathcal{H} \cap \mathcal{E}^* \)

Prove: \( \{t_1\} \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \setminus C(D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow
\]

\[
(\mathcal{E}_{N_1} \otimes \{t_1\}) \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^i \otimes \alpha) \land (\mathcal{E}_{N_2} \otimes \{t_1\}) \in \mathcal{F}_{N_2}(\mathcal{E}_{N_2}^i \otimes \alpha)
\]
Proof: By (3)2 and Lemma B.15.

(2). Q.E.D.

Proof: $\Rightarrow$-introduction

Lemma B.17.

$$\forall t \in \mathcal{H} : \forall S \subseteq \mathcal{E} : \#(S \otimes t) \in \mathbb{N} \Rightarrow S \otimes t_1 = S \otimes t \lor \exists i \in \mathbb{N} : S \otimes t_i \neq S \otimes t \land S \otimes t_{i+1} = S \otimes t$$

Proof:

(1). Assume: $t \in \mathcal{H} \land S \subseteq \mathcal{E}$

Proof: $\#(S \otimes t) \in \mathbb{N} \Rightarrow S \otimes t_1 = S \otimes t \lor \exists i \in \mathbb{N} : S \otimes t_i \neq S \otimes t \land S \otimes t_{i+1} = S \otimes t$

(2). Assume: $\#(S \otimes t) \in \mathbb{N}$

Proof: $S \otimes t_1 = S \otimes t \lor \exists i \in \mathbb{N} : S \otimes t_i \neq S \otimes t \land S \otimes t_{i+1} = S \otimes t$

(3). Assume: $S \otimes t_1 \neq S \otimes t \land \forall i \in \mathbb{N} : S \otimes t_i \neq S \otimes t \Rightarrow S \otimes t_{i+1} \neq S \otimes t$

Proof: $\bot$

(4). $S \otimes t \neq S \otimes t$

Proof: By (3)1 and the principle of mathematical induction.

(2). Q.E.D.

Proof: By (4)1 and $\bot$-introduction.

(3). Q.E.D.

Proof: Proof by contradiction.

(2). Q.E.D.

Proof: $\Rightarrow$-introduction.

Lemma B.17.

$$\forall t \in \mathcal{H} : \forall S \subseteq \mathcal{E} : \#(S \otimes t) \in \mathbb{N} \Rightarrow \exists t' \in \mathcal{H} \cap \mathcal{E}^* : S \otimes t' = S \otimes t$$

Proof. Follows from Lemma B.17.

Observation B.19. Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$ and let $\alpha$ be a queue history in $\mathcal{B}_{N_1 \cup N_2}$. Then

$$\forall t \in D_{N_1}(\mathcal{E}_{N_1}^i \otimes \alpha) :$$

$$(((\{!\} \times S \times N_1 \times N_2 \times \mathcal{Q}) \otimes t \subseteq (((\{!\} \times S \times N_1 \times N_2 \times \mathcal{Q}) \otimes \alpha) \Rightarrow$$

$$((\{!\} \times S \times N_1 \times N_2 \times \mathcal{Q}) \otimes t =$$

$$((\{!\} \times S \times N_1 \times N_2 \times \mathcal{Q}) \otimes \alpha)\lor$$

$$((\{!\} \times S \times N_1 \times N_2 \times \mathcal{Q}) \otimes \alpha)\land$$

$$((\{!\} \times S \times N_1 \times N_2 \times \mathcal{Q}) \otimes t' \subseteq t \land$$

$$((\{!\} \times S \times N_1 \times N_2 \times \mathcal{Q}) \otimes t' = (((\{!\} \times S \times N_1 \times N_2 \times \mathcal{Q}) \otimes \alpha)\lor$$

Proof:

(1). Assume: $t \in D_{N_1}(\mathcal{E}_{N_1}^i \otimes \alpha)$
Assume

Prove


\[2. \text{Q.E.D.}\]

\[1. \]

Assume

Prove

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t \sqsubseteq\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha \Rightarrow\]

\[\# t \in \mathbb{N} \Rightarrow \exists i \in \mathbb{N}:(\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t =\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[\# t = \infty \Rightarrow \forall i \in \mathbb{N}: \exists t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^*: t' \sqsubseteq t \land\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t' =\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[(2) 1. \text{Assume:} \ (\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t \sqsubseteq\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\]

Prove

\[\# t \in \mathbb{N} \Rightarrow \exists i \in \mathbb{N}:(\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t =\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[\# t = \infty \Rightarrow \forall i \in \mathbb{N}: \exists t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^*: t' \sqsubseteq t \land\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t' =\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[(3) 1. \text{Assume:} \ # t \in \mathbb{N}\]

Prove

\[\# t \in \mathbb{N} \Rightarrow \exists i \in \mathbb{N}:(\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t =\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[\# t = \infty \Rightarrow \forall i \in \mathbb{N}: \exists t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^*: t' \sqsubseteq t \land\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t' =\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[(4) 1. \text{Assume:} \ # t = \infty\]

Prove

\[\forall i \in \mathbb{N}: \exists t' \in \mathcal{H} \cap \mathcal{E}_{N_1}^*: t' \sqsubseteq t \land\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t' =\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[(5) 1. \text{Assume:} \ t \in \mathbb{N}\]

Prove

\[\forall i \in \mathbb{N}: \exists t' \in \mathcal{H} \cap \mathcal{E}_{N_1}^*: t' \sqsubseteq t \land\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t' =\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[(6) 1. \text{Assume:} \ \forall t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^*: t' \sqsubseteq t \Rightarrow\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t' \neq\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

Prove

\[\bot\]

\[(7) 1. \ \{1\} \times S \times N_1 \times N_2 \times Q) \uplus t\}_i \neq\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[(8) 1. \ \exists j \in \mathbb{N}:\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t_{ij} =\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[(9) 1. \text{Case:} \ i = 0\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus t_{ij} =\]

\[\{1\} \times S \times N_1 \times N_2 \times Q) \uplus \alpha\}_i\]

\[(10) 1. \text{Proof:} \text{By assumption (9) 1.}\]

\[(10) 2. \text{Q.E.D.}\]

\[\text{Proof:} \text{By (10) 1 and } \exists\text{-introduction.}\]

\[(9) 2. \text{Case:} \ i \neq 0\]
Lemma B.20. Let $s$ and $t$ be two infinite sequences of events. Then

$$s \neq t \Rightarrow \exists i \in \mathbb{N} : s_i = t_i \land s_{i+1} \neq t_{i+1}$$

Proof:
Lemma 5.4 Let $N_1$ and $N_2$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$ and let $\alpha$ be a queue history in $\mathcal{B}_{N_1 \cup N_2}$. Then

$$CT_{N_1 - N_2}(\alpha) \subseteq \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha) \wedge CT_{N_2 - N_1}(\alpha) \subseteq \mathcal{F}_{N_2}(\mathcal{E}_{N_2}^1 \circ \alpha)$$

Proof:

(1) $CT_{N_1 - N_2}(\alpha) \subseteq \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha) \wedge CT_{N_2 - N_1}(\alpha) \subseteq \mathcal{F}_{N_2}(\mathcal{E}_{N_2}^1 \circ \alpha)$

(2) $CT_{N_1 - N_2}(\alpha) \subseteq \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha)$

(3) Case: $CT_{N_1 - N_2}(\alpha) = \emptyset$

(4) 1. Case: $\#(\{1\} \times S \times N_1 \times N_2 \times Q) \circ \alpha \in N$

(5) 1. Let: $S = \{t \in \mathcal{H}_{N_1} \cap \mathcal{E}^* | \exists t' \in D_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha) : t \subseteq t' \wedge (\{1\} \times S \times N_1 \times N_2 \times Q) \circ t = (\{1\} \times S \times N_1 \times N_2 \times \emptyset) \circ \alpha\}$

(5) 2. Let: $S' = \{t \in \mathcal{H}_{N_1} \cap \mathcal{E}^* | \exists t' \in D_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha) : t \subseteq t' \wedge \#(\{1\} \times S \times N_1 \times N_2 \times Q) \circ t = \#(\{1\} \times S \times N_1 \times N_2 \times \emptyset) + 1\}$

(5) 3. $\bigcup_{t \in S} c(t, D_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha)) \setminus \bigcup_{t \in S'} c(t, D_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha)) \subseteq \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha)$

(6) 1. $\bigcup_{t \in S} c(t, D_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha)) \subseteq \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha)$

Proof: By (5) 1 and Corollary B.10.

(6) 2. $\bigcup_{t \in S'} c(t, D_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha)) \subseteq \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^1 \circ \alpha)$

Proof: By (5) 2 and Corollary B.10.

53
(6)3. Q.E.D.

Proof: By (6)1 and (6)2, since $F_{N_1}(E_{N_1}^1 \circ \alpha)$ is closed under set difference.

(5)4. $\bigcup_{t \in S} c(t, D_{N_1}(E_{N_1}^1 \circ \alpha)) \setminus \bigcup_{t' \in S'} c(t, D_{N_1}(E_{N_1}^1 \circ \alpha)) = CT_{N_1-N_2}(\alpha)$

(6)1. $\bigcup_{t \in S} c(t, D_{N_1}(E_{N_1}^1 \circ \alpha)) \setminus \bigcup_{t' \in S'} c(t, D_{N_1}(E_{N_1}^1 \circ \alpha)) \subseteq CT_{N_1-N_2}(\alpha)$

(7)1. Assume: $t \in \bigcup_{t' \in S} c(t', D_{N_1}(E_{N_1}^1 \circ \alpha)) \setminus \bigcup_{t' \in S'} c(t', D_{N_1}(E_{N_1}^1 \circ \alpha))$

Proof: $t \in CT_{N_1-N_2}(\alpha)$

(8)1. $t \in D_{N_1}(E_{N_1}^1 \circ \alpha)$

Proof: By assumption (7)1, (5)1 and Definition 3.1.

(8)2. $(\{!\} \times S \times N_1 \times N_2 \times Q \oplus t) \subseteq (\{!\} \times S \times N_1 \times N_2 \times Q \oplus \alpha$

(9)1. $\exists t' \in \mathcal{H}_{N_1} \cap E^*: t' \sqsubseteq t \land (\{!\} \times S \times N_1 \times N_2 \times Q) \oplus t' = (\{!\} \times S \times N_1 \times N_2 \times Q) \oplus \alpha$

Proof: By assumption (7)1 and which implies that $S \neq \emptyset$.

(9)2. Let: $t' \in \mathcal{H}_{N_1} \cap E^*$ such that $t' \sqsubseteq t \land (\{!\} \times S \times N_1 \times N_2 \times Q) \oplus t' = (\{!\} \times S \times N_1 \times N_2 \times Q) \oplus \alpha$

Proof: By (9)1.

(9)3. $\neg \exists i \in [\#t'..\#t]: k.t[i] = ! \land tr.t[i] = N_1 \land co.t[i] = N_2$

(10)1. Assume: $\exists i \in [\#t'..\#t]: k.t[i] = ! \land tr.t[i] = N_1 \land co.t[i] = N_2$

Proof: By (10)1.

(11)2. $\#(\{!\} \times S \times N_1 \times N_2 \times Q) \oplus t \geq \#(\{!\} \times S \times N_1 \times N_2 \times Q) \oplus \alpha + 1$

Proof: By (9)2 and (11)1.

(11)3. $\exists t'' \in \mathcal{H}_{N_1} \cap E^*: t'' \sqsubseteq t \land \#(\{!\} \times S \times N_1 \times N_2 \times Q) \oplus t'' = \#(\{!\} \times S \times N_1 \times N_2 \times Q) \oplus \alpha + 1$

Proof: By assumption (4)1 and (11)2.

(11)4. Let: $t'' \in \mathcal{H}_{N_1} \cap E^*$ such that $t'' \sqsubseteq t \land \#(\{!\} \times S \times N_1 \times N_2 \times Q) \oplus t'' = \#(\{!\} \times S \times N_1 \times N_2 \times Q) \oplus \alpha + 1$

Proof: By (11)3.

(11)5. $t'' \in S''$

Proof: By (11)4, (8)1 and (5)2.

(11)6. $t \in c(t'', D_{N_1}(E_{N_1}^1 \circ \alpha))$

Proof: By (11)4, (8)1 and Definition 3.1.

(11)7. $t \in \bigcup_{t' \in S'} c(t', D_{N_1}(E_{N_1}^1 \circ \alpha))$

Proof: By (11)5, (11)6 and elementary set theory.

(11)8. Q.E.D.

Proof: By assumption (7)1, (11)7 and \bot introduction.

(10)2. Q.E.D.

Proof: Proof by contradiction.

(9)4. $(\{!\} \times S \times N_1 \times N_2 \times Q) \oplus t|_{\#t'} = (\{!\} \times S \times N_1 \times N_2 \times Q) \oplus t$

Proof: By (9)3 and definition (7).

(9)5. Q.E.D.

Proof: By (9)2, (9)4 and the rule of replacement [51].
⟨ 6 2. Q.E.D. 
 Proof: By ⟨ 6 1, ⟨ 6 2 and definition (24). ⟩ 7 2. Q.E.D. 
 Proof: ⊆ rule. ⟩ 6 2. $CT_{N_1 - N_2}(α) ⊆ \bigcup_{t ∈ S} c(t, D_{N_1}(E_{N_1}^i ⊕ α)) \setminus \bigcup_{t' ∈ S'} c(t', D_{N_1}(E_{N_1}^i ⊕ α))$ 
 7 1. Assume: $t ∈ CT_{N_1 - N_2}(α)$ 
 Proof: $t ∈ \bigcup_{v ∈ S} c(t', D_{N_1}(E_{N_1}^i ⊕ α)) \setminus \bigcup_{v' ∈ S'} c(t', D_{N_1}(E_{N_1}^i ⊕ α))$ 
 8 2. $\{!\} × S × N_1 × N_2 × Q) ⊖ t \subseteq (\{!\} × S × N_1 × N_2 × Q) ⊖ α$ 
 Proof: By assumption ⟨ 7 1 and definition (24). ⟩ 8 3. $∃ i ∈ N: (\{!\} × S × N_1 × N_2 × Q) ⊖ t_i = (\{!\} × S × N_1 × N_2 × Q) ⊖ α$ 
 Proof: By assumption ⟨ 4 1, ⟨ 8 2 and Corollary B.18. ⟩ 8 4. Let: $i ∈ N$ such that $(\{!\} × S × N_1 × N_2 × Q) ⊖ t_i = (\{!\} × S × N_1 × N_2 × Q) ⊖ α$ 
 Proof: By ⟨ 8 3. ⟩ 8 5. $t ∈ \bigcup_{v ∈ S} c(t', D_{N_1}(E_{N_1}^i ⊕ α))$ 
 (9) 1. $t_i ∈ S$ 
 Proof: By ⟨ 8 1, ⟨ 8 4 and (5) 1. ⟩ 9 2. $t ∈ c(t_i, D_{N_1}(E_{N_1}^i ⊕ α))$ 
 Proof: By ⟨ 8 1 and Definition 3.1. ⟩ 9 3. Q.E.D. 
 Proof: By ⟨ 9 2, ⟨ 9 1 and elementary set theory. ⟩ 8 6. $t \notin \bigcup_{v ∈ S'} c(t', D_{N_1}(E_{N_1}^i ⊕ α))$ 
 (9) 1. Assume: $t ∈ \bigcup_{v' ∈ S'} c(t', D_{N_1}(E_{N_1}^i ⊕ α))$ 
 Proof: $\bot$ 
 (10) 1. $∃ j ∈ [i + 1..#t]: k.t[j] = ! ∧ tr.t[j] = N_1 ∧ co.t[j] = N_2$ 
 (11) 1. $∃ t' ∈ H_{N_1} \cap E_*: t' ⊑ t ∧ \#((\{!\} × S × N_1 × N_2 × Q) ⊖ t') = \#((\{!\} × S × N_1 × N_2 × Q) ⊖ α) + 1$ 
 Proof: By assumption ⟨ 9 1 and (5) 2. ⟩ 11 2. Let: $t' ∈ H_{N_1} \cap E_*$ such that $t' ⊑ t ∧ \#((\{!\} × S × N_1 × N_2 × Q) ⊖ t') = \#((\{!\} × S × N_1 × N_2 × Q) ⊖ α) + 1$ 
 Proof: By ⟨ 11 1. ⟩ 11 3. Let: $j = \#((\{!\} × S × N_1 × N_2 × Q) ⊖ t')$ 
 (11) 4. $j ∈ [i + 1..#t] ∧ k.t[j] = ! ∧ tr.t[j] = N_1 ∧ co.t[j] = N_2$ 
 Proof: By ⟨ 8 4, ⟨ 11 2 and (11) 3. ⟩ 11 5. Q.E.D. 
 Proof: By ⟨ 11 3, ⟨ 11 4 and ⊥ introduction. ⟩ 10 2. Let: $j ∈ [i + 1..#t]$ such that 
 $k.t[j] = ! ∧ tr.t[j] = N_1 ∧ co.t[j] = N_2$ 
 Proof: By ⟨ 10 1. ⟩ 10 3. $(\{!\} × S × N_1 × N_2 × Q) ⊖ t \not\subseteq (\{!\} × S × N_1 × N_2 × Q) ⊖ α$ 
 Proof: By ⟨ 8 4, ⟨ 10 2 and definition (2). ⟩ 10 4. Q.E.D. 
 Proof: By ⟨ 8 2, ⟨ 10 3 and ⊥ introduction. }
(9)2. Q.E.D.
Proof: Proof by contradiction.

(8)7. Q.E.D.
Proof: By (8)5, (8)6 and elementary set theory.

(7)2. Q.E.D.
Proof: \(\subseteq\) rule.

(6)3. Q.E.D.
Proof: By (6)1, (6)2 and the \(\equiv\) rule for sets \([29]\).

(5)5. Q.E.D.
Proof: By (5)3, (5)4 and the rule of replacement \([51]\).

(4)2. Case: \(\#(\{i\} \times S \times N_1 \times N_2 \times Q) \circ \alpha = \infty\)

(5)1. Let: \(S = \{ t \in D_{N_1}(\mathcal{E}_{N_1}^{t} \circ \alpha) \cap \mathcal{E}^* \mid (\{i\} \times S \times N_1 \times N_2 \times Q) \circ t \subseteq ((\{i\} \times S \times N_1 \times N_2 \times Q) \circ \alpha)\}\)

(5)2. \(\forall i \in N\)
Let: \(S'_i = \{ t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^* \mid \exists t \in D_{N_1}(\mathcal{E}_{N_1}^{t} \circ \alpha) : t' \subseteq t \wedge ((\{i\} \times S \times N_1 \times N_2 \times Q) \circ t \subseteq ((\{i\} \times S \times N_1 \times N_2 \times Q) \circ \alpha) \wedge ((\{i\} \times S \times N_1 \times N_2 \times Q) \circ t' = (((\{i\} \times S \times N_1 \times N_2 \times Q) \circ \alpha))\}

(5)3. \(\forall i \in N\)
Let: \(G_i = \bigcup_{t \in S'_i} c(t, D_{N_1}(\mathcal{E}_{N_1}^{t} \circ \alpha))\)

(5)4. \(S \cup \bigcap_{i=1}^{\infty} G_i \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^{i} \circ \alpha)\)

(6)1. \(S \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^{i} \circ \alpha)\)

(7)1. (\#S = N, \#S \in N), that is, \(S\) is countable.

(6)2. \(\bigcap_{i=1}^{\infty} G_i \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^{i} \circ \alpha)\)

(7)1. \(\forall i \in N : G_i \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^{i} \circ \alpha)\)
Proof: By (5)2 and Corollary B.10.

(7)2. Q.E.D.
Proof: By (7)1, since \(\mathcal{F}_{N_1}(\mathcal{E}_{N_1}^{i} \circ \alpha)\) is closed under countable union.

(6)3. Q.E.D.
Proof: By (6)1 and (6)2, since \(\mathcal{F}_{N_1}(\mathcal{E}_{N_1}^{i} \circ \alpha)\) is closed under countable union.

(5)5. \(S \cup \bigcap_{i=1}^{\infty} G_i = CT_{N_1-N_2}(\alpha)\)

(6)1. \(S \cup \bigcap_{i=1}^{\infty} G_i \subseteq CT_{N_1-N_2}(\alpha)\)

(7)1. Assume: \(t \in S \cup \bigcap_{i=1}^{\infty} G_i\)
Prove: \(t \in CT_{N_1-N_2}(\alpha)\)

(8)1. Case: \(t \in S\)

(9)1. Q.E.D.
Proof: By (5)1 and definition (24).

56
\(8\). Case: \( t \in \bigcap_{i=1}^{\infty} G_i \)
\(9\). 1. \( \forall i \in \mathbb{N} : t \in G_i \)

**Proof:** By assumption \(8\).
\(9\). 2. \( \# t = \infty \)
\(10\). 1. **Assume:** \( \# t \in \mathbb{N} \)

**Prove:** \( \perp \)
\(11\). 1. \( t \in G_{\# t+1} \)

**Proof:** By assumption \(10\) and \(9\) and \( \forall \) elimination.
\(11\). 2. \( \exists t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^*: t' \sqsubseteq t \land (\{!\} \times S \times N_1 \times N_2 \times Q) \odot t' = (\{!\} \times S \times N_1 \times N_2 \times Q) \odot \alpha) \mid_{\# t+1} \)

**Proof:** By \(11\) and \(5\).
\(11\). 3. \( \forall t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^* : t' \sqsubseteq t \land (\{!\} \times S \times N_1 \times N_2 \times Q) \odot t' = (\{!\} \times S \times N_1 \times N_2 \times Q) \odot \alpha) \mid_{\# t+1} \)

**Proof:** By \(11\).
\(12\). 1. \( \# t' = \#(\{!\} \times S \times N_1 \times N_2 \times Q) \odot t' \)

**Proof:** By definition (7).
\(12\). 2. \( \#((\{!\} \times S \times N_1 \times N_2 \times Q) \odot t') = \#((\{!\} \times S \times N_1 \times N_2 \times Q) \odot \alpha) \mid_{\# t+1} \)

**Proof:** By \(11\) and the rule of equality between functions [51].
\(12\). 3. \( \forall t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^* : t' \sqsubseteq t \land (\{!\} \times S \times N_1 \times N_2 \times Q) \odot t' = (\{!\} \times S \times N_1 \times N_2 \times Q) \odot \alpha) \mid_{\# t+1} = \# t + 1 \)

**Proof:** By assumption \(4\).
\(12\). 4. Q.E.D.

**Proof:** By \(12\) and \(12\) and \(12\) and \(9\) and \( \perp \) introduction.
\(11\). 5. \( t' \nsubseteq t \)

**Proof:** By \(11\) and definition (2).
\(11\). 6. Q.E.D.

**Proof:** By \(11\), \(11\) and \(1\) and \( \perp \) introduction.
\(10\). 2. Q.E.D.

**Proof:** Proof by contradiction.
\(9\). 3. **Assume:** \( t \notin CT_{N_1-N_2}(\alpha) \)

**Prove:** \( \perp \)
\(10\). 1. \( t \in D_{N_1}(\mathcal{E}^i) \)

**Proof:** By assumption \(8\) and \(5\) and \(5\) and Definition 3.1.
\(10\). 2. \( (\{!\} \times S \times N_1 \times N_2 \times Q) \odot t \nsubseteq (\{!\} \times S \times N_1 \times N_2 \times Q) \odot \alpha \)

**Proof:** By \(10\) and \(9\) and \(5\) and Definition (24).
\(10\). 3. \( (\{!\} \times S \times N_1 \times N_2 \times Q) \odot t = (\{!\} \times S \times N_1 \times N_2 \times Q) \odot \alpha \)

**Proof:** By \(10\) and \(2\) and definition (2).
\(10\). 4. \( \exists i \in \mathbb{N} : (\{!\} \times S \times N_1 \times N_2 \times Q) \odot t) \mid_i = (\{!\} \times S \times N_1 \times N_2 \times Q) \odot t) \mid_{i+1} \neq (\{!\} \times S \times N_1 \times N_2 \times Q) \odot \alpha) \mid_{i+1} \)

**Proof:** By \(9\) and \(4\) and \(5\) and Lemma B.20.
\(10\). 5. **Let:** \( i \in \mathbb{N} \) such that \( (\{!\} \times S \times N_1 \times N_2 \times Q) \odot t) \mid_i = \)
Proof: By (10)4.

\[ t \in G_{i+1} \]

Proof: By (10)5, (9)1 and \( \forall \) elimination.

\[ \exists t' \in H_{N_1} \cap E^*: t' \subseteq t \land (\{\} \times S \times N_1 \times N_2 \times Q) \odot t' = ((\{\} \times S \times N_1 \times N_2 \times Q) \odot \alpha)|_{i+1} \]

Proof: By (10)6, (5)3 and (5)2.

Proof: By (10)7.

\[ (((\{\} \times S \times N_1 \times N_2 \times Q) \odot t)|_{i+1} = (((\{\} \times S \times N_1 \times N_2 \times Q) \odot t)|_{i+1} = ((\{\} \times S \times N_1 \times N_2 \times Q) \odot t') \]

Proof: By (9)2 and definition (2).

\[ #(((\{\} \times S \times N_1 \times N_2 \times Q) \odot t)|_{i+1} = #(((\{\} \times S \times N_1 \times N_2 \times Q) \odot t') \]

Proof: By assumption (4)2 and definition (2).

Proof: By (13)1, (13)2 and the rule of transitivity.

Proof: By (12)1, (13)2 and the rule of transitivity.

Proof: By (10)8 and (11)1 and the rule of transitivity [51].

Proof: By (9)5, (10)9 and \( \bot \)-introduction.

Proof: Proof by contradiction.

Proof: By assumption (7)1, the cases (8)1 and (8)2 are exhaustive.

Proof: Q.E.D.

Proof: Q.E.D.

Proof: By (8)3 Q.E.D.

Proof: By (7)2 Q.E.D.

Proof: \( \subseteq \) rule.

Proof: By (6)2, (7)1. Assume: \( t \in CT_{N_1-N_2}(\alpha) \)

Prove: \( t \in S \cup \bigcap_{i=1}^{\infty} G_i \)

Proof: By (8)1 \( t \in D_{N_1}(E_{N_1}^* \odot \alpha) \land (((\{\} \times S \times N_1 \times N_2 \times Q) \odot t) \subseteq ((\{\} \times S \times N_1 \times N_2 \times Q) \odot \alpha) \)

58
Proof: By assumption (7)1 and definition (24).
(8)2. Case: \( \#t \in \mathbb{N} \)
(9)1. \( t \in S \)
    Proof: By (8)1, assumption (8)2 and (5)1.
(9)2. Q.E.D.
Proof: By (9)1 and elementary set theory.
(8)3. Case: \( \#t = \infty \)
(9)1. \( t \in \bigcap_{i=1}^{\infty} G_i \)
    (10)1. Assume: \( t \notin \bigcap_{i=1}^{\infty} G_i \)
        Proof: \( \bot \)
    (11)1. \( \exists i \in \mathbb{N} : t \notin G_i \)
        Proof: By assumption (10)1.
(12)2. Let: \( i \in \mathbb{N} \) such that \( t \notin G_i \)
    (13)3. \( \exists t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^* : t' \subset t \land \left( \{!\} \times S \times N_1 \times N_2 \times Q \right) \otimes t' = \left( \{!\} \times S \times N_1 \times N_2 \times Q \right) \otimes \alpha \vert_i \)
    (12)1. Assume: \( \exists t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^* : t' \subset t \land \left( \{!\} \times S \times N_1 \times N_2 \times Q \right) \otimes t' = \left( \{!\} \times S \times N_1 \times N_2 \times Q \right) \otimes \alpha \vert_i \)
        Proof: \( \bot \)
(13)1. Let: \( t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^* : t' \subset t \land \left( \{!\} \times S \times N_1 \times N_2 \times Q \right) \otimes t' = \left( \{!\} \times S \times N_1 \times N_2 \times Q \right) \otimes \alpha \vert_i \)
    (12)2. \( t' \in S'_i \)
        Proof: By (8)1, (13)1 and (5)2.
(13)3. \( t \in G_i \)
        Proof: By (8)1, (13)1, (13)2, Definition 3.1 and (5)3.
(13)4. Q.E.D.
        Proof: By (11)2, (13)3 and \( \bot \)-introduction.
(12)2. Q.E.D.
        Proof: Proof by contradiction.
(11)4. \( \exists t' \in \mathcal{H}_{N_1} \cap \mathcal{E}^* : t' \subset t \land \left( \{!\} \times S \times N_1 \times N_2 \times Q \right) \otimes t' = \left( \{!\} \times S \times N_1 \times N_2 \times Q \right) \otimes \alpha \vert_i \)
        Proof: By (8)1, assumption (8)3, Observation B.19 and \( \forall \)-elimination.
(11)5. Q.E.D.
        Proof: By (11)3, (11)4 and \( \bot \)-introduction.
(10)2. Q.E.D.
        Proof: Proof by contradiction.
(9)3. Q.E.D.
        Proof: By (9)1 and elementary set theory.
(8)4. Q.E.D.
        Proof: The cases (8)2 and (8)3 are exhaustive.
(7)2. Q.E.D.
        Proof: \( \subseteq \) rule.
(6)3. Q.E.D.
Lemma B.21. Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$ and let $\alpha$ be a queue history in $B_{N_1 \cup N_2}$. Then

$$\forall t_1 \in \mathcal{H} \cap \mathcal{E}^*: c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow$$

$$\mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \subseteq c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1}^l \otimes \alpha))$$

Proof:

1. Assume: $t_1 \in \mathcal{H} \cap \mathcal{E}^*$
   
   **Prove:** $c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow$
   $$\mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \subseteq c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1}^l \otimes \alpha))$$

2. Assume: $c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha))$
   
   **Prove:** $\mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \subseteq c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1}^l \otimes \alpha))$

3. Assume: $t' \in \mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))$
   
   **Prove:** $t' \in c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1}^l \otimes \alpha))$

4. Let: $t'' \in c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))$ such that $t' = \mathcal{E}_{N_1} \otimes t''$
   
   **Prove:** By (4)1.

5. $t' \in D_{N_1}(\mathcal{E}_{N_1}^l \otimes \alpha)$
   
   **Proof:** By (4)2, definition (22) and Definition 3.1.

6. $\mathcal{E}_{N_1} \otimes t'' \in D_{N_1}(\mathcal{E}_{N_1}^l \otimes \alpha)$
   
   **Proof:** By (4)2 and Definition 3.1.

7. $t_1 \subseteq t''$
   
   **Proof:** By (6)1 and definition (7).

8. $\mathcal{E}_{N_1} \otimes t_1 \subseteq t''$
   
   **Proof:** By (7) and definition (7).

9. $\mathcal{E}_{N_1} \otimes t_1 \subseteq \mathcal{E}_{N_1} \otimes t''$
   
   **Proof:** By (9) and the rule of replacement [51].
Lemma B.22. Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$ and let $\alpha$ be a queue history in $B_{N_1 \cup N_2}$. Then

$$\forall t \in \mathcal{H} \cap \mathcal{E}^* : c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow$$

$$(\forall t_1, t_2 \in \mathcal{H} : t_1 \in \mathcal{E}_{N_1} \otimes c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \land t_2 \in \mathcal{E}_{N_2} \otimes c(t, D_{N_1} \otimes D_{N_2}(\alpha))) \Rightarrow$$

$$\left( \exists \alpha' \in c(t, D_{N_1} \otimes D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \otimes t' = t_1 \land \mathcal{E}_{N_2} \otimes t' = t_2 \right)$$

**Proof:**

(1.1) Assume $t \in \mathcal{H} \cap \mathcal{E}^*$

**Proof Sketch:** By induction over the length of $t$.

(3.1) Case: $t = \emptyset$ (induction basis)

(4.1) Assume: $t_1 \in \mathcal{H} \land t_2 \in \mathcal{H}$

**Proof:** $t_1 \in \mathcal{E}_{N_1} \otimes c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \land t_2 \in \mathcal{E}_{N_2} \otimes c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow$

$$\exists \alpha' \in c(t, D_{N_1} \otimes D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \otimes t' = t_1 \land \mathcal{E}_{N_2} \otimes t' = t_2$$

(5.1) Assume: $t_1 \in \mathcal{E}_{N_1} \otimes c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \land t_2 \in \mathcal{E}_{N_2} \otimes c(t, D_{N_1} \otimes D_{N_2}(\alpha))$

**Proof:** $\exists \alpha' \in c(t, D_{N_1} \otimes D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \otimes t' = t_1 \land \mathcal{E}_{N_2} \otimes t' = t_2$

(6.1) $c(t, D_{N_1} \otimes D_{N_2}(\alpha)) = D_{N_1} \otimes D_{N_2}(\alpha)$

**Proof:** By assumption (3.1) and Definition 3.1.

(6.2) $\exists \alpha'', \alpha''' \in D_{N_1} \otimes D_{N_2}(\alpha) : \mathcal{E}_{N_1} \otimes \alpha'' = t_1 \land \mathcal{E}_{N_2} \otimes \alpha''' = t_2$

**Proof:** By assumption (5.1), (6.1) and definition (7).

(6.3) Let: $\alpha'', \alpha''' \in D_{N_1} \otimes D_{N_2}(\alpha)$ such that $\mathcal{E}_{N_1} \otimes \alpha'' = t_1 \land \mathcal{E}_{N_2} \otimes \alpha''' = t_2$

**Proof:** By (6.2)

(6.4) $t_1 \in D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)$ \land $t_2 \in D_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)$

(7.1) $t_1 \in D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)$

(8.1) $\mathcal{E}_{N_1} \otimes \alpha'' = D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)$

**Proof:** By (6.3) and definition (22).

(8.2) Q.E.D.

**Proof:** By (6.3), (8.1) and the rule of replacement [51].

(7.2) $t_2 \in D_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)$

**Proof:** Symmetrical to (7.1).

(7.3) Q.E.D.

**Proof:** By (7.1), (7.2) and \land-introduction.

(6.5) $\text{rng}_t \subseteq \mathcal{E}_{N_1} \land \text{rng}_{t_2} \subseteq \mathcal{E}_{N_2}$

**Proof:** By (6.4), Definition 5.3, definition (20) and definition (5).

(6.6) $\langle \langle \rangle \times S \times N_1 \times N_2 \times Q \rangle \times t_1 \subseteq \langle \langle \rangle \times S \times N_2 \times N_2 \times Q \rangle \otimes \alpha \land$

$\langle \langle \rangle \times S \times N_2 \times N_1 \times Q \rangle \times t_2 \subseteq \langle \langle \rangle \times S \times N_2 \times N_1 \times Q \rangle \otimes \alpha$
⟨7:1. \{(\text{!}) \times S \times N_1 \times N_2 \times Q\} \oplus t_1 \subseteq \{(\text{!}) \times S \times N_1 \times N_2 \times Q\} \oplus \alpha \\
⟨8:1. \{(\text{!}) \times S \times N_1 \times N_2 \times Q\} \oplus t'' \subseteq \{(\text{!}) \times S \times N_1 \times N_2 \times Q\} \oplus \alpha \\
Proof: By (6)3 and definition (22). \\
⟨8:2. \{(\text{!}) \times S \times N_1 \times N_2 \times Q\} \oplus t'' \subseteq \{(\text{!}) \times S \times N_1 \times N_2 \times Q\} \oplus \alpha \\
Proof: By (8)1 and definition (7). \\
⟨8:3. Q.E.D. \\
Proof: By (6)3, (8)2 and the rule of replacement [51]. \\
⟨7:2. \{(\text{!}) \times S \times N_2 \times N_1 \times Q\} \oplus t_2 \subseteq \{(\text{!}) \times S \times N_2 \times N_1 \times Q\} \oplus \alpha \\
Proof: Symmetrical to step (7)1 \\
⟨7:3. Q.E.D. \\
Proof: By (7)1, (7)2 and \&-introduction. \\
⟨6:7. ((\forall i, j \in [1..\#t_1]): i < j \Rightarrow q.t_1[i] < q.t_1[j]) \land \\
\#t_1 = \infty \Rightarrow \forall k \in Q: \exists i \in N: q.t_1[i] > k \land \\
(\forall i, j \in [1..\#t_2]): i < j \Rightarrow q.t_2[i] < q.t_2[j] \land \\
\#t_2 = \infty \Rightarrow \forall k \in Q: \exists i \in N: q.t_2[i] > k) \\
⟨7:1. (\forall i, j \in [1..\#t_1]): i < j \Rightarrow q.t_1[i] < q.t_1[j] \land \\
(\#t_1 = \infty \Rightarrow \forall k \in Q: \exists i \in N: q.t_1[i] > k) \\
⟨8:1. (\forall i, j \in [1..\#t'']: i < j \Rightarrow q.t'[i] < q.t'[j] \land \\
\#t' = \infty \Rightarrow \forall k \in Q: \exists i \in N: q.t'[i] > k \\
Proof: By (6)3 and definition (22). \\
⟨8:2. (\forall i, j \in [1..\#E_{N_1} \oplus t'']: i < j \Rightarrow q.E_{N_1} \oplus t'[i] < q.E_{N_1} \oplus t'[j] \land \\
\#E_{N_1} \oplus t' = \infty \Rightarrow \forall k \in Q: \exists i \in N: q.E_{N_1} \oplus t'[i] > k \\
Proof: By (8)1, definition (7), and constraints (8) and (9), since 
the filtering of a trace with regard to a set of events does not change 
the ordering of the remaining events in the trace. \\
⟨8:3. Q.E.D. \\
Proof: By (6)3, (8)2 and the rule of replacement [51]. \\
⟨7:2. (\forall i, j \in [1..\#t_2]): i < j \Rightarrow q.t_2[i] < q.t_2[j] \land \\
\#t_2 = \infty \Rightarrow \forall k \in Q: \exists i \in N: q.t_2[i] > k \\
Proof: Symmetrical to step (7)1 \\
⟨7:3. Q.E.D. \\
Proof: By (7)1, (7)2 and \&-introduction. \\
⟨6:8. \{q.t_1[i] | i \in [1..\#t_1]\} \cap \{q.t_2[j] | j \in [1..\#t_2]\} = \emptyset \\
Proof: By the assumption that each interface, and hence each compo-
ment, is assigned a set of time-stamps disjoint from the set of time-stamps 
assigned to every other interface or component. \\
⟨6:9. \Pi_{(1,2)}.(\Pi_{(2)}.(\{\text{!} \times M\} \oplus t_1)) \subseteq \Pi_{(1,2)}.(\Pi_{(2)}.(E_{N_1}^k \oplus \alpha)) \\
Proof: By (6)4 and constraint (11). \\
⟨6:10. \Pi_{(1,2)}.(\Pi_{(2)}.(\{\text{!} \times M\} \oplus t_2)) \subseteq \Pi_{(1,2)}.(\Pi_{(2)}.(E_{N_2}^l \oplus \alpha)) \\
Proof: By (6)4 and constraint (11). \\
⟨6:11. \exists t'' \in H : E_{N_1}^k \oplus t'' = t_1 \land E_{N_2}^l \oplus t'' = t_2 \\
Proof: By (6)7 t_1 and t_2 fulfil well-formedness constraints (8) and (9) 
with regard to time. By (6)8 their sets of time-stamps are disjoint. Hence, 
it is possible to interleave the events of t_1 and t_2 in such a way that the 
well-formedness constraints (8) and (9) are fulfilled. Furthermore, by (6)9 
the sequence of consumed messages in t_1 is a prefix of the messages in 
E_{N_2}^l \oplus \alpha when disregarding time, and vice versa for t_2 by (6)10. By (6)6
the sequence of messages transmitted from \( N_1 \) to \( N_2 \) in \( t_1 \) is a prefix of the messages transmitted from \( N_1 \) to \( N_2 \) in \( \alpha \), and vice versa for \( t_2 \). Hence, it is possible to interleave the events of \( t_1 \) and \( t_2 \) in such a way that the sequence of consumed messages sent from \( N_1 \) to \( N_2 \), is a prefix of the sequence of transmitted messages from \( N_1 \) to \( N_2 \), and vice versa, when disregarding time, fulfilling constraint (10).

(6) 12. Let: \( t'' \in \mathcal{H} \) such that \( \mathcal{E}_{N_1} \circ t'' = t_1 \land \mathcal{E}_{N_2} \circ t'' = t_2 \)

PROOF: By (6) 11.

(6) 13. \( t''' \in \mathcal{H}_{N_1 \cup N_2} \)

PROOF: By (6) 12, (6) 5 and elementary set theory.

(6) 14. \( (\{!\} \times S \times N_2 \times N_1 \times Q) \circ t''' \subseteq (\{!\} \times S \times N_2 \times N_1 \times Q) \circ \alpha \land (\{!\} \times S \times N_1 \times N_2 \times Q) \circ t''' \subseteq (\{!\} \times S \times N_1 \times N_2 \times Q) \circ \alpha \)

(6) 15. \( \mathcal{E}_{N_1} \circ t''' = D_{N_1}(\mathcal{E}_{N_1} \circ \alpha) \land \mathcal{E}_{N_2} \circ t''' = D_{N_2}(\mathcal{E}_{N_2} \circ \alpha) \)

PROOF: By (6) 12, (6) 4 and the rule of replacement.

(6) 16. \( t''' \in D_{N_1} \circ D_{N_2}(\alpha) \)

PROOF: By (6) 13, (6) 14, (6) 15 and definition (22).

(6) 17. Q.E.D.

PROOF: By (6) 16, (6) 12 and \( \exists \)-introduction.

(4) 2. Q.E.D.

PROOF: \( \forall \)-introduction.

(3) 2. CASE: \( t = t''' \sim \{e\} \) (induction step)

(4) 1. ASSUME: \( \forall t_1, t_2 \in \mathcal{H} \land \mathcal{E}^* : t_1 \in \mathcal{E}_{N_1} \circ c(t'', D_{N_1} \circ D_{N_2}(\alpha)) \land t_2 \in \mathcal{E}_{N_2} \circ c(t'', D_{N_1} \circ D_{N_2}(\alpha)) ) ) \Rightarrow \exists t' \in c(t'', D_{N_1} \circ D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \circ t' = t_1 \land \mathcal{E}_{N_2} \circ t' = t_2 \)

(induction hypothesis)

PROVE: \( \forall t_1, t_2 \in \mathcal{H} : t_1 \in \mathcal{E}_{N_1} \circ c(t, D_{N_1} \circ D_{N_2}(\alpha)) \land t_2 \in \mathcal{E}_{N_2} \circ c(t, D_{N_1} \circ D_{N_2}(\alpha)) ) ) \Rightarrow \exists t' \in c(t, D_{N_1} \circ D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \circ t' = t_1 \land \mathcal{E}_{N_2} \circ t' = t_2 \)

(induction step)

(5) 1. ASSUME: \( t_1 \in \mathcal{H} \land t_2 \in \mathcal{H} \)

PROVE: \( t_1 \in \mathcal{E}_{N_1} \circ c(t, D_{N_1} \circ D_{N_2}(\alpha)) \land t_2 \in \mathcal{E}_{N_2} \circ c(t, D_{N_1} \circ D_{N_2}(\alpha)) ) ) \Rightarrow \exists t' \in c(t, D_{N_1} \circ D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \circ t' = t_1 \land \mathcal{E}_{N_2} \circ t' = t_2 \)

(6) 1. ASSUME: \( t_1 \in \mathcal{E}_{N_1} \circ c(t, D_{N_1} \circ D_{N_2}(\alpha)) \land t_2 \in \mathcal{E}_{N_2} \circ c(t, D_{N_1} \circ D_{N_2}(\alpha)) )

PROVE: \( \exists t' \in c(t, D_{N_1} \circ D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \circ t' = t_1 \land \mathcal{E}_{N_2} \circ t' = t_2 \)

(7) 1. ASSUME: \( \neg \exists t' \in c(t, D_{N_1} \circ D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \circ t' = t_1 \land \mathcal{E}_{N_2} \circ t' = t_2 \)

PROVE: \( \bot \)

(8) 1. \( \exists t' \in c(t'', D_{N_1} \circ D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \circ t' = t_1 \land \mathcal{E}_{N_2} \circ t' = t_2 \)

(9) 1. \( t_1 \in \mathcal{E}_{N_1} \circ c(t'', D_{N_1} \circ D_{N_2}(\alpha)) \land t_2 \in \mathcal{E}_{N_2} \circ c(t'', D_{N_1} \circ D_{N_2}(\alpha)) \)

(10) 1. \( c(t, D_{N_1} \circ D_{N_2}(\alpha)) \subseteq c(t'', D_{N_1} \circ D_{N_2}(\alpha)) \)

PROOF: By assumption (3) 2 and Lemma B.5.

(10) 2. Q.E.D.
Proof: By assumption $\langle 6 \rangle 1$, $\langle 10 \rangle 1$, definition (7) and elementary set theory.

$\langle 9 \rangle 2.$ Q.E.D.

Proof: By $\langle 9 \rangle 1$ and the induction hypothesis (assumption $\langle 4 \rangle 1$).

$\langle 8 \rangle 2.$ Let $t' \in c(t'', D_{N_1} \otimes D_{N_2}(\alpha))$ such that $E_{N_1} \otimes t' = t_1 \land E_{N_2} \otimes t' = t_2$

Proof: By $\langle 8 \rangle 1$.

$\langle 8 \rangle 3.$ $t' \not\in c(t, D_{N_1} \otimes D_{N_2}(\alpha))$

Proof: By $\langle 8 \rangle 2$ and assumption $\langle 7 \rangle 1$.

$\langle 8 \rangle 4.$ $t \not\subseteq t'$

Proof: By $\langle 8 \rangle 3$ and Definition 3.1.

$\langle 8 \rangle 5.$ $t \subseteq t'$

$\langle 9 \rangle 1.$ $E_{N_1} \otimes t \subseteq t_1 \land E_{N_2} \otimes t \subseteq t_1$

Proof: By assumption $\langle 6 \rangle 1$, Definition 3.1 and definition (7).

$\langle 9 \rangle 2.$ $E_{N_1} \otimes t \subseteq E_{N_1} \otimes t' \land E_{N_2} \otimes t \subseteq E_{N_2} \otimes t'$

Proof: By $\langle 9 \rangle 1$, $\langle 8 \rangle 2$ and the rule of replacement [51].

$\langle 9 \rangle 3.$ $\text{rng.} \cdot t = E_{N_1} \cup E_{N_2}$

Proof: By $\langle 8 \rangle 2$, Definition 3.1, definition (22) and definition (20).

$\langle 9 \rangle 4.$ $\text{rng.} \cdot t' = E_{N_1} \cup E_{N_2}$

Proof: By assumption (2)1, Definition 3.1, definition (22) and definition (20).

$\langle 9 \rangle 5.$ Q.E.D.

Proof: By $\langle 9 \rangle 2$, $\langle 9 \rangle 3$, $\langle 9 \rangle 4$ and constraint (8) which ensures that events in a trace are totally ordered by time.

$\langle 8 \rangle 6.$ Q.E.D.

Proof: By $\langle 8 \rangle 5$, $\langle 8 \rangle 4$ and $\bot$-introduction.

$\langle 7 \rangle 2.$ Q.E.D.

Proof: Proof by contradiction

$\langle 6 \rangle 2.$ Q.E.D.

Proof: $\Rightarrow$-introduction.

$\langle 5 \rangle 2.$ Q.E.D.

Proof: $\forall$-introduction.

$\langle 4 \rangle 2.$ Q.E.D.

Proof: Induction step.

$\langle 3 \rangle 3.$ Q.E.D.

Proof: By induction over the length of $t$ with $\langle 3 \rangle 1$ as basis step and $\langle 3 \rangle 2$ as induction step.

$\langle 2 \rangle 2.$ Q.E.D.

Proof: $\Rightarrow$-introduction.

$\langle 1 \rangle 2.$ Q.E.D.

Proof: $\forall$-introduction.

Lemma B.23. Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$ and let $\alpha$ be a queue history in $B_{N_1 \cup N_2}$. Then

$$\forall t \in \mathcal{H} \cap \mathcal{E}^* : (c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha))$$

$$\land \exists t' \in \mathcal{H} : (\{!\} \times S \times N_1 \times N_2 \times Q) \otimes t \subseteq (\{!\} \times S \times N_1 \times N_2 \times Q) \otimes \alpha$$

$$\land t \in c(E_{N_1} \otimes t_1, D_{N_1}(E_{N_1} \otimes \alpha)) \land t \not\in E_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow$$
\[(\#t > \#(\mathcal{E}_{N_1} \otimes t_1) \wedge q.t[\#(\mathcal{E}_{N_1} \otimes t_1) + 1] < q.t_1[\#t_1])\]

**Proof:**

(1)1. Assume: \(t_1 \in \mathcal{H} \cap \mathcal{E}^*\)

Prove: \((c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha)) \wedge \exists t \in \mathcal{H}: \{!\} \times \mathcal{S} \times N_1 \times N_2 \times Q \otimes \alpha \supseteq t \in c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \wedge t \notin \mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow (\#t > \#(\mathcal{E}_{N_1} \otimes t_1) \wedge q.t[\#(\mathcal{E}_{N_1} \otimes t_1) + 1] < q.t_1[\#t_1])\)

(2)1. Assume: \(c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha)) \wedge \exists t \in \mathcal{H}: \{!\} \times \mathcal{S} \times N_1 \times N_2 \times Q \otimes \alpha \supseteq t \in c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \wedge t \notin \mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow (\#t > \#(\mathcal{E}_{N_1} \otimes t_1) \wedge q.t[\#(\mathcal{E}_{N_1} \otimes t_1) + 1] < q.t_1[\#t_1])\)

(3)1. Let: \(t \in \mathcal{H}\) such that \((\{!\} \times \mathcal{S} \times N_1 \times N_2 \times Q \otimes \alpha \supseteq t \in c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \wedge t \notin \mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))\)

Prove: By assumption (2)1.

(3)2. Assume: \(\#t \leq \#(\mathcal{E}_{N_1} \otimes t_1) \lor q.t[\#(\mathcal{E}_{N_1} \otimes t_1) + 1] \geq q.t_1[\#t_1]\)

Prove: By assumption (3)1.

(4)1. \(\mathcal{E}_{N_1} \otimes t_1 \subseteq t\)

Prove: By (3)1 \((t \in c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha))\) and Definition 3.1.

(4)2. \(\exists t' \in c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) : t = \mathcal{E}_{N_1} \otimes t'\)

(5)1. \(\exists t'' \in (t_1, D_{N_1} \otimes D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \otimes t'' = \mathcal{E}_{N_2} \otimes t' \wedge \mathcal{E}_{N_1} \otimes t'' = t\)

(6)1. \(t \in \mathcal{E}_{N_1} \otimes (D_{N_1} \otimes D_{N_2}(\alpha))\)

(7)1. \(t \in D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)\)

Prove: By (3)1 \((t \in c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha))\) and Definition 3.1.

(7)2. Q.E.D.

Prove: By (3)1 \(((\{!\} \times \mathcal{S} \times N_1 \times N_2 \times Q \otimes \alpha) \supseteq (\{!\} \times \mathcal{S} \times N_1 \times N_2 \times Q \otimes \alpha) \supseteq t \in c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha))\), (7)1 and definition (22).

(6)2. \(\forall t \in c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) : \mathcal{E}_{N_2} \otimes t \in \mathcal{E}_{N_2} \otimes (D_{N_1} \otimes D_{N_2}(\alpha))\)

(7)1. \(\forall t \in c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) : t \in (D_{N_1} \otimes D_{N_2}(\alpha))\)

Prove: By Definition 3.1.

(7)2. \(\forall \alpha \in (D_{N_1} \otimes D_{N_2}(\alpha)) : \mathcal{E}_{N_2} \otimes t \in \mathcal{E}_{N_2} \otimes (D_{N_1} \otimes D_{N_2}(\alpha))\)

Proof: By definition (22).

(7)3. Q.E.D.

Proof: By (7)1 and (7)2.

(6)3. Q.E.D.

Proof: By (6)1, (6)2, Lemma B.22 and \(\exists\) introduction.

(5)2. Let: \(t' \in c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))\) such that \(\exists t'' \in D_{N_1} \otimes D_{N_2}(\alpha) : \mathcal{E}_{N_2} \otimes t'' = \mathcal{E}_{N_2} \otimes t' \wedge \mathcal{E}_{N_1} \otimes t'' = t\)

Proof: By (5)1.

(5)3. Let: \(t'' \in D_{N_1} \otimes D_{N_2}(\alpha)\) such that \(\mathcal{E}_{N_2} \otimes t'' = \mathcal{E}_{N_2} \otimes t' \wedge \mathcal{E}_{N_1} \otimes t'' = t\)

Proof: By (5)2.

(5)4. \(t'' \in c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))\)

(6)1. \(t_1 \subseteq t''\)

(7)1. \(\forall k \in [0..q.t_1[\#t_1]] : t'' \downarrow k = t_1 \downarrow k\)

(8)1. \(\forall k \in [0..q.t_1[\#t_1]] : \mathcal{E}_{N_1} \otimes t'' \downarrow k = \mathcal{E}_{N_1} \otimes t_1 \downarrow k\)

(9)1. CASE: \(\#t \leq \#(\mathcal{E}_{N_1} \otimes t_1)\)
\[ \langle 10 \rangle 1. \ E_{N_1} \odot t'' = E_{N_1} \odot t_1 \]
\[ \langle 11 \rangle 1. \ t = E_{N_1} \odot t_1 \]
Proof: By \langle 4 \rangle 1 and assumption \langle 9 \rangle 1.
\[ \langle 11 \rangle 2. \ Q.E.D. \]
Proof: By \langle 11 \rangle 1, \langle 5 \rangle 3 and the rule of replacement [51].
\[ \langle 10 \rangle 2. \ Q.E.D. \]
Proof: By \langle 10 \rangle 1.
\[ \langle 9 \rangle 2. \ CASE: \ q.t[\#(E_{N_1} \odot t_1) + 1] \geq q.t_1[\#t_1] \]
\[ \langle 10 \rangle 1. \ ASSUME: \ \exists k \in [0..q.t_1[\#t_1]]: E_{N_1} \odot t''_{1k} \neq E_{N_1} \odot t_1_{1k} \]
Proof: \[ \langle 11 \rangle 1. \ \forall i, j \in [1..\#t_1]: i < j \Rightarrow q.t_1[i] < q.t_1[j] \]
Proof: By assumption \langle 1 \rangle 1 and requirement (8).
\[ \langle 11 \rangle 2. \ q.E_{N_1} \odot t''[E_{N_1} \odot \#t_1 + 1] \geq q.t_1[\#t_1] \]
Proof: By assumption \langle 9 \rangle 2 and \langle 5 \rangle 3.
\[ \langle 11 \rangle 3. \ \forall k \in [0..q.E_{N_1} \odot t_1[\#(E_{N_1} \odot t_1)]]: E_{N_1} \odot t''_{1k} = E_{N_1} \odot t_1_{1k} \]
\[ \langle 12 \rangle 1. \ E_{N_1} \odot t_1 \subseteq E_{N_1} \odot t'' \]
Proof: By \langle 5 \rangle 3, \langle 4 \rangle 1 and the rule of replacement [51].
\[ \langle 12 \rangle 2. \ Q.E.D. \]
Proof: By \langle 12 \rangle 1.
\[ \langle 11 \rangle 4. \ LET: \ k \in [0..q.t_1[\#t_1]] \text{ such that } E_{N_1} \odot t''_{1k} \neq E_{N_1} \odot t_1_{1k} \]
Proof: By assumption \langle 10 \rangle 1.
\[ \langle 11 \rangle 5. \ q.E_{N_1} \odot t_1[\#(E_{N_1} \odot t_1)] < k \leq q.t_1[\#t_1] \]
Proof: By \langle 11 \rangle 1, \langle 11 \rangle 2, \langle 11 \rangle 3 and \langle 11 \rangle 4.
\[ \langle 11 \rangle 6. \ q.E_{N_1} \odot t''[E_{N_1} \odot \#t_1 + 1] < q.t_1[\#t_1] \]
Proof: By \langle 11 \rangle 4 and \langle 11 \rangle 5.
\[ \langle 11 \rangle 7. \ Q.E.D. \]
Proof: By \langle 11 \rangle 2, \langle 11 \rangle 6 and \bot introduction.
\[ \langle 10 \rangle 2. \ Q.E.D. \]
Proof: Proof by contradiction.
\[ \langle 9 \rangle 3. \ Q.E.D. \]
Proof: The cases \langle 9 \rangle 1 and \langle 9 \rangle 2 are exhaustive.
\[ \langle 8 \rangle 2. \ \forall k \in [0..q.t_1[\#t_1]]: E_{N_2} \odot t''_{1k} = E_{N_2} \odot t_1_{1k} \]
\[ \langle 9 \rangle 1. \ \forall k \in [0..q.t_1[\#t_1]]: E_{N_2} \odot t_1'_{1k} = E_{N_2} \odot t_1_{1k} \]
\[ \langle 10 \rangle 1. \ t_1 \subseteq t' \]
Proof: By \langle 5 \rangle 2 and Definition 3.1.
\[ \langle 10 \rangle 2. \ Q.E.D. \]
Proof: By \langle 10 \rangle 1, since otherwise \ t' \ would not be well-formed.
\[ \langle 9 \rangle 2. \ Q.E.D. \]
Proof: By \langle 5 \rangle 3, \langle 9 \rangle 1 and the rule of replacement.
\[ \langle 8 \rangle 3. \ rng.t'' \subseteq E_{N_1} \cup E_{N_2} \]
Proof: By \langle 5 \rangle 3, Definition 5.3, definition (20) and definition (5).
\[ \langle 8 \rangle 4. \ rng.t_1 \subseteq E_{N_1} \cup E_{N_2} \]
Proof: By assumption (2)1, Definition 3.1, Definition 5.3, definition (20) and definition (5).
\[ \langle 8 \rangle 5. \ Q.E.D. \]
Proof: By \langle 8 \rangle 1, \langle 8 \rangle 2, \langle 8 \rangle 3 and \langle 8 \rangle 4.
\[ \langle 7 \rangle 2. \ Q.E.D. \]
Proof: By \langle 7 \rangle 1.
Lemma B.24. Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$ and let $\alpha$ be a queue history in $B_{N_1 \cup N_2}$. Then

$$\forall t_1 \in H \cap \mathcal{E}^*: #t_1 > 1 \land c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow$$

$$(c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha)) \cap CT_{N_1-N_2}(\alpha)) \cup \{t \in T \mid D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha) = c(t, D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha)) \cap CT_{N_1-N_2}(\alpha) \} =$$

$$\mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))$$

where $T = \{ t \in H \mid #t = #(\mathcal{E}_{N_1} \otimes t_1) + 1 \land \exists t' \in D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha) : t \subseteq t' \land$

$q.t[#(\mathcal{E}_{N_1} \otimes t_1) + 1] < q.t_1[#t_1]$}

**Proof:**

1. **Assume:** $t_1 \in H \cap \mathcal{E}^* \land #t_1 > 1$

   **Let:** $T = \{ t \in H \mid #t = #(\mathcal{E}_{N_1} \otimes t_1) + 1 \land \exists t' \in D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha) : t \subseteq t' \land$

   $q.t[#(\mathcal{E}_{N_1} \otimes t_1) + 1] < q.t_1[#t_1]$}

   **Prove:** $c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow$

   $$(c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha)) \cap CT_{N_1-N_2}(\alpha)) \cup \{c(t, D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha)) \mid t \in T \} =$$

   $$\mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))$$

2. **Assume:** $c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha))$

   **Prove:** $c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha)) \cap CT_{N_1-N_2}(\alpha) \cup \{c(t, D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha)) \mid t \not\in \mathcal{T} \} =$$

   $$\mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))$$

3. **Assume:** $t \in \mathcal{E}_{N_1} \otimes c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))$

   **Prove:** $t \in (c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha)) \cap CT_{N_1-N_2}(\alpha)) \land \not\in \mathcal{T}$

   $$(c(\mathcal{E}_{N_1} \otimes t_1, D_{N_1}(\mathcal{E}_{N_1}^{|\alpha} \otimes \alpha)) \cap CT_{N_1-N_2}(\alpha)) \not\in \mathcal{T}$$

4. **Prove:** By assumption (2) and Definition 3.1.

6. **Prove:** By definition (22).
\[7\] 3. Q.E.D.

Proof: By \(7\) 1 and \(7\) 2.

\(6\) 2. Q.E.D.

Proof: By \(6\) 1 and definition \(7\).

\(5\) 2. \(E_{N_1} \otimes (c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \subseteq c(E_{N_1} \otimes t_1, D_{N_1}(E_{N_1} \otimes \alpha))

Proof: By assumption \(1\) 1, assumption \(2\) 1 and Lemma B.21.

\(5\) 3. \(t \in c(E_{N_1} \otimes t_1, D_{N_1}(E_{N_1} \otimes \alpha)) \cap C T_{N_1-N_2}(\alpha)

Proof: By \(5\) 1, definition \(24\) and \(5\) 2.

\(5\) 4. \(t \not\in \bigcup_{t'' \in T} c(t'', D_{N_1}(E_{N_1} \otimes \alpha))

\(6\) 1. Assume: \(t \in \bigcup_{t'' \in T} c(t'', D_{N_1}(E_{N_1} \otimes \alpha))

Prove: \(\bot\)

\(7\) 1. \(\exists t' \in (c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) : t = E_{N_1} \otimes t'

Proof: By assumption \(4\) 1 and definition \(7\).

\(7\) 2. Let: \(t' \in (c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) such that \(t = E_{N_1} \otimes t'

Proof: By \(7\) 1.

\(7\) 3. \(t' \in H\)

Proof: By \(7\) 2 and Definition 3.1.

\(7\) 4. \(t_1 \sqsubseteq t'

Proof: By \(7\) 2 and Definition 3.1.

\(7\) 5. \(q.E_{N_1} \otimes t'[\#(E_{N_1} \otimes t_1) + 1] < q.t'[\#t_1]

\(8\) 1. \(t'[\#t_1] = t_1[\#t_1]

Proof: By \(7\) 4 and definition \(2\).

\(8\) 2. \(q.E_{N_1} \otimes t'[\#(E_{N_1} \otimes t_1) + 1] < q.t_1[\#t_1]

\(9\) 1. \(q.t[\#(E_{N_1} \otimes t_1) + 1] < q.t_1[\#t_1]

Proof: By assumption \(6\) 1 and assumption \(1\) 1.

\(9\) 2. Q.E.D.

Proof: By \(9\) 1, \(7\) 2 and the rule of replacement.

\(8\) 3. Q.E.D.

Proof: By \(8\) 1, \(8\) 2 and the rule of replacement [51].

\(7\) 6. \(\exists j \in [\#t_1 + 1..\#t'] : t'[j] = E_{N_1} \otimes t'[\#(E_{N_1} \otimes t_1) + 1]

\(8\) 1. \(\#(E_{N_1} \otimes t') > \#(E_{N_1} \otimes t_1)

Proof: By assumption \(6\) 1, assumption \(1\) 1, \(7\) 2 and the rule of replacement.

\(8\) 2. Q.E.D.

Proof: By \(7\) 4, \(8\) 1 and \(7\) 3, since by constraint \(8\) any event in \(\text{rng}.t' \cap E_{N_1} not in \text{rng}.t_1 must occur after \(t'[\#t_1]\) in \(t'\).

\(7\) 7. Let: \(j \in [\#t_1 + 1..\#t'] such that \(t'[j] = E_{N_1} \otimes t'[\#(E_{N_1} \otimes t_1) + 1]

Proof: By \(7\) 6

\(7\) 8. \(q.t'[j] < q.t[t_1[\#t_1]

Proof: By \(7\) 5, \(7\) 7 and the fact that every event in a trace is unique, due to the total ordering of events by time (constraint \(8\)).

\(7\) 9. \(t' \not\in H\)

Proof: By \(7\) 7, \(7\) 8 and constraint \(8\).

\(7\) 10. Q.E.D.

Proof: By \(7\) 3, \(7\) 9 and \(\bot\)-introduction.

\(6\) 2. Q.E.D.

Proof: By contradiction.

\(5\) 5. Q.E.D.
Lemma B.25. Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$ and let $\alpha$ be a queue history in $B_{N_1 \cup N_2}$. Then

$$D_{N_1}(E_{N_1}^1 \circ \alpha) \cap CT_{N_1-N_2}(\alpha) = E_{N_1} \circ (D_{N_1} \circ D_{N_2}(\alpha))$$

Proof:

1. $D_{N_1}(E_{N_1}^1 \circ \alpha) \cap CT_{N_1-N_2}(\alpha) = E_{N_1} \circ (D_{N_1} \circ D_{N_2}(\alpha))$

2. $E_{N_1} \circ (D_{N_1} \circ D_{N_2}(\alpha)) \subseteq D_{N_1}(E_{N_1}^1 \circ \alpha) \cap CT_{N_1-N_2}(\alpha)$

Proof: By assumption (3)1, definition (22), definition (24) and definition (7).

3. $E_{N_1} \circ (D_{N_1} \circ D_{N_2}(\alpha)) \subseteq D_{N_1}(E_{N_1}^1 \circ \alpha) \cap CT_{N_1-N_2}(\alpha)$

Proof: By (3)2.

4. $\exists t \in E_{N_1} \circ (D_{N_1} \circ D_{N_2}(\alpha))$ such that $t = E_{N_1} \circ t'$

Proof: By assumption (3)1 and definition (7).
Lemma B.26. Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$ and let $\alpha$ be a queue history in $B_{N_1 \cup N_2}$. Then

$$\forall t_1 \in H \cap \mathcal{E}^* \cdot x(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow$$

$$(\mathcal{E}_{N_1} \circ c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))) \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \land$$

$$(\mathcal{E}_{N_2} \circ c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))) \in \mathcal{F}_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)$$

Proof:

1. Assume: $t_1 \in H \cap \mathcal{E}^*$

   Prove: $c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow$

   $$(\mathcal{E}_{N_1} \circ c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))) \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \land$$

   $$(\mathcal{E}_{N_2} \circ c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))) \in \mathcal{F}_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)$$

2. Assume: $c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \in C(D_{N_1} \otimes D_{N_2}(\alpha))$

   Prove: $(\mathcal{E}_{N_1} \circ c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))) \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \land$

   $(\mathcal{E}_{N_2} \circ c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))) \in \mathcal{F}_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)$

3. $(\mathcal{E}_{N_1} \circ c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))) \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)$

4. Case: $\mathcal{E}_{N_1} \circ c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) = \emptyset$

5. Q.E.D.

   Proof: Since $\emptyset \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)$ by Definition 5.2 and Definition 5.3.

4. Case: $\mathcal{E}_{N_1} \circ c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) \neq \emptyset$

5. $c(\mathcal{E}_{N_1} \circ t_1, D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)) \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)$

6. $\exists t' \in D_{N_1} \otimes D_{N_2}(\alpha) : (\mathcal{E}_{N_1} \circ t' \in D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \land \mathcal{E}_{N_1} \circ t_1 \subseteq \mathcal{E}_{N_1} \circ t'$

7. $\forall t \in c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) : t \in D_{N_1} \otimes D_{N_2}(\alpha) \land t_1 \subseteq t$

   Proof: By assumption $\{2\}$, Definition 3.2 and Definition 3.1.

7. $\forall t \in D_{N_1} \otimes D_{N_2}(\alpha) \cdot (\mathcal{E}_{N_1} \circ t \in D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)$

   Proof: By definition (22).

7. Let: $t \in c(t_1, D_{N_1} \otimes D_{N_2}(\alpha))$

7. $t \in D_{N_1} \otimes D_{N_2}(\alpha) \land \mathcal{E}_{N_1} \circ t \in D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \land \mathcal{E}_{N_1} \circ t_1 \subseteq \mathcal{E}_{N_1} \circ t$

   Proof: By $\{7\}$, $\{7\}$, $\{7\}$ and $\forall$ elimination.

7. Q.E.D.

   Proof: By $\{7\}$ and $\exists$-introduction.

6. Q.E.D.

   Proof: By $\{6\}$ and Definition 3.1.

5. Case: $\mathcal{E}_{N_1} \circ c(t_1, D_{N_1} \otimes D_{N_2}(\alpha)) = c(\mathcal{E}_{N_1} \circ t_1, D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha))$

6. Q.E.D.

   Proof: By $\{5\}$ and assumption $\{5\}$.
Theorem 5.5 Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$, let $\alpha$ be a queue history in $B_{N_1 \cup N_2}$ and let $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ be a measure on
\( C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \) as defined by (25). Then the function \( \mu_{N_1} \otimes \mu_{N_2}(\alpha) \) is well defined. That is:

\[ \forall c \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) : (\mathcal{E}_{N_1} \otimes c) \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^\perp \otimes \alpha) \land (\mathcal{E}_{N_2} \otimes c) \in \mathcal{F}_{N_2}(\mathcal{E}_{N_2}^\perp \otimes \alpha) \]

**Proof.** Follows from Lemma B.16 and Lemma B.26.

**Lemma B.27.** Let \((D_1, \mathcal{F}_1, \mu_1)\) and \((D_2, \mathcal{F}_2, \mu_2)\) be measure spaces. Then

\[ \forall A_1 \in \mathcal{F}_1, A_2 \in \mathcal{F}_2 : \\
(\forall \phi \in (\mathcal{P}(D_1 \times D_2))^\omega : \forall i \in [1..\#\phi] : \phi[i] \in \mathcal{F}_1 \otimes \mathcal{F}_2 \\
\land (\forall m, j \in [1..\#\phi] : j \neq m \implies \phi[j] \cap \phi[m] = \emptyset) \\
\land \left( \bigcup^{\#\phi}_{i=1} \phi[i] = A_1 \times A_2 \land \bigcup^{\#\phi}_{i=1} \phi[i] \in \mathcal{F}_1 \otimes \mathcal{F}_2 \right) \\
\implies (\mu_1(A_1) \cdot \mu_2(A_2) = \sum^{\#\phi}_{i=1} \mu_1(\{\Pi_1.p \mid p \in \phi[i]\}) \cdot \mu_2(\{\Pi_2.p \mid p \in \phi[i]\})) \]

**Proof:**

1. Assume: \( A_1 \in \mathcal{F}_1 \land A_2 \in \mathcal{F}_2 : \\
Prove: (\forall \phi \in (\mathcal{P}(D_1 \times D_2))^\omega : \forall i \in [1..\#\phi] : \phi[i] \in \mathcal{F}_1 \otimes \mathcal{F}_2 \\
\land (\forall m, j \in [1..\#\phi] : j \neq m \implies \phi[j] \cap \phi[m] = \emptyset) \\
\land \left( \bigcup^{\#\phi}_{i=1} \phi[i] = A_1 \times A_2 \land \bigcup^{\#\phi}_{i=1} \phi[i] \in \mathcal{F}_1 \otimes \mathcal{F}_2 \right) \\
\implies (\mu_1(A_1) \cdot \mu_2(A_2) = \sum^{\#\phi}_{i=1} \mu_1(\{\Pi_1.p \mid p \in \phi[i]\}) \cdot \mu_2(\{\Pi_2.p \mid p \in \phi[i]\})) \]

2. Assume: \( \phi \in (\mathcal{P}(D_1 \times D_2))^\omega \)
Prove: \( \forall i \in [1..\#\phi] : \phi[i] \in \mathcal{F}_1 \otimes \mathcal{F}_2 \\
\land (\forall m, j \in [1..\#\phi] : j \neq m \implies \phi[j] \cap \phi[m] = \emptyset) \\
\land \left( \bigcup^{\#\phi}_{i=1} \phi[i] = A_1 \times A_2 \land \bigcup^{\#\phi}_{i=1} \phi[i] \in \mathcal{F}_1 \otimes \mathcal{F}_2 \right) \\
\implies (\mu_1(A_1) \cdot \mu_2(A_2) = \sum^{\#\phi}_{i=1} \mu_1(\{\Pi_1.p \mid p \in \phi[i]\}) \cdot \mu_2(\{\Pi_2.p \mid p \in \phi[i]\})) \]

3. Assume: \( i \in [1..\#\phi] \)
Prove: \( \phi[i] \in \mathcal{F}_1 \otimes \mathcal{F}_2 \land (\forall m, j \in [1..\#\phi] : j \neq m \implies \phi[j] \cap \phi[m] = \emptyset) \\
\land \left( \bigcup^{\#\phi}_{i=1} \phi[i] = A_1 \times A_2 \land \bigcup^{\#\phi}_{i=1} \phi[i] \in \mathcal{F}_1 \otimes \mathcal{F}_2 \right) \\
\implies (\mu_1(A_1) \cdot \mu_2(A_2) = \sum^{\#\phi}_{i=1} \mu_1(\{\Pi_1.p \mid p \in \phi[i]\}) \cdot \mu_2(\{\Pi_2.p \mid p \in \phi[i]\})) \]

4. Assume: \( \phi[i] \in \mathcal{F}_1 \otimes \mathcal{F}_2 \land (\forall m, j \in [1..\#\phi] : j \neq m \implies \phi[j] \cap \phi[m] = \emptyset) \\
\land \left( \bigcup^{\#\phi}_{i=1} \phi[i] = A_1 \times A_2 \land \bigcup^{\#\phi}_{i=1} \phi[i] \in \mathcal{F}_1 \otimes \mathcal{F}_2 \right) \\
\implies (\mu_1(A_1) \cdot \mu_2(A_2) = \sum^{\#\phi}_{i=1} \mu_1(\{\Pi_1.p \mid p \in \phi[i]\}) \cdot \mu_2(\{\Pi_2.p \mid p \in \phi[i]\})) \]

5. There is a unique measure \( \mu \) on \( \mathcal{F}_1 \otimes \mathcal{F}_2 \), such that

\[ \mu(A_1) \cdot \mu(A_2) = \mu(\bigcup^{\#\phi}_{i=1} \phi[i]) \]

**Proof:** By assumption 1, assumption 4)1 \((A_1 \times A_2 = \bigcup^{\#\phi}_{i=1} \phi[i])\) and Theorem B.4.

5.2. Let: \( \mu \) be the unique measure \( \mu \) on \( \mathcal{F}_1 \otimes \mathcal{F}_2 \), such that

\[ \mu(A_1) \cdot \mu(A_2) = \mu(\bigcup^{\#\phi}_{i=1} \phi[i]) \]

**Proof:** By 5.1.

5.3. \( \mu(\bigcup^{\#\phi}_{i=1} \phi[i]) = \sum^{\#\phi}_{i=1} \mu_1(\{\Pi_1.p \mid p \in \phi[i]\}) \cdot \mu_2(\{\Pi_2.p \mid p \in \phi[i]\}) \)
Lemma B.28. Let \( I_{N_1} \) and \( I_{N_2} \) be two probabilistic component executions such that \( N_1 \cap N_2 = \emptyset \) and let \( \alpha \) be a queue history in \( B_{N_1 \cup N_2} \). Then

\[
\forall \phi \in \mathbb{P}(\mathcal{H})^\omega : (\forall i \in [1..\#\phi] : \phi[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \\
\land (\forall m, j \in [1..\#\phi] : j \neq m \Rightarrow \phi[j] \cap \phi[m] = \emptyset) \\
\land \bigcup_{i=1}^{\#\phi} \phi[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \\
\Rightarrow (\mu_{N_1} \otimes \mu_{N_2}(\alpha)(\bigcup_{i=1}^{\#\phi} \phi[i]) = \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i]))
\]

Proof:

(1) Assume: \( \phi \in \mathbb{P}(\mathcal{H})^\omega \)

Prove: (\( \forall i \in [1..\#\phi] : \phi[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \\
\land (\forall m, j \in [1..\#\phi] : j \neq m \Rightarrow \phi[j] \cap \phi[m] = \emptyset) \\
\land \bigcup_{i=1}^{\#\phi} \phi[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \\
\Rightarrow (\mu_{N_1} \otimes \mu_{N_2}(\alpha)(\bigcup_{i=1}^{\#\phi} \phi[i]) = \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i]))

(2) Assume: \( \forall i \in [1..\#\phi] : \phi[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \\
\land (\forall m, j \in [1..\#\phi] : j \neq m \Rightarrow \phi[j] \cap \phi[m] = \emptyset) \\
\land \bigcup_{i=1}^{\#\phi} \phi[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \\
Prove: \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\bigcup_{i=1}^{\#\phi} \phi[i]) = \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i])
(3.1) $\mathcal{E}_{N_1} \otimes \bigcup_{i=1}^{\# \phi} \phi[i] \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \land \mathcal{E}_{N_2} \otimes \bigcup_{i=1}^{\# \phi} \phi[i] \in \mathcal{F}_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)$

Proof: By assumption (2.1) and Theorem 5.5.

(3.2) $\forall i \in [1..\# \phi]: \mathcal{E}_{N_1} \otimes \phi[i] \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \land \mathcal{E}_{N_2} \otimes \phi[i] \in \mathcal{F}_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)$

Proof: By assumption (2.1) and Theorem 5.5.

(3.3) $\mu_{N_1} \otimes \mu_{N_2}(\alpha)(\bigcup_{i=1}^{\# \phi} \phi[i]) = f_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)(\mathcal{E}_{N_1} \otimes \bigcup_{i=1}^{\# \phi} \phi[i]) \cdot f_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)(\mathcal{E}_{N_2} \otimes \bigcup_{i=1}^{\# \phi} \phi[i])$

Proof: By definition (25) and (3.1).

(3.4) $\sum_{i=1}^{\# \phi} f_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)(\mathcal{E}_{N_1} \otimes \bigcup_{i=1}^{\# \phi} \phi[i]) \cdot f_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)(\mathcal{E}_{N_2} \otimes \bigcup_{i=1}^{\# \phi} \phi[i]) = 4. Q.E.D.$

Proof: By assumption (4.1) and Definition 3.3.

(4.1) Case: $\bigcup_{i=1}^{\# \phi} \phi[i] \in C(D_{N_1} \otimes D_{N_2}(\alpha)) \setminus C(D_{N_1} \otimes D_{N_2}(\alpha))$

(4.1.1) Let: $\psi$ be a sequence in $(\mathcal{P}(D_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \times D_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)))^\omega$ such that

$p = \# \psi \land \forall i \in [1..\# \psi]: \psi[i] = \{(\mathcal{E}_{N_1} \otimes t, \mathcal{E}_{N_2} \otimes t) \mid t \in \phi[i]\}$

(4.1.2) $\forall i \in [1..\# \psi]: \psi[i] \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1} \otimes \alpha) \land \mathcal{F}_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)$

Proof: By assumption (3.1), (5.1), the assumption that $I_{N_1}$ and $I_{N_3}$ are probabilistic component executions, Definition 5.3, Definition 3.4, Lemma B.27 and $\forall$ elimination.

(5.3) $\sum_{i=1}^{\# \psi} f_{N_1}(\mathcal{E}_{N_1} \otimes \alpha)(\mathcal{E}_{N_1} \otimes \bigcup_{i=1}^{\# \phi} \phi[i]) \cdot f_{N_2}(\mathcal{E}_{N_2} \otimes \alpha)(\mathcal{E}_{N_2} \otimes \bigcup_{i=1}^{\# \phi} \phi[i]) = 6. Q.E.D.$

Proof: By (7.1), (5.1), assumption (3.2) and definition (7).
(6.3)  \( \bigcup_{i=1}^{\#\psi} \psi[i] = \mathcal{E}_{N_1} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \times \mathcal{E}_{N_2} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \)

(7.1)  \( \bigcup_{i=1}^{\#\psi} \psi[i] \subseteq \mathcal{E}_{N_1} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \times \mathcal{E}_{N_2} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \)

(8.1)  Assume: \( p \in \bigcup_{i=1}^{\#\psi} \psi[i] \)

**PROVE:**  \( p \in \mathcal{E}_{N_1} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \times \mathcal{E}_{N_2} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \)

(9.1)  \( \exists \in \bigcup_{i=1}^{\#\phi} \phi[i] : p = (\mathcal{E}_{N_1} \otimes t, \mathcal{E}_{N_2} \otimes t) \)

**PROOF:**  By assumption (8.1) and (5.1).

(9.2)  Let: \( t \in \bigcup_{i=1}^{\#\phi} \phi[i] \) such that \( p = (\mathcal{E}_{N_1} \otimes t, \mathcal{E}_{N_2} \otimes t) \)

**PROOF:**  By (9.1).

(9.3)  \( \mathcal{E}_{N_1} \otimes t \in \mathcal{E}_{N_1} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \) \( \land \mathcal{E}_{N_2} \otimes t \in \mathcal{E}_{N_2} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \)

**PROOF:**  By (9.2) and definition (7).

(9.4)  Q.E.D.

**PROOF:**  By (9.2), (9.3) and definition (32).

(8.2)  Q.E.D.

**PROOF:**  \( \subseteq \) rule

(7.2)  \( \mathcal{E}_{N_1} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \times \mathcal{E}_{N_2} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \subseteq \bigcup_{i=1}^{\#\psi} \psi[i] \)

(8.1)  Assume: \( p \in \mathcal{E}_{N_1} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \times \mathcal{E}_{N_2} \odot \bigcup_{i=1}^{\#\phi} \phi[i] \)

**PROOF:**  \( p \in \bigcup_{i=1}^{\#\psi} \psi[i] \)

(9.1)  \( \exists \in (\mathcal{E}_{N_1} \odot \bigcup_{i=1}^{\#\phi} \phi[i]) : \exists t_2 \in (\mathcal{E}_{N_2} \odot \bigcup_{i=1}^{\#\phi} \phi[i]) : p = (t_1, t_2) \)

**PROOF:**  By assumption (8.1) and definition (32).

(9.2)  Let: \( t_1 \in (\mathcal{E}_{N_1} \odot \bigcup_{i=1}^{\#\phi} \phi[i]), t_2 \in (\mathcal{E}_{N_2} \odot \bigcup_{i=1}^{\#\phi} \phi[i]) \) such that \( p = (t_1, t_2) \)

**PROOF:**  By (9.1).

(9.3)  \( \exists \in \bigcup_{i=1}^{\#\phi} \phi[i] : \mathcal{E}_{N_1} \otimes t = t_1 \land \mathcal{E}_{N_2} \otimes t = t_2 \)

(10.1)  \( \exists t \in \mathcal{H} \cap \mathcal{E}^* : \exists t' \in D_{N_1} \otimes D_{N_2}(\alpha) : t \subseteq t' \land c(t, D_{N_1} \otimes D_{N_2}(\alpha)) = \bigcup_{i=1}^{\#\phi} \phi[i] \)

**PROOF:**  By assumption (4.2).

(10.2)  Let: \( t \in \mathcal{H} \cap \mathcal{E}^* \) such that \( \exists t' \in D_{N_1} \otimes D_{N_2}(\alpha) : t \subseteq t' \land c(t, D_{N_1} \otimes D_{N_2}(\alpha)) = \bigcup_{i=1}^{\#\phi} \phi[i] \)

**PROOF:**  By (10.1).

(10.3)  \( \forall t_1, t_2 \in \mathcal{H} : t_1 \in \mathcal{E}_{N_1} \odot c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \land t_2 \in \mathcal{E}_{N_2} \odot c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \Rightarrow (\exists t_1 \in (\mathcal{E}_{N_1} \otimes D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \otimes t_1 = t_1 \land \mathcal{E}_{N_2} \otimes t_1 = t_2) \)

**PROOF:**  By assumption (2)1, (10)2 the rule of replacement [51] and Lemma B.22.

(10.4)  \( t_1 \in \mathcal{E}_{N_1} \odot c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \land t_2 \in \mathcal{E}_{N_2} \odot c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \land \mathcal{E}_{N_1} \odot c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \land \mathcal{E}_{N_2} \odot c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \)

**PROOF:**  By (10)2 and the rule of equality between functions [51].

(9.4)  Q.E.D.

**PROOF:**  By (9)2, (11)1 and the rule of replacement [51].

(10.5)  \( \exists t' \in c(t, D_{N_1} \otimes D_{N_2}(\alpha)) : \mathcal{E}_{N_1} \otimes t' = t_1 \land \mathcal{E}_{N_2} \otimes t' = t_2 \)

**PROOF:**  By (10)3, (10)4 and \( \forall \) elimination.

(10.6)  Let: \( t' \in c(t, D_{N_1} \otimes D_{N_2}(\alpha)) \) such that \( \mathcal{E}_{N_1} \otimes t' = t_1 \land \mathcal{E}_{N_2} \otimes t = t_2 \)

**PROOF:**  By (10)5.

(10.7)  \( t' \in \bigcup_{i=1}^{\#\phi} \phi[i] \)

75
Proof: By (10)2, (10)6 and the rule of replacement [51].

(10)8. Q.E.D.

Proof: By (10)7, (10)6 and $\exists$ introduction.

(9)4. Let: $t \in \bigcup_{i=1}^{\# \psi} \phi[i]$ such that $E_{N_1} \otimes t = t_1 \land E_{N_2} \otimes t = t_2$

Proof: By (9)3.

(9)5. $(E_{N_1} \otimes t, E_{N_2} \otimes t) \in \bigcup_{i=1}^{\# \psi} \psi[i]$

Proof: By (9)4 and (5)1.

(9)6. Q.E.D.

Proof: By (9)2, (9)4, (9)5 and the rule of replacement [51].

(8)2. Q.E.D.

Proof: $\subseteq$-rule.

(7)3. Q.E.D.

Proof: By (7)1, (7)2 and the $\Rightarrow$-rule for sets [29].

(6)4. $\bigcup_{i=1}^{\# \alpha} \psi[i] \in F_{N_1}(E_{N_1} \otimes \alpha) \times F_{N_2}(E_{N_2} \otimes \alpha)$

(7)1. $E_{N_1} \otimes \bigcup_{i=1}^{\# \phi} \phi[i] \times E_{N_2} \otimes \bigcup_{i=1}^{\# \phi} \phi[i] \in F_{N_1}(E_{N_1} \otimes \alpha) \times F_{N_2}(E_{N_2} \otimes \alpha)$

Proof: By (3)1 and Definition A.10.

(7)2. Q.E.D.

Proof: By (6)3, (7)1 and the rule of replacement.

(6)5. Q.E.D.

Proof: By (5)2, (6)1, (6)2, (6)3, (6)4 and $\Rightarrow$ elimination.

(5)4. $\forall i \in [1..\# \psi]: \{\Pi_1.p| p \in \psi[i]\} = E_{N_1} \otimes \phi[i] \land \{\Pi_2.p| p \in \psi[i]\} = E_{N_2} \otimes \phi[i]$

Proof: By (5)1.

(5)5. Q.E.D.

Proof: By (5)3, (5)4 and the rule of replacement.

(4)3. Q.E.D.

Proof: By assumption (2)1, the cases (4)1 and (4)2 are exhaustive.

(3)5. $\sum_{i=1}^{\# \phi} f_{N_1}(E_{N_1} \otimes \alpha)(E_{N_1} \otimes \phi[i]) \cdot f_{N_2}(E_{N_2} \otimes \alpha)(E_{N_2} \otimes \phi[i]) = \sum_{i=1}^{\# \phi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i])$

(4)1. $\forall i \in [1..\# \phi]: \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i]) = f_{N_1}(E_{N_1} \otimes \alpha)(E_{N_1} \otimes \phi[i]) \cdot f_{N_2}(E_{N_2} \otimes \alpha)(E_{N_2} \otimes \phi[i])$

Proof: By definition (25) and (3)2.

(4)2. Q.E.D.

Proof: By (4)1 and the rule of equality between functions [51].

(3)6. Q.E.D.

Proof: By (3)3, (3)4, (3)5 and the rule of transitivity [51].

(2)2. Q.E.D.

Proof: By $\Rightarrow$-introduction.

(1)2. Q.E.D.

Proof: By $\forall$-introduction.

Lemma 5.6 Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$ and let $\mu_{N_1} \otimes \mu_{N_2}$ be a measure on the composite extended cone set of $D_{N_1} \otimes D_{N_2}$ as defined in (25). Then, for all complete queue histories $\alpha \in B_{N_1 \cup N_2}$

1. $\mu_{N_1} \otimes \mu_{N_2}(\alpha)(\emptyset) = 0$

2. $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ is $\sigma$-additive

3. $\mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \leq 1$

Proof: (Proof of Lemma 5.6.1.)
\(\forall \alpha \in B_N: \mu_{N_1} \otimes \mu_{N_2}(\alpha)(0) = 0\)

(2.1) Assume: \(\alpha \in B_N\)

Prove: \(\mu_{N_1} \otimes \mu_{N_2}(\alpha)(0) = 0\)

(3.1) \(\mu_{N_1} \otimes \mu_{N_2}(\alpha)(0) = f_{N_1}(\mathcal{E}_{N_1}^1 \otimes \alpha)(0) \cdot f_{N_2}(\mathcal{E}_{N_2}^1 \otimes \alpha)(0)\)

(4.1) \(\mathcal{E}_{N_1} \otimes \emptyset = \emptyset \land \mathcal{E}_{N_2} \otimes \emptyset = \emptyset\)

Proof: By definition (7).

(4.2) \(\emptyset \in \mathcal{F}_{N_1}(\mathcal{E}_{N_1}^1 \otimes \alpha) \land \emptyset \in \mathcal{F}_{N_2}(\mathcal{E}_{N_2}^1 \otimes \alpha)\)

Proof: By assumption (2.1), definition (21) and Definition 5.3.

(4.3) Q.E.D.

Proof: By definition (25), (4.1),(4.2) and the rule of replacement.

(3.2) \(f_{N_1}(\mathcal{E}_{N_1}^1 \otimes \alpha)(0) = 0\)

Proof: By the assumption that \(I_{N_1}\) is a probabilistic component execution, Definition 5.3, Definition 5.1 and Definition A.6.

(3.3) \(f_{N_2}(\mathcal{E}_{N_2}^1 \otimes \alpha)(0) = 0\)

Proof: By the assumption that \(I_{N_2}\) is a probabilistic component execution, Definition 5.3, Definition 5.1 and Definition A.6.

(3.4) Q.E.D.

Proof: By (3.1), (3.2), (3.3) and elementary arithmetic.

(2.2) Q.E.D.

Proof: \(\forall\)-introduction.

(1.2) Q.E.D.

Proof. (Proof of Lemma 5.6.2.) Follows from Lemma B.28.
Lemma B.29. Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$, let $\alpha$ be a queue history in $B_{N_1 \cup N_2}$. Let $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ be a measure on $C_E(D_{N_1} \otimes D_{N_2}(\alpha))$ as defined by (25) and let $F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))$ be an extension of $C_E(D_{N_1} \otimes D_{N_2}(\alpha))$ as defined in Definition A.11. The function $\mu_{N_1} \otimes \mu_{N_2}^t(\alpha)$ defined by

$$\mu_{N_1} \otimes \mu_{N_2}(\alpha)(c) = \begin{cases} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(c) & \text{if } c \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \\ \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) - \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \setminus c & \text{if } c \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \setminus C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \end{cases}$$

is a measure on $F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))$.

Proof:

1. $\mu_{N_1} \otimes \mu_{N_2}(\alpha)(\emptyset) = 0$
2. $\emptyset \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \setminus C_E(D_{N_1} \otimes D_{N_2}(\alpha))$

   Proof: By Definition A.11.
3. $\mu_{N_1} \otimes \mu_{N_2}^t(\alpha)(\emptyset) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) - \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha))$

   Proof: By (2) and definition (33)

4. $\emptyset \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))$ and Lemma B.8.
Proof: By assumption (3)1, Proposition B.1 \((F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))) \subseteq \mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha))\) and elementary set theory.

\(4.2. \exists \phi' \in \mathcal{P}(\cup) : \forall i \in [1..\#\phi'] : \phi'[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \land \forall m, j \in [1..\#\phi'] : j \neq m \Rightarrow \phi'[j] \cap \phi'[m] = \emptyset \land \bigcup_{i=1}^{\#\phi'} \phi'[i] = \bigcup_{i=1}^{\#\phi} \phi[i]e \)

Proof: By (4)1, Lemma B.14 and Corollary B.7.

\(4.3. \text{Let: } \phi' \in \mathcal{P}(\cup) \text{ such that } \forall i \in [1..\#\phi'] : \phi'[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \land \forall m, j \in [1..\#\phi'] : j \neq m \Rightarrow \phi'[j] \cap \phi'[m] = \emptyset \land \bigcup_{i=1}^{\#\phi'} \phi'[i] = \bigcup_{i=1}^{\#\phi} \phi[i]

Proof: By (4)2.

\(4.4. \text{Case: } \bigcup_{i=1}^{\#\phi'} \phi'[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \)

\(5.1. \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\bigcup_{i=1}^{\#\phi'} \phi[i]) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\bigcup_{i=1}^{\#\phi} \phi[i]) \)

Proof: By assumption (4)4, (4)3, the rule of replacement [51] and definition (33).

\(5.2. \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\bigcup_{i=1}^{\#\phi'} \phi[i]) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\bigcup_{i=1}^{\#\phi} \phi[i]) \)

Proof: By (4)3 and the rule of equality of functions [51].

\(5.3. \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\bigcup_{i=1}^{\#\phi'} \phi[i]) = \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i]) \)

Proof: By (4)3, assumption (4)4 and Lemma B.28.

\(5.4. \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i]) = \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i]) \)

\(7.1. \forall i \in [1..\#\phi'] : \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i]) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i]) \)

Proof: By (4)3 and definition (33), since \(\phi\) and \(\phi'\) are two different partitions of the same set.

(6)4. Q.E.D.

Proof: By (7)1 and the rule of equality between functions.

\(6.3. \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i]) = \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i]) \)

Proof: By (4)3 and definition (33), since \(\phi\) and \(\phi'\) are two different partitions of the same set.

(6)4. Q.E.D.

Proof: By (6)1, (6)2 (6)3 and the rule of transitivity [51].

(5)4. Q.E.D.

Proof: By (5)1, (5)2, (5)3 and the rule of transitivity [51].

\(4.5. \text{Case: } \bigcup_{i=1}^{\#\phi'} \phi'[i] \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \setminus C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \)

\(5.1. D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcup_{i=1}^{\#\phi'} \phi'[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \)

Proof: By assumption (4)5 and Definition A.11.

\(5.2. \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\bigcup_{i=1}^{\#\phi} \phi[i]) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \cup \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \cup \bigcup_{i=1}^{\#\phi} \phi[i]) \)

Proof: By assumption (4)5, (4)3, the rule of replacement [51] and definition (33).

\(5.3. \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \cup \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \cup \bigcup_{i=1}^{\#\phi} \phi[i]) = \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i]) \)

\(6.1. \text{Let: } \psi'[i] = D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcup_{i=1}^{\#\phi} \phi[i] \cup \phi'[i] \cup \phi'[i]) \)

\(6.2. \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) = \sum_{i=1}^{\#\psi}[i] \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\psi'[i]) \)

\(7.1. \bigcup_{i=1}^{\#\psi}(i) = D_{N_1} \otimes D_{N_2}(\alpha) \)

\(8.1. \bigcup_{i=1}^{\#\phi'} \phi'[i] = \bigcup_{i=1}^{\#\phi} \phi[i] \cup D_{N_1} \otimes D_{N_2}(\alpha) \cup \bigcup_{i=1}^{\#\phi} \phi[i] \)

Proof: By (6)1.

\(8.2. \bigcup_{i=1}^{\#\phi'} \phi'[i] \cup (D_{N_1} \otimes D_{N_2}(\alpha) \cup \bigcup_{i=1}^{\#\phi} \phi[i]) = D_{N_1} \otimes D_{N_2}(\alpha) \)

79
(9)1. $\bigcup_{i=1}^{\#\phi} \phi[i] \cup (D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcup_{i=1}^{\#\phi} \phi[i]) = D_{N_1} \otimes D_{N_2}(\alpha)$

(10)1. $\bigcup_{i=1}^{\#\phi} \phi[i] \subseteq D_{N_1} \otimes D_{N_2}(\alpha)$

(11)1. $\forall A \in \mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha): A \subseteq D_{N_1} \otimes D_{N_2}(\alpha)$

Proof: By Definition A.4 and Definition A.3.

(11)2. Q.E.D.

Proof: By assumption (3)1, Proposition B.1 ($F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \subseteq \mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)$), (11)1 and $\forall$ elimination.

(10)2. Q.E.D.

Proof: By (10)1 and elementary set theory.

(9)2. Q.E.D.

Proof: By (9)1 and (4)3.

(8)3. Q.E.D.

Proof: By (8)1, (8)2 and the rule of transitivity [51].

(7)2. $\forall i \in [1..\#\psi']: \psi'[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha))$
\[\land \forall i \in [1..\#\psi']: \psi'[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha))\]

Proof: By (7)1 and Definition 3.3.

(8)2. $\forall i \in [1..\#\psi']: \psi'[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha))$

Proof: By (5)1, (4)3 and (6)1.

(8)3. $\forall i \in [1..\#\psi']: \psi'[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha))$

Proof: By (4)3 and (6)1.

(8)4. Q.E.D.

Proof: By (8)1, (8)2, (8)3 and $\land$-introduction.

(7)3. Q.E.D.

Proof: By (7)1, (7)2 and Lemma 5.6.

(6)3. $\mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) - \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcup_{i=1}^{\#\phi} \phi[i]) = \sum_{i=1}^{\#\psi'} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\psi'[i]) - \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\psi'[1])$

(7)1. $\mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcup_{i=1}^{\#\phi} \phi[i]) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\psi'[1])$

Proof: By (6)1 and the rule of equality between functions [51].

(7)2. Q.E.D.

Proof: By (7)1, (6)2 and the rule of replacement [51].

(6)4. $\sum_{i=1}^{\#\psi'} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\psi'[i]) \leq 1$

Proof: By the assumption that $\mu_{N_1} \otimes \mu_{N_2}$ is a measure on $C_E(D_{N_1} \otimes D_{N_2}(\alpha))$ as defined by (25) and Lemma 5.6.3.

(9)2. Q.E.D.

Proof: By (6)2, (9)1 and the rule of replacement [51].

(8)2. $\mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\psi'[1]) + \sum_{i=2}^{\#\psi'} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\psi'[i])$

Proof: By (6)1, (6)2 and (8)1, since the sum of the terms of a converging series is preserved when regrouping the terms in the same
Lemma B.30. Let \( I_N \) be a probabilistic component execution as defined in Definition 5.3 and let \( \alpha \) be a complete queue history \( \alpha \in B_N \). Let \( F_1(C_E(D_N(\alpha))) \) be an extension of \( C_E(D_N(\alpha)) \) as defined in Definition A.11. Then

\[
\forall B, A \in F_1(C_E(D_N(\alpha))) : (B \cap A \neq \emptyset) \Rightarrow (\{(D_N(\alpha) \setminus B) \cap (D_N(\alpha) \setminus A) = \emptyset\} \cup (((D_N(\alpha) \setminus B) \subseteq (D_N(\alpha) \setminus A)) \vee ((D_N(\alpha) \setminus A) \subseteq (D_N(\alpha) \setminus B))))
\]

Proof:

81
1. Assume: $A \in F_1(C_E(D_N(\alpha))) \land B \in F_1(C_E(D_N(\alpha)))$

   **Proof:** $(B \cap A \neq \emptyset) \Rightarrow (((D_N(\alpha) \setminus B) \cap (D_N(\alpha) \setminus A) = \emptyset) \lor

   ((D_N(\alpha) \setminus B) \subseteq (D_N(\alpha) \setminus A)) \lor

   ((D_N(\alpha) \setminus A) \subseteq (D_N(\alpha) \setminus B)))$

2. Assume: $B \cap A \neq \emptyset$

   **Proof:** $((D_N(\alpha) \setminus B) \cap (D_N(\alpha) \setminus A) = \emptyset) \lor

   ((D_N(\alpha) \setminus B) \subseteq (D_N(\alpha) \setminus A)) \lor

   ((D_N(\alpha) \setminus A) \subseteq (D_N(\alpha) \setminus B)))$

3.1. Case: $A \in C_E(D_N(\alpha)) \land B \in C_E(D_N(\alpha))$

   **Proof:** By assumption (3)1 and Corollary B.6.

3.2. Case: $(D_N(\alpha) \setminus A) \in C_E(D_N(\alpha)) \land (D_N(\alpha) \setminus B) \in C_E(D_N(\alpha))$

   **Proof:** By assumption (2)1 and elementary set theory.

3.3. Case: $(D_N(\alpha) \setminus A) \in C_E(D_N(\alpha)) \land B \in C_E(D_N(\alpha))$

   **Proof:** By assumption (2)1 and elementary set theory.

3.4. Case: $(D_N(\alpha) \setminus A) \subseteq B$

   **Proof:** By assumption (4)2 and elementary set theory.

3.5. Case: $(D_N(\alpha) \setminus A) \cap (D_N(\alpha) \setminus B) = \emptyset$

   **Proof:** By assumption (4)2 and elementary set theory.

4. Case: $B \subseteq (D_N(\alpha) \setminus A)$

   **Proof:** By assumption (2)1 and elementary set theory.

5. Case: $(D_N(\alpha) \setminus A) \cap B = \emptyset$

   **Proof:** By assumption (4)3 and elementary set theory.

4.2. Case: $A \in C_E(D_N(\alpha)) \land (D_N(\alpha) \setminus B) \in C_E(D_N(\alpha))$

   **Proof:** Symmetrical to step (3)3.

5.1. $A \subseteq B$

   **Proof:** By assumption (1)1, the cases (4)2 and (4)3 are exhaustive.

4.4. Case: $A \in C_E(D_N(\alpha)) \land (D_N(\alpha) \setminus B) \in C_E(D_N(\alpha))$

   **Proof:** By assumption (1)1, the cases (3)1, (3)2, (3)3 and (3)4 are exhaustive.

5.2. Q.E.D.

5.3. Q.E.D.

5.4. Q.E.D.

5.5. Q.E.D.

Corollary B.31. Let $I_N$ be a probabilistic component execution as defined in Definition 5.3 and let $\alpha$ be a complete queue history $\alpha \in B_N$. Let $F_1(C_E(D_N(\alpha)))$ be an extension of $C_E(D_N(\alpha))$ as defined in Definition A.11. Let $\bigcap_{i=1}^{n} A_i$ be a non-empty intersection of finitely many elements such that $\forall i \in [1..n] : A_i \in F_1(C)$. Then there is a finite sequence $\psi$ of disjoint elements in $F_1(C)$ such that

$$\# \psi \leq n \land \bigcup_{i=1}^{\# \psi} \psi[i] = D_N(\alpha) \setminus \bigcap_{i=1}^{n} A_i$$
Lemma B.32. Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$, let $\alpha$ be a queue history in $B_{N_1 \cup N_2}$. Let $\mu_{N_1} \otimes \mu_{N_2}''(\alpha)$ be a measure on $F_1(C_E(D_N(\alpha)))$ as defined in (33) and let $F_2(C_E(D_N \otimes D_N(\alpha)))$ be an extension of $F_1(C_E(D_N \otimes D_N(\alpha)))$ as defined in Definition A.11. The function $\mu_{N_1} \otimes \mu_{N_2}''(\alpha)$ defined by

\[
(34) \quad \mu_{N_1} \otimes \mu_{N_2}''(\alpha)(A) \overset{\text{def}}{=} \]
\[\begin{align*}
\mu_{N_1} \otimes \mu_{N_2}'(\alpha)(A) & \text{ if } A \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \\
\mu_{N_1} \otimes \mu_{N_2}''(\alpha)(A) - \sum_{j=1}^{n} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi[j]) & \text{ if } A \in F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \setminus F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \\
\text{where } B_1, \ldots, B_n \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) & \wedge \psi \in (F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))))^* \\
\text{so that } A & = \cap_{i=1}^{n} B_i \wedge \forall m, j \in [1..\#\psi]: j \neq m \Rightarrow \psi[j] \cap \psi[m] = \emptyset \wedge \\
\cup_{j=1}^{m} \psi[j] & = D_{N_1} \otimes D_{N_2}(\alpha) \setminus \cap_{i=1}^{n} B_i^9
\end{align*}\]

is a measure on \(F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))\).

**Proof:**

1. \(\mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\emptyset) = 0\)

   **Proof:** By Definition B.1.

2. \(\mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\emptyset) = \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\emptyset)\)

   **Proof:** By (2)1 and definition (34).

3. \(\mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\emptyset) = 0\)

   **Proof:** By Lemma B.29.

4. Q.E.D.

**Proof:** By (2)2, (2)3 and the rule of transitivity.

1.2. \(\forall \phi \in \mathbb{P}(H)^\omega : (\forall i \in [1..\#\phi]: \phi[i] \in F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \wedge (\forall m, j \in [1..\#\phi]: j \neq m \Rightarrow \phi[j] \cap \phi[m] = \emptyset) \wedge \cup_{i=1}^{\#\phi} \phi[i] \in F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))) \Rightarrow \mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\cup_{i=1}^{\#\phi} \phi[i]) = \sum_{j=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\phi[j])\)

**Proof:** By assumption (3)1, Proposition B.1 \((F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \subseteq \mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha))\) and elementary set theory.

1.2. \(\exists \phi' \in \mathbb{P}(H)^\omega : (\forall i \in [1..\#\phi']: \phi'[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \wedge (\forall m, j \in [1..\#\phi']: j \neq m \Rightarrow \phi'[j] \cap \phi'[m] = \emptyset) \wedge \cup_{i=1}^{\#\phi'} \phi'[i] = \cup_{i=1}^{\#\phi} \phi[i]\)

**Proof:** By (4)1, Lemma B.14 and Corollary B.7.

1.3. Let: \(\phi' \in \mathbb{P}(H)^\omega\) such that \(\forall i \in [1..\#\phi']: \phi'[i] \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \wedge (\forall m, j \in [1..\#\phi']: j \neq m \Rightarrow \phi'[j] \cap \phi'[m] = \emptyset)\)

---

Note that by Definition A.11, if \(A \in F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \setminus F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))\), then \(A\) corresponds to the intersection of finitely many elements in \(F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))\). Note also that there may exist several sequences \(\psi\) of disjoint elements in \(F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))\) such that \(\bigcup_{j=1}^{m} \psi[j] = D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcap_{i=1}^{n} B_i\). However, since \(\mu_{N_1} \otimes \mu_{N_2}'(\alpha)\) is a measure, by Lemma B.29, the sum of the measures of their elements will all be the same.

84
\[
\land \bigcup_{i=1}^{\#\phi'} \phi'[i] = \bigcup_{i=1}^{\#\phi} \phi[i]
\]

**Proof:** By \((4)2\)

\((4.4)\) **CASE:** \(\bigcup_{i=1}^{\#\phi'} \phi'[i] \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))\)

\((5.1)\) \(\mu_{N_1} \otimes \mu_{N_2}''(\alpha)\bigcup_{i=1}^{\#\phi} \phi[i] = \mu_{N_1} \otimes \mu_{N_2}'(\alpha)\bigcup_{i=1}^{\#\phi} \phi[i]\)

**Proof:** By assumption \((4.4), (4)3\), the rule of replacement \([51]\) and definition \((34)\).

\((5.2)\) \(\mu_{N_1} \otimes \mu_{N_2}'(\alpha)\bigcup_{i=1}^{\#\phi} \phi[i] = \mu_{N_1} \otimes \mu_{N_2}'(\alpha)\bigcup_{i=1}^{\#\phi} \phi'[i]\)

**Proof:** By \((4)3\) and the rule of equality of functions \([51]\).

\((5.3)\) \(\mu_{N_1} \otimes \mu_{N_2}'(\alpha)\bigcup_{i=1}^{\#\phi} \phi[i] = \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\phi[i])\)

**Proof:** By \((4)3\), assumption \((4)4\) and Lemma B.29.

\((6.2)\) \(\sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\phi'[i]) = \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\phi[i])\)

\((7.1)\) \(\forall i \in [1..\#\phi]: \mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\phi'[i]) = \mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\phi[i])\)

**Proof:** By \((4)3\) and Definition A.11.

\((8.2)\) Q.E.D.

**Proof:** By \((8)1\) and definition \((34)\).

\((7.2)\) Q.E.D.

**Proof:** By \((7)1\) and the rule of equality between functions.

\((6.3)\) \(\sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\phi'[i]) = \sum_{i=1}^{\#\phi} \mu_{N_1} \otimes \mu_{N_2}''(\alpha)(\phi[i])\)

**Proof:** By definition \((34)\), \((4)3\), since \(\phi\) and \(\phi'\) are two different partitions of the same set.

\((6.4)\) Q.E.D.

**Proof:** By \((6)1, (6)2, (6)3\) and the rule of transitivity \([51]\).

\((5.4)\) Q.E.D.

**Proof:** By \((5)1, (5)2, (5)3\) and the rule of transitivity \([51]\).

\((4.5)\) **CASE:** \(\bigcup_{i=1}^{\#\phi'} \phi'[i] \in F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \setminus F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))\)

\((5.5)\) Let \(A_1, \ldots, A_n \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))\) such that \(\bigcup_{i=1}^{\#\phi'} \phi'[i] = \bigcap_{i=1}^{n} A_i\)

**Proof:** By \((5)1\).

\((5.3)\) \(\bigcap_{i=1}^{n} A_i \neq \emptyset\)

**Proof:** By assumption \((4)5\) and Definition A.11.

\((5.4)\) \(\exists \psi \in \mathbb{P}(H) \land \forall i \in [1..\#\psi]: \psi[i] \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \land (\forall m, j \in [1..\#\psi]: j \neq m \Rightarrow [\psi[j] \cap \psi[m]] = \emptyset) \land \#\psi \leq n\)

\(\land \bigcup_{i=1}^{\#\psi} \psi[i] = D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcap_{i=1}^{n} A_i\)

**Proof:** By \((5)2, (5)3\) and Corollary B.31.

\((5.5)\) Let \(\psi \in \mathbb{P}(H)\) such that \(\forall i \in [1..\#\psi]: \psi[i] \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \land (\forall m, j \in [1..\#\psi]: j \neq m \Rightarrow [\psi[j] \cap \psi[m]] = \emptyset) \land \#\psi \leq n\)

\(\land \bigcup_{i=1}^{\#\psi} \psi[i] = D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcap_{i=1}^{n} A_i\)

**Proof:** By \((5)4\).

\((5.6)\) \(\mu_{N_1} \otimes \mu_{N_2}''(\alpha)\bigcup_{i=1}^{\#\psi} \psi[i] = \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) - \sum_{j=1}^{\#\psi} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi[j])\)

**Proof:** By assumption \((4)5, (5)5, (4)3\), the rule of replacement \([51]\) and definition \((33)\).

\((5.7)\) \(\mu_{N_1} \otimes \mu_{N_2}'(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) - \sum_{j=1}^{\#\psi} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi[j]) = \)

85
\[
\sum_{i=1}^{\#\psi'} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i])
\]

(6.1) LET: \( \psi' = \psi \bowtie \phi' \)

(6.2) \( \sum_{i=1}^{\#\psi'} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\psi'[i]) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \)

(7.1) \( \bigcup_{i=1}^{\#\psi'} \psi'[i] = D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcup_{i=1}^{\#\phi'} \phi'[i] \)

(8.1) \( \bigcup_{i=1}^{\#\psi'} \psi'[i] = \bigcup_{i=1}^{\#\psi'} \psi[i] \)

Proof: By (6.1).

(8.2) \( \bigcup_{i=1}^{\#\psi'} \psi'[i] = D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcup_{i=1}^{\#\phi'} \phi'[i] \)

Proof: By (5.2), (5.5) and the rule of replacement [51].

(8.3) Q.E.D.

Proof: By (8.1), (8.2) and the rule of replacement [51].

(7.2) \( \bigcup_{j=\#\psi+1}^{\#\psi'} \psi'[j] = \bigcup_{i=1}^{\#\phi'} \phi'[i] \)

Proof: By (6.1).

(7.3) \( \bigcup_{i=1}^{\#\psi'} \psi'[i] = D_{N_1} \otimes D_{N_2}(\alpha) \)

(8.1) \( \bigcup_{i=1}^{\#\psi'} \psi'[i] = (D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcup_{i=1}^{\#\phi'} \phi'[i]) \cup \bigcup_{i=1}^{\#\phi'} \phi'[i] \)

(9.1) \( \bigcup_{i=1}^{\#\psi'} \psi'[i] = (\bigcup_{i=1}^{\#\psi'} \psi'[i]) \cup (\bigcup_{j=\#\psi+1}^{\#\psi'} \psi'[j]) \)

Proof: By (5.5) and (6.1).

(9.2) Q.E.D.

Proof: By (9.1), (7.1), (7.2) and the rule of replacement [51].

(8.2) \( (D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcup_{i=1}^{\#\phi'} \phi'[i]) \cup \bigcup_{i=1}^{\#\phi'} \phi'[i] = D_{N_1} \otimes D_{N_2}(\alpha) \)

Proof: By elementary set theory.

(8.3) Q.E.D.

Proof: By (8.1), (8.2) and the rule of transitivity.

(7.4) \( \forall i \in [1..\#\psi']: \psi'[i] \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \)

(8.1) \( \bigcap_{i=1}^{\#\psi'} \psi'[i] = \bigcap_{i=1}^{\#\psi'} \psi[i] \)

(9.1) \( \bigcap_{i=1}^{\#\psi'} \psi'[i] = F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \)

Proof: By (7.3), Definition 3.3 and the fact that \( c(\psi, D_{N_1} \otimes D_{N_2}(\alpha)) = D_{N_1} \otimes D_{N_2}(\alpha) \), by Definition 3.1.

(9.2) Q.E.D.

Proof: By (9.1), Proposition B.1 \( (C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \subseteq F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \) and elementary set theory.

(8.2) \( \forall i \in [1..\#\psi']: \psi'[i] \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \)

(9.1) \( \forall i \in [1..\#\psi']: \psi'[i] \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \)

Proof: By (4.3) and Definition A.11.

(9.2) Q.E.D.

Proof: By (5.5), (9.1) and (6.1).

(8.3) \( \forall m, j \in [1..\#\psi']: j \neq m \Rightarrow \psi'[j] \cap \psi'[m] = \emptyset \)

(9.1) \( \forall m, j \in [1..\#\psi']: j \neq m \Rightarrow \psi'[j] \cap \psi'[m] = \emptyset \)

Proof: By (5.5) and (6.1).

(9.2) \( \forall m, j \in [1..\#\psi']:\psi'[j] \cap \psi'[m] = \emptyset \)

Proof: By (4.3) and (6.1).

(9.3) \( (\bigcup_{i=1}^{\#\psi'} \psi'[i]) \cap (\bigcup_{j=\#\psi+1}^{\#\psi'} \psi'[j]) = \emptyset \)

(10.1) \( (D_{N_1} \otimes D_{N_2}(\alpha) \setminus \bigcup_{i=1}^{\#\phi'} \phi'[i]) \cap \bigcup_{i=1}^{\#\phi'} \phi'[i] = \emptyset \)

Proof: By elementary set theory.

Proof: By (10.1), (7.1), (7.2) and the rule of replacement [51].
(9)4. Q.E.D.

Proof: By (9)1, (9)2, (9)3 and elementary set theory.

(8)4. Q.E.D.

Proof: By (8)1, (8)2, (8)3 and ∧-introduction.

(7)5. Q.E.D.

Proof: By (7)3, (7)4 and Lemma B.29.

(6)3. \( \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) - \sum_{i=1}^{\# \psi} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) = \sum_{i=1}^{\# \psi} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) - \sum_{i=1}^{\# \psi'} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) \)

Proof: By (6)1 and (6)2.

(6)4. \( \sum_{i=1}^{\# \psi'} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) = \sum_{i=1}^{\# \psi} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) \)

(7)1. \( \sum_{i=1}^{\# \psi} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) = \sum_{i=1}^{\# \psi'} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) \)

(8)1. \( \sum_{i=1}^{\# \psi} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) \leq 1 \)

(9)1. \( \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \leq 1 \)

(10)1. \( D_{N_1} \otimes D_{N_2}(\alpha) \in C_E(D_{N_1} \otimes D_{N_2}(\alpha)) \)

Proof: By the fact that \( c(\cdot), D_{N_1} \otimes D_{N_2}(\alpha) = D_{N_1} \otimes D_{N_2}(\alpha) \), by Definition 3.1.

(10)2. \( \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \)

Proof: By (10)1 and Lemma B.29.

(10)3. \( \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \leq 1 \)

Proof: By definition (25) and Lemma 5.6.3.

(10)4. Q.E.D.

Proof: By (10)2, (10)3 and the rule of replacement [51].

(9)2. Q.E.D.

Proof: By (6)2, (9)1 and the rule of replacement [51].

(8)2. \( \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \)

Proof: By (8)1, (6)2 and (8)1, since the sum of the terms of a converging series is preserved when regrouping the terms in the same order [50].

(8)3. \( \sum_{i=1}^{\# \psi} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) \)

Proof: By (8)2, (6)2 and the rule of transitivity [51].

(8)4. Q.E.D.

Proof: By (8)3 and elementary arithmetic. The possibility to apply the rules of elementary arithmetic follows from the fact that \( \sum_{i=1}^{\# \psi} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) \) converges to a finite number, by (8)1 and that \( \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\psi'[1]) \) and \( \sum_{i=2}^{\# \psi} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\psi'[i]) \) also converges to finite numbers by (8)3.

(7)2. \( \sum_{i=1}^{\# \psi'} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) = \sum_{i=1}^{\# \psi} \mu_{N_1} \otimes \mu_{N_2}'(\alpha)(\psi'[i]) \)

Proof: By (6)1.

(7)3. Q.E.D.

Proof: By (7)1, (7)2 and the rule of transitivity [51].

(6)5. Q.E.D.

Proof: By (6)3, (6)4 and the rule of transitivity [51].
\( \sum_{i=1}^{\#\prime} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i]) = \sum_{i=1}^{\#\prime} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i]) \)

\( \sum_{i=1}^{\#\prime} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i]) = \sum_{i=1}^{\#\prime} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i]) \)

\( \forall i \in [1..\#\prime] : \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i]) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i]) \)

\( \forall i \in [1..\#\prime] : \phi'[i] \in F_1(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \)

**Proof:** By \( \langle 4 \rangle \)3 and Definition A.11.

\( \langle 8 \rangle 2. \text{Q.E.D.} \)

**Proof:** By \( \langle 8 \rangle 1 \) and definition (34).

\( \langle 7 \rangle 2. \text{Q.E.D.} \)

**Proof:** By \( \langle 7 \rangle 1 \) and the rule of equality between functions.

\( \langle 6 \rangle 2. \sum_{i=1}^{\#\prime} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi'[i]) = \sum_{i=1}^{\#\prime} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\phi[i]) \)

**Proof:** By definition (34) and \( \langle 4 \rangle 3 \), since \( \phi' \) and \( \phi \) are two different partitions of the same set.

\( \langle 6 \rangle 3. \text{Q.E.D.} \)

**Proof:** By \( \langle 6 \rangle 1 \), \( \langle 6 \rangle 2 \) and the rule of transitivity [51].

\( \langle 5 \rangle 9. \text{Q.E.D.} \)

**Proof:** By \( \langle 5 \rangle 6 \), \( \langle 5 \rangle 7 \), \( \langle 5 \rangle 8 \) and the rule of transitivity.

\( \langle 4 \rangle 6. \text{Q.E.D.} \)

**Proof:** By assumption \( \langle 3 \rangle 1 \), the cases \( \langle 4 \rangle 4 \) and \( \langle 4 \rangle 5 \) are exhaustive.

\( \langle 3 \rangle 2. \text{Q.E.D.} \)

**Proof:** \( \Rightarrow \)-introduction.

\( \langle 2 \rangle 1. \text{Q.E.D.} \)

**Proof:** \( \forall \)-introduction.

\( \langle 1 \rangle 3. \text{Q.E.D.} \)

**Proof:** By \( \langle 1 \rangle 1 \), \( \langle 1 \rangle 2 \) and Definition A.6..

**Lemma B.33.** Let \( I_{N_1} \) and \( I_{N_2} \) be two probabilistic component executions such that \( N_1 \cap N_2 = \emptyset \), let \( \alpha \) be a queue history in \( B_{N_1 \cup N_2} \). Let \( \mu_{N_1} \otimes \mu_{N_2}(\alpha) \) be a measure on \( F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \) as defined in Lemma B.32 and let \( F_3(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \) be an extension of \( F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \) as defined in Definition A.11. The function \( \mu_{N_1} \otimes \mu_{N_2}(\alpha) \) defined by

\[
\mu_{N_1} \otimes \mu_{N_2}(\alpha)(A) \overset{\text{def}}{=} \begin{cases} 
\mu_{N_1} \otimes \mu_{N_2}(\alpha)(A) & \text{if } A \in F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \\
\sum_{i=1}^{n} \mu_{N_1} \otimes \mu_{N_2}(\alpha)(B_i) & \text{if } A \in F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \setminus F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \\
\text{where } B_1, \ldots, B_n \in F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \end{cases}
\]

so that \( A = \bigcup_{i=1}^{n} B_i^{10} \)

is a measure on \( F_3(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \).

**Proof:**

\( \langle 1 \rangle 1. \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\emptyset) = 0 \)

\( \langle 2 \rangle 1. \emptyset \in F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \)

**Proof:** By Definition A.11 and Proposition B.1.

\( \langle 2 \rangle 2. \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\emptyset) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(\emptyset) \)

\(^{10}\text{Note that by Definition A.11, if } A \in F_3(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) \setminus F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))), \text{ then } A \text{ corresponds to the union of finitely many elements in } F_2(C_E(D_{N_1} \otimes D_{N_2}(\alpha))). \)
Theorem 5.7 Let $I_{N_1}$ and $I_{N_2}$ be two probabilistic component executions such that $N_1 \cap N_2 = \emptyset$, let $\alpha$ be a queue history in $B_{N_1 \cup N_2}$, and let $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ be a measure on $C_E(D_{N_1} \otimes D_{N_2}(\alpha))$ as defined in (25). Then, there exists a unique extension $f_{N_1} \otimes f_{N_2}(\alpha)$ of $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ to the cone-$\sigma$-field $\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)$.

Proof:

1. There exists a unique extension of $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ to the cone-$\sigma$-field $\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)$

2.1. There exists a unique extension $\mu_{N_1} \otimes \mu_{N_2}''(\alpha)$ of $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ to $F_{\alpha}(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))$

3.1. There exists a unique extension $\mu_{N_1} \otimes \mu_{N_2}''(\alpha)$ of $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ to $F_{\alpha}(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))$

Proof: By Lemma B.29, Lemma B.32 and Lemma B.33.

3.2. $F_{\alpha}(C_E(D_{N_1} \otimes D_{N_2}(\alpha))) = F_{\alpha}(C_E(D_{N_1} \otimes D_{N_2}(\alpha)))$

Proof: By Proposition B.1.

3.3. Q.E.D.

Proof: By (3)1, (3)2 and the rule of replacement [51].

2. $\mu_{N_1} \otimes \mu_{N_2}''(\alpha)$ is finite.

3.1. $\mu_{N_1} \otimes \mu_{N_2}''(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \leq 1$

4.1. $D_{N_1} \otimes D_{N_2}(\alpha) \in C_E(D_{N_1} \otimes D_{N_2}(\alpha))$

5.1. $D_{N_1} \otimes D_{N_2}(\alpha) = c(\emptyset, D_{N_1} \otimes D_{N_2}(\alpha))$

Proof: By Definition 3.1.

5.2. Q.E.D.

Proof: By (5)1 and Definition 3.3.

4.2. $\mu_{N_1} \otimes \mu_{N_2}''(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha))$

Proof: By (4)1, Lemma B.29, Lemma B.32 and Lemma B.33.

4.3. $\mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \leq 1$

Proof: By definition (25) and Lemma 5.6.3.

4.4. Q.E.D.

Proof: By (4)2, (4)3 and the rule of replacement [51].

3.2. Q.E.D.

Proof: By (3)1 and Definition A.6.

2.3. $\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha) = \sigma(F_{\alpha}(C_E(D_{N_1} \otimes D_{N_2}(\alpha))))$

Proof: By definition (23) and Lemma B.2.

2.4. Q.E.D.

Proof: By (2)1, (2)2, (2)3 and Theorem B.3, define $f_{N_1} \otimes f_{N_2}(\alpha)$ to be the unique extension of $\mu_{N_1} \otimes \mu_{N_2}''(\alpha)$.

1.2. Q.E.D.

Lemma B.34. Let $D$ be a non-empty set and $\mathcal{F}$ be a $\sigma$-field over $D$ and let $f$ be a measure on $f$. If $f(D) \leq 1$, then $f$ is a conditional probability measure on $\mathcal{F}$.
Proof:

(1.1) Assume: $f(D) \leq 1$
Proof: $\forall A \in \mathcal{F}: f(A) = 0 \lor \exists c \in \{0, 1\}$ such that the function $f'$ defined by $f'(A) = f(A)/c$ is a probability measure on $\mathcal{F}$.

(2.1) Case: $f(D) = 0$
(3.1) $\forall A \in \mathcal{F}: f(A) = 0$
(4.1) $\forall A \in \mathcal{F}: A \subseteq D$
Proof: By the fact that $\mathcal{F}$ is a $\sigma$-field over $D$, Definition A.4 and Definition A.3.

(4.2) Q.E.D.
Proof: By assumption (2)1, (4)1 and Lemma B.8.

(3.2) Q.E.D.
Proof: By (3)1 and $\lor$ introduction.

(2.2) Case: $f(D) > 0$
(3.1) $\exists c \in \{0, 1\}$ such that the function $f'$ defined by $f'(A) = f(A)/c$ is a probability measure on $\mathcal{F}$
(4.1) $\exists n \in \{0, 1\}: f(D) = n$
Proof: By assumption (1)1, assumption (2)2 and $\exists$ introduction.

(4.2) Let: $c \in \{0, 1\}$ such that $f(D) = c$
Proof: By (4)1.

(4.3) Let: $f'(A) = f(A)/c$

(4.4) $f'$ is a probability measure on $\mathcal{F}$.

(5.1) $f'(\emptyset) = 0$
(6.1) $f(\emptyset) = 0$
Proof: By the fact that $f$ is a measure, and Definition A.6.

(6.2) Q.E.D.
Proof: By (4)3, (4)2, (6)1 and elementary arithmetic.

(5.2) $\forall \phi \in \mathbb{P}(\mathcal{H})^\omega: (\forall i \in [1..\#\phi]: \phi[i] \in \mathcal{F} \\
\land (\forall m, j \in [1..\#\phi]: j \neq m \Rightarrow \phi[j] \cap \phi[m] = \emptyset) \\
\land \bigcup_{l=1}^{\#\phi} \phi[i] \in \mathcal{F})$
Proof: $f'(\bigcup_{j=1}^{\#\phi} \phi[j]) = \sum_{j=1}^{\#\phi} f'(\phi[j])$

(6.1) Assume: $\phi \in \mathbb{P}(\mathcal{H})^\omega$
Proof: $(\forall i \in [1..\#\phi]: \phi[i] \in \mathcal{F} \\
\land (\forall m, j \in [1..\#\phi]: j \neq m \Rightarrow \phi[j] \cap \phi[m] = \emptyset) \\
\land \bigcup_{l=1}^{\#\phi} \phi[i] \in \mathcal{F})$
Proof: $f'(\bigcup_{j=1}^{\#\phi} \phi[j]) = \sum_{j=1}^{\#\phi} f'(\phi[j])$

(7.1) Assume: $\forall i \in [1..\#\phi]: \phi[i] \in \mathcal{F} \\
\land (\forall m, j \in [1..\#\phi]: j \neq m \Rightarrow \phi[j] \cap \phi[m] = \emptyset) \\
\land \bigcup_{l=1}^{\#\phi} \phi[i] \in \mathcal{F})$
Proof: $f'(\bigcup_{j=1}^{\#\phi} \phi[j]) = \sum_{j=1}^{\#\phi} f'(\phi[j])$
Proof: By assumption (7)1, the fact that $f$ is a measure and Definition A.6.

(8.1) $f(\bigcup_{j=1}^{\#\phi} \phi[j]) = \sum_{j=1}^{\#\phi} f(\phi[i])$
Proof: By assumption (8)1, (4)2 and elementary arithmetic.

(8.2) $f(\bigcup_{j=1}^{\#\phi} \phi[j])/c = (\sum_{j=1}^{\#\phi} f(\phi[i]))/c$
Proof: By (8)1, (4)2 and elementary arithmetic.

(9.1) $\sum_{j=1}^{\#\phi} f(\phi[i]) \leq 1$
\(\langle 10 \rangle 1. \; f(\bigcup_{j=1}^{\#\phi} \phi[j]) \leq 1\)

\(\langle 11 \rangle 1. \; \bigcup_{j=1}^{\#\phi} \phi[j] \subseteq D\)

Proof: By assumption \(\langle 7 \rangle 1\), the fact that \(\mathcal{F}\) is a \(\sigma\)-field over \(D\) Definition A.4 and Definition A.3.

\(\langle 11 \rangle 2. \; \text{Q.E.D.}\)

Proof: By assumption \(\langle 1 \rangle 1, \langle 11 \rangle 1\) and Lemma B.8.

\(\langle 10 \rangle 2. \; \text{Q.E.D.}\)

Proof: By \(\langle 8 \rangle 1, \langle 10 \rangle 1\) and the rule of replacement [51].

\(\langle 9 \rangle 2. \; \text{Q.E.D.}\)

Proof: By \(\langle 9 \rangle 1\) and elementary arithmetic.

\(\langle 8 \rangle 4. \; \sum_{j=1}^{\#\phi} (f(\phi[i])/c) = \sum_{j=1}^{\#\phi} f'(\phi[j])\)

\(\langle 9 \rangle 1. \; \forall i \in [1..\#\phi]: f'(\phi[i]) = f(\phi[i])/c\)

Proof: By \(\langle 4 \rangle 3\).

\(\langle 9 \rangle 2. \; \text{Q.E.D.}\)

Proof: By \(\langle 9 \rangle 1\) and the rule of equality between functions [51].

\(\langle 8 \rangle 5. \; \text{Q.E.D.}\)

Proof: By \(\langle 4 \rangle 3, \langle 8 \rangle 2, \langle 8 \rangle 3, \langle 8 \rangle 4\) and the rule of transitivity [51].

\(\langle 7 \rangle 2. \; \text{Q.E.D.}\)

Proof: \(\Rightarrow\) introduction.

\(\langle 6 \rangle 2. \; \text{Q.E.D.}\)

Proof: \(\forall\) introduction.

\(\langle 5 \rangle 3. \; f'(D) = 1\)

Proof: By \(\langle 4 \rangle 2, \langle 4 \rangle 3\) and elementary arithmetic.

\(\langle 5 \rangle 4. \; \text{Q.E.D.}\)

Proof: By \(\langle 5 \rangle 1, \langle 5 \rangle 2, \langle 5 \rangle 3\) and Definition A.7.

\(\langle 4 \rangle 5. \; \text{Q.E.D.}\)

Proof: By \(\langle 4 \rangle 2, \langle 4 \rangle 3, \langle 4 \rangle 4\) and \(\exists\) introduction.

\(\langle 3 \rangle 2. \; \text{Q.E.D.}\)

Proof: By \(\langle 3 \rangle 1\) and \(\lor\) introduction.

\(\langle 2 \rangle 3. \; \text{Q.E.D.}\)

Proof: By assumption \(\langle 1 \rangle 1\), the cases \(\langle 2 \rangle 2\) and \(\langle 2 \rangle 1\) are exhaustive.

\(\langle 1 \rangle 2. \; \text{Q.E.D.}\)

Proof: \(\Rightarrow\) introduction.

Corollary 5.8 Let \(f_{N_1} \otimes f_{N_2}(\alpha)\) be the unique extension of \(\mu_{N_1} \otimes \mu_{N_2}(\alpha)\) to the cone-\(\sigma\)-field \(\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)\). Then \(f_{N_1} \otimes f_{N_2}(\alpha)\) is a conditional probability measure on \(\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)\).

Proof:

\(\langle 1 \rangle 1. \; \forall A \in \mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha): f_{N_1} \otimes f_{N_2}(\alpha)(A) = 0 \lor \exists c \in \langle 0, 1 \rangle\) such that the function \(f'\) defined by \(f'(A) = f_{N_1} \otimes f_{N_2}(\alpha)(A)/c\) is a probability measure on \(\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)\).

\(\langle 2 \rangle 1. \; f_{N_1} \otimes f_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \leq 1\)

\(\langle 3 \rangle 1. \; D_{N_1} \otimes D_{N_2}(\alpha) \in C_E(D_{N_1} \otimes D_{N_2}(\alpha))\)

\(\langle 4 \rangle 1. \; D_{N_1} \otimes D_{N_2}(\alpha) = c(\langle \rangle, D_{N_1} \otimes D_{N_2}(\alpha))\)

Proof: By Definition 3.1.

\(\langle 4 \rangle 2. \; \text{Q.E.D.}\)

Proof: By \(\langle 4 \rangle 1\) and Definition 3.3.

\(\langle 3 \rangle 2. \; f_{N_1} \otimes f_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) = \mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha))\)
Lemma B.35. The function $f_{N_1} \otimes f_{N_2}(\alpha)$ is the unique extension of $\mu_{N_1} \otimes \mu_{N_2}(\alpha)$ to the cone-$\sigma$-field $\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)$ and Theorem B.3.

(3). $\mu_{N_1} \otimes \mu_{N_2}(\alpha)(D_{N_1} \otimes D_{N_2}(\alpha)) \leq 1$.

Proof: By definition (25) and Lemma 5.6.3.

(3). Proof: By (3), (3) and the rule of transitivity [51].

(2). Q.E.D.

Proof: By (2), the fact that $f_{N_1} \otimes f_{N_2}(\alpha)$ is a measure on $\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)$ by Theorem 5.7 and Lemma B.34.

(1). Proof:

1. $I_{N_1} \otimes I_{N_2}$ is a probabilistic component execution of $N_1 \cup N_2$

2. $\forall \alpha \in \mathcal{B}_{N_1 \cup N_2}: I_{N_1} \otimes I_{N_2} = (D_{N_1} \otimes D_{N_2}(\alpha), \mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha), f_{N_1} \otimes f_{N_2}(\alpha))$

Proof: By definition (26).

2. Proof: By (2)1, (2)2, $\forall$ elimination and definition (22).

3. $\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)$ is the trace set of $I_{N_1} \otimes I_{N_2}(\alpha)$.

Proof: By (2)1, (2)2, $\forall$ elimination and definition (22).

4. $\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)$ is the cone-$\sigma$-field generated by $C_{E}(D_{N_1} \otimes D_{N_2}(\alpha))$

Proof: By (2)1, (2)2, $\forall$ elimination and definition (23).

5. $f_{N_1} \otimes f_{N_2}(\alpha)$ is a conditional probability measure on $\mathcal{F}_{N_1} \otimes \mathcal{F}_{N_2}(\alpha)$.

Proof: By (2)1, (2)2, $\forall$ elimination and Corollary 5.8.

(2). Q.E.D.

Proof: By steps (2)1 to (2)5.

(1). Q.E.D.

B.4. Hiding

In the following we prove that components are closed under hiding of assets and interface names. That is, we show that hiding assets and/or interface names in a component yields a new component.

Lemma B.35. The function $f_{\delta_n:N}$ is defined for all elements in $C_{E}(D_{\delta_n:N}(\alpha)) \setminus C(D_{\delta_n:N}(\alpha))$. That is:

$$\forall t_1 \in (\mathcal{H} \cap \mathcal{E}^*): \{t_1\} \in C_{E}(D_{\delta_n:N}(\alpha)) \setminus C(D_{\delta_n:N}(\alpha)) \Rightarrow \{t \in D_N(\delta_n:\alpha)|\mathcal{E}_{\delta_n:N} \otimes t \in \{t_1\}\} \in \mathcal{F}_{N}(\delta_n:\alpha)$$

Proof:

1. Assume: $t_1 \in (\mathcal{H} \cap \mathcal{E}^*)$

Prove: $\{t_1\} \in C_{E}(D_{\delta_n:N}(\alpha)) \setminus C(D_{\delta_n:N}(\alpha))$ $\Rightarrow$ $\{t \in D_N(\delta_n:\alpha)|\mathcal{E}_{\delta_n:N} \otimes t \in \{t_1\}\} \in \mathcal{F}_{N}(\delta_n:\alpha)$

2. Assume: $\{t \in D_N(\delta_n:\alpha)|\mathcal{E}_{\delta_n:N} \otimes t \in \{t_1\}\} \in \mathcal{F}_{N}(\delta_n:\alpha)$

Prove: $\forall t \in D_N(\delta_n:\alpha)$ $\mathcal{E}_{\delta_n:N} \otimes t \in \{t_1\}$ $\in \mathcal{F}_{N}(\delta_n:\alpha)$

3. Let: $S = \{t \in \mathcal{H} \cap \mathcal{E}^*: \exists t'' \in D_N(\delta_n:\alpha): t' \subseteq t'' \wedge \mathcal{E}_{\delta_n:N} \otimes t' \in \{t_1\}\}$

4. Let: $S'' = \{t \in \mathcal{H} \cap \mathcal{E}^*: \exists t'' \in D_N(\delta_n:\alpha): t' \subseteq t'' \wedge t_1 \subseteq \mathcal{E}_{\delta_n:N} \otimes t' \wedge \#\mathcal{E}_{\delta_n:N} \otimes t' \in \{t_1\}\}$

5. $\bigcup_{t' \in S} c(t', D_N(\delta_n:\alpha)) \setminus \bigcup_{t'' \in S''} c(t'', D_N(\delta_n:\alpha)) \in \mathcal{F}_{N}(\delta_n:\alpha)$
(4.4) \( \cup_{t \in S} c(t', D_N(\delta n : \alpha)) \in \mathcal{F}_N(\delta n : \alpha) \)

**Proof:** By (3)1 and Corollary B.10.

(4.2) \( \cup_{t' \in S'} c(t'', D_N(\delta n : \alpha)) \in \mathcal{F}_N(\delta n : \alpha) \)

**Proof:** By (3)2 and Corollary B.10.

(4.3) Q.E.D.

**Proof:** By (4.1) and (4.2), since \( \mathcal{F}_N(\delta n : \alpha) \) is closed under set-difference.

(3.4) \( \bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \setminus \bigcup_{t'' \in S''} c(t'', D_N(\delta n : \alpha)) = \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in \{ t_1 \} \} \)

(4.1) \( \bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \setminus \bigcup_{t'' \in S''} c(t'', D_N(\delta n : \alpha)) \subset \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in \{ t_1 \} \} \)

(5.1) **Assume:** \( t_2 \in \bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \setminus \bigcup_{t'' \in S''} c(t'', D_N(\delta n : \alpha)) \)

**Prove:** \( t_2 \in \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in \{ t_1 \} \} \)

(6.1) **Assume:** \( t_2 \notin \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in \{ t_1 \} \} \)

**Prove:** \( \bot \)

(7.1) \( t_2 \in D_N(\delta n : \alpha) \)

**Proof:** By assumption (5.1), (3)1 and Definition 3.1.

(7.2) \( t_1 \subseteq \mathcal{E}_{\delta n : N} \otimes t_2 \)

**Proof:** By assumption (1)1, (3)1 and assumption (5.1).

(7.3) \( t_1 \neq \mathcal{E}_{\delta n : N} \otimes t_2 \)

**Proof:** By assumption (6.1) and (7.1).

(7.4) \( \# \mathcal{E}_{\delta n : N} \otimes t_2 > \# t_1 \)

**Proof:** By (7.2) and (7.3).

(7.5) \( \exists t \in S'' : t \subseteq t_2 \)

**Proof:** By (7.4) and (7.2) and (3.2).

(7.6) \( t_2 \in \bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \)

**Proof:** By (7.5) and Definition 3.1.

(7.7) Q.E.D.

**Proof:** By assumption (5.1), (7.6) and \( \bot \)-introduction.

(6.2) Q.E.D.

**Proof:** Proof by contradiction.

(5.2) Q.E.D.

**Proof:** \( \subseteq \)-rule.

(4.2) \( \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in \{ t_1 \} \} \subset \bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \setminus \bigcup_{t'' \in S''} c(t'', D_N(\delta n : \alpha)) \)

(5.1) **Assume:** \( t_2 \in \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in \{ t_1 \} \} \)

**Prove:** \( t_2 \in \bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \setminus \bigcup_{t'' \in S''} c(t'', D_N(\delta n : \alpha)) \)

(6.1) \( t_2 \in \bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \)

(7.1) \( t_2 \in D_N(\delta n : \alpha) \)

**Proof:** By assumption (5.1).

(7.2) \( \mathcal{E}_{\delta n : N} \otimes t_2 = t_1 \)

**Proof:** By assumption (5.1)

(7.3) Q.E.D.

**Proof:** By (7.1), (7.2) and (3.1).

(6.2) \( t_2 \notin \bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \)

(7.1) **Assume:** \( t_2 \in \bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \)

**Prove:** \( \bot \)

(8.1) \( \mathcal{E}_{\delta n : N} \otimes t_2 = t_1 \)

**Proof:** By assumption (5.1)

93
Lemma B.36. The function $f_{\delta n : N}$ is defined for all elements in $C(D_{\delta n : N}(\alpha))$. That is:

$$\forall t_1 \in (\mathcal{H} \cap \mathcal{E}^*) : c(t_1, D_{\delta n : N}(\alpha)) \in C(D_{\delta n : N}(\alpha)) \Rightarrow \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in c(t_1, D_{\delta n : N}(\alpha)) \} \in \mathcal{F}_N(\delta n : \alpha)$$

Proof:

(1.1) Assume: $t_1 \in (\mathcal{H} \cap \mathcal{E}^*)$

Prove: $c(t_1, D_{\delta n : N}(\alpha)) \in C(D_{\delta n : N}(\alpha))$

$$\Rightarrow \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in c(t_1, D_{\delta n : N}(\alpha)) \} \in \mathcal{F}_N(\delta n : \alpha)$$

(2.1) Assume: $c(t_1, D_{\delta n : N}(\alpha)) \in C(D_{\delta n : N}(\alpha))$

Prove: $\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in c(t_1, D_{\delta n : N}(\alpha)) \} \in \mathcal{F}_N(\delta n : \alpha)$

(3.1) Let: $S = \{ t' \in \mathcal{H} \cap \mathcal{E}^* | \exists t'' \in D_N(\delta n : \alpha) : t_1 \subseteq \mathcal{E}_{\delta n : N} \otimes t'' \subseteq t' \subseteq t'' \}$

(3.2) $\bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \in \mathcal{F}_N(\delta n : \alpha)$

Prove: By (3.1) and Corollary B.10.

(3.3) $\bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) = \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in c(t_1, D_{\delta n : N}(\alpha)) \}$

(4.1) $\bigcup_{t' \in S} c(t', D_N(\delta n : \alpha)) \subseteq \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in c(t_1, D_{\delta n : N}(\alpha)) \}$

(5.1) Assume: $t_2 \in \bigcup_{t' \in S} c(t', D_N(\delta n : \alpha))$

Prove: $t_2 \in \{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n : N} \otimes t \in c(t_1, D_{\delta n : N}(\alpha)) \}$

(6.1) $t_2 \in D_N(\delta n : \alpha)$

Prove: By assumption (5.1) and Definition 3.1.

(6.2) $\mathcal{E}_{\delta n : N} \otimes t_2 \in c(t_1, D_{\delta n : N}(\alpha))$
Proof: By (6)1 and Definition 7.1.

Proof: By assumption (5)1 and (3)1.

Proof: By (8)2 and definition (7).

Proof: By (7)2, (7)1 and the rule of replacement [51].

Proof: By (7)2, (7)1 and Definition 3.1.

Proof: By (6)1 and (6)2.

Proof: By (5)1.

Proof: By (8)1 and Definition 3.1.

Proof: By (7)2 and Definition (2).

Proof: By (7)2, (8)1 and the rule of replacement [51].

Proof: By assumption (1)1, (8)1 and the rule of replacement [51].

Proof: By (8)2, (8)1, (8)3 and ∃-introduction.

Proof: By (7)3.

Proof: By (7)1, (7)2 and (7)4.

Proof: By (7)4, (7)5 and ∃ introduction

Proof: By (6)1 and Definition 3.1.

Proof: By (6)1 and Definition 3.1.
The function $f_{\delta n:N}$ is defined for all elements in $C_E(D_{\delta n:N}(\alpha))$. That is:

$$\forall c \in \mathbb{P}(H) : c \in C_E(D_{\delta n:N}(\alpha)) \Rightarrow \\
\{ t \in D_N(\delta n: \alpha)|E_{\delta n:N} \otimes t \in c \} \in \mathcal{F}_N(\delta n: \alpha)$$

**Proof:**

\begin{enumerate}
  \item Let $c$ be a sequence of cones in $C_E(D_{\delta n:N}(\alpha))$ such that:
    \[ c = \bigcup_{i=1}^{\# \phi} \phi[i]. \]

  **Proof:** By (3)1 and Definition A.1.

  \item $\forall i \in [1..\# \phi] : \{ t \in D_N(\delta n: \alpha)|E_{\delta n:N} \otimes t \in \phi[i] \} \in \mathcal{F}_N(\delta n: \alpha)$

  **Proof:** By Lemma B.37.

  \item $\bigcup_{i=1}^{\# \phi} \{ t \in D_N(\delta n: \alpha)|E_{\delta n:N} \otimes t \in \phi[i] \} \in \mathcal{F}_N(\delta n: \alpha)$

  **Proof:** By (3)2 and (4)1, since $\mathcal{F}_N(\delta n: \alpha)$ is closed under countable union.

  \item $\bigcup_{i=1}^{\# \phi} \{ t \in D_N(\delta n: \alpha)|E_{\delta n:N} \otimes t \in \phi[i] \} = \{ t \in D_N(\delta n: \alpha)|E_{\delta n:N} \otimes t \in \bigcup_{i=1}^{\# \phi} \phi[i] \}$

  **Proof:** By definition (7).

  \item Q.E.D.

  **Proof:** By (4)2, (4)3 and the rule of replacement [51].

  \item Q.E.D.

  **Proof:** By (3)2, (3)3 and the rule of replacement [51].

  \item Q.E.D.

  **Proof:** $\Rightarrow$-introduction.

  \item Q.E.D.

  **Proof:** $\forall$-introduction.
Lemma B.39. Let \( N \) be a component and let \( \alpha \) be a queue history in \( B_N \). Then

1. \( D_{\delta n} N(\alpha) \) is a set of well-formed traces
2. \( \mathcal{F}_{\delta n} N(\alpha) \) is the cone-\( \sigma \)-field of \( D_{\delta n} N(\alpha) \)
3. \( f_{\delta n} N(\alpha) \) is a conditional probability measure on \( \mathcal{F}_{\delta n} N(\alpha) \)

PROOF: (Proof of Lemma B.39.1.)

1. \( D_{\delta n} N(\alpha) \) is a set of well-formed traces, that is, sequences of events fulfilling
   well-formedness constraints (8), (9) and (10).
2. \( D_{\delta n} N(\alpha) = \{ \mathcal{E}_{\delta n} N \otimes t | t \in D_N(\delta n : \alpha) \} \)
   PROOF: By Definition 7.1.
3. \( D_N(\delta n : \alpha) \) is a set of well-formed traces.
   PROOF: By definition (26).
4. \( \forall t \in \{ \mathcal{E}_{\delta n} N \otimes t | t \in D_N(\delta n : \alpha) \} : (\forall i, j \in \{ 1..\#t \} : i < j \Rightarrow q.t[i] < q.t[j]) \land
   \#t = \infty \Rightarrow \forall k \in Q : \exists \bar{i} \in \mathbb{N} : q.t[\bar{i}] > k) \)
   PROOF: By (3)1 and definition (7), since the filtering of a trace with regard to a
   set of events does not change the ordering of the remaining events in the trace.
5. \( \Pi_{\{1,2,3\}} (\Pi_{\{2\}} i) \subseteq \Pi_{\{1,2,3\}} (\Pi_{\{2\}} o) \), that is, the sequence of consumed mes-
   sages sent from an internal interface \( l \) to another internal interface \( m \), is a
   prefix of the sequence of transmitted messages from \( l \) to \( m \), when disregarding
   time.
   PROOF: By (3)1 this constraint is fulfilled by all traces in \( D_N(\delta n : \alpha) \). The
   new traces are obtained by filtering away messages consumed by or transmitted
   from \( n \). Hence, \( n \) is treated as an external interface. The remaining internal
   communication is not affected by the filtering of events, so the restriction is
   fulfilled by the new traces.
6. Q.E.D.
   PROOF: By (3)2, (3)4 and (3)5.
7. Q.E.D.
   PROOF: By (2)1, (2)2 and the rule of replacement [51].

PROOF: (Proof of Lemma B.39.2.)

1. \( \mathcal{F}_{\delta n} N(\alpha) = \sigma(C_E(D_{\delta n} N(\alpha))) \) that is, the cone-\( \sigma \)-field of \( D_{\delta n} N(\alpha) \).
   PROOF: By Definition 7.1.
2. Q.E.D.

PROOF: (Proof of Lemma B.39.3.)

1. \( f_{\delta n} N(\alpha) \) is a conditional probability measure on \( \mathcal{F}_{\delta n} N(\alpha) \).
2. \( f_{\delta n} N(\alpha) \) is a measure on \( \mathcal{F}_{\delta n} N(\alpha) \).
(3.1) \( f_{\delta n:N}(\alpha) \) is well defined, that is
\[
\forall c \in \mathbb{P}(\mathbb{P}(\mathbb{H}_{\delta n:N})) : c \in \mathcal{F}_{\delta n:N}(\alpha) \implies \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in c \} \in \mathcal{F}_N(\delta n: \alpha)
\]
PROOF: By Lemma B.38.
(3.2) \( f_{\delta n:N}(\alpha)(\emptyset) = 0 \)
(4.1) \( f_{\delta n:N}(\alpha)(\emptyset) = f_N(\delta n: \alpha)(\emptyset) \)
PROOF: By Definition 7.1.
(4.2) \( f_N(\delta n: \alpha)(\emptyset) = 0 \)
PROOF: By the fact that \( N \) is a component, Definition 6.1 and Definition 5.3.
(4.3) Q.E.D.

PROOF: By (4.1), (4.2) and the rule of transitivity [51].
(3.3) \( \forall \phi \in \mathbb{P}(\mathbb{H})^\omega : (\forall i \in \{ 1, \ldots, \# \phi \} : \phi[i] \in \mathcal{F}_{\delta n:N}(\alpha) \)
\[
\wedge (\forall m, j \in \{ 1, \ldots, \# \phi \} : j \neq m \implies \phi[j] \cap \phi[m] = \emptyset) \\
\wedge \bigcup_{i=1}^{\# \phi} \phi[i] \in \mathcal{F}_{\delta n:N}(\alpha)) \\
\implies f_{\delta n:N}(\alpha)(\bigcup_{i=1}^{\# \phi} \phi[j]) = \sum_{j=1}^{\# \phi} f_{\delta n:N}(\alpha)(\phi[j])
\]
(4.1) ASSUME: \( \phi \in \mathbb{P}(\mathbb{H})^\omega \)
PROOF: \( (\forall i \in \{ 1, \ldots, \# \phi \} : \phi[i] \in \mathcal{F}_{\delta n:N}(\alpha) \)
\[
\wedge (\forall m, j \in \{ 1, \ldots, \# \phi \} : j \neq m \implies \phi[j] \cap \phi[m] = \emptyset) \\
\wedge \bigcup_{i=1}^{\# \phi} \phi[i] \in \mathcal{F}_{\delta n:N}(\alpha)) \\
\implies f_{\delta n:N}(\alpha)(\bigcup_{j=1}^{\# \phi} \phi[j]) = \sum_{j=1}^{\# \phi} f_{\delta n:N}(\alpha)(\phi[j])
\]
(5.1) ASSUME: \( \forall i \in \{ 1, \ldots, \# \phi \} : \phi[i] \in \mathcal{F}_{\delta n:N}(\alpha) \)
\[
\wedge (\forall m, j \in \{ 1, \ldots, \# \phi \} : j \neq m \implies \phi[j] \cap \phi[m] = \emptyset) \\
\wedge \bigcup_{i=1}^{\# \phi} \phi[i] \in \mathcal{F}_{\delta n:N}(\alpha)) \\
\implies f_{\delta n:N}(\alpha)(\bigcup_{j=1}^{\# \phi} \phi[j]) = \sum_{j=1}^{\# \phi} f_{\delta n:N}(\alpha)(\phi[j])
\]
(6.1) \( \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \bigcup_{j=1}^{\# \phi} \phi[j] \} \subseteq \mathcal{F}_N(\delta n: \alpha) \)
PROOF: By assumption (5.1) \( \bigcup_{j=1}^{\# \phi} \phi[j] \in \mathcal{F}_{\delta n:N}(\alpha) \) and Lemma B.38.
(6.2) \( \forall i \in \{ 1, \ldots, \# \phi \} : \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \phi[i] \} \subseteq \mathcal{F}_{\delta n:N}(\alpha) \)
PROOF: By assumption (5.1) \( \forall i \in \{ 1, \ldots, \# \phi \} : \phi[i] \in \mathcal{F}_{\delta n:N}(\alpha) \) and Lemma B.38.
(6.3) \( f_{\delta n:N}(\alpha)(\bigcup_{j=1}^{\# \phi} \phi[j]) = 
\]
\[
f_N(\delta n: \alpha)(\{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \bigcup_{j=1}^{\# \phi} \phi[j] \})
\]
PROOF: By Definition 7.1 and (6.1).
(6.4) \( f_N(\delta n: \alpha)(\{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \bigcup_{j=1}^{\# \phi} \phi[j] \}) = 
\]
\[
\sum_{j=1}^{\# \phi} f_N(\delta n: \alpha)(\{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \phi[j] \})
\]
(7.1) \( \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \bigcup_{j=1}^{\# \phi} \phi[j] \} = 
\]
\[
\bigcup_{j=1}^{\# \phi} \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \phi[j] \}
\]
PROOF: By definition (7.1).
(7.2) \( f_N(\delta n: \alpha)(\bigcup_{j=1}^{\# \phi} \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \phi[j] \}) = 
\]
\[
f_N(\delta n: \alpha)(\bigcup_{j=1}^{\# \phi} \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \phi[j] \})
\]
PROOF: By (7.1) and the rule of equality between functions [51].
(7.3) \( f_N(\delta n: \alpha)(\bigcup_{j=1}^{\# \phi} \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \phi[i] \}) = 
\]
\[
\sum_{j=1}^{\# \phi} f_N(\delta n: \alpha)(\{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \phi[i] \})
\]
(8.1) \( \bigcup_{i=1}^{\# \phi} \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \phi[i] \} \subseteq \mathcal{F}_{\delta n:N}(\alpha) \)
PROOF: By (7.1), (6.1) and the rule of replacement [51].
(8.2) \( \forall j, m \in \{ 1, \ldots, \# \phi \} : j \neq m \implies \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \phi[j] \} \cap \{ t \in D_N(\delta n: \alpha)|\mathcal{E}_{\delta n:N} \otimes t \in \phi[m] \} = \emptyset \)
(9.1) ASSUME: \( \exists j, m \in \{ 1, \ldots, \# \phi \} : j \neq m \)
\[ \left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in \phi[j] \right\} \cap \\
\left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in \phi[m] \right\} \neq \emptyset \]

**Proof:**

1. \( \bot \)

10.1. Let \( j, m \in [1..\#\phi] \) such that
\[ \left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in \phi[j] \right\} \cap \\
\left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in \phi[m] \right\} \neq \emptyset \]

**Proof:** By assumption (9)1.

10.2. \( \exists t_1 \in D_N(\delta n : \alpha) : \)
\[ t_1 \in \left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in \phi[j] \right\} \land \\
\left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in \phi[m] \right\} \]

**Proof:** By (10)2.

10.3. Let \( t_1 \in D_N(\delta n : \alpha) \) such that
\[ t_1 \in \left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in \phi[j] \right\} \land \\
\left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in \phi[m] \right\} \]

**Proof:** By (10)3.

10.4. \( \mathcal{E}_{\delta n} : N \otimes t_1 \in \phi[j] \land \mathcal{E}_{\delta n} : N \otimes t_1 \in \phi[m] \)

**Proof:** By (10)2.

10.5. \( \phi[j] \cap \phi[m] \neq \emptyset \)

**Proof:** By (10)4.

10.6. Q.E.D.

**Proof:** By assumption (5)1, (10)5 and \( \bot \)-introduction.

(9)2. Q.E.D.

**Proof:** Proof by contradiction.

(8)3. Q.E.D.

**Proof:** By (8)1, (6)2 and (8)2, the fact that \( N \) is a component, Definition 6.1 and Definition 5.3.

(7)4. Q.E.D.

**Proof:** By (7)2, (7)3 and the rule of transitivity.

(6)5. \( \sum_{i=1}^{\#\phi} f_N(\delta n : \alpha) \left( \left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in \phi[i] \right\} \right) = \\
\sum_{i=1}^{\#\phi} f_N(\delta n : \alpha) (\phi[i]) \)

(7)1. \( \forall i \in [1..\#\phi] : f_N(\delta n : \alpha) (\phi[i]) = \\
f_N(\delta n : \alpha) \left( \left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in \phi[i] \right\} \right) \)

**Proof:** By Definition 7.1 and (6)2.

(7)2. Q.E.D.

**Proof:** By (7)1 and the rule of equality between functions [51].

(6)6. Q.E.D.

**Proof:** By (6)3, (6)4, (6)5 and the rule of transitivity [51].

(5)2. Q.E.D.

**Proof:** \( \Rightarrow \) rule.

(4)2. Q.E.D.

**Proof:** \( \forall \)-introduction

(3)4. Q.E.D.

**Proof:** By (3)1, (3)2 and (3)3 and Definition A.6.

(2)2. \( f_N(\delta n : N(\alpha))(D_{\delta n} : N(\alpha)) \leq 1 \)

(3)1. \( \left\{ t \in D_N(\delta n : \alpha) | \mathcal{E}_{\delta n} : N \otimes t \in D_{\delta n} : N(\alpha) \right\} \in \mathcal{F}_N(\delta n : \alpha) \)

(4)1. \( D_{\delta n} : N(\alpha) \in \mathcal{F}_{\delta n} : N(\alpha) \)

**Proof:** By Definition 7.1 \( (\mathcal{F}_{\delta n} : N(\alpha) \text{ is the cone-}\sigma\text{-field of } D_{\delta n} : N(\alpha)) \).

(4)2. Q.E.D.
Theorem 7.4

If \( N \) is a component and \( a \) is an asset, then \( \sigma a : N \) is a component.

Proof:

(1) Assume: \((I_N, A_N, cv_N, rf_N)\) is a component and \(a\) is an asset.

prove: \( \sigma a : (I_N, A_N, cv_N, rf_N) \) is a component, that is, a quadruple consisting of its probabilistic component execution, its assets, consequence function and risk function according to Definition 6.1.

(2) \( \sigma a : (I_N, A_N, cv_N, rf_N) = (I_N, \sigma a : A_N, \sigma a : cv_N, \sigma a : rf_N) \)

Proof: By Definition 7.3.

\( \sigma a : A_N, \sigma a : cv_N, \sigma a : rf_N \) is a component.

(3) \( I_N \) is a component execution.

Proof: By assumption (1)1.

(3)2. \( \sigma a : A_N \) is a set of assets.

(4) \( \sigma a : A_N = A_N \setminus \{a\} \)

Proof: By Definition 7.3.
Theorem 7.6
If $A_N$ is a set of assets.

PROOF: By assumption (1)1.

(4)3. Q.E.D.

PROOF: By (4)1 and (4)2.

(3)3. $\sigma : cv_N$ is a consequence function in $E_N \times \sigma : A_N \rightarrow N$

PROOF: By Definition 7.3.

(4)1. $\sigma : cv_N = cv_N \setminus \{(e,a) \rightarrow |e \in E \land c \in N\}$

PROOF: By Definition 7.3.

(4)2. $cv_N$ is a consequence function in $E_N \times A_N \rightarrow N$.

PROOF: By assumption (1)1.

PROOF: By (4)1 and (4)2.

(3)4. $\exists a : rf_N$ is a risk function in $N \times [0,1] \times \sigma : A_N \rightarrow N$

PROOF: By assumption (1)1.

(4)3. Q.E.D.

PROOF: By (4)1 and (4)2.

(3)5. Q.E.D.

PROOF: By (3)1, (3)2, (3)3, and (3)4.

(2)3. Q.E.D.

PROOF: By (2)1, (2)2 and the rule of replacement [51].

(1)2. Q.E.D.

PROOF: $\Rightarrow$-introduction.

Theorem 7.6 If $N$ is a component and $n$ is an interface name, then $\delta n : N$ is a component.

PROOF:
(1)1. ASSUME: $(I_N, A_N, cv_N, rf_N)$ is a component and $n$ is an interface name.

PROOF: $\delta n : (I_N, A_N, cv_N, rf_N)$ is a component.

(2)1. $\delta n : (I_N, A_N, cv_N, rf_N) = (\delta n : I_N, \sigma A_n : A_N, \sigma A_n : cv_N, \sigma A_n : rf_N)$

PROOF: By Lemma 7.2.

(2)2. $(\delta n : I_N, \sigma A_n : A_N, \sigma A_n : cv_N, \sigma A_n : rf_N)$ is a component

(3)1. $\delta n : I_N(\alpha)$ is a probabilistic component execution.

PROOF: By Definition 7.3.

(3)2. $(I_N, \sigma A_n : A_N, \sigma A_n : cv_N, \sigma A_n : rf_N)$ is a component.

(4)1. $\sigma A_n : (I_N, A_N, cv_N, rf_N)$ is a component.

PROOF: By assumption (1)1 and Theorem 7.4.

(4)2. $\sigma A_n : (I_N, A_N, cv_N, rf_N) = (I_N, \sigma A_n : A_N, \sigma A_n : cv_N, \sigma A_n : rf_N)$

PROOF: By Definition 7.3.

(4)3. Q.E.D.

PROOF: By (4)1, (4)2 and the rule of replacement.

(3)3. $\sigma A_n : A_N$ is a set of assets,

$\sigma A_n : cv_N$ is a consequence function in $E_N \times \sigma A_n : A_N \rightarrow N$ and $\sigma A_n : rf_N$ is a risk function in $N \times [0,1] \times \sigma A_n : A_N \rightarrow N$

PROOF: By (3)2 and Definition 6.1.

(3)4. Q.E.D.

PROOF: By (3)1, (3)3 and Definition 6.1.
\( \langle 2 \rangle 3 \). Q.E.D.

Proof: By \( \langle 2 \rangle 1 \), \( \langle 2 \rangle 2 \) and the rule of replacement [51].

\( \langle 1 \rangle 2 \). Q.E.D.

Proof: \( \Rightarrow \)-introduction.