STAIRS — Understanding and Developing Specifications Expressed as UML Interaction Diagrams

Doctoral Dissertation by

Ragnhild Kobro Runde

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

January 2007

Abstract

STAIRS is a method for the step-wise, compositional development of interactions in the setting of UML 2.x. UML 2.x interactions, such as sequence diagrams and interaction overview diagrams, are seen as intuitive ways of describing communication between di erent parts of a system, and between a system and its users.

STAIRS addresses the challenges of harmonizing intuition and formal reasoning by providing a precise understanding of the partial nature of interactions, and of how this kind of incomplete specifications may be consistently refined into more complete specifications.

For understanding individual interaction diagrams, STAIRS defines a denotational trace semantics for the main constructs of UML 2.x interactions. The semantic model takes into account the partiality of interactions, and the formal semantics of STAIRS is faithful to the informal semantics given in the UML 2.x standard. For developing UML 2.x interactions, STAIRS defines a number of refinement relations corresponding to basic system development steps. STAIRS also defines matching compliance relations, for relating interactions to real computer systems.

An important feature of STAIRS is the distinction between underspecification and inherent nondeterminism. Underspecification means that there are several possible behaviours serving the same overall purpose, and that it is su cient for a computer system to perform only one of these. On the other hand, inherent nondeterminism is used to capture alternative behaviours that must all be possible for an implementation. A typical example is the tossing of a coin, where both heads and tails should be possible outcomes. In some cases, using inherent nondeterminism may also be essential for ensuring the necessary security properties of a system.

Acknowledgements

First of all, I thank Ketil Stølen and Øystein Haugen for being my supervisors on this thesis work. You have been excellent supervisors, and I am grateful for your invaluable guidance, encouragement and our countless scientific discussions. Thanks also for inviting me to participate in the SARDAS project, which has provided a friendly and challenging environment for discussing my work. I thank everyone involved in the SARDAS project, including the Ph.D. students, senior researchers, guest scientists and advisory board. In particular, I thank Knut Eilif Husa for our collaboration on the early work on STAIRS, and Atle Refsdal for a number of interesting discussions and for close collaboration during the last year. I thank Manfred Broy for inviting me to Munich, and María Victoria Cengarle and Alexander Knapp for interesting discussions during my visit there. I thank the people at the Department for Cooperative and Trusted Systems at SINTEF for welcoming me at several occasions, and for providing valuable feedback on parts of my work. Thanks are also due to many other researchers who have crossed my path over the years, in person or in writing. I am grateful to the Department of Informatics at the University of Oslo for giving me the opportunity to do this thesis work, and I thank all of my colleagues for making this a wonderful place to be. In particular, I thank Anders Moen Hagalisletto, Einar Broch Johnsen and Olaf Owe for various kinds of support over the years. I thank the sta at the Informatics Library for providing me with the necessary background literature, and Dag Langmyhr for helping me with my LaTeX problems. Finally, my greatest thanks go to Hans Arild for continuous support through all of these years.

Contents

Part I

Overview

Chapter 1

Introduction

This thesis presents work on the STAIRS method, a method for the step-wise, compositional development of interactions in the setting of UML. This chapter gives a short introduction to the thesis work, together with an overview of the thesis.

1.1 UML Interactions and STAIRS

During the past decade, UML has become the de facto modelling standard used in industry. From being an approach unifying the leading modelling languages at the time, UML has gradually developed and is now in version 2.1 [OMG06].

UML 2.1 interactions, such as sequence diagrams and interaction overview diagrams, are seen as intuitive ways of describing communication between different parts (e.g. components or objects) of a system, and between a system and its users. According to the UML 2.1 standard [OMG06], an interaction describes a set of valid and a set of invalid traces, i.e. system behaviours. Interactions are usually incomplete specifications, meaning that there will typically be many traces that are not described by the interaction at all, and it is impossible to know whether these are valid or not.

A problem with UML 2.1 interactions is that their semantics is only explained in natural language, and for a given interaction it is often difficult, or even impossible, to know its precise meaning. Another aspect not addressed by the UML 2.1 standard is the relationships between different interactions for the same system, i.e. what it means for one interaction to be a refinement of another interaction, or for two interactions to describe the system from different viewpoints. Also, it is not defined what it means for a computer system to be in compliance with an interaction, i.e. when is a computer system a valid implementation of a specification in the form of UML 2.1 interactions. In particular, it is not clear whether the valid traces of an interaction describe behaviours that *must* or *may* be present in the final system.

As long as these aspects are not addressed properly, different people tend to interpret the same interaction differently. This leads to confusion and misunderstanding, where the end result might be that the systems being built are not the ones requested by the customers.

Another problem is the lack of tools supporting system development using UML 2.1 interactions. With a proper formal semantics, and with precise definitions of viewpoint correspondence, refinement and compliance, it is possible to make advanced tools for e.g. automatic analysis and

consistency checking. Errors are an inevitable part of any system development process, but with adequate tool support, fewer errors may be introduced during the development process, and the errors that are made may be discovered earlier, possibly resulting in substantially reduced development costs.

The STAIRS method presented in this thesis defines a denotational trace semantics for the main constructs of UML 2.1 interactions. The semantic model takes into account the partial nature of interactions, and the formal semantics of STAIRS corresponds closely to the informal semantics given in the UML 2.1 standard. STAIRS also defines a number of refinement relations for relating interactions made at different stages of the development process, and corresponding compliance relations.

Our vision is that the intuitive feeling of UML 2.1 interactions should be maintained while also providing the means for formal analysis such as security analysis, testing and automatic transformation into executable code.

1.2 Contribution and Overview of the Thesis

This thesis is organised as a collection of eight papers presenting work on STAIRS, together with an introductory part providing the context of this work. The organisation of the rest of this introductory part is as follows: In chapter 2 we present our research method, while chapter 3 gives a thorough problem analysis, presenting the setting of this thesis together with the goals and success criteria for the STAIRS method. Based on this problem analysis, in chapter 4 we discuss the state of the art that are relevant for this thesis work. Chapter 5 gives a summary of STAIRS, while chapter 6 provides a brief overview of the papers included in this thesis. Chapter 7 discusses the results obtained and recent related work on UML 2.1 interactions. Finally, ideas for future work is presented in chapter 8.

STAIRS, and the papers included in this thesis, is the result of a collaborative effort in a group of researchers led by Ketil Stølen and Øystein Haugen. In the following, we list the main contributions of STAIRS together with references to the papers where these are described. In chapter 6, the particular contributions of Ragnhild Kobro Runde are described for each of the eight papers in question.

- STAIRS improves the language of UML 2.1 interactions by providing additional mechanisms for
  - distinguishing between mandatory alternatives (e.g. inherent nondeterminism) and potential alternatives (e.g. underspecification). Described in: Papers 1, 2, 4, 5 and 6.
  - using two different negation operators depending on the desired positive behaviours. Described in: Paper 3.
  - distinguishing between the reception and the consumption of a message, an essential feature when working with time constraints. Described in: Paper 2.
- STAIRS provides a precise understanding of the partial nature of UML interactions by
  - defining a semantic model for UML 2.1 interactions with the extensions listed above. Described in: Paper 1.

  - defining a denotational trace semantics for the most commonly used parts of UML 2.1 interactions, including time, guarded alternatives and the extensions listed above. Described in: Papers 1, 2, 3, 4 and 5.

- STAIRS supports stepwise and compositional development of interactions by defining basic refinement relations that
  - take the partiality of interactions into account, together with all the features mentioned above. Described in: Papers 1, 2, 4, 5 and 7.
  - are sound, meaning that the desirable mathematical properties of reflexivity, transitivity and monotonicity hold. Described in: Papers 2, 3, 4 and 7.

- STAIRS defines what it means for a computer system to be compliant with an interaction by
  - explaining how computer systems may be understood in terms of our semantic model. Described in: Paper 7.
  - defining sound compliance relations corresponding to the di erent refinement relations. Described in: Paper 7.

- STAIRS provides methodological guidelines for how to create and refine interactions using the main principles of STAIRS. Described in: Papers 5 and 8.

The research on STAIRS has been partly carried out within the context of the SARDAS project [SAR], which is funded by the Research Council of Norway under the IKT-2010 programme. The overall goal of SARDAS is to improve on state of the art for the specification, design and development of systems with high availability. The main goal of STAIRS has been to provide a firm foundation for the other activities to build on.

STAIRS has been successfully used both in other parts of the SARDAS project, and in related projects. Mass Soldal Lund has developed an operational semantics for STAIRS [LS06b] and used it for building a tool for test case generation from UML 2.1 interactions [LS06a]. Atle Refsdal and Knut Eilif Husa have developed probabilistic STAIRS [RHS05, RRS06], which extends STAIRS with probabilistic choice and soft real-time constraints. Fredrik Seehusen has used STAIRS for defining secure information flow property preserving refinement and transformation of interaction diagrams [SS06].

Chapter 2

Research Method

This chapter presents the research method on which this thesis work has been based.

Compared to most other sciences, computer science is a relatively new discipline and without an established research method [DC02]. Even the question of whether or not computer science qualifies as a science in the traditional sense is still being debated [Den05]. Computer science has ancestors in disciplines as diverse as mathematics, physics, engineering and social science. Consequently, researchers in computer science have to a varying degree adapted research methods used within each of these disciplines. However, computer science research is also often performed in an ad hoc manner and without a clear thought about research method [Gla95].

Much of computer science research, including this thesis work, fall into the category called technological research in [SS07]. While the aim of classical research in e.g. the natural and social sciences is to achieve more knowledge about some existing part of the world, the aim of technological research is to create new or improved artefacts. In computer science research, such artefacts may be e.g. programs, programming languages, security protocols, hardware processors, or methods as in our case.

Similar to classical research methods, the technological research method advocated in [SS07] is an iterative process consisting of three main steps: problem analysis, innovation and evaluation. These steps are very similar to the phases proposed in [Gla95]. Problem analysis corresponds to the informational phase, i.e. gathering or aggregating information. Innovation covers both the propositional and the analytical phase, i.e. proposing a hypothesis, method, etc, and exploring this proposition. Finally, evaluation corresponds to the evaluative phase, where the proposition is evaluated by e.g. experimentation or observation.

In the following sections, we describe how each of the steps in [SS07] have been instantiated in this thesis work.

2.1 Problem Analysis

First, as documented in chapter 3, we investigated the current situation in model-based system development using UML and identified the need for a new artefact — a method giving a precise semantics for UML 2.x, definitions of model relations and methodological guidelines for using UML 2.x for specifications. As this would be too much to cover within the scope of one thesis, we identified the STAIRS method for developing UML 2.x interactions as the artefact to be created

by this thesis work. Also, we formulated a number of requirements that should be fulfilled by STAIRS.

Next, as documented in chapter 4, we investigated existing theories and methods in order to evaluate to what extent these fulfilled our requirements for the formal framework. The main conclusion from this investigation was that these theories and methods were not su cient, and that new research was needed in order to create the required framework.

2.2 Innovation

The innovation part of this thesis work has been to create the STAIRS method as a response to the identified need for a formal framework for UML 2.x interactions. A summary of STAIRS is given in chapter 5, and a more thorough description is given in the attached papers 1–7.

The development of STAIRS has been performed by treating the most basic parts of interactions first, and then iteratively adding more and more features. Similar to what often happens in iterative system development, the addition of new features to STAIRS has sometimes led to minor modifications of previous work. The choice of what new features to include in each step has been guided by feedback received when presenting our work to other researchers in the field, and by ourselves identifying weaknesses and additional needs when trying to use STAIRS on small toy examples.

Even though we have not reached all of the initial goals described in section 3.3, our claim is that STAIRS satisfies the main requirements and provides a useful basis for further research in this area.

2.3 Evaluation

In this thesis work, the evaluation has been performed alongside the development of the STAIRS method, and is documented in the attached papers. Also, a summarizing discussion may be found in chapter 7.

For evaluation, there exists a number of methods and techniques that may be categorized in di erent ways. In [McG84], a distinction is made between eight di erent evaluation methods, each with its own advantages and disadvantages. A perfect method leads to results that are both general, realistic and precise. However, [McG84] points out that no such perfect method exists, and that trying to increase one of these properties invariably results in reducing one or both of the other properties. The key, then, is to use di erent methods that complement each other. Several factors influence the exact choice of evaluation methods, including the time and resources required to carry out each of the methods. Also, the stated requirements are important as the chosen method must be able to both verify and falsify the claim that the artefact meets these requirements.

The STAIRS method consists mainly of a formal foundation for UML 2.x interactions, and it has therefore been natural to use formal theory and mathematical proofs for establishing desirable properties of this formalization. The main properties required of STAIRS are described in section 3.3, and proved in the attachments of papers 2, 4, 6 and 7.

For evaluating the usefulness of STAIRS, the results from formal theory have been supplemented with the use of STAIRS in several examples. Papers 1–7 all contain running examples

illustrating the main points. As STAIRS is meant to be a general method for UML 2.x interactions, these examples are on purpose taken from quite di erent domains as can be seen from the following overview:

Paper 1 describes an automatic teller machine, focusing on withdrawal of money.

Paper 2 elaborates on the original STAIRS example in [HS03], the making of dinner at an ethnic restaurant.

Paper 3 illustrates its main point using a small example with a vending machine selling tea and co ee.

Paper 4 uses an example of network communication.

Paper 5 is a tutorial paper using an appointment system as its running example.

Paper 6 describes the games of playing tic-tac-toe, flipping a coin and throwing a dice.

Paper 7 describes aspects of a gambling machine.

In addition to these examples, a larger case study has been performed. The case study describes a system for automatically matching service providers with users of those services. Based on the case study, we performed an evaluation of STAIRS. Both the case study and the following evaluation are documented in paper 8. The case study was performed by ourselves, which gave complete control over the setting of the case study. This ensured that the subsequent evaluation really evaluated STAIRS, and not some other uncontrollable factor. A natural next step in the evaluation of STAIRS, would be to perform a field study where STAIRS is used in the development of a real system.

Chapter 3

Problem Analysis

This chapter presents the result of the problem analysis. Sections 3.1 and 3.2 outline the problem to be solved in this thesis, i.e. describing the need the STAIRS method is supposed to meet. Section 3.3 refines this overall outline into a set of success criteria that the STAIRS method should fulfil.

3.1 Problem Specification

During the past decade, UML has become the de facto modelling standard used in industry. For this thesis work, we started out using the U2 Partners' submission [OMG03a] for the UML 2.0 superstructure. In 2005, a revised version of this proposal became an adopted OMG specification [OMG05], and UML is now being updated to version 2.1 [OMG06]. For our purposes, there are no significant di erences between these versions, and we will refer to them collectively as "the UML 2.x standard" or simply "the standard".[1]

Using the classification of Martin Fowler [Fow03], there are three primary ways of thinking about UML. In UmlAsSketch, UML diagrams are used informally for discussing selected aspects of a system to be built, or for explaining parts of an existing system. Using UmlAsBlueprint, the UML diagrams serve as a detailed specification and/or documentation of the system, containing all major design decisions. From the specification, programming the system should be pretty straightforward. Finally, UmlAsProgrammingLanguage is the idea that UML may be used as a high-level programming language, and that tools may automatically transform the UML models into executable code.

We believe that one of the main reasons for the attractiveness of UML, is its intuitive use in informal sketches. Our vision is that this intuitive feeling should be kept while at the same time improving UML for better use in blueprints and as a programming language.

All UML users, including those who use UML only for informal sketches, desire tool support for drawing diagrams. More advanced users will also need tools for analysing UML diagrams and for transforming them into executable code. Such tools, and in particular interoperability between such tools, can only be achieved if the interpretation of a given UML diagram is unam-

[1] Note that there are important di erences between UML 1.x and UML 2.x, in particular with respect to interactions which is the topic of this thesis. The various composition operators introduced with UML 2.x interactions have received particular attention in this thesis work.

biguous, i.e. tool vendors need UML to have a well-defined, precise, semantics. As Bran Selic points out in [Sel04], UML is not completely without a semantics, but the semantics given in the UML 2.x standard is not detailed or precise enough to answer questions about the exact meaning of more complicated diagrams. If the UML diagrams are to be used as blueprints, the lack of precise semantics is problematic not only because of little tool support, but also because it leads to situations where di erent users interpret the same diagram slightly di erently. The result may be that the system being built is not the same system as the one intended by the person(s) making the blueprint.

Partly due to the fact that UML is a language and not a methodology, very little is found in the UML 2.x standard with respect to the semantic relationship between di erent UML diagrams of the same system, and between UML diagrams and computer systems. Understanding such relationships is important for ensuring that the initial requirements are contained also in later versions of the specification, for performing consistency checks across several diagrams, and for enabling reuse of analysis results obtained on diagrams created early in the development process.

3.2 Limiting the Scope of this Thesis

Currently, the UML 2.x standard describes 13 di erent diagram types. Clearly, it is out of scope for this thesis to cover all of UML 2.x and all possible relations between di erent UML models. The purpose of this section is to define the limits for this thesis work. We discuss UML diagrams in section 3.2.1, before turning to model relations and relationships between UML diagrams and computer systems in section 3.2.2.

3.2.1 UML Diagrams

In the context of UML, much work has been done on diagram types for describing system structure, but less has been done on diagram types for describing behaviour (see e.g. [Whi02], which gives an overview of formal approaches to UML). For behaviour, the UML 2.x standard describes altogether seven di erent diagram types, including state machine diagrams and four kinds of interaction diagrams (interaction overview diagrams, sequence diagrams, communication diagrams, and timing diagrams).

While state machines describe the complete behaviour of one part of the system (typically one or more objects), interactions are partial descriptions of communication between di erent parts of the system. We expect that if we first understand how to deal with the partiality of interactions, it will be relatively easy to generalize the obtained results to complete descriptions in the form of e.g. state machines. Consequently, interactions have been chosen as the focus of this thesis. Chapter 7.3 provides a more detailed discussion of to what extent the obtained results generalize to other specification techniques.

Interactions are appealing, as they are seen to be intuitive and easy to understand and thus useful for communicating both with customers and within the development team. In addition to being used for capturing and analysing requirements, interactions are very useful as system documentation as they describe how the di erent components or objects in the system interact to achieve the required behaviour. Also, test scenarios may be specified and documented using interactions.

As an example interaction, the sequence diagram in figure 3.1 (taken from [PP05]) describes how a workstation sends two ping packets to the server. The two responses may take di erent routes in the network, leading to the second response message arriving at the workstation before the first response. The workstation and the server are called lifelines, the arrows represent the messages sent between them and the arrowheads indicate the direction of each message. The open arrowheads indicate that the messages are asynchronous, meaning that the sender of a message is not required to wait for a reply before continuing with its behaviour.

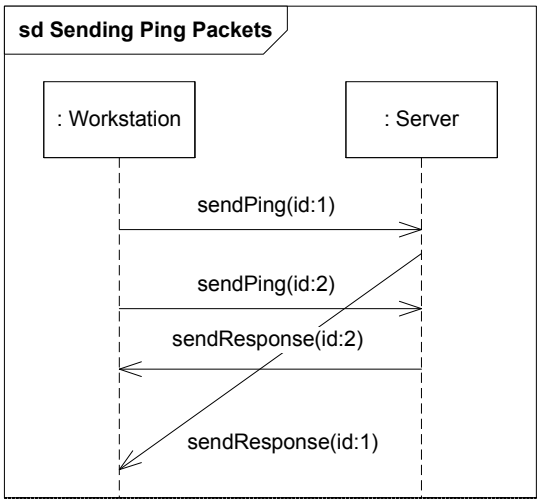

Figure 3.1: Example interaction

A message consists of two events, the send event and the receive event. The implicit composition operator in UML 2.x interactions is weak sequencing, where events on the same lifeline are ordered from the top and downwards. Events on di erent lifelines may happen in any order (with the obvious restriction that a message must be sent before it may be received). For the interaction in figure 3.1, this means for instance that the workstation may send both ping messages before any one of them is received by the server, or that the server may receive the first message (and possibly also send the response) before the workstation sends the second ping message.

The interaction in figure 3.1 is obviously not a complete specification of how ping packets may be sent. Is the workstation allowed to send more ping messages if it takes too much time before it receives any response? May the workstation send only one ping message? What if the server is down, and the workstation never receives any response message at all? May the server respond to some, but not all, of the ping messages? The given interaction does not provide an answer to any of these questions. This does not mean that the interaction is wrong, but rather suggests that the interaction is a partial specification as it only describes a few example scenarios.

Another example interaction is given in figure 3.2. In this sequence diagram, a time constraint is used to describe that after the server has received the request from the client, it should take between zero and five time units before the server sends the response back. Again, this is not a complete specification. The sequence diagram in figure 3.2 does not describe what should happen if, for instance, the server receives a second request or if the server fails to respond within the given time limit. Also, from the UML 2.x standard it is not clear whether the time constraint

refers to the point in time when the request arrives in the input queue of the server, or to the point in time when the sever consumes the message from the bu er and starts the handling of the request.

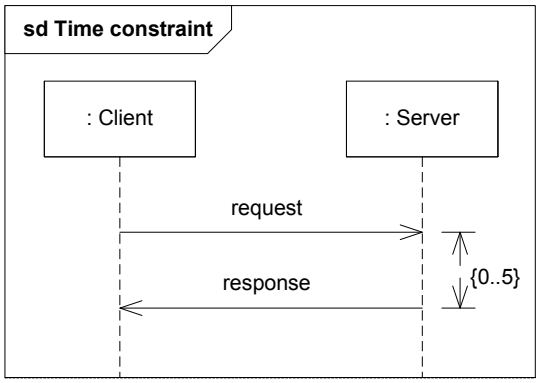

Figure 3.2: Interaction with time constraint

For UML 2.x interactions, the standard also includes a number of operators for specifying e.g. parallel execution and alternative behaviours. Figure 3.3 (taken from paper 5) is an example interaction using the alt-operator when describing how a client may interact with an appointment system in order to book an appointment. This interaction describes altogether four di erent system behaviours. What is not clear from the UML 2.x standard, is whether a system

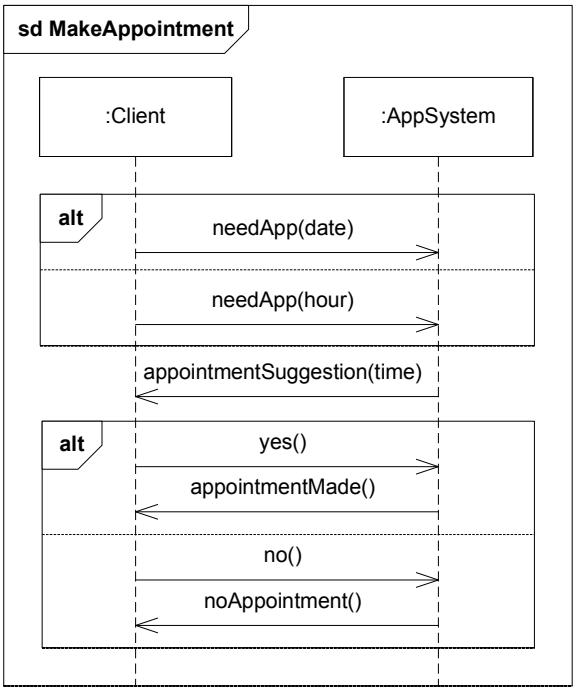

Figure 3.3: Alternative behaviours

compliant with the interaction must be able to perform all of these alternatives or if performing only one (or maybe even none) is su cient. In other words, does the alt operator specify alternative behaviours that are mandatory (i.e. required) or potential (i.e. optional)? Currently, both interpretations are being used by UML practitioners.

UML 2.x also provides operators for specifying negative behaviour in the interactions. As an example, consider the interaction in figure 3.4 (which is a simplified version of an example given in paper 3). Here, the neg operator of UML 2.x is used to specify that if a user orders co ee from a vending machine, he should not receive tea. The exact interpretation of this interaction is not given by the informal semantics in the UML 2.x standard. For instance, is the behaviour where the user orders co ee and then nothing more happens positive? What about the behaviour where the user orders co ee and then receives cappuccino?[2] Does the interaction describe any positive behaviours at all? And what is the exact set of negative behaviours described by the interaction? For instance, if the user orders co ee and then receives both tea and co ee, is that behaviour also negative?

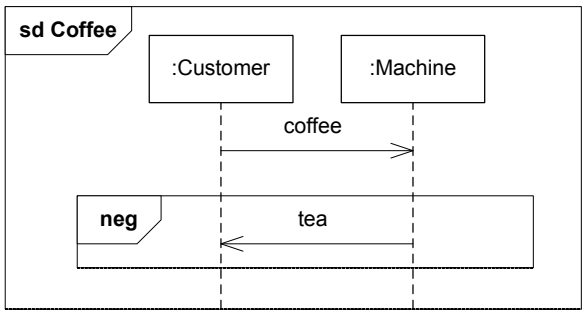

Figure 3.4: Negative behaviours

The discussion of these examples demonstrates that a precise semantics for UML 2.x interactions is needed in order to understand the exact sets of positive and negative behaviours described by an interaction. In particular, the following is a list of features of interactions that must be addressed by the work in this thesis:

- An interaction may describe both positive (i.e. valid) and negative (i.e. invalid) system behaviour.
- Interactions are partial, meaning that there are behaviours not described as either positive or negative.
- Interactions may specify both mandatory (i.e. required) and potential (i.e. optional) behaviour.
- Interactions may specify time constraints, and these may be related to both the sending, the arrival (i.e. reception) and the consumption of messages.

[2]Some readers may find that this is a strange question to ask for the given interaction. However, one possible interpretation of the standard is that *everything* except the behaviour inside neg should be positive, i.e. that everything is positive except from the user receiving tea.

3.2.2 Model Relations

We find it useful to distinguish between three basic ways that two interactions describing the same system may be related:

1. One of the interactions is a corrected version of the other, for instance correcting errors or taking into account changed requirements.
2. One of the interactions is a refinement, i.e. a more detailed description, of the other interaction.
3. The two interactions describe the system addressing di erent concerns, i.e. using di erent viewpoints.

Although correction is a central part of system development, relating the original and the corrected model is interesting mainly for ensuring that important refinement relations and viewpoint correspondences still hold after the correction. In general, using viewpoints are particularly useful when specifying large and complex systems, while refinement is important in any step-wise and incremental development process.

Consequently, the focus of this thesis is on refinement relations. The concept of refinement has been studied within the area of formal methods since the early 1970s (see section 4.4), and one of the challenges of this thesis is to find out how the essence of this theory may be explained and used in the practical setting of UML 2.x interactions.

From formal methods, we know that in order to support step-wise, compositional development, the semantics of UML 2.x interactions should be compositional. Following [dRdBH+01], this means that the semantics of an interaction should be a function of the semantics of each of its sub-interactions and the operator(s) used for composing them. No more knowledge about the operands is required. Also, the composition operators should be monotonic with respect to refinement, meaning that separate refinement of each operand results in a refinement of the complete interaction. As demonstrated in e.g. [dR85], [Col93] and [MS00], achieving compositionality is not straightforward, and great care should be taken when formalizing the semantics of UML 2.x interactions. For the refinement relations, we also know that they should be transitive, as this ensures that the result of several successive refinement steps is a refinement also of the original interaction.[3]

In addition to refinement, we also address the notion of compliance for relating UML 2.x interactions and computer systems. The final system should be in compliance with all of the interactions made during the step-wise development process, meaning that compliance should be a special case of refinement. As UML 2.x interactions does not prescribe any particular technology to be used for the final system, the notion of compliance should not depend on any particular kind of technology either.

3.3 Goals and Success Criteria

The overall goal of this thesis is to improve system development processes that use UML. Based on the above analysis, we conclude that there is a need for a method that may be used by both

[3] For simplicity, we often refer to monotonicity as a property of the refinement relations, and list monotonicity together with other refinement properties such as transitivity and reflexivity.

tool vendors and UML users. In particular, tool vendors need:

- A precise semantics for UML 2.x, making it possible to create tools that may assist when creating and analysing UML models, and possibly also perform automatic analysis of UML models.
- Precise definitions of possible model relationships, enabling tools to perform consistency checking and assist in creating and validating refinements.

In addition to improved tool support, UML users need

- Methodological guidelines for creating, developing, implementing and maintaining specifications expressed as UML models.

Looking at the combined needs of tool vendors and UML users, and narrowing it down to the scope of this thesis, there is a need for a formal foundation for UML 2.x interactions, consisting of:

- A formal definition of the semantics of UML 2.x interactions.
- An explanation of useful refinement relations for interactions, with corresponding formal definitions.
- An explanation of the relation between interactions and computer systems, i.e. an answer to the question of when a given system is in compliance with a given interaction. Again, the informal explanation should be accompanied by corresponding formal definitions.
- A methodology explaining how to use this formal foundation in practical system development with UML 2.x interactions.

In order to verify that the STAIRS method presented in this thesis provides the needed foundation, based on the above analysis we also formulate a set of success criteria for such a foundation.

- The formal semantics should
  - take into account the partiality of UML 2.x interactions.
  - handle both positive and negative behaviour.
  - handle both mandatory and potential behaviour.
  - include a notion of time.
  - be compositional, i.e.
    - the meaning of an interaction should be completely determined by the semantics of its sub-interactions and the composition operators used.
    - the composition operators should be monotonic with respect to refinement.
  - be in accordance with the UML 2.x standard.
- The refinement relations should

  - take into account the partiality of interactions.
  - handle both positive and negative behaviour.
  - handle both mandatory and potential behaviour.
  - capture the main refinement notions known from classical formal methods.
  - be transitive, thus enabling stepwise development.
- The compliance relation between interactions and computer systems should
  - be a special case of refinement.
  - be independent of technology used for the computer system.
- The methodology should
  - be conservative, i.e. based on existing UML methodology.
  - be useful without thorough knowledge of the formal definitions.

Chapter 4

State of the Art

This chapter presents state of the art representing background material for this thesis work. In particular, we investigate to what extent existing theories and methods fulfil the requirements from section 3.3.

In section 4.1 we consider methods for defining semantics, and in section 4.2 we investigate alternative ways of introducing time in the semantics. Section 4.3 considers di erent ways to specify alternative behaviours, i.e. di erent kinds of nondeterminism. Section 4.4 provides an overview of refinement in various formal methods, focusing on to what extent they include mechanisms for supporting the special features of interactions described in section 3.2.1. In section 4.5, we consider UML-related methodologies and other relevant work on UML. When advancing UML from version 1.5 [OMG03b] to version 2.0 [OMG04], sequence diagrams underwent a major revision, strongly influenced by Message Sequence Charts (MSC) [ITU99]. MSC is treated in section 4.6.

Our overall conclusion is that these theories and methods provide useful background material, but they are not su cient for fulfilling the requirements formulated for STAIRS in section 3.3.

4.1 Semantics

The semantics of a language defines the meaning of statements formed using the syntactical constructs of that language. In computer science, it is common to categorize semantics definition methods as axiomatic, denotational or operational (see e.g. [Sch96]). In this section we give a brief overview of each of these methods and evaluate their suitability for defining the semantics of UML 2.x interactions.

In axiomatic semantics, the meaning of individual statements is defined indirectly by describing logical axioms and rules that apply to these statements. Axiomatic semantics are mainly used for deriving or proving desirable properties of the statements, such as in Hoare logic [Hoa69]. The axioms given are usually concise and understandable, but the descriptions tend to be very large and complex for real languages with many basic constructs [Pri00]. As a result, we find axiomatic semantics less suitable for defining the semantics of interactions.

In denotational semantics, the meaning of statements in the language is given as a mathematical function from syntactical expressions to a well-known domain. The main challenge using

denotational semantics is deciding on the target domain that should be used. On the one hand, the domain should be well-known and thus understandable, while on the other hand the semantic function should be as simple as possible for improving readability. A common approach is therefore to first define a specialized target language (using e.g. axiomatic semantics), and then basing the denotational semantics on this specialized language [Pri00]. In the UML 2.x standard, the semantics of interactions is explained using traces of event occurrences, meaning that such traces would form a natural target domain for a formal denotational semantics of interactions.

Compared to axiomatic and denotational semantics, operational semantics is closer to a real implementation. In operational semantics, the meaning of individual statements are defined in terms of how these statements may be executed using an abstract interpreter (which in turn must have a well-defined semantics). Operational semantics provides a good formalization of implementation, and is often easily understandable for tool developers. However, it may be difficult to derive formal proofs from an operational semantics [Pri00], making it less suitable for our purposes.

To conclude, a formal semantics for UML 2.x interactions should most probably use a denotational semantics based on traces of events.

4.2 Time

There exists several techniques for introducing time in the syntax and semantics of specification languages. In [Lam05], a distinction is made between explicit- and implicit-time descriptions. In explicit-time descriptions, time is introduced by a special variable representing the current time. The passing of time is modelled by an action incrementing this special variable. The advantage of using explicit-time is that the time variable is treated in the same way as other variables by both the language and its tools. Explicit-time works well for e.g. state machines, but is less suitable for trace-based formalisms and other formalisms containing no explicit notion of a global state. Instead, such formalisms usually use implicit-time descriptions, in which the language is extended with special constructs for expressing timing properties.

[AH92] contains a survey of possible formal semantics for real-time systems. A general semantics based on interval sequences is described, together with four semantical choices leading to a total of sixteen possible kinds of formal semantics. The first choice is whether to use state sequences or observation sequences in the semantics. In section 4.1, we have already concluded that traces of events, i.e. observation sequences, should be used. Secondly, we will make the assumption that events are instantaneous, meaning that using time points is sufficient and that the more general time intervals are not needed.

The third choice in [AH92] is whether to use strictly monotonic or weakly monotonic time. Weakly monotonic time means that adjacent time instants may be identical, and is necessary for modelling interleaving of simultaneous events. This is the case for interactions, where two events on different lifelines may occur at the same time. Finally, there is a choice between real-numbered and integer time. [AH92] states that integer time is sufficient for synchronous systems. However, interactions typically describe asynchronous messages, meaning that real-numbered time seems to be the best choice.

4.3 Nondeterminism

In some sense, potential and mandatory behaviours may be understood as di erent kinds of nondeterminism. Potential behaviour is similar to nondeterminism in the form of underspecification, where an implementer is free to choose only one of the given behaviours. As for mandatory behaviour, inherent nondeterminism requires that all of the behaviours given in the specification are also reflected in the final system.

As pointed out in e.g. [Ros95] and [Jür01], distinguishing between underspecification and inherent nondeterminism (also called unpredictability) may be essential for ensuring certain security properties of a system. For instance, when creating keys and nounces, the set of possible outcomes should be su ciently large to make them unguessable by an adversary. If the di erent alternatives are specified using underspecification, this set may be significantly reduced in the final system leading to an insecure system. However, most formalisms do not distinguish between underspecification and inherent nondeterminism. In [Jür01], unpredictability is ensured using specific primities instead of relying on nondeterministic choice being interpreted as inherent nondeterminsm.

In the setting of algebraic specifications, [WM01] argues for using explicit nondeterminism in cases where underspecification might in fact lead to overspecification. However, the resulting system may still be deterministic.

In VDM-SL, a similar distinction is made between underdeterminedness (i.e. underspecification) and nondeterminism when interpreting looseness in specifications (i.e. specifications allowing alternative behaviours) [LAMB89]. Looseness in function definitions is interpreted as underdeterminedness, meaning that the exact definition is chosen at implementation time. Looseness in operations is interpreted as nondeterminism where the choice may be delayed until run-time, meaning that the final system may be either deterministic or nondeterministic.

CSP [Hoa85, Ros98] includes two di erent operators for nondeterminism. With internal nondeterminism, the system is free to choose whether it should o er 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 and the system must be able to perform all alternatives.

A similar distinction is that between angelic and demonic choice made in e.g the refinement calculus [BvW98]. With angelic choice, the choice between the alternatives is made by the system with the goal of establishing a given postcondition. This means that if the behaviours are similar up to some point, the choice between them may be deferred as long as possible in order to increase the chances of obtaining the desired end result. Demonic choice, on the other hand, 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.

[SBDB97] extends the process algebraic language LOTOS [ISO89] with a disjunction operators for specifying implementation freedom (i.e. underspecification), leaving the LOTOS choice operator to be used for inherent nondeterminism. With this new disjunction operator, exactly one of the alternatives may be implemented, in contrast to the usual interpretation of underspecification which also allows implementations with several of the alternative behaviours.

To conclude, the described formalisms include a variety of operators for specifying non-

determinism and choice, but the distinction between potential and mandatory behaviour is not fully covered by any of these approaches.

4.4 Refinement

[Dij68], [Wir71] and [DDH72] introduced the notion of stepwise program construction/refinement in the setting of sequential programs. In each refinement step, one or more high-level instructions may be decomposed into more detailed instructions, refining also the data structure whenever necessary. The notion of stepwise refinement was then formalized in e.g. [Hoa72] and [Jon72].

Later, methods such as CSP [Hoa85, Ros98], TLA [AL91, Lam02] and F [BS01] have investigated refinement in the setting of concurrent and reactive systems. Common for all of these methods is that in each refinement step, properties are added to the specification in order to make it more precise or deterministic. This means that the properties of the refinement should imply the properties of the original specification. For trace-based formalisms this corresponds to trace inclusion, i.e. all traces of the refinement must also be traces of the original specification.

In addition to reducing the set of possible behaviours, a refinement step may also make changes to the representation of data. This is captured by the notion of interface refinement in e.g. [Lam02] and [BS01], where the correspondence between the two representations is given as part of the refinement step. A unifying treatment of data refinement may be found in [dRE98]. [BS01] also defines a more general notion called conditional refinement, in which additional assumptions may be made about the environment.

Traditional refinement methods assume that the specifications are complete, meaning that if a behaviour does not have the properties required by the specification, that behaviour is negative and should not be exhibited by the specified system. As we have seen in section 3.2.1, this "closed world assumption" does not hold for interactions in general. As a consequence, the methods described so far cannot be directly applied in the setting of interactions.

In traditional pre/post-specifications [Hoa69], arbitrary behaviour is allowed if the pre-condition is false. Still, all behaviours are categorized as either positive or negative. Positive behaviours are those where the pre-condition is false, together with the behaviours satisfying both the pre- and the post-condition. Behaviours that satisfy the pre-condition but not the post-condition are negative. Refinement means to reduce the set of positive behaviours, either by strengthening the post-condition or weakening the pre-condition.

In [MBD00], pre/post-specifications in Z are given a three-valued interpretation, the third truth value being "don't care" or undefinedness. In this interpretation, behaviours with a false pre-condition are undefined and may later be refined as either positive or negative. To restrict the undefinedness, more general guards are introduced in addition to the pre-conditions. If a guard is false, the behaviour is negative, while if the guard is true and the pre-condition is false, the behaviour is undefined. Refinement now means removing undefinedness by either weakening the pre-condition or strengthening the guard, or removing underspecification by strengthening the post-condition.

[MBD00] also presents an alternative three-valued interpretation in order to capture required nondeterminism, i.e. mandatory behaviour. In this interpretation, a behaviour is mandatory if the pre-condition is true, while behaviours where the pre-condition is false and the guard is true

are potential behaviours. A refinement step may no longer strengthen the post-condition, but the gap between the guard and the pre-condition may be made smaller as long as it is not eliminated entirely.

These results concerning pre/post-specifications cannot be directly applied to specifications in the form of interactions. Typically, pre-conditions apply to the input values of a (sub-)system, whereas post-conditions constrain the set of valid output values. In other words, if the pre-condition holds, the post-condition categorizes all possible output behaviours as either positive or negative. For an interaction, it is the complete sequence of events that are either positive, negative, or not described by the interaction. As a result, an interaction may be incomplete also with respect to the allowed output behaviours for some valid input.

[dJvdPH00] presents a formal method for refining incomplete requirements specifications, incomplete meaning "yet unfinished". However, the incompleteness addressed is a controlled form of incompleteness, limited to additional nondeterminism (intended or unintended) and parameterized specifications where the parameters refer to parts that are not specified yet. General incompleteness, such as may be the case with interactions, is not addressed.

4.5 UML-Related Methodologies

There exists a number of UML-related methodologies, including the Unified Process [JBR99], the Rational Unified Process (RUP) [Kru04], Catalysis [DW99], KobrA [ABB+02] and Agile Modeling [Amb02]. As representatives, we have here chosen to focus on RUP and Catalysis. RUP is a specialization of the Unified Process, and these two methodologies are the ones most closely related to UML. Catalysis is chosen as it includes the most thorough treatment of refinement.

RUP [Kru04] is a generic software engineering process developed as a complement to UML. While UML is only a language that may be used for expressing models, RUP describes which models should be created at each stage in the development process. RUP divides the development process into four main phases consisting of a sequence of iterations, and nine disciplines that cut across these iterations. Although prescribing the creation of a number of UML models (and other documents), RUP is vague on the relationship between these. [Kru04] states a few places that two or more models should be consistent, or that one model should refine another, but this is not formalized and the concept of refinement is not explored in any depth.

Catalysis [DW99] is a method for object and component-based development using UML 1.x. For the development process, Catalysis gives a number of process patterns that may be adapted by a concrete system development project. Catalysis include many of the same ideas as RUP, such as phases, iterations, and the various activities performed within each of the iterations. For our purposes, the most interesting part of Catalysis is its treatment of refinement. Four basic kinds of refinement are defined:

- Operation refinement: Refining the behaviour of a type, i.e. its operation specifications (usually in the form of pre- and post-conditions).

- Model refinement: Refining the attributes of a type, i.e. its static model, allowing the implementation attributes to be di erent from those of the model.

- Action refinement: Refines a single action into a set of actions or a complex protocol of interactions between objects.
- Object refinement: Refines a single object into a set of objects.

Most refinements can be understood as combinations of these four, either in sequence ("big refinements") or performed together (for instance combining action and object refinement). The same refinement relations are used both for relating two specifications at di erent level of abstraction, and for relating specifications and computer systems. Several examples of Java implementations are given.

Catalysis does not precisely define the various kinds of refinement as the aim is a "more practical solution". Instead, [DW99] gives an overview of questions that must be answered in order to justify that a basic refinement step is correct. For action refinement, which is the most relevant for interactions, the question to be answered is: "What sequences of detailed actions will realize the e ect of the abstract action?". In addition to writing down a justification for the refinement, one should also write down the reasons from choosing this realization instead of one of the alternatives.

Neither of the methodologies mentioned above provides any attempt to do anything about the lack of a precise semantics for UML 2.x. However, there have been many other attempts to formalize UML (see e.g. [Whi02]). Much of the work on UML have focused on the static diagram types (such as class diagrams), and are not relevant for our work on interactions.

[Öve99] gives meaning to UML 1.x collaborations (which are described using interaction diagrams) by defining what it means for at set of objects to conform to a collaboration. Being based on message sequences, and not sequences of events, there is no way to express weak sequencing. The partiality of interactions are taken care of, in that objects may participate in other collaborations as well. As UML 1.x does not include any operators for negation, the concept of negative behaviours is not treated. For relating two collaborations, [Öve99] defines the notion of specialization. The specialization must include all sequences of the original collaboration, but may add both new sequences and new messages interleaved in the original sequences.

A more formal treatment of UML 1.x collaborations is given in [Kna99], using temporal logic and also incorporating semantics of actions in the form of transition systems. [Kna99] also provides a semantics based on pomsets, which are seen as closer to the informal semantics given in the UML 1.x standard. Again, the concept of negative behaviours is not treated. Neither is there any notion of refinement.

In [GHK99], the behavioural diagram types of UML 1.x (including state diagrams and collaboration diagrams) are given a common semantics in the form of constraint processes in cTLA (compositional TLA). Partial specifications are handled, in that each process constrains only those actions that are relvant with respect to the given diagram. However, negative behaviours are not treated, and neither are refinement.

[Jür02] (later published in a revised form as [Jür05]) provides a formal semantics for a restricted and simplified part of several kinds of UML 1.x diagrams, including sequence diagrams. The semantics is based on Abstract State Machines, and does not include explicit negative behaviours. Corresponding to the similar notions in [BS01], [Jür02] defines behavioural and interface refinement for UML specifications. In addition, delayed refinement is defined for being able to introduce time delays. However, the partiality of interactions is not taken into account. Neither is there any distinction between mandatory and potential behaviours.

4.6 Message Sequence Charts

UML 2.x interactions are strongly influenced by Message Sequence Charts (MSC) [ITU99]. Although there are some differences in terminology and the graphical syntax used, the two sequence diagram dialects are very similar [Hau04]. Simple MSCs correspond to the sequence diagrams of UML 2.x, and includes operators for specifying alternatives, parallel composition, iteration, exceptions and optional behaviour. All of these operators are included in UML 2.x interactions, where the operator break corresponds to MSC exceptions [Hau04]. UML 2.x interaction overview diagrams are a slightly generalized version of high-level MSCs, which provides an overview of how the basic MSCs are composed.

MSC does not provide any constructs for defining negative behaviour. Instead, [Hau97] proposes a methodology where different MSC documents (a number of MSCs) may be categorized as describing possible, not possible or all possible sequences of the system.

The MSC specification [ITU99] includes informal descriptions of the intended semantics. This has not been formalized, but [ITU98] contains an official operational semantics for [ITU96], a previous version of MSC. The semantics is defined compositionally, and includes definitions for the time concepts given in [ITU96].

The only notion of refinement mentioned in [ITU99] is that of instance decomposition. This refinement is fairly weak, as it does not require behavioural refinement but only that there is some structural similarity between the decomposed instance and its decomposition. The formalization in [ITU98] does not treat refinement at all.

The work in [Krü00] is very relevant in our setting as it contains both a formal semantics, refinement notions and a methodology for MSC. While the operational semantics in [ITU98] is based on process algebra, [Krü00] defines a denotational semantics based on streams where the semantics of a given MSC specification is a set of channel and state valuations. The main difference between the two is that [Krü00] uses strict and not weak sequencing. As noted in section 3.2.1, weak sequencing is important for UML 2.x interactions. To achieve weak sequencing, [Krü00] requires that this should be explicitly stated in the MSC. The semantics in [Krü00] is defined compositionally, and important properties such as associativity and commutativity of the operators are established. The inclusion of time is discussed informally, but is not part of the formal semantics.

[Krü00] defines four different refinements notions for MSCs. [Krü00] allows references to non-existing MSCs, which are interpreted as arbitrary behaviour. By reference binding, this arbitrary behaviour may then be refined by defining the missing MSCs. Property refinement means removing underspecification by reducing the possible behaviours of the overall system, while message refinement means substituting an interaction sequence or protocol for a single message. Finally, structural refinement means replacing a simple component with a set of other components, similar to instance decomposition in MSC (and UML 2.x). Both message refinement and structural refinement are global substitutions, i.e. all occurrences of the message or component must be replaced by the same sequence or decomposition. Property refinement is reflexive, transitive and monotonic with respect to all MSC operators.

Related to implementations, [Krü00] defines four possible interpretations of a single MSC, such that within one specification the different MSCs may have different interpretations. While an existential MSC describes partial system behaviour that *may* occur during execution of the system, a universal MSC describes behaviour that *must* occur at some point in time in all executions

of the system. An exact MSC prohibits all other behaviours than the ones specified by the MSC, while a negated MSC describes negative behaviours. Except from existential interpretation, the notions are monotonic with respect to property refinement.

Live Sequence Charts (LSC, [DH01, HM03]) is an extension of MSC focusing on the ability to specify liveness, i.e. things that must occur. LSC has influenced the work in [Krü00], and to some extent also UML 2.x interactions.

LSC distinguishes between existential and universal diagrams. An existential diagram specifies possible behaviour, i.e. example scenarios that must be satisfied by at least one execution of the system. A universal diagram specifies necessary behaviour, i.e. behaviour that must be fulfilled by all executions. A universal diagram consists of two parts, a prechart and the main chart. If the system at some point in time fulfils the prechart, then the main chart must also be fulfilled. However, if the prechart is never satisfied, the diagram imposes no restriction on the system behaviour. [DH01] also discusses the closing of a specification with respect to a system, which means that for all possible executions of the system at least one LSC is satisfied, i.e. the system cannot exhibit behaviour not described by any of the diagrams.

Within a single diagram, LSC elements may be specified as cold (meaning that they *may* occur) or hot (meaning that they *must* occur). LSC does not include the standard MSC operators for control structures like alternatives and iteration. Instead, cold conditions may be used for specifying this. Hot conditions may be used to specify anti-scenarios (i.e. negative behaviour) by including the unwanted behaviour in the prechart of a universal diagram where the main chart contains a single false hot condition.

[HM03] provides an operational semantics for LSC, tailored towards their Play-Engine tool which provides means for capturing requirements by using a graphical user interface and afterwards executing the resulting LSCs. For simplifying the tool, the LSC semantics di er from MSC on important aspects. For instance, synchronization between all lifelines is performed at the beginning of each sub-diagram. As a result, the semantics of a loop is not the same as the semantics of writing its content in full. Also, the temperature (hot/cold) of messages is only syntactic sugar without any semantic implication, as the temperature of the corresponding locations (for the send and the receive event) is given higher priority.

[HM03] also includes more advanced constructs such as variables and time. Even though there is a notion of sub-diagrams, there are no construct for referencing one LSC within another or for composing LSCs. LSC contains no notion of refinement, but [HM03] mentions object refinement (i.e. decomposition) as an important area for further research.

Chapter 5

STAIRS

In this chapter we give an introduction to the STAIRS method. First, in section 5.1 we describe the initial work on STAIRS by Øystein Haugen and Ketil Stølen. In section 5.2, we give a brief summary of STAIRS as it appears today.

5.1 The First STAIRS

At the UML 2003 conference, Øystein Haugen and Ketil Stølen presented the first paper on STAIRS, called "STAIRS — Steps to Analyze Interactions with Refinement Semantics" [HS03]. This paper contained the basic ideas of STAIRS, which has been further developed by the work presented in this thesis.

The main ideas in [HS03] can be summarized as follows:

Mandatory vs potential behaviour. A specification may employ nondeterminism in order to capture two very di erent kinds of requirements. First, nondeterminism in the form of under-specification is used where there are alternative behaviours serving the same purpose and a correct implementation is only required to fulfil one of these. The alternative behaviours then represent what is referred to as potential behaviour.

On the other hand, explicit nondeterminism is used where the nondeterminism is required also by a correct implementation. As an example, every lottery ticket in a lottery should have the possibility to win the prizes, even though only one ticket is drawn for each prize. Explicit nondeterminism is referred to as mandatory behaviour.

For describing potential behaviour, the common UML alt operator is used, while a new operator called xalt is introduced in order to capture mandatory behaviour and distinguish this from potential behaviour.

Negative, positive and inconclusive behaviour. An interaction is understood as describing a set of positive (i.e. valid, legal, or desirable) traces, and a set of negative (i.e. invalid, illegal, or undesirable) traces. Traces not considered by the interaction are referred to as inconclusive.

Supplementing, narrowing and detailing. Three basic refinement relations are described, corresponding to basic system development steps:

Supplementing categorizes earlier inconclusive traces as either positive or negative, recognizing that early specifications in the form of interactions are usually incomplete. Positive traces remain positive, and negative traces remain negative.

Narrowing reduces the set of positive traces by redefining some of them as negative, capturing new design decisions or matching the problem better. Inconclusive traces remain inconclusive and negative traces remain negative.

Detailing introduces a more detailed description without significantly altering the externally observable behaviour.

Semantics. Trace semantics for simple interactions using the operators seq (weak sequencing), ref (interaction occurrence), par (parallel combined fragment), neg (negation), alt (potential behaviour) and xalt (mandatory behaviour) are explained informally using small example interactions.

A semantic model for interactions is proposed, capturing all of the ideas described above and in particular the distinction between mandatory and potential behaviour. In our later work on formalizing STAIRS, we chose a slightly di erent semantic model. The reasons for this, and a comparison between the di erent models, may be found in section 7.1. We also chose another interpretation of the neg operator, as discussed in paper 3

5.2 STAIRS Today

As explained in section 2, the STAIRS method has been developed by iteratively adding more and more features (e.g. new operators, refinement relations or methodological advice) to it. As our understanding of system development using UML 2.x interactions has improved, some minor adjustments have been made to the formal definitions in order to reflect this. In this section, we give a brief overview of the main syntax, semantic model and refinement relations as they appear in STAIRS today. For motivation, examples, and more details we refer to the attached papers. Also, the pragmatical guidelines from paper 5 are not repeated, neither are the compliance relations from paper 7.

5.2.1 Syntax

The syntax of basic STAIRS interactions is defined by the BNF-grammar in figure 5.1. `Signal` represents the actual content of a message, `Lifeline` is the name of a lifeline (representing a component) in the diagram and `Set` should be an expression that evaluates to a subset of $\mathbb{N}_0 \cup \{\infty\}$ (the natural numbers including 0, and ∞).

In addition to the operators in figure 5.1, veto and opt are defined as high-level operators by:

$$\mathsf{veto}\ d \stackrel{\mathsf{def}}{=} \mathsf{alt}\ [\mathsf{refuse}\ [d], \mathsf{skip}] \tag{5.1}$$

$$\mathsf{opt}\ d \stackrel{\mathsf{def}}{=} \mathsf{alt}\ [d, \mathsf{skip}] \tag{5.2}$$

The ref construct is seen as a syntactical short-hand for the contents of the referenced interaction. Gates are treated in appendix A of paper 2.

⟨Interaction⟩ → ⟨Empty⟩ | ⟨Event⟩ | ⟨Refuse⟩ | ⟨Assert⟩ |
⟨Weak sequencing⟩ | ⟨Parallel execution⟩ |
⟨Loop⟩ | ⟨Potential alternatives⟩ |
⟨Mandatory alternatives⟩
⟨Empty⟩ → skip
⟨Event⟩ → (⟨Kind⟩ , ⟨Message⟩)
⟨Kind⟩ → ⟨Transmission⟩ | ⟨Reception⟩
⟨Transmission⟩ → !
⟨Reception⟩ → ?
⟨Message⟩ → (Signal , ⟨Transmitter⟩ , ⟨Receiver⟩)
⟨Transmitter⟩ → Lifeline
⟨Receiver⟩ → Lifeline
⟨Refuse⟩ → refuse [⟨Interaction⟩]
⟨Assert⟩ → assert [⟨Interaction⟩]
⟨Weak sequencing⟩ → seq [⟨Interaction list⟩]
⟨Parallel execution⟩ → par [⟨Interaction list⟩]
⟨Loop⟩ → loop Set [⟨Interaction⟩]
⟨Potential alternatives⟩ → alt [⟨Interaction list⟩]
⟨Mandatory alternatives⟩ → xalt [⟨Interaction list⟩]
⟨Interaction list⟩ → ⟨Interaction⟩ | ⟨Interaction list⟩ , ⟨Interaction⟩

Figure 5.1: Syntax of basic STAIRS interactions

In our work, we have defined two orthogonal extensions of STAIRS, TimedSTAIRS (paper 2) and guarded STAIRS (paper 4). For TimedSTAIRS, the syntax is extended as defined by the BNF-grammar in figure 5.2. Nonterminals that are unchanged from the syntax of basic STAIRS in figure 5.1 are not repeated. In TimedSTAIRS, every event is decorated with a unique timestamp tag (TimestampTag) as a placeholder for real timestamp values. TimeConstraint is a boolean expression on such timestamp tags. In TimedSTAIRS, we also distinguish between the reception of a message (the arrival of the message in the input bu er of the lifeline) and the consumption of the message (when it is taken from the input bu er and processed by the lifeline).

For interactions with data and guards, the syntax is extended as defined by the BNF-grammar in figure 5.3. Variable should be either a global variable or a variable local to the lifeline on which the assignment is placed (not shown in our textual syntax), while Expression is a mathematical expression and Constraint is a boolean expression on variables. In guarded STAIRS, alt/xalt-operands without an explicit guard, are interpreted as having the boolean constant *true* as guard.

Except from having a few additional operators, we only address sequence diagrams that are considered syntactically correct in UML 2.x. Also, we do not handle extra global combined fragments, and for all operators except from seq and par we assume that each operand consists of complete messages only. We also require that for diagrams consisting of more than one event, the message should be complete if both the transmitter and the receiver lifelines are present in the diagram. These requirements are written down formally in paper 2.

⟨Interaction⟩ → ⟨Timed interaction⟩
⟨Timed interaction⟩ → ⟨Empty⟩ | ⟨Event⟩ | ⟨Refuse⟩ | ⟨Assert⟩|
⟨Weak sequencing⟩ | ⟨Parallel execution⟩ |
⟨Loop⟩ | ⟨Potential alternatives⟩ |
⟨Mandatory alternatives⟩ |
⟨Time-constrained interaction⟩
⟨Time-constrained interaction⟩ → ⟨Timed Interaction⟩ `tc TimeConstraint`
⟨Event⟩ → `(` ⟨Kind⟩ `,` ⟨Message⟩ `,` `TimestampTag )`
⟨Kind⟩ → ⟨Transmission⟩ | ⟨Reception⟩ | ⟨Consumption⟩
⟨Transmission⟩ → `!`
⟨Reception⟩ → ∼
⟨Consumption⟩ → `?`

Figure 5.2: Syntax of TimedSTAIRS interactions

⟨Interaction⟩ → ⟨Empty⟩ | ⟨Assignment⟩ | ⟨Constraint⟩ | ⟨Event⟩ |
⟨Refuse⟩ | ⟨Assert⟩ | ⟨Weak sequencing⟩ |
⟨Parallel execution⟩ | ⟨Loop⟩ | ⟨Guarded alt⟩ |
⟨Guarded xalt⟩
⟨Assignment⟩ → `assign ( Variable , Expression )`
⟨Constraint⟩ → `constr ( Constraint )`
⟨Guarded alt⟩ → `alt [` ⟨Guarded list⟩ `]`
⟨Guarded xalt⟩ → `xalt [` ⟨Guarded list⟩ `]`
⟨Guarded list⟩ → ⟨Guarded interaction⟩ |
⟨Guarded list⟩ `,` ⟨Guarded interaction⟩
⟨Guarded interaction⟩ → ⟨Guard⟩ → ⟨Interaction⟩
⟨Guard⟩ → `Constraint`

Figure 5.3: Syntax of guarded STAIRS interactions

5.2.2 Semantics

STAIRS defines denotational trace semantics for interactions that are using the syntax of section 5.2.1. Our semantic domain is the set of all well-formed traces, denoted $\mathcal{H}$. In basic STAIRS, a trace is well-formed if, for each message, the send event is ordered before the corresponding receive event. For TimedSTAIRS, we also have additional requirements ensuring e.g. that all events in a trace are ordered by time. These requirements may be found in paper 2.

The semantics of an interaction is given as a set of m interaction obligations, where an interaction obligation is a pair of trace-sets (p, n) which gives a classification of all traces in $\mathcal{H}$ into three categories: the positive traces p, the negative traces n, and the inconclusive traces $\mathcal{H} \setminus (p \cup n)$. Visually, we illustrate an interaction obligation as an oval divided into three regions as shown in figure 5.4.

Intuitively, each interaction obligation represents a mandatory alternative that must be present in any computer system compliant with the interaction. In earlier work on STAIRS, we classified

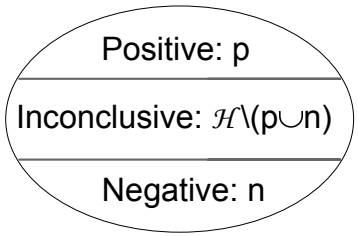

Figure 5.4: Illustrating an interaction obligation

an interaction obligation with $p \cap n \neq \emptyset$ as contradictory. We no longer find this term useful, as the same trace may very well be both positive and negative in the same interaction obligation when e.g. abstracting away guards or other details.

When defining the semantics of the individual operators, we have used the following main principles:

- The definitions should be in accordance with the informal semantics and explanations given in the UML 2.x standard.
- The definitions should be context-free, i.e. the meaning of an operator should not depend on the context in which it is used.
- All traces that are not explicitly described in a (sub-)interaction should be inconclusive for that (sub-)interaction.
- In order to support compositional development of interactions, the composition operators should be monotonic with respect to refinement.

A few times, these principles have been in conflict, and we have had to make minor compromises. In our work, monotonicity has been an important principle, as we find it essential that di erent parts of a specification may be developed separately. As proved in the attached papers, all operators except assert are monotonic with respect to refinement.[1] For assert, we have monotonicity in the special case of narrowing.

For the formal definitions of each operator, we refer to the attached papers, and in particular papers 1, 2 and 4. Some definitions are slightly di erent in the various papers. This is mainly structural di erences, such as defining operators also on the level of sets of interaction obligations or changing the number of operands (which makes no di erence due to the associativity results in paper 2). In the following, we comment on the more significant changes that have been made during the development of STAIRS.

First of all, papers 1 and 2 used the UML operator neg for specifying negative behaviour. In paper 3, we investigated alternative formal definitions for neg, and argued for replacing neg

[1] Actually, each of the attached papers includes only a subset of the total set of operators covered by STAIRS, depending on the focus of that particular paper. All operators (except assert) are proved to be monotonic with respect to the most general refinement relation (section 5.2.3, definition (5.7) with definition (5.5)). However, in the papers introducing other refinement relations, monotonicity is only proved for the operators included in each of these papers.

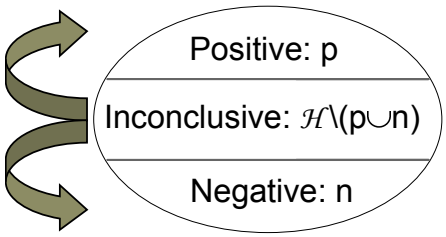

Figure 5.5: Supplementing of interaction obligations

with two new negation operators called refuse and veto. The operator refuse is the most basic operator, used in the definition of veto (which corresponds to our initial definition of neg). All later papers use refuse and veto instead of neg.

In the definition of guarded alt in paper 4, we followed the UML 2.x standard in that if all guards in an alt-construct is false, the empty trace (i.e. doing nothing) should be positive. However, the resulting definition is not associative, di cult to read and understand, and also contradicts our principle that traces not explicitly described should be inconclusive. As a result, in paper 5 we changed the definition so that the implicit empty trace was no longer included. The resulting definition is more readable, and also re-establishes associativity of alt. If the empty trace should be positive in case all other guards are false, this is easily specified by including skip with the guard *else* as the last alt-operand. For alt-constructs where the guards are exhaustive, the definitions in papers 4 and 5 are equivalent.

5.2.3 Refinement

In STAIRS, we have defined several refinement relations for relating UML 2.x interactions made at di erent stages of the development process. The refinement relations are found at two di erent levels: refinement of interactions and refinement of individual interaction obligations.

For individual interaction obligations, we have formalized the notions of supplementing, narrowing, and detailing described in [HS03]. In addition, in paper 7 we introduced the notion of restricted refinement.

Supplementing means adding more behaviours to the specification by reducing the set of inconclusive behaviours. This is illustrated in figure 5.5 and formally defined by:

$$(p, n) \rightsquigarrow_s (p', n') \stackrel{\text{def}}{=} p \subseteq p' \ \wedge \ n \subseteq n' \tag{5.3}$$

Narrowing means reducing underspecification by redefining positive traces as negative. This is illustrated in figure 5.6 and formally defined by:

$$(p, n) \rightsquigarrow_n (p', n') \stackrel{\text{def}}{=} p' \subseteq p \ \wedge \ n' = n \cup p \setminus p' \tag{5.4}$$

Combining supplementing and narrowing results in our most common refinement relation, formally defined by:

$$(p, n) \rightsquigarrow_r (p', n') \stackrel{\text{def}}{=} n \subseteq n' \ \wedge \ p \subseteq p' \cup n' \tag{5.5}$$

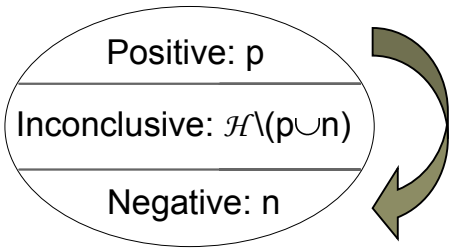

Figure 5.6: Narrowing of interaction obligations

In restricted refinement, supplementing traces as positive is not allowed:

$$(p, n) \rightsquigarrow_{rr} (p', n') \stackrel{\text{def}}{=} (p, n) \rightsquigarrow_{r} (p', n') \ \wedge \ p' \subseteq p \tag{5.6}$$

Detailing means reducing the level of abstraction by decomposing one or more lifelines, corresponding to structural decomposition in UML 2.x. The decomposition may result in a change in the sender/receiver of the messages, and also in internal messages, guards, etc being revealed. For a formal definition of detailing, we refer to paper 5.

For interactions which may have several interaction obligations as their semantics, we have defined two refinement notions: general and limited refinement. In general refinement, each interaction obligation for the original interaction must be refined by an interaction obligation for the refining interaction, but new interaction obligations may be added freely:

$$d \rightsquigarrow_{g} d' \stackrel{\text{def}}{=} \forall o \in [\![d]\!] : \exists o' \in [\![d']\!] : o \rightsquigarrow o' \tag{5.7}$$

where the refinement relation $\rightsquigarrow$ is one of the refinement relations given for interaction obligations above.

Limited refinement restricts the possibility of adding new interaction obligations by also requiring that each interaction obligation for the refining interaction must be a refinement of and interaction obligation for the original interaction:

$$d \rightsquigarrow_{l} d' \stackrel{\text{def}}{=} d \rightsquigarrow_{g} d' \ \wedge \ \forall o' \in [\![d']\!] : \exists o \in [\![o]\!] : o \rightsquigarrow o' \tag{5.8}$$

Chapter 6

Overview of the Papers

The main results of this thesis work are documented in the eight papers found in part II. In the following, we list the publication details of each paper, together with a short description of its main research contributions. For each paper, my contribution is specified.

Paper 1: Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 22(4):349–458, 2005.

This paper formalizes the main ideas of STAIRS, based on the presentation in [HS03]. This includes formal definitions of the main UML 2.x operators, the distinction between mandatory and potential alternatives, and the refinement relations supplementing, narrowing and detailing. In particular, all of the definitions take into account the partial nature of interactions. The semantic model used in this paper and all our subsequent work on STAIRS is slightly di erent from the one proposed in [HS03]. This di erence is discussed in section 7.1.

My contribution: One of four main authors, responsible for approximately 25% of the work (with an emphasis on the more formal aspects).

Paper 2: Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309. Extended and revised version of: Why timed sequence diagrams require three-event semantics. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 1–25. Springer, 2005.

This paper extends STAIRS with time and three-event semantics, arguing that distinguishing between the reception and the consumption of an event is essential for evaluating whether a system fulfils the specified time constraints or not. This paper also provides formal definitions for the treatment of gates, and for additional operators such as assert and loop. The other main contribution of this paper is the appendices (not included in the LNCS-publication) containing several proofs for the soundness of the given formalization. We have associativity, commutativity and distributivity where expected, and the refinement relations are reflexive, transitive and monotonic with respect to the defined operators, enabling step-wise and compositional development of interactions.

My contribution: One of four main authors of the original paper, and the main author behind the revised report. My main responsibility was the writing up of the detailed proofs. Altogether, I was responsible for approximately 35% of the work.

Paper 3: Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. How to transform UML neg into a useful construct. In *Norsk Informatikkonferanse NIK'2005*, pages 55–66. Tapir, 2005.

This paper contains a discussion of alternative definitions of the neg operator used for negation in UML 2.x. The main conclusion is that having only one operator for negation is not su cient to capture the di erent uses, and we propose replacing neg with the two operators refuse and veto (where veto is defined in terms of refuse).

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

Paper 4: Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. Refining UML interactions with underspecification and nondeterminism. Technical Report 325. Extended and revised version of: Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.

First of all, this paper contains a thorough discussion of the di erence between potential alternatives in the form of underspecification, and mandatory alternatives in the form of inherent nondeterminism. Secondly, STAIRS is extended with data and guards. Also, the notion of limited refinement is introduced, restricting the possibilities for increasing the inherent nondeterminism required of the specified system. Again, the appendices (not included in the journal-publication) contain the necessary proofs that transitivity and monotonicity holds also for this extended version of STAIRS, and that the definitions of guarded alternatives are consistent with the original definitions given for unguarded alternatives.

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

Paper 5: Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. The pragmatics of STAIRS. In *Proc. Formal Methods for Components and Objects (FMCO 2005)*, volume 4111 of *LNCS*, pages 88–114. Springer, 2006.

This is a tutorial paper giving pragmatical guidelines for creating and refining interactions. Also, the refinement relations of STAIRS are described in more detail.

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

Paper 6: Atle Refsdal, Ragnhild Kobro Runde, and Ketil Stølen. Underspecification, inherent nondeterminism and probability in sequence diagrams. Technical Report 335. Extended version of: Underspecification, inherent nondeterminism and probability in sequence diagrams. In *Proc. Formal Methods for Open Object-Based Distributed Systems (FMOODS 2006)*, volume 4037 of *LNCS*, pages 138–155. Springer, 2006.

This paper provides a further discussion of the di erent kinds of nondeterminism, and in particular how to understand interactions where these are combined in various ways. In addition to underspecification and inherent nondeterminism, we also discuss probabilistic choice, which may be understood as a generalization of inherent nondeterminism. The

report version includes an appendix containing proofs of claims made in the main body of the paper. Probabilistic STAIRS first appeared in [RHS05], and is slightly revised in this paper. In the setting of this thesis, it is interesting how probabilistic choice relates to underspecification and inherent nondeterminism, but we are not interested in probabilities as such. Hence, probabilities has not been a theme in the other parts of this introductory part.

My contribution: One of two main authors. The paper was written in close collaboration, and I was responsible for approximately 45% of the work.

Paper 7: Ragnhild Kobro Runde, Atle Refsdal and Ketil Stølen. Relating computer systems to sequence diagrams with underspecification, inherent nondeterminism and probabilistic choice. Part 1: underspecification and inherent nondeterminism. Technical Report 346.

In this paper we define how to understand computer systems in terms of the semantic model of STAIRS, and give criteria for when a system is correct with respect to a given interaction containing di erent kinds of nondeterminism. We discuss di erent refinement relations and corresponding compliance relations for relating computer systems to interactions, and we explore the mathematical properties of these relations. This is part 1 of the work, discussing underspecification and inherent nondeterminism. In part 2, we discuss probabilistic choice. As this thesis focuses on underspecification and inherent nondeterminism, and not on probabilities or probabilistic choice, this second part is not included here.

My contribution: The paper was written in close collaboration, with me as the main author responsible for approximately 65% of the work.

Paper 8: Ragnhild Kobro Runde. STAIRS case study: The BuddySync System. Technical Report 345.

This is the last paper in the thesis, presenting a case study demonstrating the practical usefulness of STAIRS. The paper also describes some weaknesses identified during the work on the case study, and provides suggestions for improving these weaknesses.

My contribution: I was the sole author.

Chapter 7

Discussion

This chapter contains a more detailed discussion of STAIRS than what has been possible within the context of the attached papers. In section 7.1 we compare the original semantic model of STAIRS as proposed in [HS03] to the one used in our work. In section 7.2 we evaluate STAIRS with respect to the success criteria in section 3.3, while in section 7.3 we discuss to what extent the results obtained for interaction diagrams may be generalized to other diagram types such as UML state machine diagrams. Finally, in section 7.4 we discuss related work.

7.1 The Original Semantic Model of STAIRS

The semantic model proposed in [HS03] is slightly di erent from our semantic model as presented in section 5.2.2. In this section we first present the semantic model of [HS03] in section 7.1.1. Our main motivation for using another semantic model comes from being able to refine and implement the di erent xalt-operands separately. Hence, in section 7.1.2 we define the refinement notion supplementing and narrowing for the semantic model in [HS03], while section 7.1.3 contains a discussion of alternative definitions of xalt.

7.1.1 Semantic Model

The semantic model proposed in [HS03] di ers from our semantic model as presented in section 5.2.2 in that in [HS03] there are several positive trace-sets (corresponding to our interaction obligations), but only one negative trace-set. This alternative semantic model is illustrated in figure 7.1.

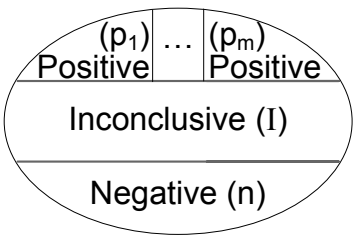

Figure 7.1: Illustrating the semantic model of [HS03]

Formally, the semantics described in [HS03] is a pair (P, n), where P is at set $\{p_1, \ldots, p_m\}$ of m sets of positive traces, while n is a set of negative traces. The inconclusive traces are those that are neither in any of the positive trace-sets nor in the negative trace-set, i.e. all traces in the set $I = \mathcal{H} \setminus (n \cup \bigcup_{i \in [1,m]} p_i)$. The intuition behind this semantic model is that a correct implementation should be able to perform at least one trace from each of the positive trace-sets, but none of the negative traces.

Using only one negative trace-set is intuitively appealing, since defining a trace as negative would mean that the system should definitely not be able to perform that trace. In contrast, in our semantic model a trace that is negative in one interaction obligation may still be positive or inconclusive in another, and the trace may or may not be performed by the system.

7.1.2 Refinement

Refinement is only informally explained in [HS03], and not all special cases are treated. The following formalization constitutes our interpretation of the informal exposition in [HS03].

Supplementing means to reduce the set of inconclusive traces by adding former inconclusive traces to the negative trace-set, to one of the positive trace-sets, or as a new positive trace-set. Formally:

$$\begin{aligned}(P, n) \rightsquigarrow_s (P', n') \quad \stackrel{\text{def}}{=} \quad & n \subseteq n' \wedge n' \setminus n \subseteq I \\ & \wedge \forall p \in P : \exists p' \in P' : p \subseteq p' \wedge p' \setminus p \subseteq I\end{aligned} \tag{7.1}$$

Narrowing means reducing underspecification by reducing one or more of the positive trace-sets without making any of them empty. If, as a result of this, a former positive trace is no longer in any of the positive trace-sets, the trace should be included in the negative trace-set. Formally:

$$\begin{aligned}(P, n) \rightsquigarrow_n (P', n') \quad \stackrel{\text{def}}{=} \quad & n' = n \cup (\bigcup_{p_i \in P} p_i \setminus \bigcup_{p'_i \in P'} p'_i) \\ & \wedge \forall p \in P : \exists p' \in P' : p' \subseteq p \wedge p' \neq \emptyset\end{aligned} \tag{7.2}$$

One problem with these refinement definitions is that combining supplementing and narrowing is not transitive with respect to the proposed notion of compliance. As an example, let t_1 and t_2 be two di erent, well-formed, traces and consider the three interactions d_1, d_2 and d_3 with semantics as follows:

$$\begin{aligned}[\![\, d_1 \,]\!] &= (\{\{t_1\}\}, \emptyset) \\ [\![\, d_2 \,]\!] &= (\{\{t_1, t_2\}\}, \emptyset) \\ [\![\, d_3 \,]\!] &= (\{\{t_2\}\}, \{t_1\})\end{aligned}$$

It is straightforward to see that d_3 is a narrowing refinement of d_2, which is a supplementing refinement of d_1. A system performing only the trace t_2 is in compliance with d_3 (and d_2), but not d_1 as it contains no trace from the positive trace-set $\{t_1\}$.

This problem could be solved for instance by a more relaxed notion of compliance where for each of the positive trace-sets, the system must perform either one of the positive traces or an inconclusive trace. This notion would be closer to our compliance relation for interaction obligations.

7.1.3 Definition of xalt

We now turn to discussing possible definitions of xalt, using the semantic model described above. From the explanations in [HS03], we get that the new set of positive trace-sets is the union of the two sets given by each of its operands. However, it is not obvious what the new negative trace-set should be. The two most obvious choices are using disjunction or union of the two operand sets.

In our chosen semantic model in section 5.2.2, consisting of a set of interaction obligations, a trace is negative for a system only if it is negative in all interaction obligations. In the semantic model of [HS03], this would correspond to using disjunction of the negative trace-sets of the two operands:

$$[\![\ \mathsf{xalt}\ [d_1, d_2]\ ]\!] \quad \stackrel{\mathsf{def}}{=} \quad (P_1 \cup P_2, n_1 \cap n_2) \tag{7.3}$$

where we assume that $[\![\ d_1\ ]\!] = (P_1, n_1)$ and $[\![\ d_2\ ]\!] = (P_2, n_2)$.

However, this definition is not monotonic with respect to the refinement definitions (7.1) and (7.2), meaning that d_1 and d_2 cannot be developed separately. One problem is that in the original specification, a trace may be inconclusive in one operand and positive in the other. As a simple example, consider the interaction

$$d = \mathsf{xalt}\ [t_1, \mathsf{alt}\ [t_2, t_3]]$$

with semantics

$$[\![\ d\ ]\!] = (\{\{t_1\}, \{t_2, t_3\}\}, \emptyset)$$

A valid narrowing refinement of the operand alt $[t_2, t_3]$ is the interaction
alt $[t_2, \mathsf{refuse}\ [t_3]]$. Replacing alt $[t_2, t_3]$ with its refinement in the original interaction, we get the interaction

$$d' = \mathsf{xalt}\ [t_1, \mathsf{alt}\ [t_2, \mathsf{refuse}\ [t_3]]]$$

with semantics

$$[\![\ d'\ ]\!] = (\{\{t_1\}, \{t_2\}\}, \emptyset)$$

However, d' is not a valid refinement of d as the trace t_3 has been moved from positive to inconclusive which is not allowed by either supplementing or narrowing.

Another problem with definition (7.3) is that when refining each of the xalt-operands separately, the positive traces of the first operand may be supplemented as positive in the second operand, and vice versa. Consider again the interaction d in the previous example. A valid supplementing refinement of the operand alt $[t_2, t_3]$ is the interaction alt $[t_1, \mathsf{alt}\ [t_2, t_3]]$. Substituting this into the original interaction, we get the interaction

$$d'' = \mathsf{xalt}\ [t_1, \mathsf{alt}\ [t_1, \mathsf{alt}\ [t_2, t_3]]]$$

with semantics

$$[\![\ d''\ ]\!] = (\{\{t_1\}, \{t_1, t_2, t_3\}\}, \emptyset)$$

Again, this is not a valid refinement of d. We also see that a system performing only the trace t_1 is in compliance with d'' but not with the original interaction d.

Using union instead of disjunction for the negative trace-set, xalt could be defined by:

$$[\![\ \mathsf{xalt}\ [d_1, d_2]\ ]\!] \quad \stackrel{\mathsf{def}}{=} \quad (P_1 \cup P_2, n_1 \cup n_2) \tag{7.4}$$

This definition solves the monotonicity problems with respect to the first, but not the second of the examples above. In fact, with the semantic model used in this section, it is not possible to define xalt in a way that ensures monotonicity when supplementing each operand separately. Instead, a possible solution is using di erent negative trace-sets for each positive trace-set, as with our interaction obligations.

7.2 Evaluating STAIRS with Respect to the Success Criteria

In this section, we evaluate STAIRS with respect to each of the success criteria given in section 3.3.

- **The formal semantics should take into account the partiality of UML 2.x interactions.**

  This is achieved by the semantic model of STAIRS, categorizing behaviours as either positive, inconclusive or negative. As explained in section 5.2.2, one of the main principles behind our formalization is that all traces not explicitly described as positive or negative in a (sub-)interaction are inconclusive for that (sub-)interaction.

- **The formal semantics should handle both positive and negative behaviour.**

  Again, this is achieved by the semantic model of STAIRS as described for the previous success criteria.

- **The formal semantics should handle both mandatory and potential behaviour.**

  Mandatory behaviour, also referred to as inherent nondeterminism, is specified using xalt, and is reflected in the di erent interaction obligations in the semantic model. Potential behaviour, or underspecification, is specified using alt (or through weak sequencing), and is reflected in each interaction obligation having a set of positive behaviours.

- **The formal semantics should include a notion of time.**

  Time is included by giving all events a timestamp in the form of a positive, real number (see paper 2). All events in a trace must be ordered by time, but two events may happen at the same time. Time constraints restrict the valid timestamps of one or more events such that traces with valid timestamps are defined as positive, while traces with invalid timestamps are defined as negative.

- **The formal semantics should be compositional.**

  The semantics of an interaction is given in terms of the semantics of its sub-interactions, and the operators used for composing these. All operators except assert are monotonic with respect to general refinement, meaning that di erent sub-interactions may be developed separately. This is proved in papers 2, 4 and 7, which also contains monotonicity proofs for the operators and refinement relations covered by each particular paper. For assert, we have monotonicity in the special case of narrowing. As explained in paper 2, the lack of monotonicity with respect to supplementing is not important, as assert is usually used to state that all positive behaviours have been defined and that no more supplementing is needed.

- **The formal semantics should be in accordance with the UML 2.x standard.**

  We believe that the formalization in STAIRS is faithful to the underlying ideas of UML 2.x, including the partial nature of interactions and using weak sequencing as the main composition operator. However, it has not always been possible to follow the standard in every respect. In particular, we have defined two different operators for negation as explained in paper 3, extended the language with the xalt operator, and given a slightly different interpretation of guarded alt in paper 5.

  The UML 2.x standard is vague regarding how the different composition operators should be understood with respect to negative behaviours. In this situation, we have tried to follow the other principles given in section 5.2.2, and in particular that of monotonicity.

- **The refinement relations should take into account the partiality of interactions.**

  All of the refinement relations in STAIRS acknowledges that an interaction usually does not provide a complete description of the system. In particular, the notion of supplementing may be used to define more traces as positive or negative. Also, the notion of general refinement may be used to add more mandatory behaviours to the interaction.

- **The refinement relations should handle both positive and negative behaviour.**

  As the semantic model of STAIRS includes both positive and negative behaviour, so does the refinement relations. Within a single interaction obligation, negative behaviour is interpreted as behaviour that *must not* be present in the system, while positive behaviour is interpreted as behaviour that *may* be present.

- **The refinement relations should handle both mandatory and potential behaviour.**

  Refinement of both mandatory and potential behaviour is handled by defining relations for refining both individual interaction obligations and sets of interaction obligations (see section 5.2.3). By narrowing, positive traces in an interaction obligation may be redefined as negative, hence reducing the set of potential behaviours. General and limited refinement requires that all mandatory behaviour specified in the original interaction must be present also in the refinement.

- **The refinement relations should capture the main refinement notions known from classical formal methods.**

  Classical refinement in the sense of reducing underspecification is captured by the STAIRS notion of narrowing. In addition, STAIRS includes the notion of supplementing, for adding earlier inconclusive traces to the specification. Interface refinement is partially captured by the STAIRS notion of detailing, but including a more general notion of message refinement in STAIRS would probably be useful.

- **The refinement relations should be transitive.**

  All refinement relations are proved to be transitive, see papers 2, 4 and 7.

- **The compliance relation between interactions and computer systems should be a special case of refinement.**

The compliance relations in paper 7 are all special cases of refinement. With the exception of restricted compliance, all compliance relations are special cases of the corresponding refinement relations. In addition, it is proved that all systems that are in compliance with a refinement will also be in compliance with the original interaction.

- **The compliance relation between interactions and computer systems should be independent of technology used for the computer system.**

  Technology independence is achieved by assuming that a computer system is given by its set of traces, and then translating this trace-set into the semantics model of STAIRS.

- **The methodology should be conservative, i.e. based on existing UML methodology.**

  Within the limits of this thesis, there has not been room for creating a thorough methodology based on STAIRS. What we have, is a number of pragmatic guidelines for creating and refining UML 2.x interactions (see paper 5). In paper 8 we give some initial ideas of how these may be used together with existing development methodologies such as e.g. RUP.

- **The methodology should be useful without thorough knowledge of the formal definitions.**

  The guidelines in paper 5 were proven useful in the case study in paper 8. However, they are not complete, and more guidelines must be defined before the principles of STAIRS may be applied to system development by people without knowledge of the formal definitions.

7.3 Generalization of the Results

In our work, we have focused exclusively on UML 2.x interactions in the form of sequence diagrams and interaction overview diagrams. This means that we have only studied semantics and refinement with respect to dynamic behaviour, and not static structure. The static structure diagrams of UML 2.x, such as class diagrams and object diagrams, are also partial descriptions in the same manner as interactions, and the distinction between positive (i.e. possible or valid), inconclusive and negative (i.e. impossible or invalid) may be useful also for such description techniques. However, it is not common to talk about negation in this setting, and it is not obvious how the results of STAIRS may be used for these diagram types.

The interaction diagrams of UML 2.x include also communication diagrams and timing diagrams. These have not been explicitly treated in our work, but it is straightforward to use these diagram types within the context of STAIRS. As with sequence diagrams, communication diagrams specify interaction between objects. The di erence between the two is that while sequence diagrams focus on the ordering of events, communication diagrams focus on the relationships between the objects. However, for the behavioural description, any communication diagram may be translated into an equivalent sequence diagram. Similarly, timing diagrams are a special kind of sequence diagrams, explicitly showing the time ticks and also state changes in the lifelines.

The main principles of STAIRS generalize nicely to other approaches for behavioural specification, both within and outside UML. The exact semantics must of course be defined in each

case, but the basic semantic model consisting of several interaction obligations, each distinguishing between positive, inconclusive and negative behaviour, should be applicable for a wide range of languages. In particular, our results with respect to refinement and compliance are applicable for other behavioural specifications as soon as the basic understanding of their semantics is obtained. Usually, only a simpler notion of refinement is needed, as most behavioural description techniques are seen as giving complete specifications where the set of inconclusive behaviour is empty. In those cases, the most interesting contribution of STAIRS is the distinction between mandatory and potential behaviour, and how this di erence is treated in the semantic model and by the refinement relations.

7.4 Related Work

In this section we discuss closely related work on UML 2.x interactions. For a general treatment of related work, such as other kinds of sequence diagrams and work on the di erent kinds of nondeterminism, we refer to chapter 4 and the sections on related work in the attached papers.

7.4.1 Harald Störrle: Trace Semantics of Interactions in UML 2.0

In [Stö04], Harald Störrle defines trace semantics for interactions, based on the UML 2.0 standard [OMG04]. The basic concepts are very similar to those in STAIRS, and the two approaches mainly agree with respect to the definitions of positive behaviour. Neither approach treats extra global combined fragments, but Störrle defines semantics for operators not covered by STAIRS, including consider, ignore and variants of break and critical. Guards are not treated in [Stö04], while time is treated in a manner very similar to that of STAIRS.

Positive behaviours are in [Stö04] interpreted as *must*, similar to our xalt operator and analogous to negative behaviours being interpreted as *must not* (in both approaches). Underspecification is not treated in [Stö04].

For negative behaviours, there are also interesting di erences. STAIRS and [Stö04] mainly agree on the definition of assert, where the only positive traces are the positive traces of the operand. The di erence between the two definitions is that if a trace is both positive and negative in the operand, the trace will remain both positive and negative in STAIRS, whereas it becomes only positive in [Stö04].

With respect to neg, Störrle investigates three alternative definitions, but neither of these corresponds to the STAIRS definitions of refuse and veto. For all of the alternatives in [Stö04], the negative traces of neg are the positive traces of its operand (and not also the negative behaviours as in STAIRS). The di erence between the three alternatives is with respect to what constitutes the positive behaviours. The first alternative in [Stö04], loose negate, is similar to our refuse in that the set of positive behaviours are empty. With this definition, the negative traces of the operand is "lost", i.e. they become inconclusive. Therefore, a second alternative, strict negate, is proposed for which all traces that are not negative in the operand are taken as positive (i.e. neg is taken as the opposite of assert). The third alternative, flip negate, takes the negative traces of the operand as positive, corresponding to negation in classical logic. This alternative seems to be the one favoured by Störrle. Our reasons for not choosing this definition are given in paper 3.

[Stö04] also discusses some problems of combining neg with other operators. For instance,

an interaction such as seq $[d_1, \text{neg } [d_2]]$ intuitively means that the behaviours of d_2 should *not* follow the positive behaviours of d_1. However, using the definitions in [Stö04], the only negative behaviours of this interaction is the *negative* behaviours of d_1 sequenced with the negative behaviours of d_2. These problems lead Störrle to conclude that "maybe, the concept of negative traces is not such a good idea after all". We do not agree with this statement, and the formal definitions in STAIRS do not have similar problems as the ones discussed in [Stö04].

For refinement, [Stö04] defines refinement of traces, refinement of interactions, and refinement of time constraints. A single trace may be refined by adding new events to it, or replacing coarse-grained events by sequences of fine-grained events. This may be seen as a generalization of detailing refinement in STAIRS. For refining interactions, Störrle only considers a notion similar to STAIRS supplementing, as there are no underspecification in the interactions. Finally, a time constraint may be refined by narrowing the valid timestamps for the events constrained by it. In STAIRS, this is captured by the general definition of narrowing.

7.4.2 María Victoria Cengarle and Alexander Knapp: UML 2.0 Interactions — Semantics and Refinement

Another trace-based formalization of UML 2.0 interactions is the work by Cengarle and Knapp in [CK04]. The operators consider and ignore are treated, but not break or critical as in [Stö04]. Extra global combined fragments are not treated, neither are time nor guards.

For the positive traces of an interaction, the semantics in [CK04] mainly coincides with that in [Stö04] (and STAIRS), except that having several positive traces is interpreted as underspecification, similar to alt in STAIRS. Also, the empty trace is taken as the only positive trace for the interaction neg $[d]$, similar to our veto operator.

For the negative traces of an interaction, there are several di erences between STAIRS and the approach in [CK04]. The definitions of assert are similar, but for negation, Cengarle and Knapp takes only the positive traces of the operand as negative, similar to Störrle. For alt, a trace is taken as negative only if it is negative in both operands.

Cengarle and Knapp pose an interesting question with respect to the use of negation in specifications: Should a trace be negative if a prefix of it is specified as negative? Their answer is essentially yes, proposing an even stronger approach where a trace is taken as negative as soon as it has completed a negative sub-interaction. An advantage of this is that it allows for earlier identification (or even prevention) of negative traces. With this approach, there will typically be many traces that are negative for an interaction even though the traces are not explicitly described in the diagram. In STAIRS, we follow our main principles as stated in section 5.2.2 and regard these traces as inconclusive. For a further discussion of these alternative approaches, see the discussion of related work in paper 3.

For Cengarle and Knapp, a valid implementation of an interaction must show at least one positive trace and no negative traces. This is similar to the STAIRS notion of restricted compliance (see paper 7), which is indeed inspired by the notion in [CK04].

The refinement notion in [CK04] is based on a model-theoretic view, and states simply that one interaction is a refinement of another interaction if all valid implementations of the refinement are also valid implementations of the original interaction. This implies that at least one of the positive traces of the initial specification must remain positive during all of the develop-

ment process, and that supplementing inconclusive traces as positive is not allowed (similar to our notion of restricted refinement). In contrast to all of the refinement relations in STAIRS, refinement in [CK04] allows positive traces to become inconclusive.

A major disadvantage with the refinement relation in [CK04] is that it does not give monotonicity for all of the composition operators. In an attempt to remedy this, Cengarle and Knapp define a restricted refinement notion, positive refinement, where the set of positive traces are kept unchanged during refinement (i.e. the only possible refinement step is supplementing traces as negative). This refinement notion gives monotonicity for all operators under the assumption that all refinements are implementable and does not contain the same trace as both positive and negative. With this refinement notion, an implementation may still remove underspecification by implementing only some of the positive traces, but refinement can no longer be used to resolve this at the specification level.

7.4.3 Other Work on UML 2.x Interactions

Other recent work on UML 2.x interaction include [GS05], [EFM+05] and [HM06]. As these deviate more from the UML 2.x standard, they are treated here with less detail than the two approaches above.

[GS05] interprets positive and negative interactions as specifying liveness and safety properties, respectively. This is a much stronger interpretation than the traditional use of sequence diagrams for illustrating example runs. Based on a large amount of transformation, [GS05] then defines the semantics of interactions in the form of two Büchi automata, one for the positive and one for the negative behaviour. Refinement is defined as language inclusion, and is monotonic with respect to the most common composition operators.

[EFM+05] uses Petri Nets to give semantics to UML 2.x interactions. However, the approach deviates from UML on important points. First of all, it is assumed that all possible behaviours are explicitly described in the diagram (i.e. there are no inconclusive behaviours), meaning that operators such as assert and neg are not defined as there is no need for specifying negative behaviours. Also, in contrast to the UML 2.x standard, synchronization between all lifelines are assumed at the beginning of each sub-interaction. Still, [EFM+05] is interesting as it includes aspects of sequence diagrams not treated in most other formal approaches, including lost and found messages, and the creation and destruction of lifelines. Data in sequence diagrams is also covered, but not time. No notion of refinement is included in [EFM+05].

[HM06] argues that the assert and neg operators are insu cient for specifying required and forbidden behaviours, and proposes Modal UML Sequence Diagrams (MUSD) as an extension to UML 2.x sequence diagrams. Based on LSC ([DH01, HM03], see section 4.6), MUSD allows sequence diagram elements to be specified as either hot (universal) or cold (existential). With this approach, assert is interpreted as specifying that all of the events in the operand should be hot, while neg is interpreted as if the condition *false* was added immediately after the last event(s) in the operand. This interpretation of neg leads to the same approach as that proposed in [CK04], where a trace is negative as soon as it has completely traversed a negative (sub-)interaction.

Chapter 8

Future Work

As mentioned in chapter 1 this work has already been used as a basis for other work on UML 2.x interactions including test case generation [LS06a], probabilistic sequence diagrams [RHS05, RRS06] and for defining secure information flow [SS06]. Also, we have been involved in work on extending sequence diagrams with time exceptions for handling violation of time constraints [HRH07].

This work may also be extended in several other directions, some of which are hinted at in sections 7.2 and 7.3. One possible direction is continuing the work on providing a formal semantics for UML 2.x interactions. Although we have covered what we believe is the most commonly used parts of interactions, interesting aspects such as critical region and extra global combined fragments are not treated, and their formalization is not entirely obvious. In section 3 we identified a need for new or improved UML tools based on a precise semantics for UML 2.x. Providing a semantics for interactions is only the first step. Covering all of UML 2.x is a large and complex work, which is currently being addressed by the UML 2 semantics project [BCD+07].

A more interesting direction is extending our work on refinement, both with respect to the formal definitions and with respect to the pragmatic guidelines explaining them. Formally, we believe that we have covered the interesting variations of behavioural refinement (i.e. supplementing, narrowing and restricted refinement), but more work is needed with respect to message refinement. As a step towards defining more pragmatical guidelines for refinement, developing a formal refinement calculus for STAIRS would be very useful. Also, guidelines could be developed with respect to when the di erent refinement and compliance notions are most successfully used in practical system development.

For relating di erent specifications of the same system there is also the notion of viewpoint correspondences. For a simple notion of viewpoint consistency, we may use STAIRS and say that two specifications are consistent if there exists a specification that is a valid refinement of both of the original specifications. However, viewpoint correspondences involves more than just simple consistency. We believe that STAIRS provides a useful basis for investigating these issues, but much more work is needed here.

Bibliography

[ABB+02] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, and Jörg Zettel. *Component-based Product Line Engineering with UML*. Addison-Wesley, 2002.

[AH92] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Real-Time: Theory in Practice*, volume 600 of *LNCS*, pages 74–106. Springer, 1992.

[AL91] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[Amb02] Scott W. Ambler. *Agile Modeling: E ective Practices for eXtreme Programming and the Unified Process*. Wiley, 2002.

[BCD+07] Manfred Broy, Michelle L. Crane, Juergen Dingel, Alan Hartman, Bernhard Rumpe, and Bran Selic. 2^{nd} UML 2 semantics symposium: Formal semantics for UML. In *Models in Software Engineering*, volume 4364 of *LNCS*, pages 318–323. Springer, 2007.

[BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems :* F *on Streams, Interfaces, and Refinement*. Springer, 2001.

[BvW98] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction*. Springer, 1998.

[CK04] María Victoria Cengarle and Alexander Knapp. UML 2.0 interactions: Semantics and refinement. In *Proc. Critical Systems Development with UML (CSDUML'04)*, Technical report TUM-I0415, pages 85–99. Institut für Informatik, Technische Universität München, 2004.

[Col93] Pierre Collette. Application of the composition principle to unity-like specifications. In *Proc. TAPSOFT'93: Theory and Practice of Software Development*, volume 668 of *LNCS*, pages 230–242. Springer, 1993.

[DC02] Gordana Dodig-Crnkovic. Scientific methods in computer science. In *Proc. Conf. for the promotion of research in IT at new universites and at university colleges in Sweden*, 2002.

[DDH72] Ole-Johan Dahl, Edsger W. Dijkstra, and C. A. R. Hoare. *Structured Programming*. Academic Press, 1972.

[Den05] Peter J. Denning. Is computer science science? *Communications of the ACM*, 48(4):27–31, 2005.

[DH99] Werner Damm and David Harel. LSCs: Breathing life into Message Sequence Charts. In *Proc. Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, pages 293–312. Kluwer, 1999.

[DH01] Werner Damm and David Harel. LSCs: Breathing life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001. A preliminary version of this paper appeared as [DH99].

[Dij68] Edsger W. Dijkstra. Stepwise program construction. Published as [Dij82], February 1968.

[Dij82] Edsger W. Dijkstra. Stepwise program construction. In *Selected Writings on Computing: A Personal Perspective*, pages 1–14. Springer, 1982.

[dJvdPH00] Edwin de Jong, Jaco van de Pol, and Jozef Hooman. Refinement in requirements specification and analysis: A case study. In *Proc. Engineering of Computer Based Systems (ECBS 2000)*, pages 290–298. IEEE Computer Society, 2000.

[dR85] Willem-Paul de Roever. The quest for compositionality — A survey of assertion-based proof systems for concurrent programs. Part 1: Concurrency based on shared variables. Technical Report RUU-CS-85-2, Department of Computer Science, University of Utrecht, 1985.

[dRdBH+01] Willem-Paul de Roever, Frank de Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency verification: Introduction to Compositional and Noncompositional Methods*. Cambridge University Press, 2001.

[dRE98] Willem-Paul de Roever and Kai Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, 1998.

[DW99] Desmond Francis D'Souza and Alan Cameron Wills. *Objects, Components, and Frameworks with UML: the Catalysis Approach*. Addison-Wesley, 1999.

[EFM+05] Christoph Eichner, Hans Fleischhack, Roland Meyer, Ulrik Schrimpf, and Christian Stehno. Compositional semantics for UML 2.0 sequence diagrams using Petri Nets. In *Proc. SDL 2005: Model Driven Systems Design*, volume 3530 of *LNCS*, pages 133–148. Springer, 2005.

[Fow03] Martin Fowler. Uml bliki: UmlMode, May 2003. http://www.martinfowler.com/bliki/UmlMode.html.

[GHK99] Günter Graw, Peter Herrmann, and Heiko Krumm. Constraint-oriented formal modelling of OO-systems. In *Proc. Distributed Applications and Interoperable Systems (DAIS 99)*, pages 345–358. Kluwer, 1999.

[Gla95] Robert L. Glass. A structure-based critique of contemporary computing research. *Journal of Systems and Software*, 28(1):3–7, 1995.

[GS05] Radu Grosu and Scott A. Smolka. Safety-liveness semantics for UML 2.0 sequence diagrams. In *Proc. Applications of Concurrency to System Design (ACSD'05)*, pages 6–14. IEEE Computer Society, 2005.

[Hau97] Øystein Haugen. The MSC-96 distillery. In *SDL'97: Time for Testing: SDL, MSC and Trends*. Elsevier, 1997.

[Hau04] Øystein Haugen. Comparing UML 2.0 interactions and MSC-2000. In *Proc. System Analysis and Modeling (SAM 2004)*, volume 3319 of *LNCS*, pages 65–79. Springer, 2004.

[HM03] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.

[HM06] David Harel and Shahar Maoz. Assert and negate revisited: Modal semantics for UML sequence diagrams. In *Proc. Scenarios and State Machines: Models, Algorithms, and Tools (SCESM'06)*, pages 13–20. ACM, 2006.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[Hoa72] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HRH07] Oddleif Halvorsen, Ragnhild Kobro Runde, and Øystein Haugen. Time exceptions in sequence diagrams. In *Models in Software Engineering*, volume 4364 of *LNCS*, pages 131–142. Springer, 2007.

[HS03] Øystein Haugen and Ketil Stølen. STAIRS — Steps to analyze interactions with refinement semantics. In *Proc. "UML 2003" — The Unified Modeling Language: Modeling Languages and Applications*, volume 2863 of *LNCS*, pages 388–402. Springer, 2003.

[ISO89] International Standards Organization. *Information Processing Systems – Open Systems Interconnection – LOTOS – a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour – ISO 8807*, 1989.

[ITU96] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1996.

[ITU98] International Telecommunication Union. *Recommendation Z.120 — Annex B: Formal semantics of Message Sequence Charts*, 1998.

[ITU99] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1999.

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[Jon72] Cli B. Jones. Formal development of correct algorithms: An example based on Earley's recogniser. *ACM SIGPLAN Notices*, 7(1):150–169, 1972.

[Jür01] Jan Jürjens. Secrecy-preserving refinement. In *Proc. FME 2001: Formal Methods for Increasing Software Productivity*, volume 2021 of *LNCS*, pages 135–152. Springer, 2001.

[Jür02] Jan Jürjens. *Principles for Secure Systems Design*. PhD thesis, Oxford University Computing Laboratory, 2002.

[Jür05] Jan Jürjens. *Secure Systems Development with UML*. Springer, 2005.

[Kna99] Alexander Knapp. A formal semantics for UML interactions. In *Proc. "UML"'99: The Unified Modeling Language: Beyond the Standard*, volume 1723 of *LNCS*, pages 116–130. Springer, 1999.

[Krü00] Ingolf Heiko Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Institut für Informatik, Technische Universität München, 2000.

[Kru04] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, third edition, 2004.

[Lam02] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[Lam05] Leslie Lamport. Real time is really simple. Technical Report MSR-TR-2005-30, Microsoft Research, 2005.

[LAMB89] Peter Gorm Larsen, Michael Meincke Arentoft, Brian Q. Monahan, and Stephen Bear. Towards a formal semantics of the BSI/VDM specification language. In *Information processing 89: Proc. IFIP 11th World Computer Congress*, pages 95–100. Elsevier, 1989.

[LS06a] Mass Soldal Lund and Ketil Stølen. Deriving tests from UML 2.0 sequence diagrams with neg and assert. In *Proc. 1st International Workshop on Automation of Software Test (AST'06)*, pages 22–28. ACM Press, 2006.

[LS06b] Mass Soldal Lund and Ketil Stølen. A fully general operational semantics for UML 2.0 sequence diagrams with potential and mandatory choice. In *Proc. FM 2006: Formal Methods*, volume 4085 of *LNCS*, pages 380–395. Springer, 2006.

[MBD00] Ralph Miarka, Eerke Boiten, and John Derrick. Guards, preconditions, and refinement in Z. In *Proc. ZB 2000: Formal Specification and Development in Z and B*, volume 1878 of *LNCS*, pages 286–303. Springer, 2000.

[McG84] Joseph Edward McGrath. *Groups: Interaction and Performance*. Prentice-Hall, 1984.

[MS00] David Meier and Beverly Sanders. Composing leads-to properties. *Theoretical Computer Science*, 243(1-2):339–361, 2000.

[OMG03a] Object Management Group. *U2 Partners' UML 2.0 Superstructure Specification*, document: ad/03-04-01 edition, 2003.

[OMG03b] Object Management Group. *UML 1.5 Specification*, document: formal/03-03-01 edition, 2003.

[OMG04] Object Management Group. *UML 2.0 Superstructure Specification*, document: ptc/04-10-02 edition, 2004.

[OMG05] Object Management Group. *UML 2.0 Superstructure Specification*, document: ptc/05-07-04 edition, 2005.

[OMG06] Object Management Group. *UML 2.1 Superstructure Specification*, document: ptc/06-04-02 edition, 2006.

[Öve99] Gunnar Övergaard. A formal approach to collaborations in the Unified Modeling Language. In *Proc. "UML"'99: The Unified Modeling Language: Beyond the Standard*, volume 1723 of *LNCS*, pages 99–115. Springer, 1999.

[PP05] Dan Pilone and Neil Pitman. *UML 2.0 in a Nutshell*. O'Reilly, 2005.

[Pri00] Andreas Prinz. Formal semantics of specification languages. *Telektronikk*, (4), 2000.

[RHS05] Atle Refsdal, Knut Eilif Husa, and Ketil Stølen. Specification and refinement of soft real-time requirements using sequence diagrams. In *Proc. Formal Modeling and Analysis of Timed Systems (FORMATS 2005)*, volume 3829 of *LNCS*, pages 32–48. Springer, 2005.

[Ros95] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. 1995 IEEE Symposium on Security and Privacy*, pages 114–127. IEEE Computer Society, 1995.

[Ros98] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1998.

[RRS06] Atle Refsdal, Ragnhild Kobro Runde, and Ketil Stølen. Underspecification, inherent nondeterminism and probability in sequence diagrams. In *Proc. Formal Methods for Open Object-Based Distributed Systems (FMOODS 2006)*, volume 4037 of *LNCS*, pages 138–155. Springer, 2006.

[SAR] The SARDAS project: Securing availability by robust design, assessment and specification. http://heim.ifi.uio.no/~ketils/sardas/sardas.htm.

[SBDB97] Maarten Steen, Howard Bowman, John Derrick, and Eerke Boiten. Disjunction of LOTOS specifications. In *Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE X / PSTV XVII '97)*, pages 177–192. Chapman & Hall, 1997.

[Sch96] David A. Schmidt. Programming language semantics. *ACM Computing Surveys*, 28(1):265–267, 1996.

[Sel04] Bran Selic. On the semantic foundations of standard UML 2.0. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *LNCS*, pages 181–199. Springer, 2004.

[SS06] Fredrik Seehusen and Ketil Stølen. Information flow property preserving transformation of UML interaction diagrams. In *Proc. Symposium on Access Control Models and Technologies (SACMAT 2006)*, pages 150–159. ACM, 2006.

[SS07] Ida Solheim and Ketil Stølen. Technology research explained. Technical Report A313, SINTEF ICT, January 2007.

[Stö04] Harald Störrle. Trace semantics of interactions in UML 2.0. Technical Report TR 0403, University of Munich, 2004.

[Whi02] Jon Whittle. Formal approaches to systems analysis using UML: An overview. In *Advanced Topics in Database Research, Vol. 1*, pages 324–341. Idea Group, 2002.

[Wir71] Niklaus Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, 1971.

[WM01] M. Walicki and S. Meldal. Nondeterminism vs. underspecification. In *Proc. Systemics, Cybernetics and Informatics (ISAS-SCI 2001)*, pages 551–555. IIIS, 2001.