Software & System Modeling (2005) 00: 1–13
DOI 10.1007/s10270-005-0087-0

SPECIAL ISSUE PAPER

Øystein Haugen · Knut Eilif Husa ·
Ragnhild Kobro Runde · Ketil Stølen

STAIRS towards formal design with sequence diagrams

Received: date / Revised version: date / Published online: 15 June 2005

Abstract The paper presents STAIRS [1], an approach to the compositional development of UML interactions supporting the specification of mandatory as well as potential behavior. STAIRS has been designed to facilitate the use of interactions for requirement capture as well as test specification. STAIRS assigns a precise interpretation to the various steps in incremental system development based on an approach to refinement known from the field of formal methods and provides thereby a foundation for compositional analysis. An interaction may characterize three main kinds of traces. A trace may be (1) positive in the sense that it is valid, legal or desirable, (2) negative meaning that it is invalid, illegal or undesirable, or (3) inconclusive meaning that it is considered irrelevant for the interaction in question. The basic increments in system development proposed by STAIRS, are structured into three main kinds referred to as supplementing, narrowing and detailing. Supplementing categorizes inconclusive traces as either positive or negative. Narrowing reduces the set of positive traces to capture new design decisions or to match the problem more adequately. Detailing involves introducing a more detailed description without significantly altering the externally observable behavior.

Keywords UML interactions · Formal semantics · Explicit non-determinism · Refinement · Sequence diagrams

Ø. Haugen
Department of Informatics, University of Oslo
E-mail: oysteinh@ifi.uio.no

K. E. Husa
Department of Informatics, University of Oslo and Ericsson
E-mail: knutehu@ifi.uio.no

R. K. Runde
Department of Informatics, University of Oslo
E-mail: ragnhilk@ifi.uio.no

K. Stølen (✉)
Department of Informatics, University of Oslo and SINTEF ICT, Norway
E-mail: Ketil.Stolen@sintef.no

1 Introduction

A UML interaction is a specification of how messages are sent between objects or other instances to perform a task. Interactions are used in a number of different situations. They are used to get a better grip of a communication scenario for an individual designer or for a group that needs to achieve a common understanding of the situation. Interactions are also used during the more detailed design phase where the precise inter-process communication must be set up according to formal protocols. When testing is performed, the behavior of the system can be described as interactions and compared with those of the earlier phases.

Interactions seem to have the ability to be understood and produced by professionals of computer systems design as well as potential end-users and stakeholders of the (future) systems.

Interactions will typically not tell the complete story. There are normally other legal and possible behaviors that are not contained within the described interactions. Some people find this disturbing and some project leaders have even tried to request that all possible behaviors of a system should be documented through interactions in the form of e.g. sequence diagrams or similar notations.

Our position is that UML interactions are expressed through notations that lend themselves well to conveying important information about the interplay between objects, but interactions are not so well suited to define the complete behavior.

Partial information is not worthless because it is incomplete. Most statements about a system are partial in their nature. The informal statement "When pushing the ON-button on the television, the television should show a program" is definitely not a complete definition of a television set, but it is a piece of requirement that the TV vendors should take into account. The same can be said for interactions; they are partial statements about a system (or a part of a system) defining properties of the system, but not necessarily all relevant properties.

This paper advocates an approach, in the following referred to as STAIRS, aiming to provide a formal foundation for the use of UML interactions in step-wise, incremental system development. STAIRS views the process of developing the interactions as a process of learning through describing. From a fuzzy, rough sketch, the aim is to reach a precise and detailed description applicable for formal handling. To come from the rough and fuzzy to the precise and detailed, STAIRS distinguishes between three main sub-activities: (1) supplementing, (2) narrowing and (3) detailing.

Supplementing categorizes (to this point) inconclusive behavior as either positive or negative recognizing that early descriptions normally lack completeness. The initial requirements concentrate on the most obvious normal situations and the most obvious exceptional ones. Supplementing supports this by allowing less obvious situations to be treated later. Narrowing means reducing the allowed behavior to match the problem better. Detailing involves introducing a more detailed description without significantly altering the externally observable behavior.

Although the starting point for STAIRS is UML interactions in shape of UML 2.0 sequence diagrams, the approach should be considered generic to all types of behaviors.

The remainder of the paper is structured into seven sections. Section 2 provides further motivation and background in the form of requirements we would like STAIRS to fulfill. Section 3 explains how STAIRS meets these requirements. Sections 4–6 spell out STAIRS in an example-driven manner addressing respectively the notions of supplementing, narrowing and detailing. Section 7 describes the formal semantics of STAIRS. Section 8 provides a brief summary and relates STAIRS to approaches known from the literature.

2 Requirements to STAIRS

In order to explain its overall structure and architecture, we formulate and motivate a number of requirements that STAIRS has been designed to fulfill.

1. Should allow specification of potential behavior. Under-specification is a well-known feature of abstraction. In the context of interactions, "under-specification" means specifying several behaviors, each representing a potential alternative serving the same purpose, and that fulfilling only some of them (more than zero but not all) is acceptable for an implementation to be correct.
2. Should allow specification of mandatory behavior. Under-specification as described in the previous paragraph gives rise to non-determinism in the specification. Under-specification allows the system developer to choose between several potential behaviors. Sometimes, however, it is essential to retain non-determinism in the implementation reflecting choice. For example, in a lottery, it is critical that every lottery ticket has the possibility to win the prizes. Otherwise, the lottery is not fair. This means that every behavior given by the different tickets should appear as possibilities in an implementation even though in any given execution only a few prizes are awarded. It seems unproblematic to reduce non-determinism if the different alternatives represent implementation dependent variations of the same behavior. It is quite different, however, to reduce non-determinism if each alternative represents a distinct and intended behavior. As a consequence, we need to distinguish explicit non-determinism capturing mandatory behavior from non-determinism expressing potential behavior.
3. Should allow specification of negative behavior in addition to positive behavior. Interactions are not only suited to capture system requirements. They may just as well describe illegal or undesirable behavior. For example, security is a major issue in most modern systems. To identify security requirements, risk analysis is a well-known measure. To be able to assign risk values to risks we need a clear understanding of the circumstances under which the risks may appear, and the impact they may have on system assets. Interactions are well suited to describe this kind of threat scenario. Hence, we need an integrated approach to specifying negative as well as positive behavior.
4. Should capture the notion of refinement. The notion of refinement was formalized in the early 1970s [2–4], and has since then been thoroughly investigated within numerous so-called formal methods. STAIRS should build on this theory, but the theory must be adapted to take into account that interactions may be partial, describe positive as well as negative situations, and may be used to formalize both potential and mandatory behavior.
5. Should formalize aspects of incremental development. Incremental development of interactions involves various sub-activities as described informally in the introduction. STAIRS should provide precise and intuitive definitions of these activities.
6. Should support compositional analysis, verification and testing. Models are of little help if they cannot be used as a basis for analysis, verification and testing. STAIRS should provide a foundation for these activities facilitating compositionality [5, 6] in the sense that components can be developed independently from their specifications.

3 How STAIRS meets the requirements

The most visible aspects of a UML interaction are the messages between the lifelines. The sequence of the messages is considered important for the understanding of the situation. The data that the messages convey may also be very important, but the interactions do not focus on the manipulation of data even though data can be used to decorate the diagrams.

The sequencing is the heart of what is explained through an interaction. The possible flows of control throughout the process are described in two dimensions, the horizontal dimension showing the different active objects, and the vertical dimension showing the ordering in time.

Interactions focus on the interplay between objects. In the tradition of telecommunications these objects are independent and themselves active as stand-alone processes. Therefore, when a message is sent from one lifeline to another, what happens on the sending lifeline is independent from what happens on the receiving side. The only invariant is that the sending of a message must occur before the reception of that very message. Most people find this obvious.

The sending of a message and the reception of a message are examples of what we call events. An event is something that happens on a lifeline at one point in time. An event has no duration.

A trace is a sequence of events ordered by time. A trace describes the history of message-exchange corresponding to a system run. A trace may be partial or total. Interactions may be timed in the sense that they contain explicit time constraints. Although STAIRS with some minor adjustments carry over to timed interactions (see [7]), such interactions are not treated in this paper.

3.1 Spelling out the trace semantics of UML 2.0

In this section we will give a very brief introduction to the trace semantics of UML 2.0 interactions expressed in sequence diagrams and interaction overview diagrams [8]. For more on UML 2.0, see also [9].

The interaction in Fig. 1 is almost the simplest interaction there is – only one message from one lifeline to another. Following our introduction above, this message has two events – the sending event on $L1$ (which we here choose to denote $!x$) and the reception event on $L2$ (which we choose to denote $?x$). The sending event must come before the receiving event and the semantics of this interaction is described by one single trace which we denote $\langle !x, ?x\rangle$.

The interaction of Fig. 2 shows two messages both originating from $L1$ and targeting $L2$. The order of the events on each lifeline is given by their vertical positions, but the two lifelines are independent. Each of the messages has the semantics given for the message in Fig. 1, and they are combined with what is called weak sequencing. Weak sequencing takes into account that $L1$ and $L2$ are independent. The weak sequencing operator on two interactions as operands is defined by the following invariants:

1. The ordering of events within each of the operands is maintained in the result.
2. Events on different lifelines from different operands may come in any order.

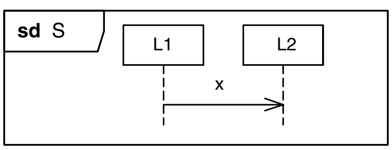

Fig. 1 Simple interaction with only one message

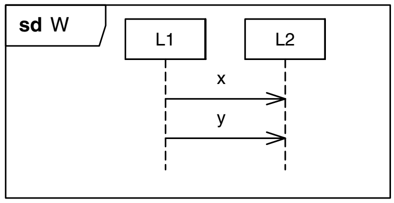

Fig. 2 Weak sequencing

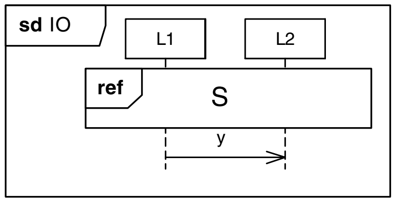

Fig. 3 Interaction occurrence

3. Events on the same lifeline from different operands are ordered such that an event of the first operand comes before that of the second operand.

Thus, if we denote the weak sequencing operator by seq according to UML 2.0, we get:

$$\begin{aligned} W &= \langle !x, ?x\rangle \, \mathsf{seq} \, \langle !y, ?y\rangle \\ &= \{\langle !x, ?x, !y, ?y\rangle, \langle !x, !y, ?x, ?y\rangle\} \end{aligned}$$

The sending of x must be the first event to happen, but after that either $L1$ may send y or $L2$ may receive x.

In Fig. 3 we show a construct called an interaction occurrence. The interaction S specified in Fig. 1 is referenced from within IO. Intuitively, an interaction occurrence is merely shorthand for the contents of the referenced interaction. Semantically we get that:

$$\begin{aligned} IO &= S \, \mathsf{seq} \, \langle !y, ?y\rangle \\ &= \langle !x, ?x\rangle \, \mathsf{seq} \, \langle !y, ?y\rangle \\ &= \{\langle !x, ?x, !y, ?y\rangle, \langle !x, !y, ?x, ?y\rangle\} \end{aligned}$$

In Fig. 4 we introduce another construct called combined fragment. Combined fragments are expressions of interactions combined differently according to which operator is used. In fact, also weak sequencing is such an operator. In Fig. 4 we have an alternative combined fragment, and its definition is simply the union of the traces of its operands. The dashed vertical line separates the operands. We get:

$$\begin{aligned} A &= \langle !x, ?x\rangle \, \mathsf{alt} \, \langle !y, ?y\rangle \\ &= \{\langle !x, ?x\rangle, \langle !y, ?y\rangle\} \end{aligned}$$

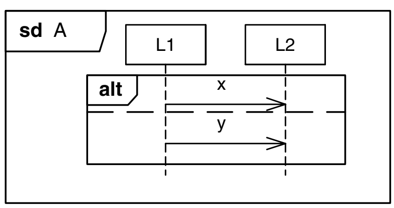

Fig. 4 Combined fragment (alternative)

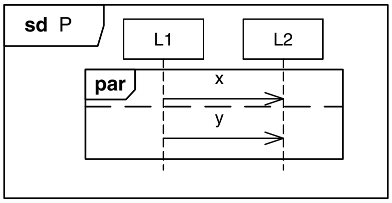

Fig. 5 Parallel combined fragment

UML 2.0 defines also a number of other operators, but in this paper we only apply one other, namely the parallel merge operator as depicted in Fig. 5. The definition of parallel merge says that a parallel merge defines a set of traces that describes all the ways that events of the operands may be interleaved without obstructing the order of the events within the operand. This gives the following traces for P:

$$\begin{aligned} P &= \langle !x, ?x\rangle \,\mathsf{par}\, \langle !y, ?y\rangle \\ &= \{\langle !x, ?x, !y, ?y\rangle, \langle !x, !y, ?x, ?y\rangle, \\ &\quad \langle !x, !y, ?y, ?x\rangle, \langle !y, ?y, !x, ?x\rangle, \\ &\quad \langle !y, !x, ?y, ?x\rangle, \langle !y, !x, ?x, ?y\rangle\} \end{aligned}$$

Finally we show the alternative syntax of interaction overview diagrams for these combined fragments. Figure 6 presents a more complicated interaction with the definition:

$$IOD = S\, seq\, (IO\, \mathsf{par}\, W)\, seq\, (IO\, \mathsf{alt}\, W)$$

where the diamond joins represent alternatives and the vertical bars represent parallel merge. We have not taken the time and space to calculate the explicit traces, but it is a mechanical task that only requires patience or tool support.

3.2 Capturing positive behavior

To illustrate our approach we use an everyday example that we hope seems intuitive. We describe the behavior of an

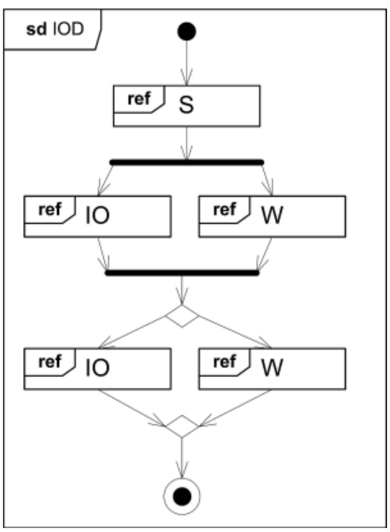

Fig. 6 Interaction overview diagram

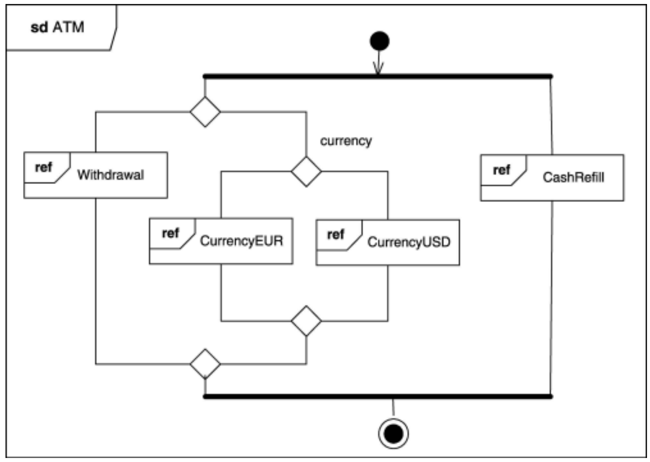

Fig. 7 Automatic teller machine

ATM (Automatic Teller Machine). The ATM offers withdrawal of native money or the purchase of a number of foreign currencies. We have specified euros (EUR) or US dollars (USD). The ATM must also have cash refill such that the customers can get what they order.

In an interaction overview diagram the behavior of the ATM may look like Fig. 7. To look into the details down to the events, we can follow e.g. Withdrawal as shown in Fig. 8.

Our withdrawal sequence diagram shows a trace that makes up some of the traces of the full ATM. Our withdrawal sequence is a positive one, one that is acceptable to the customers and as such desirable. It does not define all possible scenarios of a withdrawal of native money.

Each of the interaction occurrences in Fig. 7 represents a set of positive traces. The vertical bars between the withdrawal side and refill side represents a parallel merge combination meaning that all traces of the withdrawal are braided (interleaved) with every trace of the refill order. This must in practice be restricted, but that is not significant for the subject of this paper. The branching within the withdrawal side represents alternative choices and their combination is essentially a union of traces. Abbreviating the subsequences such that W stands for Withdrawal, E for CurrencyEUR, U for CurrencyUSD and C for CashRefill and then using the operators of UML 2.0 combined fragment, the following formula defines the positive traces:

$$ATM = ((W\, \mathsf{alt}\, (E\, \mathsf{alt}\, U))\, \mathsf{par}\, C)$$

To expand this to a set of traces, all the interactions referenced must be defined and the operations applied according to the UML 2.0 definition. We believe that the contained traces of the behavior of the ATM are intuitively understood.

Summary 1 *Semantically, each positive behavior is represented by a trace. Considering positive (potential) behavior only, the semantics of an interaction may be represented by a set of traces, each capturing a (potential) positive behavior.*

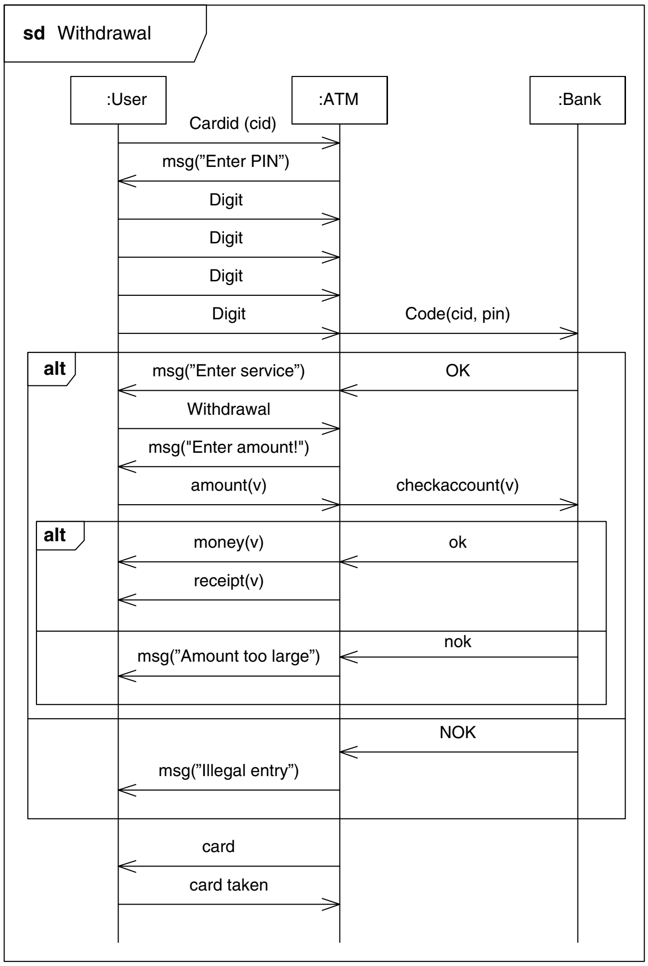

Fig. 8 Withdrawal (positive traces)

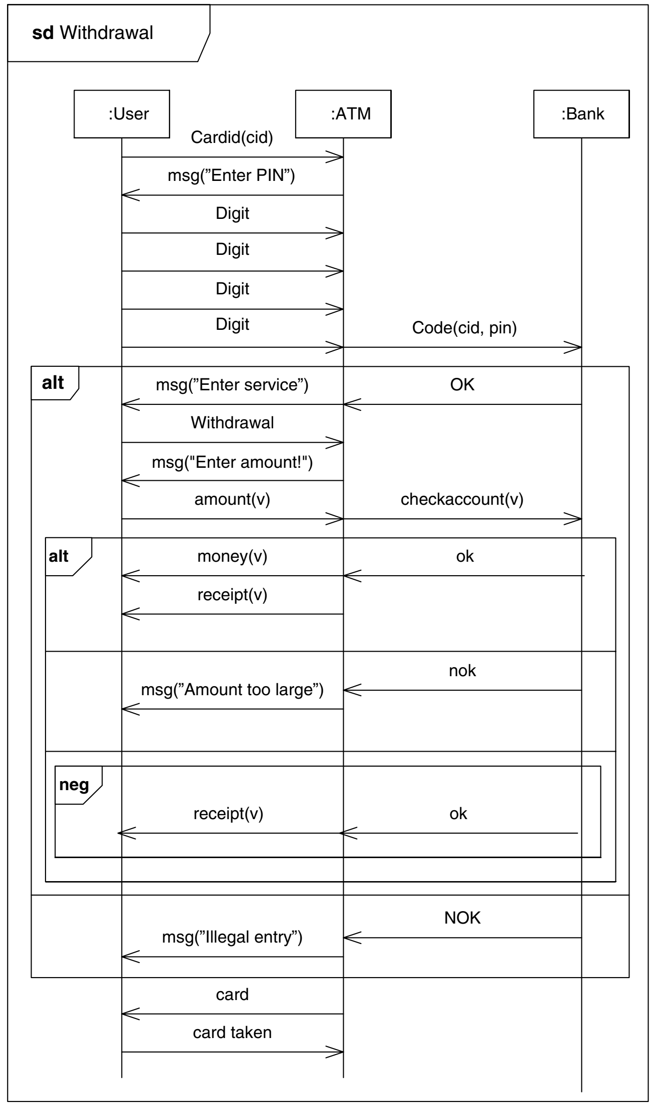

Fig. 9 Withdrawal with negative traces

3.3 Capturing negative behavior

In general the withdrawal procedure may include more than a single trace which is shown in Fig. 8 by the fact that there are alternative courses of the traces given by the alternative combined fragments. We have described both situations where the user is eligible for money and when his identification or funds are inadequate. These are all normal situations of an ATM and as such are considered "positive" as they will occur in an implementation.

On the other hand what should not be expected of an ATM is that the machine pretends to function correctly, but the user receives no money. In Fig. 9 we show a more elaborated scenario where this negative scenario has been included through a combined fragment with the operator neg.

This indicates that all traces in this fragment are "negative" or undesirable. In combination with other traces this negative fragment gives negative traces for every trace leading up to it. The diagram in Fig. 9 also defines a set of positive traces that just omit the negative fragment. This also includes the trace where the card is returned directly after the code has been entered. This trace is included because the neg fragment also introduces the empty but positive trace $\langle\rangle$.

In our example the intuition is simple; any trace that gives no money back when the receipt says it should have, is a negative scenario. The subtraces that will follow the negative fragment, the return of the card, certainly does not make the scenario less negative. Still we have not defined all possible scenarios of withdrawing money in an ATM. At this stage it is up to our imagination and the scope of our specification what cases we care to describe. It may or may not be relevant to specify what happens when the customer leaves the ATM without taking the card. Our diagram in Fig. 9 leaves that scenario inconclusive.

Summary 2 *Semantically, each negative behavior is represented by a trace. Ignoring mandatory behavior that is the issue for the next section, but considering both positive and negative behavior, the semantics of an interaction may be*

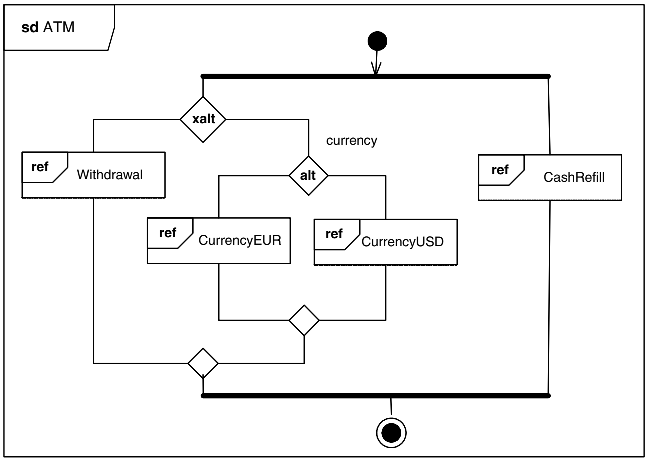

Fig. 10 Mandatory alternatives (xalt)

represented by a pair of sets (p, n) where n contains the negative traces and p contains the positive traces. The same trace should not be both positive and negative. Traces that are neither negative nor positive are inconclusive, i.e. considered irrelevant for the specification.

3.4 Distinguishing mandatory from potential behavior

Assume that we intend to use our ATM scenario as a requirement specification for purchasing ATMs. The question then becomes whether every ATM needs to be able to perform every positive trace. This would mean that every ATM must be able to offer both euros and US dollars. This would in some places be cumbersome and costly. Thus, this is not an adequate interpretation. On the other hand, we would like to convey that every ATM should offer withdrawal of native money. We specify that this is a mandatory requirement. We need a way to say that it is provisional whether both euros and US dollars are offered, but there is no choice not to offer withdrawal of native money. The latter distinction cannot be expressed directly by the operators of UML 2.0, but we have introduced a small extension and called this choice between alternatives that are mandatory as xalt. We have shown our modified specification in Fig. 10.

The semantics of ATM now can be described by an expression on the form:

$$ATM = ((W \; \mathsf{x}alt \; (E \; \mathsf{alt} \; U)) \; \mathsf{par} \; C)$$

In terms of our semantic model this is captured by traces residing in so called *interaction obligations* discriminated by the xalt operation. Thus, we have for ATM, two such interaction obligations, one for withdrawal and one for foreign currency. Any correct implementation must support both. The currency obligation may, however, be refined to support only euros or only US dollars.

Summary 3 *Semantically, we represent an interaction as a set of interaction obligations $O = \{o_1, \ldots, o_n\}$, where $o_j = (p_j, n_j)$, and p_j and n_j are the sets of positive and negative traces, respectively. An implementation satisfying the specification must fulfill each interaction obligation. Each interaction obligation represents potential variations of mandatory behavior that must be kept separate from other interaction obligations representing variations of other mandatory behavior. The traces within the same interaction obligation serve the same overall purpose.*

4 STAIRS spelled out: supplementing

Supplementing categorizes inconclusive traces as either positive or negative recognizing that early descriptions normally lack completeness. Supplementing supports the incremental process of requirements capture. The initial requirements concentrate on the most obvious normal situations and the most obvious exceptional ones. Supplementing supports this by allowing less obvious situations to be treated later. Hence, in the course of interaction development the overall picture may be filled in with more situations.

In our ATM example specified in Fig. 7, we may supplement the services by offering more kinds of foreign currency such as Danish kroner, or Japanese yen. We may likewise offer completely new services such as paying bills.

Furthermore, we may supplement the detailed production traces with more unwanted scenarios like when the user leaves without his card or forgets to key in the right number of digits in his personal identification number. We illustrate this with a Venn-diagram in Fig. 11 where the ovals and their subdivisions represents sets of traces. The traces of interest are explicitly named.

Summary 4 *Supplementing means reducing the set of inconclusive traces by defining more traces as either positive or negative. Any originally positive trace remains positive, and any originally negative trace remains negative.*

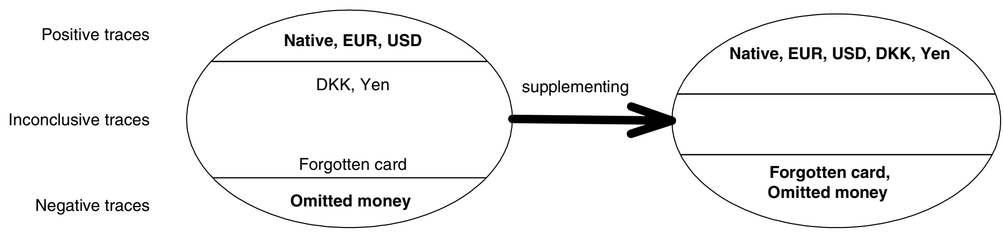

Fig. 11 Supplementing

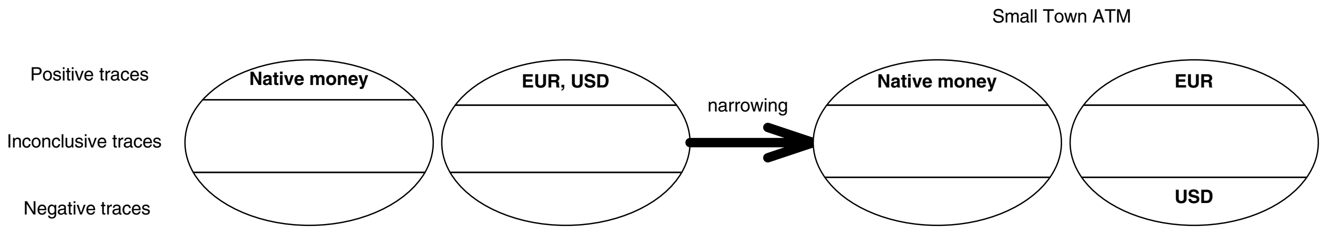

Fig. 12 Narrowing (small town ATM)

5 STAIRS spelled out: narrowing

When the designers have reached a description that they consider sufficiently complete, they will focus on making the descriptions suitable for implementation. Typically an implementation may decline to produce every positive potential trace. We define narrowing to mean reducing under-specification by eliminating positive traces without really changing the effect of the system.

Narrowing is a relation between descriptions such that the refined description has less variability/under-specification than the former. In our context of interactions, reducing the variability/under-specification means to move traces from the sets of positive traces to the set of negative. A narrowing cannot eliminate traces of the negative trace set since that would mean that some traces specified as illegal would suddenly be acceptable. This would be simply ignoring the specification. In our ATM example specified in Fig. 10, narrowing could mean that some ATMs would eliminate a subset of the possible foreign currencies. We illustrate this by a Venn-diagram in Fig. 12.

Summary 5 *Narrowing means, within one or more interaction obligations, reducing the set of positive traces, and at the same time, moving any trace deleted from the set of positive traces to the set of negative traces. Any inconclusive trace remains inconclusive and any negative trace remains negative.*

6 STAIRS spelled out: detailing

Detailing involves introducing a more detailed description without significantly altering the externally observable behavior. In Fig. 13 we have chosen to decompose the ATM as it actually consists of a number of components.

The resulting activity of the diagram in Fig. 13 is that of the :ATM lifeline of Fig. 8. This is done through the decomposition mechanism of UML 2.0. In addition we have detailed the outcoming result by specifying that the money is delivered in distinct notes. We show the simple message translation in the separate diagram. Clearly, the external behavior of ATM_Withdrawal is the same as the external behavior of :ATM in Fig. 8 given the Notes_Translator translation on the outcome. This shows how STAIRS supports the decomposition of the money message in Withdrawal into the different notes values messages in ATM_Withdrawal. The Notes_Translator diagram documents this decomposition and is in contrast to ATM_Withdrawal not supposed to be implemented.

Summary 6 *Detailing means that the sets of positive, negative and inconclusive traces are refined with respect to a translation between the more detailed and the given interaction.*

7 Formal foundation

Here, we demonstrate how the concepts in this paper may be formalized. For more details, we refer to [7].

7.1 Representing runs by traces

As explained in Sect. 3, a trace is a sequence of events, used to represent a system run. In each trace, a send event (tagged by an !) should always be ordered before the corresponding receive event (tagged by ?). We let H denote the set of all traces that complies with this requirement.

A message is a triple (s, tr, re) of a signal s, a transmitter tr, and a receiver re. M denotes the set of all messages. The transmitters and receivers are lifelines. L denotes the set of all lifelines. An event is a pair of kind and message

$$(k, m) \in \{!, ?\} \times M$$

E denotes the set of all events. We define the functions

$$k.\_ \in E \rightarrow \{!, ?\}, \quad tr.\_, re.\_ \in E \rightarrow L$$

to yield the kind, transmitter and receiver of an event, respectively.

For concatenation of sequences, filtering of sequences, and filtering of pairs of sequences, we have the functions $\frown$, Ⓢ, and Ⓣ, respectively.

Concatenating two sequences implies gluing them together. Hence, $a_1 \frown a_2$ denotes a sequence that equals a_1 if a_1 is infinite. Otherwise, $a_1 \frown a_2$ denotes a sequence that is prefixed by a_1 and suffixed by a_2. In both cases, the length of $a_1 \frown a_2$ is equal to the sum of the lengths of a_1 and a_2.

The filtering function Ⓢ is used to filter away elements. By B Ⓢ a we denote the sequence obtained from the sequence a by removing all elements in a that are not in the set of elements B. For example, we have that

$$\{1, 3\} \circledS \langle 1, 1, 2, 1, 3, 2\rangle = \langle 1, 1, 1, 3\rangle$$

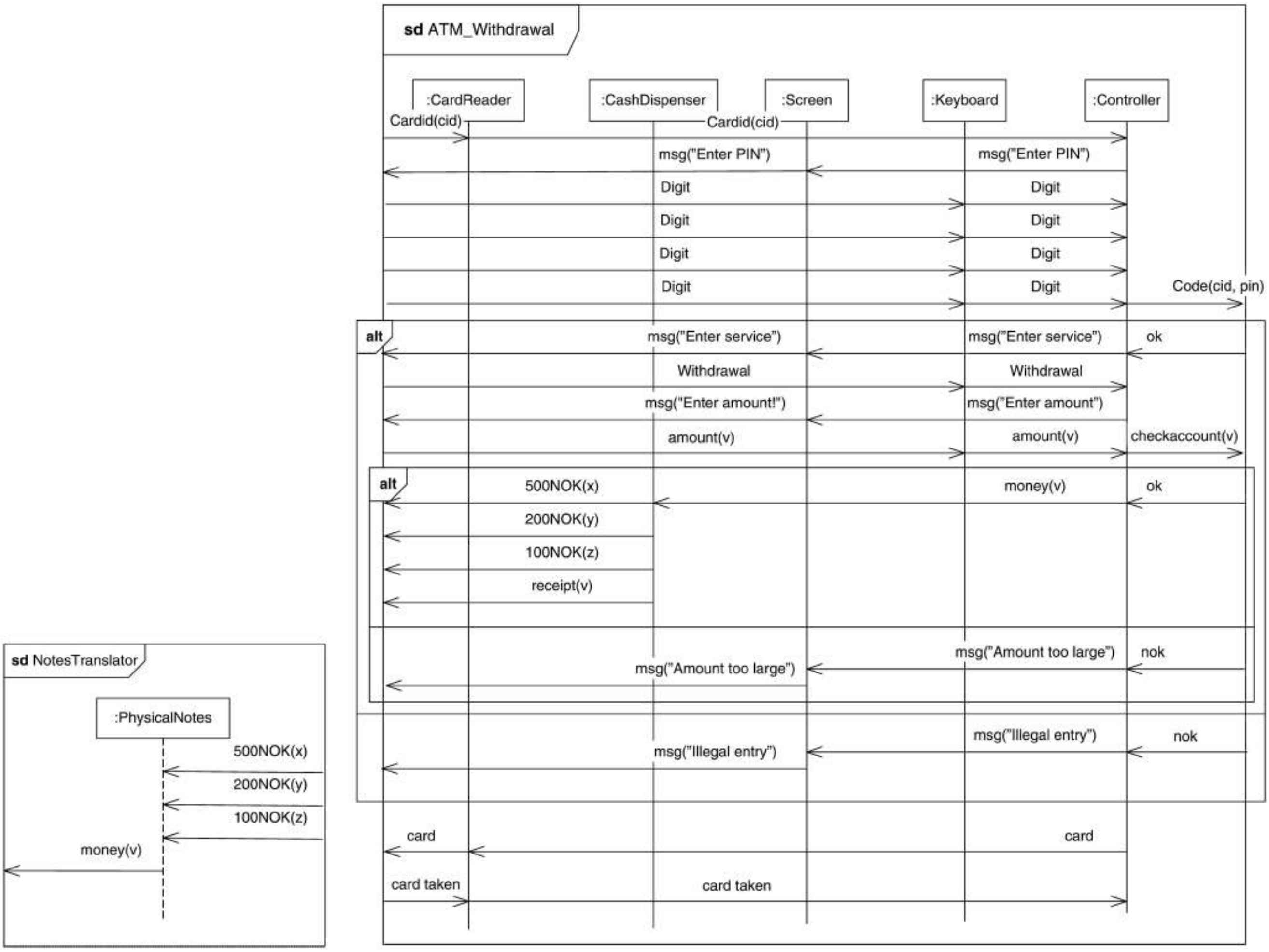

Fig. 13 Detailing the ATM

The filtering function $\textcircled{T}$ may be understood as a generalization of $\textcircled{S}$. The function $\textcircled{T}$ filters pairs of sequences with respect to pairs of elements in the same way as $\textcircled{S}$ filters sequences with respect to elements. For any set of pairs of elements P and pair of sequences t, by $P \textcircled{T} t$ we denote the pair of sequences obtained from t by

- truncating the longest sequence in t at the length of the shortest sequence in t if the two sequences are of unequal length;
- for each $j \in [1 \ldots k]$, where k is the length of the shortest sequence in t, selecting or deleting the two elements at index j in the two sequences, depending on whether the pair of these elements is in the set P.

For example, we have that

$$\{(1, f), (1, g)\} \textcircled{T} (\langle 1, 1, 2, 1, 2\rangle, \langle f, f, f, g, g\rangle) \\ = (\langle 1, 1, 1\rangle, \langle f, f, g\rangle)$$

7.2 Semantics of sequence diagrams

The semantics of sequence diagrams is defined by a function $[\![\;]\!]$ that for any sequence diagram d yields a set $[\![d]\!]$ of interaction obligations. As explained in Sect. 3.4, an interaction obligation is a pair (p, n) of sets of traces where the first set is interpreted as the set of positive traces and the second set is the set of negative traces. The term obligation is used to explicitly convey that any implementation of a specification is obliged to fulfill each specified alternative.

For a sequence diagram consisting of a single event e, its semantics is given by:

$$[\![e]\!] \stackrel{\text{def}}{=} \{(\{\langle e\rangle\}, \emptyset)\}$$

More complex sequence diagrams are constructed through the application of various operators. We focus on the operators that we find most essential, namely negation (neg), potential choice (alt), mandatory choice (xalt), parallel execution (par), and weak sequencing (seq).

As can be expected, we have associativity of alt, xalt, par and seq. We also have commutativity of alt, xalt and par. Proofs can be found in [7].

7.2.1 Negation

The neg construct defines negative traces:

$$[\![\mathsf{neg}\ d]\!] \stackrel{\text{def}}{=} \{(\{\langle\rangle\}, p \cup n) \mid (p, n) \in [\![d]\!]\}$$

Notice that a negative trace cannot be made positive by reapplying neg. Negative traces remain negative. Negation is an operation that characterizes traces absolutely and not relatively. The intuition is that the focus of the neg construct is on characterizing the positive traces in the operand as negative. Negative traces will always propagate as negative to the outermost level. The neg construct defines the empty trace as positive. This facilitates the embedding of negs in sequence diagrams also specifying positive behavior.

7.2.2 Potential choice

The alt construct defines potential traces. The semantics is the union of the trace sets for both positive and negative:

$$[\![d_1\ \mathsf{alt}\ d_2]\!] \stackrel{\text{def}}{=} \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in [\![d_1]\!] \wedge (p_2, n_2) \in [\![d_2]\!]\}$$

7.2.3 Mandatory choice

The xalt construct defines mandatory choices. All implementations must be able to handle every interaction obligation.

$$[\![d_1\ \mathsf{xalt}\ d_2]\!] \stackrel{\text{def}}{=} [\![d_1]\!] \cup [\![d_2]\!]$$

7.2.4 Parallel execution

The par construct represents a parallel merge.

In order to define par, we first define parallel execution on trace sets:

$$s_1 \parallel s_2 \stackrel{\text{def}}{=} \{h \in H \mid \exists\, p \in \{1,2\}^\infty : \pi_2((\{1\} \times E) \circledT (p, h)) \in s_1 \wedge \pi_2((\{2\} \times E) \circledT (p, h)) \in s_2\}$$

In this definition, we make use of an oracle, the infinite sequence p, to resolve the non-determinism in the interleaving. It determines the order in which events from traces in s_1 and s_2 are sequenced. π_2 is a projection operator returning the second element of a pair.

The par construct itself is defined as:

$$[\![d_1\ \mathsf{par}\ d_2]\!] \stackrel{\text{def}}{=} \{o_1 \parallel o_2 \mid o_1 \in [\![d_1]\!] \wedge o_2 \in [\![d_2]\!]\}$$

where parallel execution of interaction obligations is defined as:

$$(p_1, n_1) \parallel (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \parallel p_2, (n_1 \parallel p_2) \cup (n_1 \parallel n_2) \cup (p_1 \parallel n_2))$$

Note how any trace involving a negative trace will remain negative in the resulting interaction obligation.

7.2.5 Weak sequencing

Weak sequencing is the implicit composition mechanism combining constructs of a sequence diagram.

First, we define weak sequencing of trace sets:

$$s_1 \gtrsim s_2 \stackrel{\text{def}}{=} \{h \in s_1 \parallel s_2 \mid \exists\, h_1 \in s_1, h_2 \in s_2 : \forall l \in L : e.l \circledS h = e.l \circledS h_1 \frown e.l \circledS h_2\}$$

where $e.l$ denotes the set of events that may take place on the lifeline l. Note that weak sequencing degenerates to parallel execution if the operands have disjoint sets of lifelines.

The seq construct itself is defined as:

$$[\![d_1\ \mathsf{seq}\ d_2]\!] \stackrel{\text{def}}{=} \{o_1 \gtrsim o_2 \mid o_1 \in [\![d_1]\!] \wedge o_2 \in [\![d_2]\!]\}$$

where weak sequencing of interaction obligations is defined as:

$$(p_1, n_1) \gtrsim (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \gtrsim p_2, (n_1 \gtrsim p_2) \cup (n_1 \gtrsim n_2) \cup (p_1 \gtrsim n_2))$$

Note how anything involving a negative trace remains negative.

7.3 Refinement

Refinement means to add information to a specification such that the specification becomes closer to an implementation. Supplementing and narrowing are special cases of this general notion. Detailing is defined in terms of lifeline decomposition and interface refinement that are both defined in terms of the basic notion of refinement.

An interaction obligation (p_2, n_2) is a refinement of an interaction obligation (p_1, n_1), written

$$(p_1, n_1) \rightsquigarrow (p_2, n_2)$$

iff

$$n_1 \subseteq n_2 \wedge p_1 \subseteq p_2 \cup n_2$$

A sequence diagram d' is a refinement of a sequence diagram d, written $d \rightsquigarrow d'$, iff

$$\forall\, o \in [\![d]\!] : \exists\, o' \in [\![d']\!] : o \rightsquigarrow o'$$

The refinement semantics supports the classical notions of compositional refinement providing a firm foundation for compositional analysis, verification and testing. In [7] we prove that refinement as defined above is transitive as well as monotonic with respect to the operators defined in Sect. 7.2.

7.3.1 Supplementing

Supplementing categorizes inconclusive behavior as either positive or negative. An interaction obligation (p_2, n_2) supplements an interaction obligation (p_1, n_1), written $(p_1, n_1) \rightsquigarrow_s (p_2, n_2)$, iff

$$\begin{array}{ccc}(n_1 \subset n_2 & \wedge & p_1 \subseteq p_2) \\ & \vee & \\ (n_1 \subseteq n_2 & \wedge & p_1 \subset p_2)\end{array}$$

7.3.2 Narrowing

Narrowing reduces the allowed (positive) behavior to match the problem better. An interaction obligation (p_2, n_2) narrows an interaction obligation (p_1, n_1), written $(p_1, n_1) \rightsquigarrow_n (p_2, n_2)$, iff

$$p_2 \subset p_1 \quad \wedge \quad n_2 = n_1 \cup (p_1 \setminus p_2)$$

7.3.3 Black-box refinement

Black-box refinement may be understood as refinement restricted to the externally visible behavior. We define the function

$$ext \in H \times \mathbb{P}(L) \rightarrow H$$

to yield the trace obtained from the trace given as first argument by filtering away those events that are internal with respect to the set of lifelines given as second argument, i.e.:

$$ext(h, l) \stackrel{\text{def}}{=} \{e \in E \mid (k.e = ? \wedge tr.e \notin l) \vee (k.e = ! \wedge re.e \notin l)\} \circledS h$$

The ext operator is overloaded to sets of traces and pairs of sets of traces in the standard pointwise manner, e.g.: $ext(s, l) \stackrel{\text{def}}{=} \{ext(h, l) \mid h \in s\}$.

A sequence diagram d' is a black-box refinement of a sequence diagram d, written $d \rightsquigarrow_b d'$, iff

$$\forall o \in [\![d]\!] : \exists o' \in [\![d']\!] : ext(o, ll(d)) \rightsquigarrow ext(o', ll(d'))$$

where the function ll yields the set of lifelines of a sequence diagram.

7.3.4 Detailing

When we increase the granularity of sequence diagrams we call this a detailing of the specification. The granularity can be altered in two different ways: either by decomposing the lifelines such that their inner parts and their internal behavior are displayed in the diagram or by changing the data-structure of messages such that they convey more detailed information.

Black-box refinement is sufficiently general to formalize lifeline decompositions that are not externally visible. However, many lifeline decompositions are externally visible. As an example of a lifeline decomposition that is externally visible, consider the decomposition of the ATM in Fig. 13. The messages that originally (in Fig. 8) had :ATM as sender/receiver, now have the different components of the ATM (such as :CardReader or :Screen) as sender/receiver.

To allow for this, we extend the definition of black-box refinement with the notion of a lifeline substitution. The resulting refinement relation is called lifeline decomposition. A lifeline substitution is a partial function of type $L \rightarrow L$. LS denotes the set of all such substitutions. We define the function

$$subst \in D \times LS \rightarrow D$$

such that $subst(d, ls)$ yields the sequence diagram obtained from d by substituting every lifeline l in d for which ls is defined with the lifeline $ls(l)$.

We then define that a sequence diagram d' is a lifeline decomposition of a sequence diagram d with respect to a lifeline substitution ls, written $d \rightsquigarrow_l^{ls} d'$, iff

$$d \rightsquigarrow_b subst(d', ls)$$

Changing the data-structure of messages may be understood as black-box refinement modulo a translation of the externally visible behavior. This translation is specified by a sequence diagram t, and we refer to this as an interface refinement.

We define that a sequence diagram d' is an interface refinement of a sequence diagram d with respect to a sequence diagram t, written $d \rightsquigarrow_i^t d'$, iff

$$d \rightsquigarrow_b (t \ \mathsf{seq} \ d')$$

Detailing may then be defined as the transitive and reflexive closure of lifeline decomposition and interface refinement.

8 Conclusions

We have presented STAIRS, a formal approach to the stepwise, incremental development of interactions. It is based on trace semantics. Traces are sequences of events. Events are representations of sending and receiving messages. STAIRS meets the requirements of Sect. 2 in the following sense:

1. Different potential behaviors are expressed through lifeline independence and by alt combined fragments. Semantically, each potential behavior is represented by a trace.
2. Different mandatory behaviors are expressed using combined fragments with xalt. Semantically, different mandatory behaviors are separated by placing them in the different interaction obligations.
3. The potential and the mandatory behavior constitute the positive behavior. Negative behavior may be specified by the neg-construct. Also negative behavior is represented semantically by a set of traces in every interaction obligation.

4. The classical notion of refinement is supported. Firstly, under-specification in the form of potential behavior may be reduced (narrowing). This corresponds to refinement by strengthening the post-condition in traditional pre/post specification [10]. Secondly, the scope of the specification may be enlarged (supplementing). This corresponds to refinement by weakening the pre-condition in traditional pre/post specification [10]. Thirdly, the granularity and data-structure of messages may be altered (detailing). This corresponds to classical data-refinement [2], or more exactly, to the more recent form of interface refinement as e.g. in TLA [11] and Focus [12].
5. Incremental development of interactions in the form of supplementing, narrowing and detailing has been formalized.
6. The underlying semantics supports the classical notations of compositional refinement providing a firm foundation for compositional analysis, verification and testing. In [7] we show that the basic notions of supplementing and narrowing are reflexive, transitive and monotonic with respect to the operators specified in Sect. 7. The same holds for detailing modulo the specified translation.

8.1 Related work

To consider not only positive traces, but also negative ones, has been suggested before. In [13] the proposed methodology stated that specifying negative scenarios could be even more practical and powerful than only specifying the possible or mandatory ones. It was made clear that the MSC-92 standard [14] was not sufficient to express the intention behind the scenarios and that the MSC documents had to be supplemented with informal statements about the intended interpretation of the set of traces expressed by the different MSCs.

The algebraic semantics of MSC-92 [15] gave rise to a canonical logical expression restricted to the strict sequencing operator and a choice operator. When the MSC standard evolved with more advanced structuring mechanisms, the formal semantics as given in [16] and [17] was based on sets of traces, but it was still expressed in algebraic terms. The MSC approach to sequence diagram semantics is an interleaving semantics based on a fully compositional paradigm. The set of traces denoting the semantics of a message sequence chart can be calculated from its constituent parts based on definitions of the semantics of the structuring concepts as operators. This is very much the approach that we base our semantics on as we calculate our semantics of an interaction fragment from the semantics of its internal fragments. The notion of negative traces, and the explicit distinction between mandatory and potential behavior is beyond the MSC language and its semantics. The Eindhoven school of MSC researchers led by Sjouke Mauw concentrated mainly on establishing the formal properties of the logical systems used for defining the semantics, and also how this could be applied to make tools.

The need for describing also the intention behind the scenarios motivated the so-called "two-layer" approaches. In [18] they showed how MSC could be combined with languages for temporal logics such as CTL letting the scenarios constitute the atoms for the higher level of modal descriptions. With this one could describe that certain scenarios should appear or should never appear.

Damm and Harel brought this further through their augmented MSC language LSC (Live Sequence Charts) [19]. This may also be characterized as a two-layer approach as it takes the basic message sequence charts as starting point and add modal characteristics upon those. The modal expressiveness is strong in LSC since charts, locations, messages and conditions are orthogonally characterized as either mandatory or provisional. Since LSC also includes a notion of subchart, the combinatory complexity can be quite high. The "inline expressions" of MSC-96 (corresponding to combined fragments in UML 2.0) and MSC documents as in MSC-2000 [20] (corresponds to classifier in UML 2.0) are, however, not included in LSC. Mandatory charts are called universal. Their interpretation is that provided their initial condition holds, these charts must happen. Mandatory as in LSC should not be confused with mandatory as in STAIRS, since the latter only specifies traces that must be present in an implementation while the first specifies all allowed traces. Hence, mandatory as in STAIRS does not distinguish between universal or existential interpretation, but rather gives a restriction on what behaviors that must be kept during a refinement. Provisional charts are called existential and they may happen if their initial condition holds. Through mandatory charts it is of course indirectly also possible to define scenarios that are forbidden or negative. Their semantics is said to be a conservative extension of the original MSC semantics, but their construction of the semantics is based on a two-stage procedure. The first stage defines a symbolic transition system from an LSC and from that a set of runs accepted by the LSC is produced. These runs represent traces where each basic element is a snapshot of a corresponding system.

The motivation behind LSC is explicitly to relate sequence diagrams to other system descriptions, typically defined with state machines. Harel has also been involved in the development of a tool-supported methodology that uses LSC as a way to prescribe systems as well as verifying the correspondence between manually described LSCs and State Machines [21].

Our approach is similar to LSC since it is basically interleaving. STAIRS is essentially one-stage as the modal distinction between the positive and negative traces in principle is present in every fragment. The final modality results directly from the semantic compositions. With respect to language, we consider almost only what is UML 2.0, while LSC is a language extension of its own. LSC could in the future become a particular UML profile. Furthermore, our focus is on refinement of sequence diagrams as a means for system development and system validation. This means that in our approach the distinction between mandatory and provisional is captured through interaction obligations.

The work by Krüger [22] addresses similar concerns as the ones introduced in this article and covered by the LSC-approach of Harel. Just as with LSC MSCs can be given interpretations as existential or universal. The exact and negative interpretations are also introduced. Krüger also proposes notions of refinement for MSCs. Binding references, interface refinement, property refinement and structural refinement are refinement relations between MSCs at different level of abstraction. Narrowing as described in STAIRS corresponds closely to property refinement in [22] and detailing corresponds to interface refinement and structural refinement. However, Krüger does not distinguish between intended non-determinism and non-determinism as a result of under–specification in the refinement relations.

Although this paper presents STAIRS in the setting of UML 2.0 sequence diagrams, the underlying principles apply just as well to MSC given that the MSC language is extended with an xalt construct similar to the one proposed above for UML 2.0. STAIRS may also be adapted to support LSC. STAIRS is complementary to software development processes based on use-cases, and classical object-oriented approaches such as the Unified Process [23]. STAIRS provides formal foundation for the basic incremental steps of such processes.

Acknowledgements The research on which this paper reports has partly been carried out within the context of the IKT-2010 project SARDAS (15295/431) funded by the Research Council of Norway. We thank Mass Soldal Lund, Fredrik Seehusen and Ina Schieferdecker for helpful feedback.

References

1. Haugen, Ø. and Stølen, K.: STAIRS—Steps to analyze interactions with refinement semantics. In: Sixth International Conference on UML (UML'2003), no. 2863 in Lecture Notes in Computer Science, pp. 388–402. Springer (2003)
2. Hoare, C.A.R.: Proof of correctness of data representations. Acta Informatica **1**, 271–282 (1972)
3. Jones, C.B.: Formal development of correct algorithms: An example based on Earley's recogniser. In: ACM Conference on Proving Assertions about Programs, no. 7 in SIGPLAN Notices, pp. 150–169 (1972)
4. Milner, R.: An algebraic definition of simulation between programs. In: International Joint Conference on Artificial Intelligence, pp. 481–489. Kaufmann (1971)
5. de Roever, W.-P.: The quest for compositionality: A survey of assertion-based proof systems for concurrent programs: Part 1. In Formal Models in Programming, pp. 181–205. North-Holland (1985)
6. Jones, C.B.: Development Methods for Computer Programs Including a Notion of Interference. PhD thesis, Oxford University (1981)
7. Haugen, Ø., Husa, K.E., Runde, R.K., Stølen, K.: Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo (2004)
8. Object Management Group.: UML 2.0 Superstructure Specification, document: ptc/04-10-02 edition (2004)
9. Haugen, Ø., Møller-Pedersen, B., Weigert, T.: Structural modeling with UML 2.0. In: UML for Real, pp. 53–76. Kluwer (2003)
10. Jones, C.B.: Systematic Software Development Using VDM. Prentice-Hall (1986)
11. Abadi, M., Lamport, L.: Conjoining specifications. ACM Trans. Prog. Lang. Sys. **17**, 507–533 (1995)
12. Broy, M., Stølen, K.: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer, Berlin Heidelberg New York (2001)
13. Haugen, Ø.: Using MSC-92 effectively. In: 7th SDL Forum (SDL'95), pp. 37–49. North-Holland (1995)
14. International Telecommunication Union.: Recommendation Z.120—Message Sequence Chart (MSC) (1993)
15. International Telecommunication Union.: Recommendation Z.120 Annex B: Algebraic Semantics of Message Sequence Charts (1994)
16. International Telecommunication Union.: Recommendation Z.120 Annex B: Formal Semantics of Message Sequence Charts (1998)
17. Reniers, M.A.: Message Sequence Chart: Syntax and Semantics. PhD thesis, Eindhoven University of Technology (1998)
18. Combes, P., Pickin, S., Renard, B., Olsen, F.: MSCs to express service requirements as properties on an SDL model: Application to service interaction detection. In: 7th SDL Forum (SDL'95), pp. 243–256. North-Holland (1995)
19. Damm, W., Harel, D.: LSCs: Breathing life into message sequence charts. In: Formal Methods for Open Object-Based Distributed Systems (FMOODS'99), pp. 293–311. Kluwer (1999)
20. Haugen, Ø.: MSC-2000 interaction diagrams for the new millennium. Computer Networks **35**, 721–732 (2001)
21. Harel, D., Marelly, R.: Specifying and executing behavioral requirements: The play-in/play-out approach. Soft. Sys. Model. **2**, 82–107 (2003)
22. Krüuger, I.: Distributed System Design with Message Sequence Charts. PhD thesis, Technische Universität München (2000)
23. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley (1999)

Øystein-Haugen

Knut-Eilif-Husa

Ragnhild-Kobro-Runde

Ketil-Stølen