STAIRS – Steps to Analyze Interactions with Refinement Semantics

Øystein Haugen[1] and Ketil Stølen[2]

[1] Ericsson NorARC and University of Oslo, Norway
oystein.haugen@ericsson.com
[2] SINTEF Telecom & Informatics and University of Oslo, Norway
ketil.stolen@sintef.no

Abstract. The paper presents STAIRS, 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) considered irrelevant for the interaction in question. This categorization corresponds well with that of testing where the verdict of a test execution is either pass, fail or inconclusive. The basic increments in system development are structured into three 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.

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 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.

The remainder of the paper is structured into six 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, 5 and 6 spell out STAIRS in an example-driven manner addressing respectively the notions of supplementing, narrowing and detailing. Section 7 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. Under-specification is caused in part by independence between lifelines and in part by alternative combined fragments. Under-specification gives the implementer freedom to select between several potential behaviors.

2. *Should Allow Specification of Mandatory Behavior.* Under-specification as described in the previous paragraph appears as non-determinism. This non-determinism is not essential and should not be present in a correct implementation. Sometimes, however, non-determinism is intended meaning that every correct implementation is required to reflect each 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 eliminate a run that as a whole produces the same results as some other behavior, which is also included. It is different, however, to eliminate a behavior that have little in common with those remaining. It is possible to imagine that a system could work even if only one of these situations were handled, but the point of describing them both in a "mandatory" alternative is that both of them should be implemented. They represent distinct behaviors, but both of them should be found in an executing system. This means that 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 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 scenarios. 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 [12] 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 Incremental Development.* Incremental development of interactions involves various sub-activities. Important examples of such sub-activities are supplementing, narrowing and detailing, specified informally above. 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 basis for analysis, verification and testing. STAIRS should provide a foundation for these activities facilitating compositionality 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, such interactions are not within the scope of 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 [13][1]. For more on UML 2.0, see also [8].

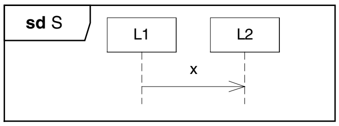

Fig. 1. Simple interaction with only one message

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 <!x,?x>.

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 we call weak sequencing. Weak sequencing takes into account that L1 and L2 are independent. The weak se-

[1] The reference is to the document that was recommended for adoption at the OMG meeting in Paris in June 2003 submitted by the U2 Partners. It is informally known as "UML 2.0".

quencing 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.
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.

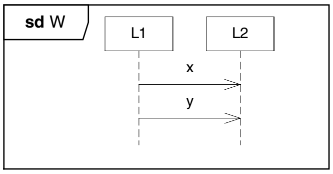

Fig. 2. Weak sequencing

Thus, if we denote the weak sequencing operator by **seq** according to UML 2.0, we get: W=<!x,?x> **seq** <!y,?y> = {<!x,?x,!y,?y>, <!x,!y,?x,?y>}. The sending of x must be the first event to happen, but after that either L1 may send y or L2 may receive x.

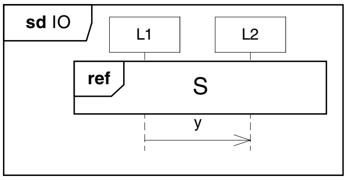

Fig. 3. Interaction occurrence

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 IO=S **seq** <!y,?y>=<!x,?x> **seq** <!y,?y> = {<!x,?x,!y,?y>, <!x,!y,?x,?y>}.

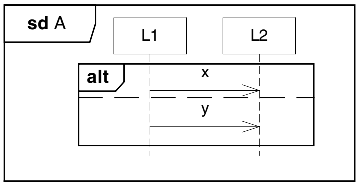

Fig. 4. Combined fragment (alternative)

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 A = <!x,?x> **alt** <!y,?y> = {<!x,?x>,<!y,?y>}.

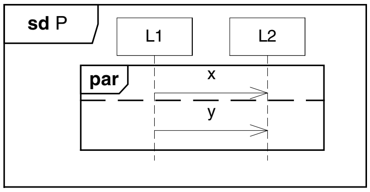

Fig. 5. Parallel combined fragment

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=<!x,?x> **par** <!y,?y> = {<!x,?x,!y,?y>, <!x,!y,?x,?y>, <!x,!y,?y,?x>, <!y,?y,!x,?x>, <!y,!x,?y,?x>, <!y,!x,?x,?y>}.

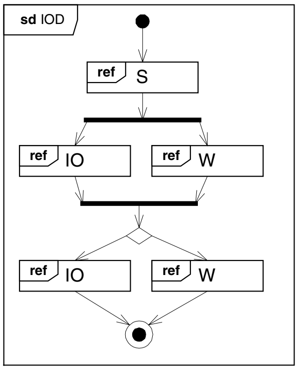

Fig. 6. Interaction overview diagram

Finally we show an alternative syntax for these combined fragments. Fig. 6 presents a more complicated interaction with the definition IOD=S **seq** (IO **par** W) **seq** (IO **alt** W). 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 will seem intuitive. We describe the behavior of a restaurant serving dinner. We specify that a restaurant will in general have a dinner menu specifying that dinners will have a salad as starter followed by a main course consisting of an entree[2] and a side order.

The entree may be vegetarian, beef or pork. The side order is baked potato, rice or French fries. In an interaction overview diagram the make-up of a dinner may look like Fig. 7. To look into the details down to the events, we can follow e.g. Beef as shown in Fig. 8.

[2] We use the American term "entree" here for the main dish.

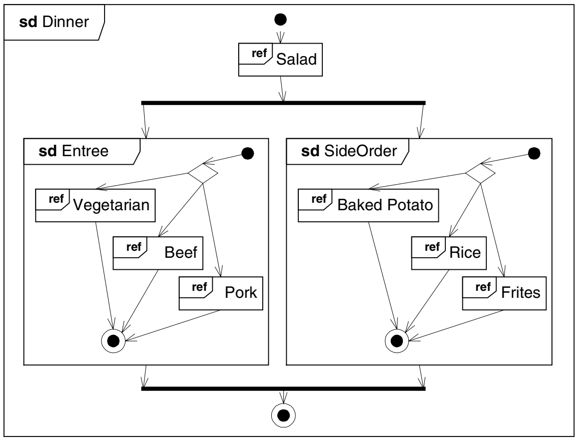

Fig. 7. Dinner

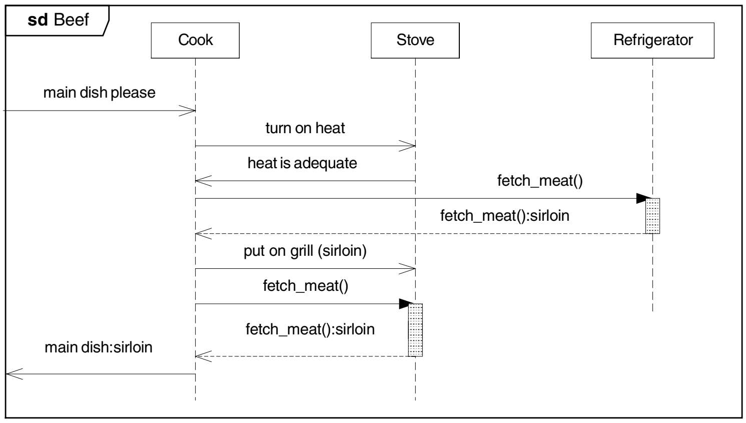

Fig. 8. Beef (positive traces)

Our Beef sequence diagram shows a trace that is a subsequence of a Dinner sequence[3]. Our Beef 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 restaurant beef entree production.

Each of the interaction occurrences in Fig. 7 represents a set of positive traces. The sequencing of the salad and the main course in this case is essentially appending traces. The fork and join between entree and side order represents a parallel merge combination meaning that all traces of the entree are braided (interleaved) with every trace of the side order. The branching within entree (or side order) represents alternative choices and their combination is essentially a union. Abbreviating the dishes and using the operators of UML 2.0 combined fragment, the following formula defines

[3] Formally, the oven may not receive the sirloin before the cook initiates taking it off, but nobody wants that rare meat, so for simplicity we shall disregard that extra trace.

the positive traces of Dinner: S **seq** ((V **alt** B **alt** P) **par** (BP **alt** R **alt** F)). 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 making a dinner are intuitively understood.

Summary: *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.*

3.3 Capturing Negative Behavior

In general the Beef procedure may include more than the single trace shown in Fig. 8. In Fig. 9 we show a more general variant. The cook may want to try and work faster by turning on the heat more or less at the same time as he is fetching the meat from the refrigerator. Thus we want to express that the two different messages may be sent in any order. Likewise we specify that he may get the meat before realizing that the oven is ready, or in the opposite order. This variability is expressed through two coregions indicating that events in the region can be permuted in any way. This is shorthand on one lifeline for the parallel merge combination.

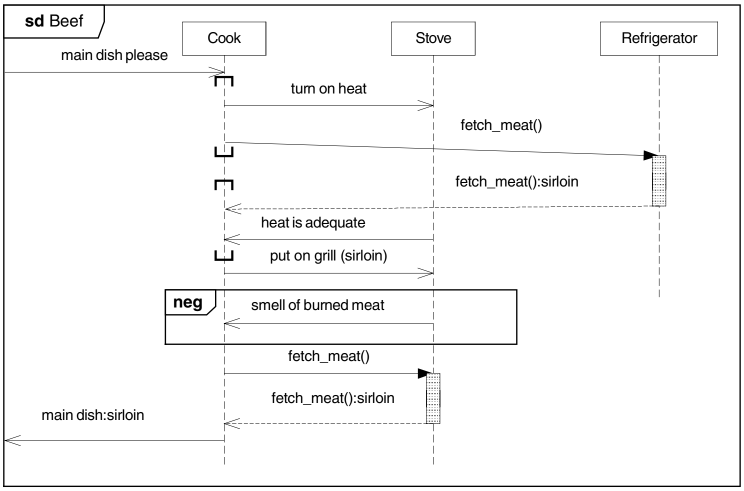

Fig. 9. Beef (with coregions and negative traces)

We notice in Fig. 9 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 appended with the traces of the negative fragment. The diagram in Fig. 9 also defines a set of positive traces that just omit the negative fragment.

In our example the intuition is simple, any trace that results in the smell of burned meat (after having put meat on the grill) is a negative scenario. Anything may happen after burning the meat, it will never make it positive.

Still we have not defined all possible scenarios of preparing sirloin beef. 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 that the cook forgets to put the sirloin steak on the grill. Sometimes in restaurants this seems to be a relevant scenario. Our diagram in Fig. 9 leaves that scenario inconclusive.

Summary: *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 represented by a pair of sets (P, N) where N contains the negative traces and P contains the positive traces. The same trace cannot be both positive and negative. Traces that are neither negative nor positive are inconclusive.*

3.4 Distinguishing Mandatory from Potential Behavior

Assume that we intend to use our Dinner scenario as a requirement specification for restaurants. The question then becomes whether every restaurant needs to be able to perform every positive trace. This would mean that every restaurant must be able to serve beef and pork dishes. This would not suit Indian and Muslim restaurants. Thus this is not an adequate interpretation. Rather we would like to convey that every restaurant should offer vegetarian as well as meat dishes. This is probably not good either, but for the sake of argument, pretend it is reasonable. We need a way to say that the distinction between Pork and Beef is provisional, but the choice between vegetarian and meat is a choice where both alternatives should be present. 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.

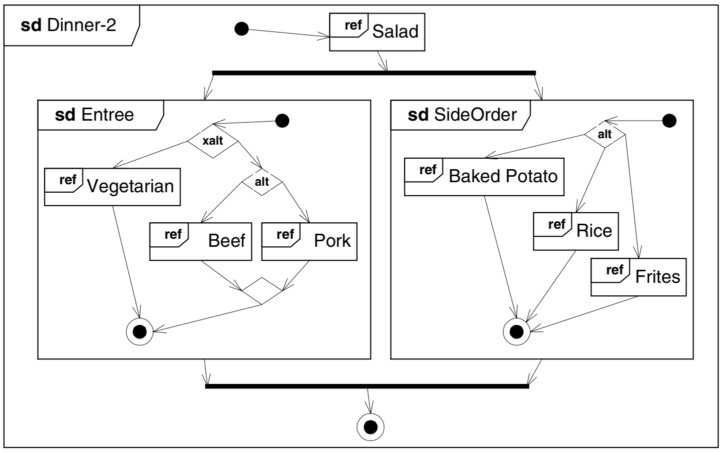

Fig. 10. Mandatory alternatives (xalt)

The semantics of Dinner-2 can be described as: S **seq** ((V **xalt** (B **alt** P)) **par** (BP **alt** R **alt** F)). In terms of our semantic model this is captured by the positive traces residing in sets discriminated by the **xalt** operation. Thus we have for Dinner-2, two such mandatory sets (inner sets), one including the vegetarian entree and one including a meat entree. Each of these sets contains a number of traces since the production of side-orders may be combined with the production of the entree in several ways.

Summary: *Semantically, we still represent an interaction as a pair of sets (P, N) (where N may be empty if no negative behavior is specified). However, in contrast with the previous section, P is now redefined to be a set of sets of traces {P1,...,Pn}. An implementation satisfying the specification must be able to perform at least one trace from each of the n inner sets Pj. Each inner set represents potential variations of a mandatory behavior that must be kept separate from the other inner sets representing variations of other mandatory behaviors. The traces within the same inner set 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 restaurant example, we may supplement the menu of potential entrees with meat dishes of turkey or tuna fish. We may likewise supplement with side orders such as mashed potatoes and spaghetti.

Furthermore, we may supplement the detailed production traces with more unwanted scenarios like when the cook forgets the sirloin or chops his finger off into the vegetarian entree. We illustrate this with a Venn-diagram in Fig. 11.

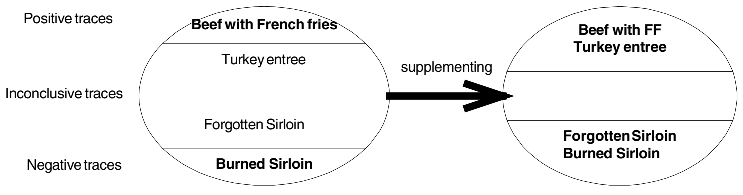

Fig. 11. Supplementing

Summary: *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.*

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 than the former. In our context of interactions, reducing the variability 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 restaurant example narrowing could mean Indian restaurants eliminating the Beef scenarios and the Muslim restaurants removing the Pork scenarios. The Indian restaurants would keep the pork and the Muslims the beef. We illustrate this by a Venn-diagram in Fig. 12.

In a more detailed scenario like our general Beef diagram in Fig. 9 we could narrow this by demanding that the cook fetches the meat before he starts heating the oven or the opposite as shown in Fig. 8.

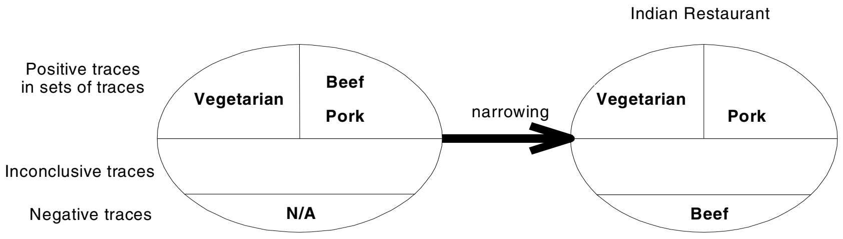

Fig. 12. Narrowing (Indian restaurant)

Summary: *Narrowing means reducing the set of positive traces without making any inner set empty, 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 cook since it actually appears that the cook consists of a chef and his apprentice.

The combined activity of the diagrams in Fig. 13 is that of the Cook lifeline of Fig. 9. This is done through the decomposition mechanism of UML 2.0. In addition we have detailed the outcoming result, namely the main dish as it is actually consisting of the meat and the gravy. We show the simple message translation in the separate diagram. Clearly, the external behavior of Beef_Cook is the same as the external behavior of Cook in Fig. 9 given the main_dish_translate translation on the outcome. This shows how STAIRS supports the decomposition of the maindish message in Cook into the meat and the additions messages in Beef_Cook. The main_dish_translate diagram documents this decomposition and is in contrast with Beef_Cook not supposed to be implemented.

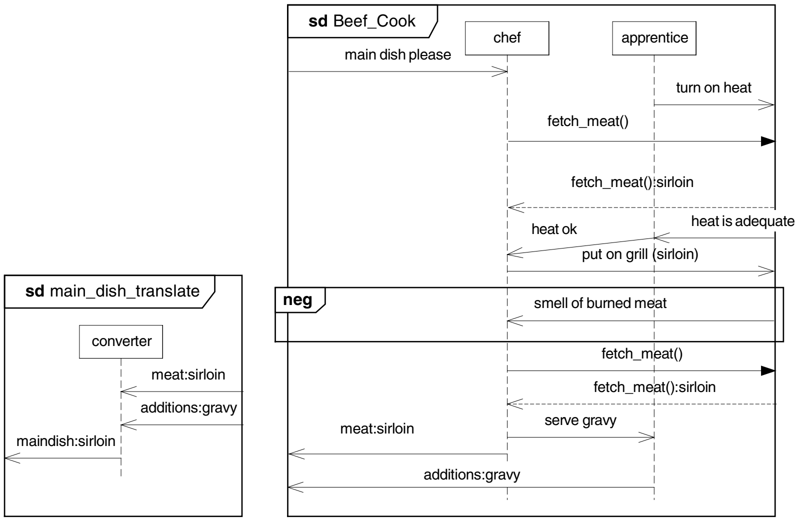

Fig. 13. Detailing the cook and the resulting dish message

Summary: *Detailing means that the sets of positive, negative and inconclusive traces are unchanged with respect to a translation between the more detailed and the given interaction. In practice a detailing will often occur in combination with a supplementing or a narrowing.*

7 Conclusions

We have presented STAIRS, a formal approach to the step-wise, 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 Section 389 in the following sense:

1. Potential behavior is expressed through lifeline independence and by **alt** combined fragments. Semantically, each potential behavior is represented by a trace.
2. Mandatory behavior is expressed using combined fragments with **xalt**. Semantically, different mandatory behaviors are separated by placing them in the different inner sets of positive behavior.
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 traces, but the negative traces are kept in a separate set.

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 [11]. Secondly, the scope of the specification may be enlarged (supplementing). This corresponds to refinement by weakening the pre-condition in traditional pre/post-specification [11]. Thirdly, the granularity and data-structure of messages may be altered (detailing). This corresponds to classical data-refinement [9], or more exactly, to the more recent form of interface refinement as e.g. in TLA [1] and Focus [2].
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 particular, although a formal proof is outside the scope of this paper, the basic notions of supplementing and narrowing are reflexive, transitive and monotonic with respect to parallel composition. The same holds for detailing modulo the specified translation.

7.1 Related Work

To consider not only positive traces, but also negative ones, has been suggested before. In [6] 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 [15] 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 [16] 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 [17] and [14] 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 distinction between mandatory and potential behavior was 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 [3] 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) [4]. 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 [7] (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. 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 [5].

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 the sets of sets of positive traces.

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 [10]. 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, Ragnhild Kobro Runde and Ina Schieferdecker for helpful feedback.

References

1. Abadi, M., Lamport, L. Conjoining specifications. ACM Transactions on Programming Languages and Systems 17, pages 507–533, (1995).
2. Broy, M., Stølen, K. Specification and development of interactive systems: Focus on streams, interfaces and refinement. Springer, (2001).
3. 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 Proc. 7th SDL Forum (SDL'95), pages 243–256. North-Holland, (1995).
4. Damm, W., Harel, D. LSCs: Breathing life into message sequence charts. In Proc. Formal Methods for Open Object-Based Distributed Systems (FMOODS'99), pages 293–311. Kluwer, (1999).
5. Harel, D., Marelly, R. Specifying and executing behavioral requirements: the play in/play-out approach. To appear in Software and System Modeling, (2003).
6. Haugen, Ø. Using MSC-92 effectively. In Proc. 7th SDL Forum (SDL'95), pages 37–49. North-Holland, (1995).
7. Haugen, Ø. MSC-2000 interaction diagrams for the new millennium. Computer Networks 35, pages 721–732, (2001).
8. Haugen, Ø., Møller-Pedersen, B., Weigert, T. Structural Modeling with UML 2.0. In UML for Real. Lavagno, L., Martin, G. and Selic, B. (eds.). Kluwer, (2003).
9. Hoare, C. A. R. Proof of correctness of data representations. Acta Informatica 1, pages 271–282, (1972).
10. Jacobson, I., Booch, G., Rumbaugh, J. The Unified Software Development Process. Addison Wesley, (1999).
11. Jones, C. B. Systematic software development using VDM. Prentice-Hall, (1986).
12. Milner, R. An algebraic definition of simulation between programs. In Proc. International Joint Conference on Artificial Intelligence, pages 481–489. Kaufmann, (1971).
13. OMG, Unified Modeling Language: Superstructure. OMG ad/03-04-01, (2003).
14. Reniers, M. A. Message Sequence Chart: Syntax and Semantics. PhD thesis, Departement of Computer Science, Eindhoven University of Technology, (1998).
15. Z.120, Message Sequence Charts (MSC). Rudolph, E. (ed.). ITU, (1993).
16. Z.120 Annex B, Algebraic Semantics of Message Sequence Charts. Mauw, S. (ed.). ITU, (1994).
17. Z.120 Annex B, Formal Semantics of Message Sequence Charts. Mauw, S. et al. (eds.). ITU, (1998).