Why Timed Sequence Diagrams Require Three-Event Semantics

Øystein Haugen[1], Knut Eilif Husa[1,2], Ragnhild Kobro Runde[1], and Ketil Stølen[1,3]

[1] Department of Informatics, University of Oslo
[2] Ericsson
[3] SINTEF ICT, Norway

Abstract. STAIRS is an approach to the compositional development of sequence diagrams supporting the specification of mandatory as well as potential behavior. In order to express the necessary distinction between black-box and glass-box refinement, an extension of the semantic framework with three event messages is introduced. A concrete syntax is also proposed. The proposed extension is especially useful when describing time constraints. The resulting approach, referred to as Timed STAIRS, is formally underpinned by denotational trace semantics. A trace is a sequence of three kinds of events: events for transmission, reception and consumption. We argue that such traces give the necessary expressiveness to capture the standard UML interpretation of sequence diagrams as well as the black-box interpretation found in classical formal methods.

1 Introduction to STAIRS

Sequence diagrams have been used informally for several decades. The first standardization of sequence diagrams came in 1992 [ITU93] – often referred to as MSC-92. Later we have seen several dialects and variations. The sequence diagrams of UML 1.4 [OMG00] were comparable to those of MSC-92, while the recent UML 2.0 [OMG04] has upgraded sequence diagrams to conform well to MSC-2000 [ITU99].

Sequence diagrams show how messages are sent between objects or other instances to perform a task. They are used in a number of different situations. They are for example used by an individual designer to get a better grip of a communication scenario or by a group to achieve a common understanding of the situation. Sequence diagrams 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 sequence diagrams and compared with those of the earlier phases.

Sequence diagrams 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. Even though sequence diagrams are intuitive – a property which is always exploited, it is not always obvious how one goes about making the sequence diagrams when a certain situation is analyzed. It is also the case that intuition is not always the best guide for a precise interpretation of a complicated scenario. Therefore we have brought forth an approach for reaching a sensible and fruitful set of sequence diagrams, supported by formal reasoning. We called this approach STAIRS – Steps To Analyze Interactions with Refinement Semantics [HS03].

STAIRS distinguishes between positive and negative traces and accepts that some traces may be inconclusive meaning that they have not yet or should not be characterized as positive or negative. 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 inconclusive behavior as either positive or negative. 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 reduces the allowed behavior to match the problem better. Detailing involves introducing a more detailed description without significantly altering the externally observable behavior.

STAIRS distinguishes between potential alternatives and mandatory or obligatory alternatives. A special composition operator named xalt facilitates the specification of mandatory alternatives.

Figure 1 shows our STAIRS example – an interaction overview diagram description of the making of a dinner at an ethnic restaurant.

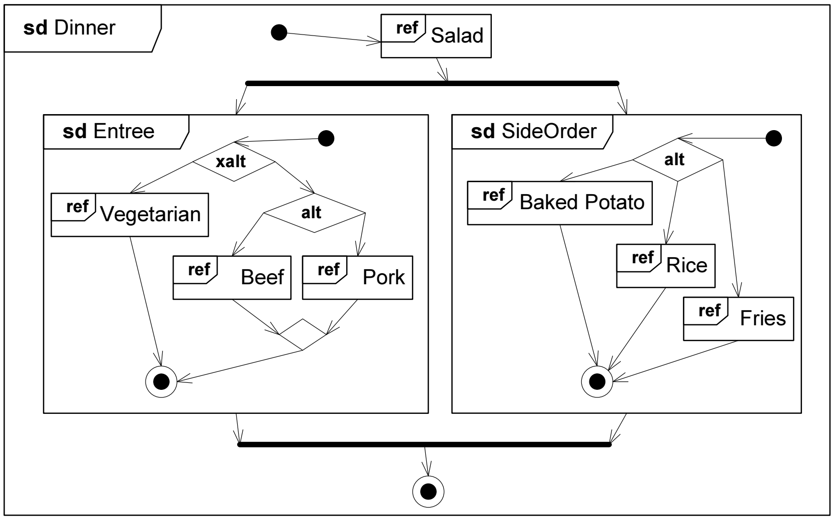

Fig. 1. Interaction overview diagram of a dinner

The dinner starts with a salad and continues with a main course that consists of an entree and a side order, which are made in parallel. For the side order there is a simple choice between three alternatives and the restaurant is not obliged to have any particular of them available. Supplementing side orders could be to offer soya beans in addition, while narrowing would mean that the restaurant could choose only to serve rice and never potatoes nor fries. It would still be consistent with the specification and a valid refinement. On the other hand, the entree has more absolute requirements. The restaurant is obliged to offer vegetarian as well as meat, but it does not have to serve both beef and pork. This means that Indian as well as Jewish restaurants are refinements (narrowing) of our dinner concept, while a pure vegetarian restaurant is not valid according to our specification.

The remainder of the paper is divided into six sections: Section 2 motivates the need for a three event semantics for sequence diagrams. Section 3 introduces the formal machinery; in particular, it defines the syntax and semantics of sequence diagrams. Section 4 defines two special interpretations of sequence diagrams, referred to as the standard and the black-box interpretation, respectively. Section 5 demonstrates the full power of Timed STAIRS as specification formalism. Section 6 introduces glass-box and black-box refinement and demonstrates the use of these notions. Section 7 provides a brief conclusion and compares Timed STAIRS to other approaches known from the literature.

2 Motivating Timed STAIRS

STAIRS works well for its purpose. However, there are certain things that cannot be expressed within the framework as presented in [HS03]. For instance time constraints and the difference between glass-box and black-box view of a system. This section motivates the need for this extra expressiveness.

Let us now look closer at the details of making the Beef in Figure 1.[1] From Figure 2 it is intuitive to assume that the working of the Cook making Beef can be explained by the following scheme: The Cook receives an order for main dish (of type Beef) and then turns on the heat and waits until the heat is adequate. Then he fetches the sirloin meat from the refrigerator before putting it on the grill. Then he fetches the sirloin from the stove (hopefully when it is adequately grilled). He then sends the steak to the customer.

We reached this explanation of the procedures of the cook from looking locally at the cook's lifeline in the Beef diagram. The input event led to one or more outputs, before he again would wait for an input. We found it natural to assume that the input event meant that the cook handled this event, consumed it and

[1] This sequence diagram is not a complete specification of Beef. The supplementing has not yet been finished. From a methodological point of view, the diagram should be "closed" with an assert when the supplementing has been finished. This to state that what is still left as inconclusive behavior should from now on be understood as negative. Otherwise, we do not get the semantics intended by Figure 1.

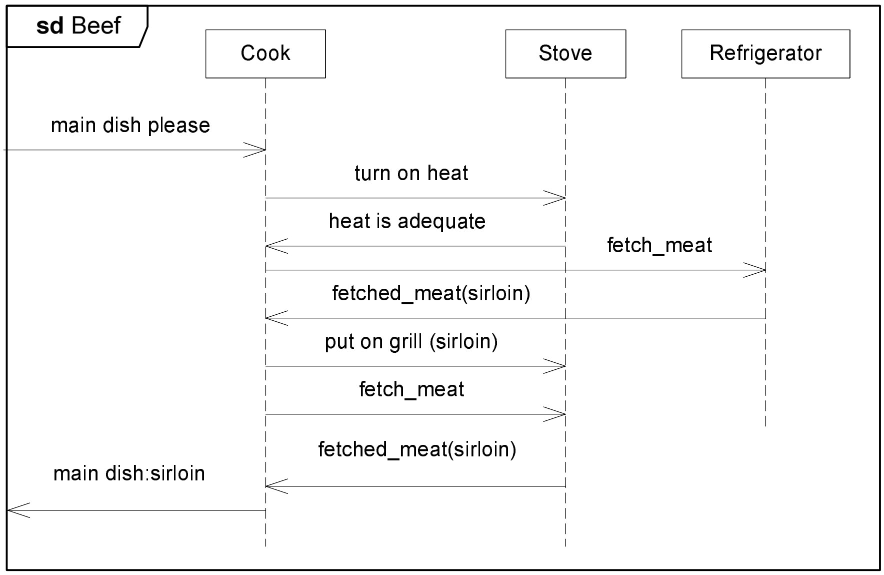

Fig. 2. Sequence diagram of Beef

acted upon it. This intuition gives rise to what we here will call the standard interpretation of sequence diagrams where an input event is seen as consumption of the event, and where the directly following output events of the trace are causally linked to the consumption. Thus, we can by considering each separate lifeline locally determine the transitions of a state machine describing the lifeline.

Our description of how the beef is made comes from a quiet day, or early in the evening when there were not so many customers and the kitchen had no problems to take care of each order immediately. Furthermore our description was probably made for one of those restaurants where the customers can look into the kitchen through glass. It was a glass-box description. We want, however, to be able to describe the situation later in the evening when the restaurant is crammed with customers and in a restaurant where there is only a black door to the kitchen. We would like to assume that even though the restaurant is full, the kitchen will handle our order immediately, but alas this is of course not the case. We can only observe the kitchen as a black-box. We observe the waiters coming through the door as messengers – orders one way and dishes the other. From these observations we could make estimates of the efficiency of the kitchen. Notice that the efficiency of the kitchen cannot be derived from when the customers placed the orders because the waiters may stop at several tables before they enter the kitchen. Comparing black-box observations of the kitchen with our glass-box one, we realize that in the glass-box description no event was attached to passing through the door. The order was sent by the customer and consumed by the chef. The passing through the door represents that the kitchen is receiving the message but not necessarily doing something with it. As long as you are not interested in timing matters, the difference is seldom practically significant, but when time matters, the difference between when a message is received and when it is consumed is crucial. How is the kitchen organized to handle the orders in a swift and fair manner?

Motivated by this we will use three events to represent the communication of a message: the sending event, the receiving event and the consumption event, and each of these events may have a timestamp associated. We will introduce concrete syntax for sequence diagrams to capture this and the distinction is also reflected in the semantics. This will give us sufficient expressiveness to describe a black-box interpretation as well as the standard glass-box interpretation.

3 Formal Foundation

In the following we define the notion of sequence diagram. In particular, we formalize the meaning of sequence diagrams in denotational trace semantics.

3.1 Syntax of Sequence Diagrams

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.

We distinguish between three kinds of events; a transmission event tagged by an exclamation mark "!", a reception event tagged by a tilde "$\sim$", or a consumption event tagged by a question mark "?". K denotes $\{!, \sim, ?\}$.

Every event occurring in a sequence diagram is decorated with a unique timestamp. T denotes the set of timestamp tags. We use logical formulas with timestamp tags as free variables to impose constraints on the timing of events. By $\mathbb{F}(v)$ we denote the set of logical formulas whose free variables are contained in the set of timestamp tags v.

E denotes the set of all events. Formally, an event is a triple of kind, message and timestamp tag

$$E = K \times M \times T$$

We define the functions

$$k.\_ \in E \to K, \quad m.\_ \in E \to M, \quad t.\_ \in E \to T, \quad tr.\_, re.\_ \in E \to L$$

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

$\mathbb{N}$ denotes the set of natural numbers, while $\mathbb{N}_0$ denotes the set of natural numbers including 0.

The set of syntactically correct sequence diagrams D is defined inductively. D is the least set such that:

- $E \subset D$
- $d \in D \Rightarrow \mathsf{neg}\, d \in D \wedge \mathsf{assert}\, d \in D$

- $d_1, d_2 \in D \Rightarrow d_1 \,\mathsf{alt}\, d_2 \in D \wedge d_1 \,\mathsf{xalt}\, d_2 \in D \wedge d_1 \,\mathsf{seq}\, d_2 \in D \wedge d_1 \,\mathsf{par}\, d_2 \in D$
- $d \in D \wedge C \in \mathbb{F}(tt(d)) \Rightarrow d \,\mathsf{tc}\, C \in D$

where $tt(d)$ yields the set of timestamp tags occurring in d. The base case implies that any event is a sequence diagram. Any other sequence diagram is constructed from the basic ones through the application of operators for negation, assertion, potential choice, mandatory choice, weak sequencing, parallel execution and time constraint. The full set of operators as defined by UML 2.0 [OMG04] is somewhat more comprehensive, and it is beyond the scope of this paper to treat them all. We focus on the operators that we find most essential.

We define the function

$$ll \in D \rightarrow \mathbb{P}(L)$$

to yield the set of lifelines of the sequence diagram constituting its argument.

3.2 Representing Executions by Traces

We will define the semantics of sequence diagrams by using sequences of events.

By A^{∞} and A^{ω} we denote the set of all infinite sequences and the set of all finite and infinite sequences over the set A, respectively. We define the functions

$$\#\_ \in A^{\omega} \rightarrow \mathbb{N}_0 \cup \{\infty\}, \quad \_[\_] \in A^{\omega} \times \mathbb{N} \rightarrow A$$

to yield the length and the nth element of a sequence. Hence, $\#a$ yields the number of elements in a, and $a[n]$ yields a's nth element if $n \leq \#a$.

We also need functions for concatenation, truncation and filtering:

$$\_\frown\_ \in A^{\omega} \times A^{\omega} \rightarrow A^{\omega}, \quad \_|\_ \in A^{\omega} \times \mathbb{N}_0 \rightarrow A^{\omega},$$
$$\_Ⓢ\_ \in \mathbb{P}(A) \times A^{\omega} \rightarrow A^{\omega}, \quad \_Ⓣ\_ \in \mathbb{P}(A \times B) \times (A^{\omega} \times B^{\omega}) \rightarrow A^{\omega} \times B^{\omega}$$

Concatenating two sequences implies gluing them together. Hence, $a_1 \frown a_2$ denotes a sequence of length $\#a_1 + \#a_2$ that equals a_1 if a_1 is infinite, and is prefixed by a_1 and suffixed by a_2, otherwise. For any $0 \leq i \leq \#a$, we define $a|_i$ to denote the prefix of a of length i.

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

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

The filtering function Ⓣ may be understood as a generalization of Ⓢ. The function Ⓣ filters pairs of sequences with respect to pairs of elements in the same way as Ⓢ filters sequences with respect to elements. For any set of pairs of elements P and pair of sequences t, by P Ⓣ 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

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

For a formal definition of $\circledT$, see [BS01].

We are mainly interested in communication scenarios. The actual content of messages is not significant for the purpose of this paper. Hence, we do not give any semantic interpretation of messages as such. The same holds for events except that the timestamp tag is assigned a timestamp in the form of a real number. $\mathbb{R}$ denotes the set of all timestamps. Hence:[2]

$$[\![E]\!] \overset{\mathsf{def}}{=} \{(k, m, t \mapsto r) \mid (k, m, t) \in E \wedge r \in \mathbb{R}\} \tag{1}$$

We define the function

$$r.\_ \in [\![E]\!] \rightarrow \mathbb{R}$$

to yield the timestamp of an event. Moreover, for any lifeline l and kind s, let $E(l, s)$ be the set of all events $e \in [\![E]\!]$ such that $tr.e = l$ and $k.e = s$.

A trace h is an element of $[\![E]\!]^{\omega}$ that satisfies a number of well-formedness conditions. We use traces to represent executions. By an execution we mean the trace of events resulting from an execution of the specified system. We require the events in h to be ordered by time: the timestamp of the ith event is less than or equal to the timestamp of the jth event if $i < j$. Formally:

$$\forall\, i, j \in [1..\#h] : i < j \Rightarrow r.h[i] \leq r.h[j]$$

This means that two events may happen at the same time.

The same event takes place only once in the same execution. Hence, we also require:

$$\forall\, i, j \in [1..\#h] : i \neq j \Rightarrow h[i] \neq h[j]$$

The following constraint makes sure that time will eventually progress beyond any finite point in time:

$$\#h = \infty \Rightarrow \forall\, t \in \mathbb{R} : \exists\, i \in \mathbb{N} : r.h[i] > t$$

That is, for any timestamp t in an infinite trace h there is an event in h whose timestamp t' is greater than t.

[2] The functions k, m, t, tr, re on E are lifted to $[\![E]\!]$ in the obvious manner.

For any single message, transmission must happen before reception, which must happen before consumption. However, in a particular sequence diagram we may have only the transmitter or the receiver lifeline present. Thus we get the following well-formedness requirements on traces, stating that if at any point in the trace we have a transmission event, up to that point we must have had at least as many transmissions as receptions of that particular message, and similarly for reception events with respect to consumptions:

$$\forall\, i \in [1..\#h] : k.h[i] = ! \Rightarrow \tag{2}$$

$$\#(\{!\} \times \{m.h[i]\} \times U) \circledS h|_i > \#(\{\sim\} \times \{m.h[i]\} \times U) \circledS h|_i$$

$$\forall\, i \in [1..\#h] : k.h[i] = \sim \Rightarrow \tag{3}$$

$$\#(\{\sim\} \times \{m.h[i]\} \times U) \circledS h|_i > \#(\{?\} \times \{m.h[i]\} \times U) \circledS h|_i$$

where $U \stackrel{\mathsf{def}}{=} \{t \mapsto r \mid t \in T \wedge r \in \mathbb{R}\}$. H denotes the set of all well-formed traces.

We define three basic composition operators on trace sets, namely parallel execution, weak sequencing, and time constraint denoted by $\|$, $\succsim$, and $\wr$, respectively.

Informally, $s_1 \parallel s_2$ is the set of all traces such that

- all events from s_1 and s_2 are included (and no other events),
- the ordering of events from s_1 and from s_2 is preserved.

Formally:

$$\begin{aligned} s_1 \parallel s_2 \quad \stackrel{\mathsf{def}}{=} \quad & \{h \in H \mid \exists\, p \in \{1,2\}^{\infty} : \\ & \pi_2((\{1\} \times [\![\, E \,]\!]) \circledT (p,h)) \in s_1 \wedge \\ & \pi_2((\{2\} \times [\![\, E \,]\!]) \circledT (p,h)) \in s_2\} \end{aligned} \tag{4}$$

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.

For $s_1 \succsim s_2$ we have the constraint that events on a lifeline from s_1 should come before events from s_2 on the same lifeline:

$$\begin{aligned} s_1 \succsim s_2 \quad \stackrel{\mathsf{def}}{=} \quad & \{h \in H \mid \exists\, h_1 \in s_1, h_2 \in s_2 : \forall\, l \in L : \\ & e.l \circledS h = e.l \circledS h_1 \frown e.l \circledS h_2\} \end{aligned} \tag{5}$$

$e.l$ denotes the set of events that may take place on the lifeline l. Formally:

$$e.l \stackrel{\mathsf{def}}{=} \{e \in [\![\, E \,]\!] \mid (k.e = ! \wedge tr.e = l) \vee (k.e \in \{\sim, ?\} \wedge re.e = l)\} \tag{6}$$

Time constraint is defined as

$$s \wr C \overset{\mathsf{def}}{=} \{h \in s \mid h \models C\} \tag{7}$$

where $h \models C$ holds if for all possible assignments of timestamps to timestamp tags done by h, there is an assignment of timestamps to the remaining timestamp tags in C (possibly none) such that C evaluates to true. For example, if

$$h = \langle (k_1, m_1, t_1 \mapsto r_1), (k_2, m_2, t_2 \mapsto r_2), (k_3, m_3, t_2 \mapsto r_3) \rangle, \qquad C = t_1 < t_2$$

then $h \models C$ if $r_1 < r_2$ and $r_1 < r_3$.

3.3 Interaction Obligations

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.

O denotes the set of interaction obligations. Parallel execution, weak sequencing and time constraint are overloaded from sets of traces to interaction obligations as follows:

$$(p_1, n_1) \parallel (p_2, n_2) \overset{\mathsf{def}}{=} (p_1 \parallel p_2, (n_1 \parallel (p_2 \cup n_2)) \cup (n_2 \parallel p_1)) \tag{8}$$

$$(p_1, n_1) \succsim (p_2, n_2) \overset{\mathsf{def}}{=} (p_1 \succsim p_2, (n_1 \succsim (n_2 \cup p_2)) \cup (p_1 \succsim n_2)) \tag{9}$$

$$(p, n) \wr C \overset{\mathsf{def}}{=} (p \wr C, n \cup (p \wr \neg C)) \tag{10}$$

An obligation pair (p, n) is contradictory if $p \cap n \neq \varnothing$.

The operators for parallel execution, weak sequencing and time constraint are also overloaded to sets of interaction obligations:

$$O_1 \parallel O_2 \overset{\mathsf{def}}{=} \{o_1 \parallel o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \tag{11}$$

$$O_1 \succsim O_2 \overset{\mathsf{def}}{=} \{o_1 \succsim o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \tag{12}$$

$$O_1 \wr C \overset{\mathsf{def}}{=} \{o_1 \wr C \mid o_1 \in O_1\} \tag{13}$$

We also define an operator for inner union of sets of interaction obligations:

$$O_1 \uplus O_2 \overset{\mathsf{def}}{=} \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in O_1 \wedge (p_2, n_2) \in O_2\} \tag{14}$$

3.4 Semantics of Sequence Diagrams

The semantics of sequence diagrams is defined by a function

$$[\![\_]\!] \in D \to \mathbb{P}(O)$$

that for any sequence diagram d yields a set $[\![d]\!]$ of interaction obligations.

An event is represented by infinitely many unary positive traces – one for each possible assignment of timestamp to its timestamp tag:

$$[\![\ (k, m, t)\ ]\!] \overset{\mathsf{def}}{=} \{(\{\langle (k, m, t \mapsto r)\rangle \mid r \in \mathbb{R}\}, \varnothing)\} \quad \text{if } (k, m, t) \in E \tag{15}$$

The neg construct defines negative traces:

$$[\![\ \mathsf{neg}\, d\ ]\!] \overset{\mathsf{def}}{=} \{(\{\langle\rangle\}, p \cup n) \mid (p, n) \in [\![\ d\ ]\!]\} \tag{16}$$

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.

The assert construct makes all inconclusive traces negative. Except for that the sets of positive and negative traces are left unchanged:

$$[\![\ \mathsf{assert}\, d\ ]\!] \overset{\mathsf{def}}{=} \{(p, n \cup (H \setminus p)) \mid (p, n) \in [\![\ d\ ]\!]\} \tag{17}$$

Note that contradictory obligation pairs remain contradictory.

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

$$[\![\ d_1\ \mathsf{alt}\ d_2\ ]\!] \overset{\mathsf{def}}{=} [\![\ d_1\ ]\!] \uplus [\![\ d_2\ ]\!] \tag{18}$$

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

$$[\![\ d_1\ \mathsf{xalt}\ d_2\ ]\!] \overset{\mathsf{def}}{=} [\![\ d_1\ ]\!] \cup [\![\ d_2\ ]\!] \tag{19}$$

Notice that the sets of negative traces are not combined as is the case with the alt. This is due to the fact that we want to allow behaviors that are positive in one interaction obligation to be negative in another interaction obligation. The intuition behind this is as follows: All positive behaviors in an interaction obligation serve the same overall purpose, e.g. different ways of making beef. Alternative ways of making beef can be introduced by the alt operator. Hence, a behavior cannot be present in both the positive and negative trace sets of an interaction obligation as this would lead to a contradictory specification. However, behaviors specified by different interaction obligations are meant to serve different purposes, e.g. make beef dish and make vegetarian dish. There is nothing wrong about stating that a behavior which is positive in one interaction obligation is negative in another. E.g. steak beef would definitely be positive in a beef context and negative in a vegetarian context. By insisting on separate negative sets of interaction obligations we achieve this wanted property.

The par construct represents a parallel merge. Any trace involving a negative trace will remain negative in the resulting interaction obligation:

$$\llbracket\ d_1 \,\mathsf{par}\, d_2\ \rrbracket \quad \overset{\mathsf{def}}{=} \quad \llbracket\ d_1\ \rrbracket \parallel \llbracket\ d_2\ \rrbracket \tag{20}$$

The seq construct defines weak sequencing which is the implicit composition mechanism combining constructs of a sequence diagram. For explicit composition, the combined fragments are used:

$$\llbracket\ d_1 \,\mathsf{seq}\, d_2\ \rrbracket \quad \overset{\mathsf{def}}{=} \quad \llbracket\ d_1\ \rrbracket \succsim \llbracket\ d_2\ \rrbracket \tag{21}$$

The tc construct defines the effect of a time constraint. The positive traces of the operand that do not fulfill the constraint become negative in the result. The negative traces of the operand remain negative regardless of whether they fulfill the constraint:

$$\llbracket\ d \,\mathsf{tc}\, C\ \rrbracket \quad \overset{\mathsf{def}}{=} \quad \llbracket\ d\ \rrbracket \wr C \tag{22}$$

4 Two Abstractions

An example to illustrate the importance of distinguishing between the message reception and the message consumption event when dealing with timed specifications goes as follows: A restaurant chain specifies in a sequence diagram (see Figure 3) that it should never take more than 10 minutes to prepare a beef dish. The specification is handed over to the local restaurant owner who takes these requirements as an input to the design process of her/his local restaurant. When testing the time it takes to prepare a beef the restaurant finds that it is in accordance with the timing requirements. However, when the restaurant chain inspector comes to verify that the timing policies of the chain are obeyed in the operational restaurant he finds that it takes much longer time than 10 minutes to prepare the beef. Thus, the inspector claims that the restaurant is not working according to the timing requirements while the restaurant owner claims they are working according to the requirements. Who is right? According to UML both are right as there is no notion of buffering of communication in UML. Whether the message arrival of "main dish please" to the kitchen shall be regarded as message reception or consumption is not defined in the semantics of UML, and hence, it is up to the users of the diagrams to interpret the meaning.

In this section we define two abstractions over the triple event semantics that match the two different views in the example above, namely the standard interpretation and the black-box interpretation.

4.1 Standard Interpretation

The standard interpretation is meant to represent the traditional way of interpreting graphical sequence diagrams, namely that the input event of a message

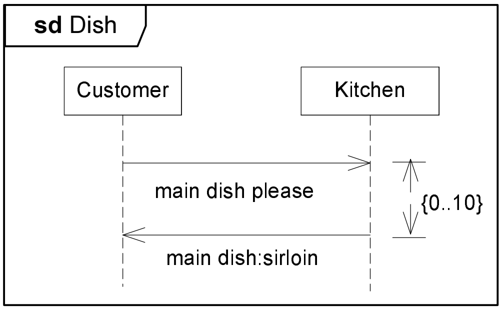

Fig. 3. Restaurant specification with time constraint

at a lifeline represents a consumption. We then only take send (!) and consume (?) events into consideration. Thus, we abstract away the fact that a message will arrive and be stored before it is consumed by the object. The standard interpretation sees graphical sequence diagrams like the diagram in Figure 3 as "standard diagrams".

The semantics of standard diagrams is defined in exactly the same manner as for general sequence diagrams in Section 3, except that the semantics of events is redefined as follows:

$$[\![\ (k, m, t) \ ]\!] \overset{\mathsf{def}}{=} \{(\{h' \frown \langle (k, m, t \mapsto r) \rangle \frown h'' \in H \mid h', h'' \in E(l, \sim)^{\omega} \wedge \#h' < \infty \wedge r \in \mathbb{R}\}, \varnothing)\} \tag{23}$$

where $l = tr.e$ if $k.e = !$ and $l = re.e$ if $k.e = ?$.

This definition says essentially that in a standard diagram, reception events may happen anywhere on the relevant lifeline (as long as the well-formedness conditions (2) and (3) are obeyed) since they are considered irrelevant in this setting.

If we apply the standard interpretation to the diagram in Figure 3, every positive trace h is such that

$$\{e \in [\![\ E \ ]\!] \mid k.e \neq \sim\} \circledS h = \langle (!, m, t_1 \mapsto r_1), (?, m, t_3 \mapsto r_3), (!, n, t_4 \mapsto r_4), (?, n, t_6 \mapsto r_6) \rangle$$

where $r_4 \leq r_3 + 10$, m stands for "main dish please" and n stands for "main dish:sirloin". The implicit reception of m can happen at any point between the corresponding transmission and consumption events, and similarly for n (and any other message).

4.2 Black-Box Interpretation

The black-box interpretation represents the view where the input event of a message at a lifeline represents a reception event. The black-box interpretation sees graphical sequence diagrams like the diagram in Figure 3 as "black-box diagrams".

As with standard diagrams, the semantics of black-box diagrams is defined in exactly the same manner as for general sequence diagrams in section 3 except that the semantics of events is redefined as follows:

$$\llbracket\ (k, m, t)\ \rrbracket \overset{\mathsf{def}}{=} \{(\{h' \frown \langle (k, m, t \mapsto r) \rangle \frown h'' \in H \mid h', h'' \in E(l, ?)^{\omega} \wedge \#h' < \infty \wedge r \in \mathbb{R}\}, \varnothing)\} \tag{24}$$

where $l = tr.e$ if $k.e = !$ and $l = re.e$ if $k.e = \sim$.

If we apply the black-box interpretation to the diagram in Figure 3, every positive trace h is such that

$$\{e \in \llbracket\ E\ \rrbracket \mid k.e \neq ?\} \circledS h = \langle (!, m, t_1 \mapsto r_1), (\sim, m, t_2 \mapsto r_2), (!, n, t_4 \mapsto r_4), (\sim, n, t_5 \mapsto r_5) \rangle$$

where $r_4 \leq r_2 + 10$. Note that we do not impose any constraint on the implicit consumption events, except that the consumption cannot take place before its reception (if it takes place at all).

5 The General Case

We have shown that input events are most naturally (standard) interpreted as consumption when they appear on lifelines that represent atomic processes and their concrete implementations should be derived from the lifelines. We have also shown that there are reasons, e.g. timing constraints, that sometimes make it necessary to consider the input event as representing the reception. Moreover, we have seen that timing constraints may also make good sense when applied to consumption events.

In fact we believe that notation for both reception and consumption events are necessary, but that most often for any given message a two-event notation will suffice. Sometimes the message will end in the reception and sometimes in the consumption, but seldom there is a need to make both the reception and the consumption explicit. There are, however, exceptions where all three events must be present to convey the exact meaning of the scenario. Hence, we will in the following introduce graphical notation in order to be able to explicitly state whether a message input event at a lifeline shall be interpreted as a reception event or a consumption event. That is, whether standard or black-box interpretation shall be applied.

Figure 4 shows the graphical notation to specify that a message input event at a lifeline shall be interpreted as a consumption event. Syntactically this notation is equal to the one applied for ordinary two-event sequence diagrams.

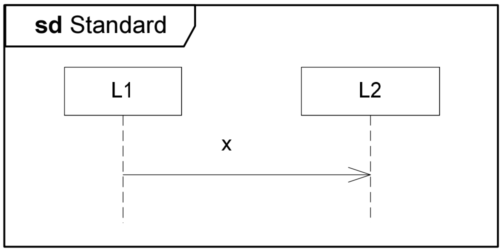

Fig. 4. Graphical syntax for specifying standard interpretation

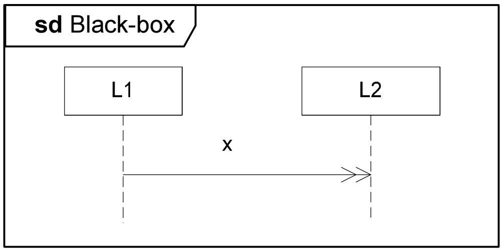

Fig. 5. Graphical syntax for specifying black-box interpretation

We express that a message input event at a lifeline shall be interpreted as a reception event and thus be given black-box interpretation by the double arrowhead as shown in Figure 5. We will in the following give some examples of the full approach describing reception events as well as consumption events explicitly.

Let us return to the dinner example where we may assume that the cook is not really one single person, but actually a chef and his apprentice. We may decompose the cook lifeline into new sequence diagrams where the chef and apprentice constitute the internal lifelines. We have shown this in Figure 6 for the preparation of beef shown originally in Figure 2. Let us assume that the apprentice wants to go and get the meat before heating the stove. His priorities may be so because heating the stove is more of a burden, or because the refrigerator is closer at hand. For our purposes we would like to describe a scenario that highlights that the apprentice fetches the meat before heating the stove even though he received the order to turn on the heat first.

In Figure 6 we have shown some explicit reception events, but we have chosen not to show explicitly the corresponding consumptions. This is because our needs were to describe the relationship between the receptions and the actions (outputs) of the apprentice.

The consumptions were considered less important. The disadvantage of this is that we cannot from Figure 6 deduce whether the apprentice actually fetched the meat because he received the order "go fetch meat" or the order "go turn on heat". The reader should appreciate that the "fetch_meat" message crosses the other messages only due to the need to graphically let the formal gates match the

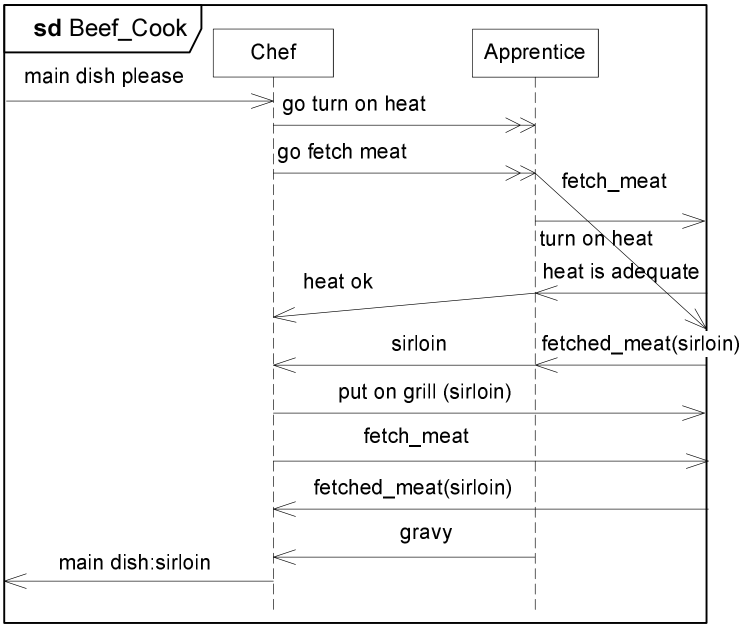

Fig. 6. Prioritizing to fetch the meat

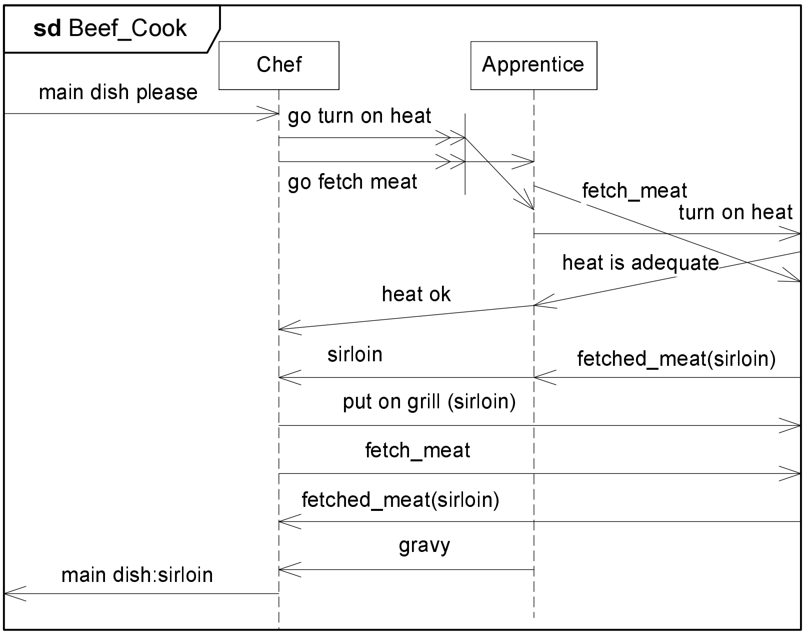

Fig. 7. The whole truth

events on the decomposed Cook lifeline shown in Figure 2. Semantically there is no ordering between gates. For a formal treatment of gates, see [HHRS04].

If we want to give an even more detailed account of the apprentice's options, we may introduce both reception and consumption events. We have done so in Figure 7.

In Figure 7 we see that the chef instructs the apprentice to go turn on heat and to go and fetch meat. The apprentice makes independent decisions for the order of consumption. Here he has decided to consume the order to go fetch meat before consuming go turn on heat. Now we can easily see that the apprentice reacts adequately to the consumptions. It is of course rather risky for the apprentice not to react immediately to the chef's order to turn on the heat, but we may remedy this by timetagging the message receptions of "go turn on heat" and "go fetch meat". Then we specify that the scenario is only valid if these receptions are sufficiently close together in time by a formula including these time tags.

As the examples in this section demonstrate, we have cases where we need to explicitly describe both reception and consumption events in the same diagram, but seldom for the same message. This means that general diagrams may contain standard, black-box as well as three-event arrows. The semantics of such diagrams is fixed by the definitions in Section 3 with two exceptions:

- The semantics of a consumption event of a standard arrow should be as for consumption events in the standard case (see Section 4.1).
- The semantics of a receive event of a black-box arrow should be as for receive events in the black-box case (see Section 4.2).

6 Refinement

Refinement means to add information to a specification such that the specification becomes closer to an implementation. The set of potential traces will be narrowed and situations that we have not yet considered will be supplemented. We define formally two forms of refinement - glass-box refinement which takes the full semantics of the diagram into account, and black-box refinement which only considers changes that are externally visible.

Negative traces must always remain negative in a refinement, while positive traces may remain positive or become negative if the trace has been cancelled out. Inconclusive traces may go anywhere.

6.1 Definition of Glass-Box Refinement

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

$$n_1 \subseteq n_2 \quad \wedge \quad p_1 \subseteq p_2 \cup n_2 \tag{25}$$

A set of interaction obligations O'_1 is a glass-box refinement of a set O_1, written $O_1 \rightsquigarrow_g O'_1$, iff

$$\forall\, o \in O_1 : \exists\, o' \in O'_1 : o \rightsquigarrow_r o' \tag{26}$$

A sequence diagram d' is then a glass-box refinement of a sequence diagram d, written $d \rightsquigarrow_g d'$, iff

$$[\![\ d \ ]\!] \rightsquigarrow_g [\![\ d' \ ]\!] \tag{27}$$

The refinement semantics supports the classical notion of compositional refinement providing a firm foundation for compositional analysis, verification and testing. In [HHRS04] we prove that refinement as defined above is transitive. We also prove that it is monotonic with respect to the operators presented in Section 3.4, except from assert. For assert, we have monotonicity in the special case of narrowing defined below.

6.2 Supplementing and Narrowing

Supplementing and narrowing are special cases of the general notion of refinement. 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

$$(n_1 \subset n_2 \quad \wedge \quad p_1 \subseteq p_2) \qquad \vee \qquad (n_1 \subseteq n_2 \quad \wedge \quad p_1 \subset p_2) \tag{28}$$

Narrowing reduces the allowed 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

$$p_2 \subset p_1 \quad \wedge \quad n_2 = n_1 \cup (p_1 \setminus p_2) \tag{29}$$

6.3 Example of Glass-Box Refinement

We want to refine the Beef_Cook diagram presented in Figure 7. In a glass-box refinement we are interested in the complete traces described by the diagram, and a selection and/or a supplement of these traces.

Figure 8 is a glass-box refinement of Figure 7. In this diagram we state that we no longer want gravy, but Beárnaise sauce instead. Defining gravy as negative is a narrowing, as it means to reduce the set of positive traces of the original specification. The traces with Beárnaise sauce was earlier considered inconclusive (i.e. neither positive nor negative), but are now defined as positive. This is an example of supplementing. In addition, the diagram in Figure 8 permits using no sauce at all. This is because the neg fragment also introduces the empty trace ($\langle\rangle$) as positive. We summarize these changes in Figure 9.

6.4 Definition of 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.:

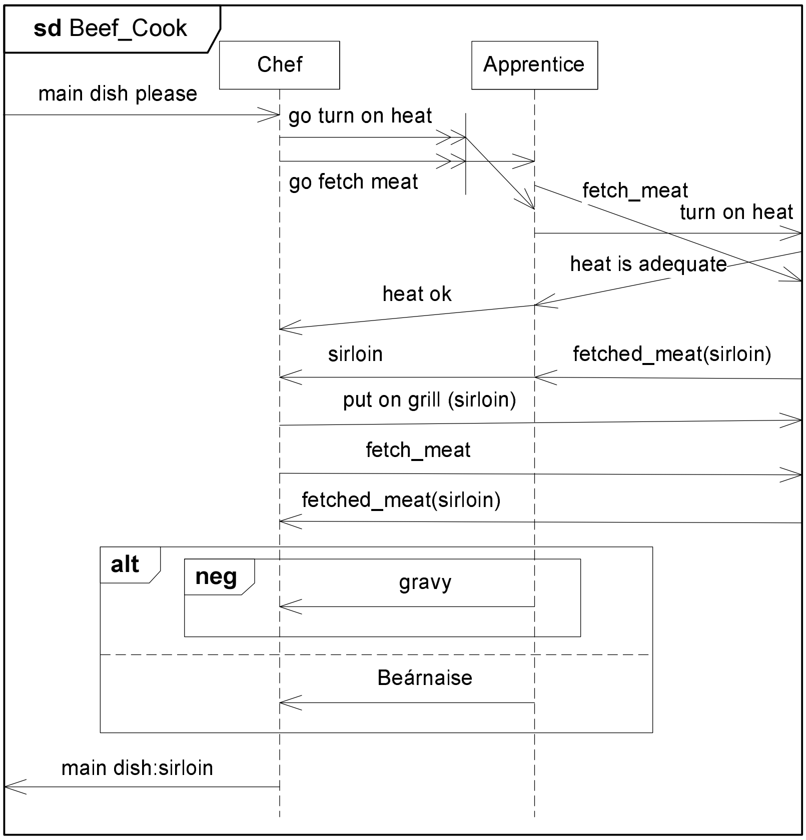

Fig. 8. Glass-box refinement of Beef_Cook

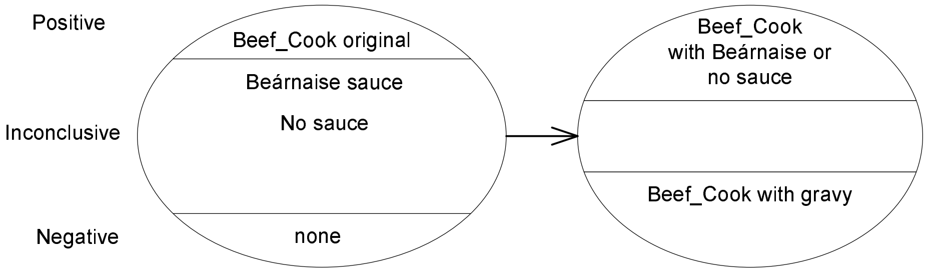

Fig. 9. Summary glass-box refinement

$$ext(h, l) \overset{\mathsf{def}}{=} \{e \in [\![E]\!] \mid (tr.e \notin l \vee re.e \notin l) \wedge k.e \neq ?\} \circledS h \qquad (30)$$

The *ext* operator is overloaded to sets of traces, to pairs of sets of traces, and sets of pairs of sets of traces in the standard pointwise manner, e.g.:

$$ext(s, l) \overset{\mathsf{def}}{=} \{ext(h, l) \mid h \in s\} \tag{31}$$

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

$$\forall\, o \in [\![\ d\ ]\!] : \exists\, o' \in [\![\ d'\ ]\!] : ext(o, ll(d)) \rightsquigarrow_r ext(o', ll(d')) \tag{32}$$

Notice that the *ext* operator also filters away all consumption events regardless of lifeline, as was the case with black-box interpretation of sequence diagrams. Thus, black-box refinement is mainly relevant in the context of black-box interpretation (even though it may also be applied to standard diagrams).

6.5 Example of Black-Box Refinement

It is obvious from the definition of black-box refinement that any glass-box refinement is also a black-box refinement. What would be a black-box refinement in our Beef_Cook context, but not a glass-box refinement? If we in a refinement of the specification in Figure 7 had just replaced the gravy with Bearnaise, this change would not affect the externally visible behavior of Beef_Cook as it is defined, and would therefore be a legal black-box refinement. However, it would not be a glass-box refinement since the traces involving gravy have been lost (they are now inconclusive), and this violates the definition.

6.6 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 Beef_Cook in Figure 6. The messages that originally (in Figure 2) had the Cook as sender/receiver, now have the chef or the apprentice 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 understood as the transitive and reflexive closure of lifeline decomposition and interface refinement.

6.7 Refinement Through Time Constraints

Having given examples of refinement in terms of pure event manipulation and trace selection, we go on to present an example where time constraints represent the refinement constructs.

We will now introduce two time refinements as indicated in Figures 10 and 11. First we would like to make sure that beefs are neither burned nor raw when fetched from the stove. To make sure that this constraint holds we will put the time constraint on the consumption event of the "put on grill" message. This is because it is the time that the beef is actually present on the stove that matters with respect to how much it is grilled, not the time the beef lies on a plate beside the stove waiting for free space on the stove. All behaviors that do not meet this time constraint are considered negative according to definition (22) of time constraint semantics. Traces that originally were positive are because of the new time constraint now defined as negative. Thus, this step constitutes a glass-box refinement according to definition (27). In fact, it is a narrowing as defined by definition (29). Since the consumption events and transmit events locally define the object behavior, it is only the behavior of the stove that is affected by this refinement step, and not the environment. Using double arrowhead on the "put on grill" message we would not be able to express the intended refinement because it is necessary to talk about message consumption. On the other hand, comparing Figure 10 with the original diagram in Figure 2, we have replaced a standard arrow with a three-event arrow. This is a valid refinement, as it means to make explicit one of the implicit reception events that are already present in the semantics of Figure 2.

Next we would like to limit the overall time it takes to prepare a beef. This represents a customer requirement on the kitchen as illustrated in Figure 11. However, the customer does not care about the details of beef preparation, just that it is prepared in time. As seen from Figure 11 this can be interpreted as a time constraint on the reception event. In the same manner as with the glass-box refinement above, the introduction of the time constraint is a narrowing that "moves" traces from the set of positive traces to the set of negative traces. We are not concerned about where the beef spends its time in the kitchen during the preparation process, just that it is prepared in time.

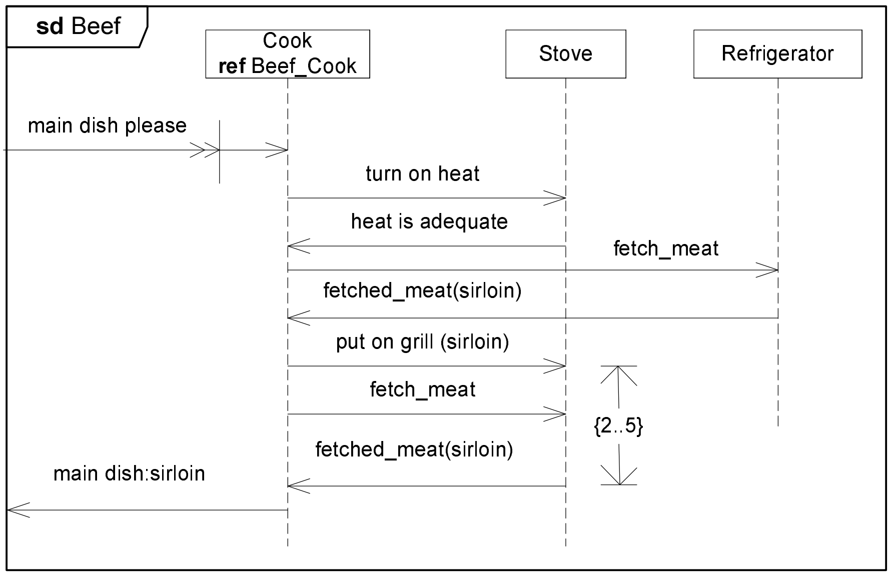

Fig. 10. Imposing constraints on timing

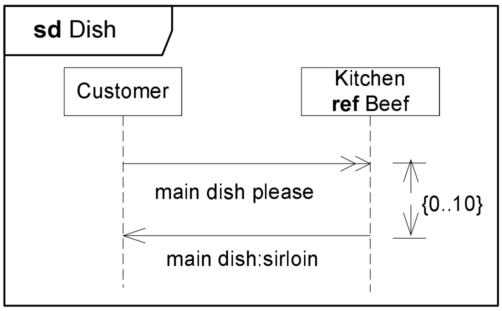

Fig. 11. Customer requirement on the beef preparation time

7 Conclusions and Related Work

We have presented Timed STAIRS, a formal approach to the step-wise, incremental development of timed sequence diagrams. It is based on trace semantics. Traces are sequences of events. Events are representations of sending, receiving and consuming messages.

Three event semantics of sequence diagrams has been considered before. In [EMR97] the event ordering imposed by the MSCs is used to determine the physical architecture needed for implementing the specified behavior such as a FIFO buffer between each of the processes. In Timed Stairs we implicitly assume that every object has one associated input buffer unless something else is explicitly specified in the diagrams. Thus, we do not deduce the communication architecture from the sequence diagrams but instead make it an option for the designer to explicitly specify the wanted architecture in the diagrams. The main rationale for introducing the three event semantics in Timed STAIRS is to be able to distinguish between reception and consumption of messages in order to specify time-constraints on black-box behavior as well as message consumption. Hence, the purpose of the three event semantics is quite different from [EMR97] where time and black-box behavior is not considered.

To consider not only positive traces, but also negative ones, has been suggested before. In [Hau95] 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 [ITU93] 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 [ITU94] 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 [ITU98] and [Ren98] 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 [CPRO95] 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) [DH99]. 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 [Hau01] (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 Timed 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 Timed 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 executions accepted by the LSC is produced. These executions 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 [HM03].

Our approach is similar to LSC since it is basically interleaving. Timed 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 interaction obligations.

The work by Krüger [Krü00] 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 Timed STAIRS corresponds closely to property refinement in [Krü00] 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 relation.

Although this paper presents Timed 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. Timed STAIRS may also be adapted to support LSC. Timed STAIRS is complementary to software development processes based on use-cases, and classical object-oriented approaches such as the Unified Process [JBR99]. Timed 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, Atle Refsdal, Ina Schieferdecker and Fredrik Seehusen for helpful feedback.

References

[BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement.* Springer, 2001.

[CPRO95] P. Combes, S. Pickin, B. Renard, and F. Olsen. MSCs to express service requirements as properties on an SDL model: Application to service interaction detection. In *7th SDL Forum (SDL'95)*, pages 243–256. North-Holland, 1995.

[DH99] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, pages 293–311. Kluwer, 1999.

[EMR97] A. Engels, S. Mauw, and M.A. Reniers. A hierarchy of communication models for message sequence charts. In *Formal Description Tecniques and Protocol Specification, Testing and Verification*, pages 75–90. Chapman & Hall, 1997.

[Hau95] Ø. Haugen. Using MSC-92 effectively. In *7th SDL Forum (SDL'95)*, pages 37–49. North-Holland, 1995.

[Hau01] Ø. Haugen. MSC-2000 interaction diagrams for the new millennium. *Computer Networks*, 35:721–732, 2001.

[HHRS04] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2004.

[HM03] D. Harel and R. Marelly. Specifying and executing behavioral requirements: The play-in/play-out approach. *Software and System Modeling*, 2:82–107, 2003.

[HS03] Ø. Haugen and K. Stølen. STAIRS – Steps to analyze interactions with refinement semantics. In *Sixth International Conference on UML (UML'2003)*, number 2863 in Lecture Notes in Computer Science, pages 388–402. Springer, 2003.

[ITU93] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1993.

[ITU94] International Telecommunication Union. *Recommendation Z.120 Annex B: Algebraic Semantics of Message Sequence Charts*, 1994.

[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] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[Krü00] I. Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.

[OMG00] Object Management Group. *Unified Modeling Language, Version 1.4*, 2000.

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

[Ren98] M. A. Reniers. *Message Sequence Chart: Syntax and Semantics*. PhD thesis, Eindhoven University of Technology, 1998.