Semantics of Interactions in UML 2.0

Harald Störrle
Ludwig-Maximilians-Universität München
Oettingenstr. 67
80538 München
GERMANY
stoerrle@informatik.uni-muenchen.de

Abstract

The recent major revision of the UML (see [4]) has introduced significant changes and additions. In particular, Message Sequence Charts (MSC) according to the ISO standard (see [2])have been integrated. In UML, the concept underlying these notations is called interaction. In this paper I shall look at its definition, defining a formal, yet straightforward trace semantics, including time.

1. Introduction

There are many variants of Message Sequence Charts (MSCs, see [2])[1], including that defined in the UML. The UML is the "*lingua franca of software engineering*", and it has recently undergone a major revision (advancing from version 1.4 to version 2.0). As expected, the proposal of the "U2-group" has been chosen from among the contending proposals for the standard, including a complete redefinition of Interactions.[2] Unfortunately, the standard has yet again failed to define a formal semantics, as would be necessary to take full advantage of the UML, e.g., in automated tools.

A semantics is a transformation of entities with intuitively explained meaning into entities with formally defined meaning. By definition, such a transformation cannot be proven right (or wrong). The only hope one may have is to be able to make the transition convincing to such an extent, that there can be no reasonable doubt concerning its correctness. Completeness, in the other hand, is trivial.

[1] Unfortunately, I lack the space to discuss them properly here. See [3] and [5] for exhaustive surveys.

[2] Throughout this paper, I shall print UML metaclasses like Interaction in sans serif in order to distinguish them from ordinary vocabulary and to avoid awkward and tedious expressions like "interactions in the sense of UML" or similar.

In this paper I first explain the new definition of interactions and the differences to the corresponding definitions in UML 1.4, which is the current standard. Then I define a straightforward semantics of Interactions including time.

2. Interactions in UML 2.0 vs. UML 1.4

In the remainder I shall refer to the UML 2.0 as the "new standard" or simply "the standard" while I refer to the version 1.4 as the "old standard".

2.1. Concrete Syntax

First of all, all diagrams now have a frame around them and a compartment displaying its type and name which makes it easier to refer to it, e.g. as a subdiagram or companion diagram.

In the old standard, there were two types of interaction diagrams, namely sequence and collaboration diagrams which both are based on the same metamodel concepts (see below). So called "*metric sequence diagrams*"had been mentioned in UML 1.3, but neither defined nor explained, and have been abandoned in UML 1.4.

In the new standard, collaboration diagrams have been renamed to communication diagrams. On the same level of communication and sequence diagrams, there are now also timing diagrams as known in many engineering disciplines (see Figure 1). Timing diagrams may be considered as an elaboration of metric sequence diagrams. While communication diagrams and sequence diagrams focus on structure and message exchange, respectively, timing diagrams focus on state and state change across time.

All these interaction diagrams may be combined ad lib by a given set of InteractionOperators. The notation is similar to that of interaction diagrams in general (see Figure 1). If there are two arguments to an InteractionOperator, they are divided by a dashed line. The notation is

strongly reminiscent of MSCs. Also, in the new standard there are now interaction overview diagrams which are basically activity diagrams where the interaction diagrams are activities. They correspond to High-Level MSCs (see [2]).

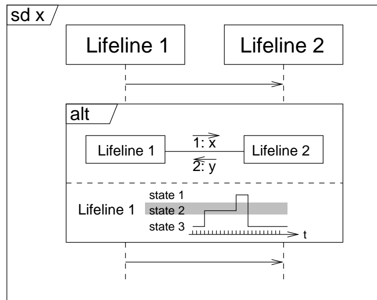

Figure 1. A UML 2.0 interaction diagram, including the high-level operator alt**, a sequence diagram (called** sd x**), a communication diagram (top) and a timing diagram (bottom).**

Another novelty in UML 2.0 interaction diagrams are so called gates. Sometimes, interactions become rather large, or contain the same idiom over and over again. In these cases, it is convenient to split the interaction up over several diagrams and link them together. The mechanism to do this are Gates, which serve as a kind of symbolic link.

Gates split up messages, that is, in one diagram, a message may end at a labeled gate, while in another diagram, the same message may start at a gate with the same label. This is not only convenient for splitting up large diagrams, it also solves the problem of where the initiating stimulus comes from in a sequence diagram. Gates have no syntax as such: if a message arrow simply ends at the bounding box of a diagram, then there has to be an underlying gate (see Figure 2).

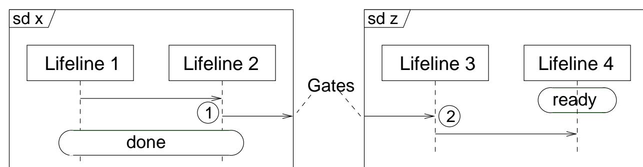

Figure 2. The rectangles with rounded edges denote states of (sets of) entities represented by lifelines. Arrows ending at diagram borders represent gates. The circled numbers indicate Event occurrences.

Time annotations can now be introduced into interaction diagrams either as a metric time scale in timing diagrams (see Figure 1), or as constraints on the duration of states or message transmissions, see Figure 3.

Yet another improvement in UML 2.0 sequence diagrams are state annotations (see Figure 2). Since these do not affect the sets (sic!) of traces, but serve as annotations for better understanding by users, they may be ignored for the time being.

2.2. Abstract Syntax

In this section, I give only a brief account of the most important portions of the metamodel concerning Interactions. The metamodel underlying interaction diagrams has completely changed. Compared to the metamodel of UML 1.4, the notion of Collaboration has gone, and the notions of Message and Interaction have been redefined. The informal notion of lifeline has now become a metaclass, and there is a large number of new notions, including EventOccurrence, InteractionFragment, CombinedFragment, MessageEnd and Gate, which I shall now explain.

An Interaction has sets of Lifelines, Messages and InteractionFragments. Since Interaction is a subclass of InteractionFragment, Interactions may contain any number and kind of Interactions. One important kind of Interactions are CombinedFragments, which consist of an InteractionOperator and a number of InteractionOperands, which may be either plain Interactions or again CombinedFragments.Thus, CombinedFragments really are expressions of Interactions. Figure 4 illustrates this portion of the metamodel.

An Interaction basically correspond to simple interactions, as known from UML 1.4, plus the high-level operators of MSCs. A Lifeline represents the EventOccurrences taking place at one of the entities involved in the Interaction.

Messages consist of up to two MessageEnds that represent either an EventOccurrence like sending, receiving, loosing or finding a message, or a Gate, that is, a reference to some other interaction diagrams (see Figure 4). There is only an indirect connection between Messages and the Connectors on which they are exchanged. Messages of an Interaction may be partially ordered by a GeneralOrdering.

2.3. Intuitive Semantics

In the old standard, relatively little had been said about the semantics of Interactions, and both true concurrency and interleaving semantics have been proposed (see e.g. [6] and [7], respectively). In the new standard, this is different. Elementary InteractionFragments still are defined as having a

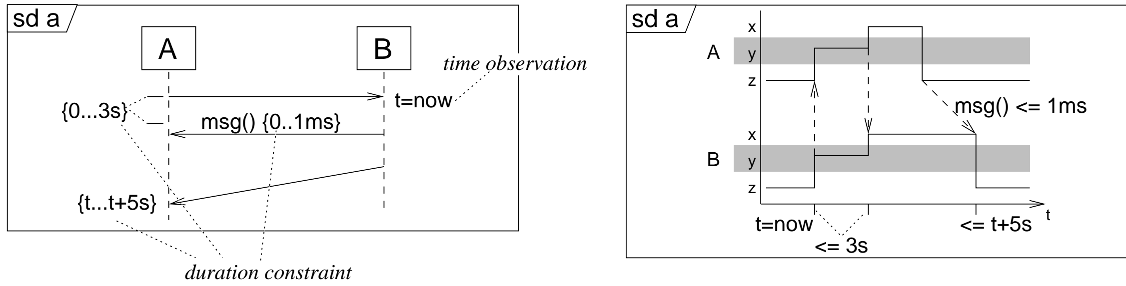

Figure 3. Sequence diagram with timing constraints (left); the same interaction presented as a timing diagram (right). Note that I have replaced the rather cumbersome "$\{a \dots a + b\}$" by the equally expressive and more succinct "$\leq b$". Note also, that I have introduced arrows for sending messages.

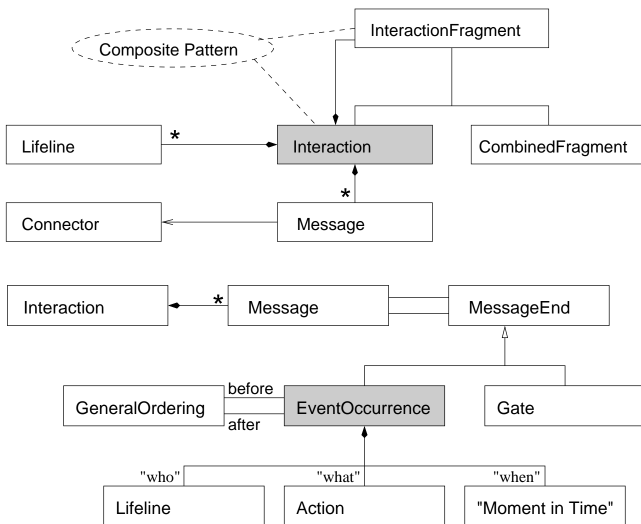

Figure 4. The portion of the UML 2.0 metamodel around Interaction **(top) and** EventOccurrence **(bottom).**

partial order semantics defined by the messages exchanged between the participants of the interactions, supplemented by a GeneralOrdering that "*[...] provides the ability to define partial orders of EventOccurrences[...]* " (cf. [4, p. 377]). However, "*the sequences of EventOccurrences are the meanings of Interactions*" (cf. [4, p. 367]). Thus, the overall semantics are simply interleaving traces. More precisely, "*the semantics of an Interaction is given as a* pair *of sets of traces*" (cf. [4, p. 378, emphasis added]), representing "*valid traces and invalid traces*" (ibid.), respectively. "*The traces that are not included [in the union of th two] are not described [...] and we cannot know whether they are valid or invalid*" (ibid.).

This point of view is obviously adopted form Life Sequence Charts (cf. [1]). This gives riese to a number of problems in connection with the assert- and negate-operators (see Section 4.3). As a convention, I write pairs of valid and invalid traces of an interaction P as $\langle P^+, P^- \rangle$. The usual operations on sets may be extended canonically to the componentwise application of these operations of pairs of sets, e.g., $\langle A^+, A^- \rangle \cup \langle B^+, B^- \rangle = \langle A^+ \cup B^+, A^- \cup B^- \rangle$, respecting order, where appropriate.

Intuitively, one might view an Interaction as a set of observations of a system using probes on different parts of it. Every part that is being probed is represented as a lifeline. The measurements of all the probes are recorded one at a time.

3. Basic features of UML 2.0 interactions

In this section I deal only with plain InteractionFragments and CombinedFragments made up from InteractionFragments and InteractionOperators. For the time being, I ignore all the difficult cases like timing constraints, message contents, gates and so on. I shall treat them in the following section.

3.1. Semantic domains and basic definitions

I now define a number of semantic domains on which to base the formalization. All the following quotations are taken from [4, p. 371ff.].

Since "*an EventOccurrence is the basic semantic unit of Interactions*", this is where I start off. The standard declares that "*EventOccurrences represent moments in time to which Actions are associated*". Thus, I define the domains of Timepoints (written $\mathcal{T}$) and Actions (written $\mathcal{A}$). For the time being, timestamps are just symbolic labels that allow us to identify the respective systems' state so as to satisfy

the requirement that "*an event occurrence will also include information about the values of all relevant objects at this point in time*" (cf. [4, p. 379]). Also, every EventOccurrence must belong to some participant in the Interaction, which are represented as Lifelines, and so I define the domains Lifelines (written $\mathcal{L}$), and thus Event Occurrences (written $\mathcal{EO}$) as $\mathcal{EO} = \mathcal{T} \times \mathcal{A} \times \mathcal{L}$.

The standard declares that "*an EventOccurrence may have associated a Message that defines its content*", but I am just interested in the semantics, so that I may safely ignore the contents. However, "*that message has information about the sender and the receiver of the Message*" called MessageEnds, and these I have to take into account. MessageEnds come in two flavors, either as the participants in the communication, represented as a Lifeline, which I have already covered above. Or they come as a reference to some other Interaction, i.e., as a Gate. So I define the domain of Gates (written $\mathcal{G}$).

As "*the sequence of EventOccurences are the meanings of Interactions*", I can now finally define the domain of traces (written $\mathcal{SEQ}$) as $\mathcal{SEQ} = (\mathcal{EO} \cup \mathcal{G})^*$.

Also, the new standard commits to an interleaving semantics: "*The sequences of EventOccurences are the meanings of Interactions*" (cf. [4, p. 375]). This means that in the system modeled, there can be only ever one event happening at any given point in time, i.e. there are never concurrent events.

3.2. Plain Interaction Fragments

I first deal with plain InteractionFragments, that is, InteractionFragments without high-level operators. Such an InteractionFragment is a set of Lifelines, each of which is a sequence of EventOccurrences. Here, as in UML 1.4, the new standard defines a partial order-semantics. According to the metamodel, an Interaction contains a set of Lifelines each of which is a sequence of EventOccurrences connected by Messages. Such a sequence can be easily translated into an ordering relation $S \subseteq \mathcal{EO} \times \mathcal{EO}$ by a function order defined as

$$\begin{aligned} \mathsf{order}(\epsilon) &= \emptyset \\ \mathsf{order}(x) &= \{\langle x, x\rangle\} \\ \mathsf{order}(x.y.w) &= (\{\langle x, x\rangle, \langle x, y\rangle\} \cup \mathsf{order}(y.w)^* \end{aligned}$$

where x^* is the transitive closure of x.

Additionally, there may be GeneralOrderings between pairs of arbitrary EventOccurrences, and there may be Messages connecting a pair of a sending and a receiving EventOccurrences. I assume that these two additional sources of constraints come together as an ordering relation $A \subseteq \mathcal{EO} \times \mathcal{EO}$. So, the partial order yielded from a plain InteractionFragment is made up of the set $EO \subseteq \mathcal{EO}$ of event occurrences in the interaction and the partial ordering relation A.

This may now be sequentialized to yield the set of traces that is to be the semantics of the Interaction.

Definition 3.1 (sequentialization of partial order)
Let $p = \langle \Sigma_p, <_p \rangle$ be a partial order. Then the set of all sequential traces defined by p (written as $\mathsf{seq}(p)$) is defined as $\mathsf{seq}(p) = \{w \in \Sigma_p^* \mid \forall_{0<i<j\leq|w|} : w_i <_p^* w_j \implies i < j\}$ where $<_p^*$ is the transitive closure of $<_p$. □

With this definition, the mapping from the metamodel to a mathematical domain is now simply

$$[\![\mathsf{InteractionFragment}]\!] = \langle \mathsf{seq}(\langle EO, O\rangle), \emptyset\rangle$$

with EO the set of event occurrences as defined above, and $O = S_{[\![\mathsf{InteractionFragment}]\!]} \cup A_{[\![\mathsf{InteractionFragment}]\!]}$ for a plain InteractionFragment x, where S_x and A_x like S and A defined above.

3.3. Combined Fragments

I can now turn to the more complex InteractionFragments. They are basically expressions made up of an InteractionOperator and one or two InteractionFragments. I define the semantics casewise by the operators.

The definitions as given in the new standard are now examined and translated in turn. In the remainder, P and Q denote interaction fragments or simple activities.

3.3.1 Parallel composition

The standard declares: "*The EventOccurrences of the different operands can be interleaved in any way [. . .]* " (cf. [4, p. 369]). This calls for a simple shuffle-operator.

Definition 3.2 (shuffle)
Let $v, w \in \Sigma^*$ and ϵ the empty sequence. The **shuffling** of v and w (written $v \sqcup\!\sqcup\, w$) is defined recursively by

$$\begin{aligned} \epsilon \sqcup\!\sqcup\, w &= w \\ v \sqcup\!\sqcup\, \epsilon &= v \\ xv \sqcup\!\sqcup\, yw &= \{x(v \sqcup\!\sqcup\, yw), y(xv \sqcup\!\sqcup\, w)\} \end{aligned}$$

I also use the canonical extension to languages (i.e., sets of words), defined as $A \sqcup\!\sqcup\, B = \{a \sqcup\!\sqcup\, b \mid a \in A, b \in B\}$, and pairs of sets of words, defined as $\langle A^+, A^-\rangle \sqcup\!\sqcup\, \langle B^+, B^-\rangle$. □

Now parallel composition is easy to define as

$$[\![\mathsf{par}(P, Q)]\!] = [\![P]\!] \sqcup\!\sqcup\, [\![Q]\!].$$

3.3.2 Optional choice

The standard declares that "*an option is semantically equivalent to an alternative [. . .] where [. . .] the second operand is empty*" (cf. [4, p. 369]). So I have

$$[\![\mathsf{opt}(P)]\!] = [\![P]\!] \cup \langle\{\epsilon\}, \emptyset\rangle,$$

where $\langle P^+, P^-\rangle \cup \langle Q^+, Q^-\rangle = \langle P^+ \cup Q^+, P^- \cup Q^-\rangle$ for pairs of words.

3.3.3 Alternative choice

The standard declares: "*[. . .] The set of traces that defines a choice is the union of the (guarded) traces of the operands*" (cf. [4, p. 369]), which means

$$[\![\mathsf{alt}(P, Q)]\!] = [\![P]\!] \cup [\![Q]\!].$$

Guards are properties about the state of the entity represented by a lifeline. They may be added as state annotations in diagrams (see Figure 2), and may be referenced by the lifeline and the (symbolic or actual) time, which together form a unique key. However, since the states are only implicit in the traces, they are not be represented in the semantics at all (see also Section 5).

3.3.4 Strict sequencing

Strict sequencing is close to the intuitive notion of sequencing familiar from sequential systems - the first operand is executed first, and then the second. The standard defines: "*The semantics of the strict operator defines a strict ordering of the operands [. . .]* " (cf. [4, p. 369]). So the resulting set of traces is just the concatenation of every pair of traces from the first and the second set or

$$[\![\mathsf{strict}(P, Q)]\!] = [\![P]\!].[\![Q]\!]$$

where $V.W = \{v.w \mid v \in V, w \in W\}$ for languages V and W from an alphabet like $\mathcal{EO}$, and where concatenation is extended canonically for tuples.

However, the quotation given above is not quite complete. In full, it reads "*the semantics of the strict operator defines a strict ordering of the operands on the first level within the CombinedInteraction with operator strict*" (cf. [4, p. 369]). The meaning of this statement is unclear to me.

3.3.5 Weak sequencing

This form of sequencing is weak in the sense that only the EventOccurrences on Lifelines belonging to both operands are sequenced, while EventOccurrences on other Lifelines are interleaved. The standard imposes three conditions on the result of weak sequencing. "*(1) The ordering of EventOccurrences within each of the operands are maintained in the result. (2) EventOccurrences on different lifelines from different operands may come in any order.*" These two are easy to maintain.

The third condition is more tricky: "*(3) EventOccurrences on the same lifeline from different operands are ordered such that an EventOccurrence of the first operand comes before that of the second operand.*" The wording of the new standard allows two interpretations of this condition. First, it might be taken to mean that for those Lifelines shared among the two operators, strict sequencing applies. Under this interpretation, weak sequencing could easily be formalized as $[\![\mathsf{seq}(v.y, x.w)]\!] = (v \sqcup\!\sqcup\, x).(y \sqcup\!\sqcup\, w)$, where y and x are the largest suffixes and prefixes, respectively without EventOccurrences on the shared Lifelines. However, there is a stronger interpretation, for the third condition could also be interpreted to mean that the EventOccurrences of the first operands precede the EventOccurrences of the second operand only individually on the shared Lifelines, rather than collectively for *all* shared Lifelines. Figure 5 illustrates this difference.

The second interpretation allows for stronger parallelization, and hence a quicker execution of a concurrent system. Thus it is to be preferred, and so the semantics of weak sequencing is

$$[\![\mathsf{seq}(P, Q)]\!] = \langle P^+ \lhd Q^+, P^- \lhd Q^-\rangle$$

with $P \lhd Q =$

$$\left\{\begin{matrix} x \in p \sqcup\!\sqcup\, q \text{ where } p \in [\![P]\!], q \in [\![Q]\!] \text{ such that} \\ \forall_{l \in L} : \exists a, b, c \in \mathcal{SEQ} : x = a.\mathsf{max}(l, p).b.\mathsf{min}(l, q).c \end{matrix}\right\}$$

where L is the set of lifelines shared among P and Q, $\mathsf{max}(l, u) = \mathsf{min}(l, u^{\mathsf{rev}})$, $\mathsf{min}(l, \epsilon) = \epsilon$ and

$$\mathsf{min}(l, x.u) = \begin{cases} x & \text{if } x \text{ is occurring at } l \\ \mathsf{min}(l, u) & \text{otherwise} \end{cases}$$

for $l \in \mathcal{L}$, $x \in \mathcal{EO}$ and $u \in \mathcal{SEQ}$.

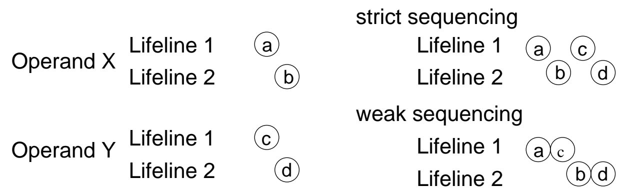

Figure 5. The difference between strict and weak sequencing.

3.3.6 Looping

With loop, iteration of subsequences may be expressed: "*The loop operand will be repeated a number of times*" (cf. [4, p. 371]). The number of iterations is defined by a pair $\langle min, max \rangle$ of numbers, where *max* may also be $*$, denoting infinity.[3]

This means, that both the minimum and maximum number of loops are determined, but if they are not identical, for the number of loops in between, the new standard *allows* looping without *prescribing* it. As for infinity as *maxint*, the proper modeling of this would be to include infinite traces.However, since both observations (i.e. "*emergent behaviour*", as the standard puts it) and system runs are finite (if of arbitrary length) in the real world. As nothing is to be gained from an overgeneralization here, I settle for the usual notion of finite traces.

The standard also defines, that loop ought to be expressed using the seq InteractionOperator: "*The loop construct represents a recursive application of the seq operator [...]* " (cf. [4, p. 371]). This leaves us with the following definition.

$$\begin{aligned}
&[\![\mathsf{loop}(P, min, max)]\!] = [\![\mathsf{loop}(P, min, max, 0)]\!]\\
&[\![\mathsf{loop}(P, min, max, i)]\!] =\\
&\quad\begin{cases}
[\![\mathsf{seq}(P, \mathsf{loop}(P, min, max, i+1))]\!] & \text{if } i < min\\
[\![\mathsf{opt}(\mathsf{loop}(P, min, max, i+1))]\!] & \text{if } min \leq i < max\\
\{\epsilon\} & \text{if } i \geq max
\end{cases}
\end{aligned}$$

where $i < *$ for all naturals i.

3.3.7 Exception handling

The standard declares: "*The break operator is a shorthand for an Alternative operator where one operand is given and the other assumed to be the rest of the enclosing InteractionFragment*" (cf. [4, p. 369]). In order to formally define "*the rest of*", I introduce a prefix function.

Definition 3.3 (prefix)
Let Σ be an alphabet and $u, w \in \Sigma^*$ be words. Then u is a **prefix** of w (written $u \in \mathsf{prefix}(w)$) iff there is some $v \in \Sigma^*$ such that $u.v = w$. This definition is easily extended to languages: let $U, W, \subseteq \Sigma^*$ be languages, then U is a prefix of W (written $U \in \mathsf{pre}(W)$) iff for all words in $w \in W$, there is a prefix in $u \in U$, and conversely all words $u \in U$ are prefixes of some word $w \in W$. Formally:

$$\begin{aligned}
U \in \mathsf{pre}(W) \iff \quad & \forall_{u \in U} : \exists w \in W \colon u \in \mathsf{pre}(w)\\
& \wedge \forall_{w \in W} : \exists u \in U \colon u \in \mathsf{pre}(w).
\end{aligned}$$

Note that $\epsilon \in \mathsf{pre}(v)$ for all $v \in \Sigma^*$ and $\{\epsilon\} \in \mathsf{pre}(V)$ for all $V \subseteq \Sigma^*$. □

[3] There are some further variants but since these are just syntactic sugar they are omitted here.

Thus I may define $[\![\mathsf{brk}(P, Q)]\!] = [\![\mathsf{strict}(\mathsf{pre}(P), Q)]\!]$.

4. Advanced features of UML 2.0 interactions

4.1. Gates

Large interactions may be split up along messages with so called gates. Gates are purely syntactic: using them does not alter the semantics in any way. Thus, for lack of space, I shall assume that no gates occur, so that diagrams are connected to each other only by the macro expansion explained below.

4.2. Define/Call

Interactions sport a macro-like mechanism for modularity, using the ref-command. It defines the in-lining of an Interaction declared by the command sd somewhere else.I assume that all used definitions are listed explicitly as a finite sequence of the form $\mathsf{sd}(\delta_0, P_0)\mathsf{sd}(\delta_1, P_1)\ldots$.

The standard does not exclude recursive definitions, but doesn't explicitly mention it either. In order to avoid fixpoint semantics, I exclude recursive definitions here, and assume, that in the sequence of sd-declarations, no backward-references may occur. In order to properly interact with operators like par, these macros must be expanded before the semantic translation is applied.

Semantically, this requires a kind of global memory to store all names used and their meaning. So I introduce the domain of Diagrams Environment (written Δ) with $\Delta = \Sigma \rightarrow \mathcal{SEQ}$ where Σ is a set of names for interaction diagrams. Thus, referring to another diagram becomes a lookup in the environment, and defining a diagram adds to the environment.

In order to deal with Δ, I need some more notation. For (possibly overlapping) alphabets Σ and Γ, and a replacement function $r : \Sigma \rightarrow \Gamma$, applying r to some $w \in \Sigma^*$ is written as $w\ [r]$. Replacement functions may be written as sets of mappings like $\{a \mapsto b\}$ for the replacement of all as by bs. Note that here, a tag will be replaced by an uninterpreted Interaction.

At this point, I need to introduce an auxiliary semantic function $[\![_]\!]_\Delta$ to gather the namespace. For the declaration of an interaction diagram, I may thus define

$$\begin{aligned}
[\![\epsilon]\!]_\Delta &= \emptyset\\
[\![\mathsf{sd}(\delta, P).REST]\!]_\Delta &= \{\delta \mapsto P\} + [\![P]\!]_\Delta + [\![REST]\!]_\Delta.
\end{aligned}$$

It simply adds the current definition ($\delta \mapsto P$) to the environment, and goes on recursively for P. The rest of the operators is treated by simply extracting the relevant information, for instance $[\![\mathsf{alt}(P, Q)]\!]_\Delta = [\![P]\!]_\Delta \cup [\![Q]\!]_\Delta$ and

$[\![\mathsf{opt}(P)]\!]_\Delta = [\![P]\!]_\Delta$. On the other hand, referencing another interaction simply inserts its name: $[\![\mathsf{ref}(\delta)]\!] = \delta$.

The Δ thus created may now be used to first expand the macros, and translate the overall expression afterwards by

$$[\![\mathsf{sd}(\delta, P)]\!] = [\![P\ [\ [\![P]\!]_\Delta\ ]]\!]. \quad (1)$$

The rest of the function $[\![_]\!]_\Delta$ is defined in the obvious way, i.e., $[\![\mathsf{alt}(P,Q)]\!]_\Delta = [\![P]\!]_\Delta \cup [\![Q]\!]_\Delta$ and so on. Note again, that the environment is always finite, and that there are neither recursion nor backward references.

4.3. Assert/Negate

The standard declares that "*the interaction operator neg designates that the combined fragment represents traces that are defined to be invalid*" (cf. [4, p. 370]), thus

$$[\![\mathsf{negate}(P)]\!] = \langle \emptyset, [\![P]\!] \rangle.$$

However, it is unclear what happens if there are several negate-operators, possibly nested and interspersed with other operators? And what does it mean, if negation does not occur at top-level, e.g., what is the intuitive meaning of $\mathsf{alt}(\mathsf{negate}(P), Q)$?

Closely related is the assert-operator, reminiscent of Life Sequence Charts (cf. [1]). The standard declares that "*the sequences of the operand of the assertion are the only valid continuations. All other continuations result in an invalid trace*" (cf. [4, p. 371]). As a first remark, implication as an operator is binary not unary. In classical logic, $(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\alpha \vee \beta)$, so one would expect $\mathsf{assert}(P,Q)$ to be equivalent in some sense to $\mathsf{alt}(P, \mathsf{negate}(Q))$. For instance, in the sense that $[\![\mathsf{assert}(P,Q)]\!] = [\![\mathsf{alt}(P, \mathsf{negate}(Q))]\!]$, or, given the above problem, rather

$$[\![\mathsf{assert}(P,Q)]\!] = \langle [\![\mathsf{Q}]\!], [\![\mathsf{P}]\!] \rangle.$$

However, one might also interpret assert as

$$[\![\mathsf{assert}(P,Q)]\!] = \{q \in [\![Q]\!] \mid \exists p \in [\![P]\!]\colon p \in \mathsf{prefix}(q)\}.$$

One might also consider both the assert- and negate-operators as being meta-logical in the sense that they express properties of traces rather than defining or modifying traces. Thus, they should not be modeled as InteractionOperators, but rather as boolean attributes of Interactions. Syntactically, this could be achieved by modifiers to the sd-command. For practical purposes, this seems to be the best solution. Anyway, the standard definitely needs clarification wrt. the operators assert and negate.

4.4. Ignore/Consider

With ignore (and its dual consider), the set of admissible messages is manipulated: "*One can understand ignore to mean that the messages that are ignored can appear anywhere in the trace*" (cf. [4, p. 370]). So far, the contents of messages has played no role in our semantics, so that I have to introduce some auxiliary notions here. First, assume that the ignore-command carries three arguments: an InteractionFragment, a set $M \subseteq \mathcal{M}$ of messages to be ignored in it, and a mapping $\mu : \mathcal{M} \to 2^{\mathcal{EO}}$ that links sets of EventOccurrences to the Message that caused them. I assume, that μ may be taken directly from the metamodel-instance.

I also need a function filter to remove the event occurrences that are to be ignored, defined as $\mathsf{filter}(\epsilon) = \epsilon$ and

$$\mathsf{filter}(x.w, \Gamma) = \begin{cases} \mathsf{filter}(w, \Gamma) & \text{if } x \in \Gamma \\ x.\mathsf{filter}(w, \Gamma) = & \text{otherwise} \end{cases}$$

with $\Gamma \subseteq \Sigma$, $x \in \Sigma$ and $w \in \Sigma^*$. Then, ignore may be defined as

$$[\![\mathsf{ignore}(P, M, \mu)]\!] \quad = \quad \mathsf{filter}([\![P]\!], \mu(M))$$

with M and μ as explained above, and $\mu(M) := \bigcup_{m \in M} \mu(m)$. Finally, since consider is dual to ignore, I can treat it indirectly by

$$[\![\mathsf{consider}(P, M, \mu)]\!] \quad = \quad [\![\mathsf{ignore}(P, \mathcal{M} - M, \mu)]\!].$$

4.5. Critical region

With critical regions, atomic subsequences are declared: "*A critical region means that the traces of the region cannot be interleaved by other* EventOccurrence*s on those lifelines covered by the region*" (cf. [4, p. 370]). Obviously, both the par and the ignore operators may thus break a critical region, so that establishing a region in an interaction must take place when no further operators may be applied. This may be done by replacing regions by tags, storing their contents in the environment and expand all such tags as the *last* step in the semantics. To this avail, I can use a similar approach to the one used for sd/ref, only that the expansion is now after translation rather than before, and, accordingly, traces have to be stored rather than uninterpreted InteractionFragments.

Thus, I have another global environment Γ with $\Gamma : \Sigma \to SEQ$, and a semantic function $[\![_]\!]_\Gamma$ similar to $[\![_]\!]_\Delta$ defined above, i.e., $[\![\mathsf{alt}(P,Q)]\!] = [\![P]\!]_\Gamma \cup [\![Q]\!]_\Gamma$ and so on.

The region as such is covered by $[\![\mathsf{region}(P)]\!] = \alpha$ and $[\![\mathsf{region}(P)]\!]_\Gamma = \{\alpha \mapsto [\![P]\!]\} + [\![P]\!]_\Gamma$, where α is a globally unique name for P. Equation (1) from above has to be replaced by $[\![\mathsf{sd}(\delta, P)]\!]_\Delta = (\ [\![P\ [\ [\![P]\!]_\Delta\ ]]\!]\ )\ [\ [\![P]\!]_\Gamma\ ]$.

4.6. Time

In UML 2.0, there are two different kinds of time condition one may express on interactions (see also Figure 3

above). First, one might want to express that a certain state persists for a duration or time interval. Second, one may want to express that the transmission of a message takes a certain duration or time interval. All of these cases may be reduced to intervals between pairs of event occurrences, so that I define the domain $\mathcal{TC}$ of Timing constraints as $\mathcal{TC} = \Sigma \times \Sigma \times (\mathbb{R} \times \mathbb{R})$, where the first two items are consecutive event occurrences, and the third item is interval of the minimum and maximum durations between the two event occurrences.

So far, I have treated timestamps as uninterpreted symbols from some alphabet Σ. Thus, traces from $\mathcal{SEQ}$ are really just trace schemas, since they each define a whole family of actual traces, where the uninterpreted timepoints are fixed to actual times. Such an interpretation function t simply maps symbolic to actual times, i.e. $t : \Sigma \rightarrow \mathbb{R}$. Given a trace $v \in \mathcal{SEQ}$ and an interpretation t, t is said to be **consistent**, iff the actual times provided do not violate the causal order represented in v.

So, a timed trace is a pair of a trace and a temporal interpretation, and thus I define the domain Timed Trace (written $\mathcal{SEQ}_T$) as $\mathcal{SEQ}_T = \mathcal{SEQ} \times (\Sigma \rightarrow \mathbb{R})$.

Let $u = \langle v, t \rangle \in \mathcal{SEQ}_T$ be a timed trace and $c = \langle \alpha, \beta, [min, max] \rangle$ be a time constraint. Then v is said to satisfy c (written $v \models_t c$), if the following holds.

$$v \models_t \langle \alpha, \beta, i \rangle :\iff t(\beta) - t(\alpha) \in i.$$

5. Conclusion

In this paper I have examined interactions the new UML 2.0 standard by defining a straightforward semantics. It closely follows the new standard in that it defines a partial order semantics for plain Interactions, embedded in a trace semantics of CombinedFragments.

The semantics presented here does not deal with gates, parameters to interactions and recursive definitions of interactions. It also omits states and guards, which may lead to strange semantics in exceptional cases (e.g. when the condition on one branch of the alt-operator is set to [false]). The current paper does deal, however, with all operators defined in the new standard, including loops and regions, and also timing annotations.

First of all, parameter passing has not been covered yet, as there arise a number of problems, in particular in conjunction with the ref-operator, as it may introduce recursion (see Figure 6). These questions have not been addressed in the new standard, and it is likely that they have been overlooked. It is also not entirely clear, whether having the power of recursion is a good thing in the first place, since it would introduce a whole new level of complexity into any semantics by implying fixed points.

Then, an obvious weak spot is the treatment of assert

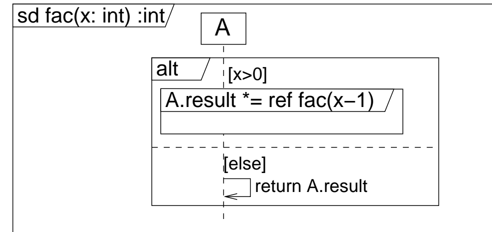

Figure 6. Example of a recursive definition.

and negate. Here, a comparison with exisiting solutions, in particular for LSCs might provide a line of attack.

Then there are "*extra global defined* CombinedFragment*s*" (cf. [4, p. 393]), whose benefit and meaning is completely left open to speculation by the standard.

the InteractionOperator "filter" mentioned on [4, p. 386],

Semantics of cverview diagrams however (in particular their semantic relationship to activity diagrams), seem to be necessary, yet less urgent. Then, notions of refinement need be discussed, in particular wrt. to assert/negate, and the time-interpretation needs to be explored further. Also, defining an alternative partial-order semantics might shed light on the meaning of strict. Finally, of course there is the integration with other semantics (like for StateMachine), and tool support that need be addressed

References

[1] W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. In *Proc. 3rd Intl. Conf. Formal Methods for Open Object-based Distributed Systems*. IFIP, 1999.

[2] ITU-T. *Recommenration Z.120. Message Sequence Charts (MSC)*. International Telecommunication Union, 2000.

[3] I. Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, TU München, 2000.

[4] OMG. Unified Modeling Language: Superstructure version 2.0 (2nd revised submission of April 10, 2003), April 2003.

[5] M. A. Reniers. *Message Sequence Charts - Syntax and Semantics*. PhD thesis, TU Eindhoven, 1999.

[6] H. Störrle. *Models of Software Architecture. Design and Analysis with UML and Petri-nets*. PhD thesis, LMU München, Inst. f. Informatik, December 2000. ISBN 3-8311-1330-0.

[7] M. Wirsing and A. Knapp. A formal approach to object-oriented software engineering. *J. Theoretical Computer Science*, 285:519–560, 2002.