A formal semantics for UML interactions

Alexander Knapp

Angaben zur Veröffentlichung / Publication details:

Knapp, Alexander. 1999. “A formal semantics for UML interactions.” *Lecture Notes in Computer Science* 1723: 116–30. https://doi.org/10.1007/3-540-46852-8_9.

A Formal Semantics for UML Interactions

Alexander Knapp[⋆]

Ludwig–Maximilians–Universität München
knapp@informatik.uni-muenchen.de

Abstract. The UML abstract syntax and semantics specification distinguishes between the statics and the dynamics of collaborations: the rôle context and interactions. We propose a formal semantics of interactions based on the abstract syntax and directly reflecting the specification. The semantics is both parametric in the notion of context and in semantic details that are intentionally left open by the specification, but resolves true inconsistencies. The formalisation uses temporal logic formulae in the style of Manna and Pnueli. We illustrate the flexibility of our semantics by discussing instantiations for a running example; its intuitiveness is substantiated by proving that the temporal formulae give rise to partial orders that also directly can be inferred from interactions.

Introduction

The object-oriented software modelling language UML, the "Unified Modeling Language", supports behavioural modelling, amongst a variety of different techniques, by the stimulus-based notion of interactions in collaborations. In an UML model, collaborations specify how an operation or an use case of the model is realised by a cooperation of several instances of model elements. For the static aspects of such a realisation a collaboration defines a context of class and association rôles describing which features actually participating instances have to show. For the dynamic aspects a collaboration defines an interaction, specifying which actions have to be performed by participating instances, which stimuli to other participating instances these actions have to dispatch, and in which order these stimuli can be sent, sequentially or concurrently.

The UML specification [10] defines the concrete and abstract syntax for collaborations and interactions and gives a description of the intended semantics, but entirely lacks a formal semantics. This omission does not only seriously limit the employment of UML for the construction of analysable and testable software designs in general; for collaborations and interactions the situation is aggravated by a gross vagueness of the specification itself. In particular, the informal semantics falls short of describing how participating instances of an interaction should actually react to an incoming stimulus, when actions are complete, and how local state information is to be treated; for some other, similar, problems see e.g. [9].

[⋆] This work was carried out during a stay at the Computer Science Laboratory of SRI International as part of the Visitor Exchange Program P-1-3334. It was supported by a DAAD scholarship and partially by the Bayerische Forschungsstiftung.

We therefore propose a formalisation of interactions in UML collaborations that clarifies at least some of the ambiguities of the specification but abstracts from semantic details that are intentionally left open. We claim that our formalisation directly, and intuitively, captures the informal semantics given in the UML specification. In order to reflect the semantic requirements following the specification as closely as possible, the formalisation is based on the UML meta-model. The concurrency constraints of UML interactions are expressed as temporal logic formulas in the style of Manna and Pnueli [8]. These formulae only take into account the sending of stimuli according to actions and the receiving of stimuli by instances. In particular, the formalisation is parametric in the notion of rôle context and in the more detailed execution of actions beyond the sending of stimuli which is under-specified by the UML specification. In order to corroborate our claim that our formalisation captures the UML specification, we provide an alternative approach to the semantics by unfolding interactions into partial orders, using Pratt's pomset framework [11]. We prove that the temporal formulae give rise to these partial orders.

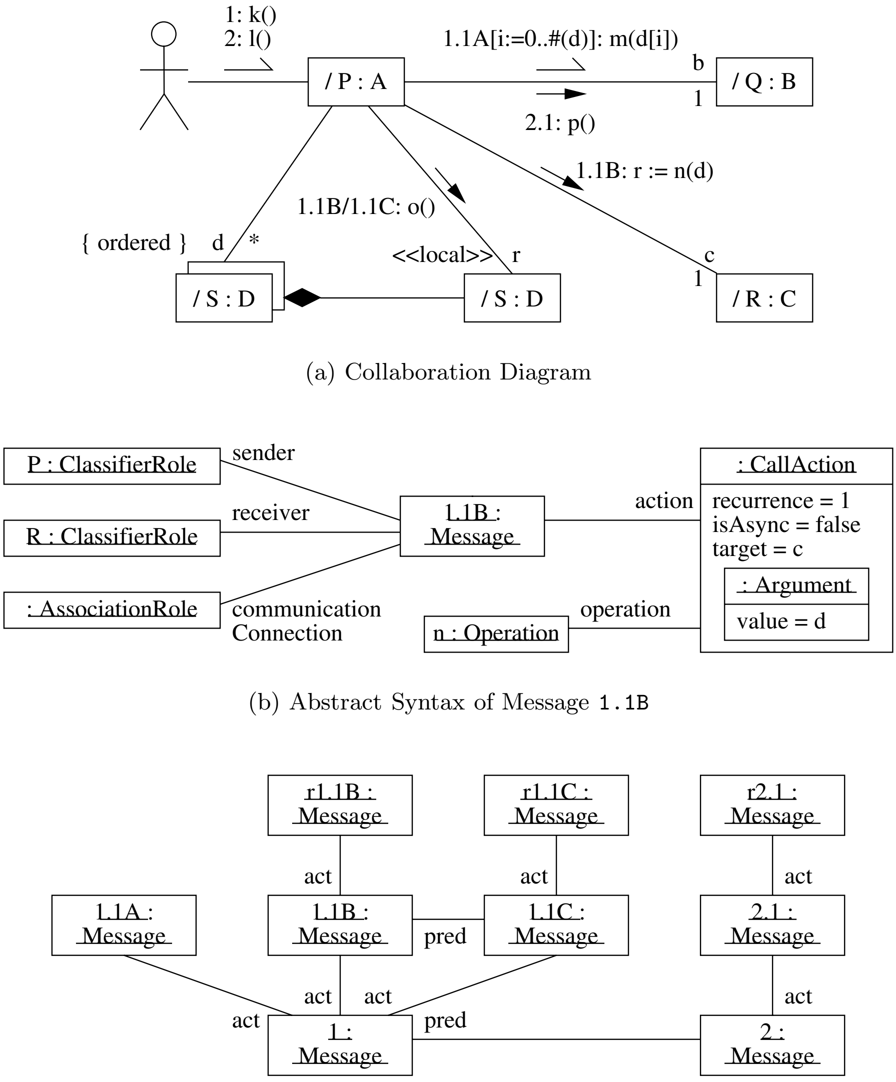
Several formal semantics of interactions have already been investigated, both of the specific notion in UML and of several closely related techniques: Araújo [2] translates a subset of UML sequence diagrams into temporal logic formulas. Gehrke, Goltz, and Wehrheim [6] sketch a translation of UML collaboration diagrams to Petri nets, but do not base their considerations on the meta-model. Wirsing and the author [12] model interaction diagrams of OOSE, one of the main predecessors of UML interactions, by asynchronously communicating finite automata; however, it is unclear whether this approach can be extended to the broader notion of UML interactions. For the equally closely related Message Sequence Charts semantical models based on process algebra, Petri nets, and automata have been investigated, for an overview see e.g. [7]; none of these approaches refers to an object-oriented setting. The semantics of rôle modelling in general is extensively discussed by Andersen [1]; the precise connections to UML collaborations, however, remain to be explored.

The remainder of this paper is structured as follows: In Sect. 1 we summarise UML interactions' abstract syntax and intended semantics. Section 2 presents the generation of temporal formulae from interactions in collaborations that precisely define the semantics of interactions. In Sect. 3 we assign partial orders to interactions and prove that the temporal logic formalisation indeed yields these partial orders. We conclude with an outlook to possible integrations of our semantics with more general approaches.

We assume some familiarity with the UML notation and a superficial knowledge of the UML abstract syntax [10].

1 UML Interactions

We briefly recall the parts of the UML meta-model pertaining to collaborations and interactions and their intended semantics [10] by means of a simple example. Notwithstanding the problems with the mapping from concrete to abstract syntax as described in the UML notation guide we more conveniently present the example in diagram form, but actually discuss the abstract syntax thereby defined. Consider the collaboration diagram in Fig. 1(a); some relevant fragments of its abstract syntax are presented graphically in Fig. 1(b) and Fig. 1(c).



(a) Collaboration Diagram

(b) Abstract Syntax of Message `1.1B`

(c) Abstract Syntax of the Interaction

Fig. 1. Example of an UML Collaboration

1.1 Abstract Syntax and Semantics According to the UML Specification

For the static aspects of a collaboration, rôles are specified that have to be filled in order to perform the task of the collaboration, describing the features collaborators have to show. Rôles are based on elements of a surrounding UML model; omitting the details, we require that ClassifierRole[1] `P` actually has two Operations `k` and `l`, that there is an AssociationRole with AssociationEndRoles having type `P` and `Q`, and so forth.

The dynamic part of a collaboration is described by Messages gathered in an Interaction. An interaction declares how and in which order stimuli complying to its messages are to be exchanged. Each Message has a reference to a sender ClassifierRole, a receiver ClassifierRole, a communication connection AssociationRole, an Action, a set of predecessor Messages, and, optionally, an activator Message. Semantically, the "activator is the message that invoked the procedure which in turn invokes the current message" [10, p. 2-115]; before a message can be executed all its predecessor messages have to be completed. There are several constraints on these predecessor and activator relations; with M the set of messages of an interaction the following properties have to be satisfied:

i. The graph (M, P) with P the predecessor relation on M is acyclic.
ii. The graph (M, A) with A the activator relation on M is a forest.
iii. For $m', m'' \in M$, if m'' transitively precedes m' then either both have the same activator $m \in M$ or both have no activator.

Executing a message actually means executing its action possibly resulting in the sending of a stimulus that complies to the message and therefore its action. An Action can be a CallAction, a ReturnAction, and of several other kinds, we do not discuss here. Each Action has a target expression that resolves to a set of target instances on evaluation; a recurrence expression that determines how the target set is iterated (sequentially or in parallel); and argument expressions that yield the actual arguments of the Action. Furthermore, an Action can be asynchronous, i.e., on execution no results are awaited, or synchronous, i.e., awaiting results. A CallAction refers additionally to an Operation, a stimulus complying to such an action will call that operation with the evaluated argument expressions as actual parameters. A ReturnAction has no explicit target; it returns the actual arguments to a caller.

According to (our interpretation of) the UML specification the intended semantics of the interaction described in Fig. 1(a) is the following: At start, only stimuli complying to `1` can occur since it is the only message that has no predecessors and no activators. Such stimuli can only be created by an execution of `1`'s action. Execution of this action means the creation of a single stimulus μ that is sent from the instance playing the actor rôle to the instance playing `P` and that bears an asynchronous call action of operation `k` with no actual arguments. Since the action of `1` is declared as asynchronous, no return stimulus is awaited and message `1` is completed; thus, stimuli complying to `2` may now occur. Furthermore, on receipt of stimulus μ, `1.1A`, `1.1B`, and `1.1C` are activated; the actions

[1] As in the UML specification meta-classes are written with initial capital letters.

of `1.1A` and `1.1B` can now be executed concurrently, `1.1C` has to wait for the completion of `1.1B`. The action of `1.1B` calls `n` with actual argument `d`; it is synchronous, hence a return stimulus with a value for `r` is awaited and only on receipt `1.1B` is completed. On completion of `1.1B`, the action of `1.1C` can be executed analogously. However, note that `1.1B` and `1.1C` share the local link `r`; this has to be stored such that the return stimulus for `1.1B` and the stimulus for `1.1C` have access. The action of `1.1A` asynchronously calls `m` as many times as `d` has elements with actual argument `d[i]` where `i` varies with the number of calls; the message is completed when all elements of `d` are processed. Meanwhile the action of `2` may have been executed in the same vein, activating `2.1`. —

Interactions may be composed inside the same collaboration by sharing rôles; or, transgressing the boundaries of one collaboration, by messages sharing actions. In the latter case, execution of an action will generate stimuli complying to all messages sharing this action.

1.2 Problems of the UML Specification

This exposition deviates from or interprets the UML specification in the following points: According to [10, p. 2-115] the activator relationship of messages in an interaction imposes a tree; we relax this condition to forests to allow multiple initial messages. The completion of a message is identified with the completion of its action, which in turn is determined by the termination of its recurrence expression; this remains open in [10]. By [10, pp. 2-99ff.], return actions do not send stimuli; in order to express our semantics more compactly, we require that they do. This also resolves the problem, left open in [10], by which means a synchronous call action learns of the results it is waiting for; we thus require a message with a return action that is activated by the message bearing the call action and that is the final message activated by the call message (see [10, p. 3-128] for a motivation). This leads to the following two additional constraints on the messages M of an interaction:

iv. For $m \in M$, if the action of m is a return then there is no $m' \in M$ such that m activates m'.

v. For $m \in M$, if the action of m is a synchronous call then there is an $r \in M$ such that the action of r is a return, m activates r and m' transitively precedes r for all $m' \in M$ such that m activates m'.

Finally, given our interpretation that shared actions lead to stimuli complying to several messages, the UML specification neglects the consequences of sharing, viz., that shared actions can only be executed if the corresponding messages do not depend on each other. Dependencies in the presence of sharing may be of the form: messages m_0 and m_1 precede messages m'_0 and m'_1, resp., but m_0 and m'_1 share the same action and also m'_0 and m_1 share the same action; none of these action can be executed in this situation. We therefore require that after identifying all those messages of an interaction that share the same action and extending this identification to the predecessor and activator relations, the new predecessor relation has to be acyclic, the activator relation has to form a forest, etc. More precisely, we regard an interaction as a graph (M, R) with the messages M as vertices and the activator and predecessor relations as edges; each vertex

m is labelled by $\alpha(m)$, the action of m, and each edge by whether its an activator or a predecessor edge:

vi. The quotient of an interaction (M, R) by α satisfies constraints (i–v).

The UML abstract syntax and its contextual constraints sketched so far serve as a basis for our formalisation of UML interactions. In particular, we will only consider CallActions and ReturnActions, all other kinds may be treated similarly; we do not include the script attribute of Action. Moreover, we do not consider the composition of interactions; we henceforth assume that messages sharing actions are contained in the same interaction.

2 Formal Semantics

We formalise the semantics of UML interactions as formulae in a temporal logic in the style of Manna and Pnueli [8]. We try to argue that this kind of formalisation allows us to capture the informal semantic requirements of the UML specification directly and intuitively. A state variable yields the global system state; functions for local states of actions and stimuli sent and received partially characterise this state. Execution of actions is described by transition systems on the local states of actions. Temporal formulae over the system state constrain the concurrent execution of actions; activations are directly reflected by number sequences. Thus, the global state may change not only subject to the execution of actions in a given interaction, as a collaboration may be embedded in a more comprehensive model. We leave open the representation of instances playing the rôles of an interaction's context and some details of executing actions, like the evaluation of arguments. These are under-specified in the UML specification and we will discuss plausible choices thus demonstrating the flexibility of our approach.

2.1 Semantic Domains

Formally, we require the following semantic domains: A domain Σ of global system states representing the states of instances playing the different rôles in the context of an interaction; for each action a a domain Λ_a of local action states comprising information from the recurrence and the target expression of a; for each action a a domain M_a for stimuli complying to a; and for each message m a domain M_m for stimuli complying to m and the action $\alpha(m)$ attached to it, such that there is a map $\alpha : M_m \to M_{\alpha(m)}$ exhibiting information that is shared by several stimuli. The semantic domain Σ is equipped with maps

$$\begin{aligned} &a_p : \Sigma \to \{\bot, \downarrow\} \uplus (\mathbb{N} \times \Lambda_a) \qquad \text{and} \\ &m_p : \Sigma \to \{\bot, \downarrow\} \uplus M_m \end{aligned}$$

for every action a, every message m, and every $p \in (\mathbb{N} \times \mathbb{N})^+$. The number pair sequences p are used to distinguish different occurrences of actions and stimuli; nesting is reflected by the length of a sequence, the pairs reflect the possibly parallel occurrences on a given nesting level. Intuitively, the map a_p

either yields the local action state of the pth occurrence of a together with a "program counter" that is used to create stimuli; or a_p yields that the pth occurrence of a is undefined ($\bot$) in a system state; or has already terminated ($\downarrow$). Analogously, the pth stimulus occurrence for a message m has been sent but not yet received if m_p yields an element of M_m; this stimulus has not been sent if m_p yields $\bot$; and this stimulus has already been received if m_p yields $\downarrow$.

2.2 Transition Semantics for Actions

For each action a we assume that its semantics is given by some initial local action state depending on the system state

$$\lambda^0_a : \Sigma \to \Lambda_a$$

and transition relations

$$\longrightarrow^\sigma_a \subseteq (\Lambda_a \times (\{\downarrow\} \uplus \Lambda_a)) \uplus (\Lambda_a \times (\Lambda_a \times \wp M_a))$$

parameterised over the global system state $\sigma \in \Sigma$.

Action occurrences will be created in an initial state; note, however, that such a notion is not required in the UML specification and may therefore be omitted. After creation, an action occurrence may proceed either by terminating and we write $\lambda{\downarrow}^\sigma_a$ if $(\lambda, \downarrow) \in {\longrightarrow^\sigma_a}$; or it may proceed by a silent step changing only its local state and we write $\lambda \longrightarrow^\sigma_a \lambda'$ if $(\lambda, \lambda') \in {\longrightarrow^\sigma_a}$ with $\lambda' \neq {\downarrow}$; or it may proceed by changing its local state and sending stimuli and we write $\lambda \longrightarrow^\sigma_a \lambda', M$ if $(\lambda, (\lambda', M)) \in {\longrightarrow^\sigma_a}$. An action's recurrence expression may allow for sending several different stimuli in one transition step.

The flexibility offered by such a general semantics for actions does indeed seem to be necessary: Let a be the call action of message `1.1A` in our running example (Fig. 1) having: as its recurrence expression `i:=0..#(d)`, as its target expression `b`, as its operation a reference to `m`, and as its single argument expression `d[i]`. Various choices for a formal semantics are possible: For one instance, we could assume that the target expression is only evaluated once, when the action is created, and that the evaluation of the actual argument is atomic. Then we would choose $\Lambda_a = \mathbb{N} \times O$ where O is some semantic domain of object identifiers and $M_a = V \times O$ with an additional semantic domain V for values such that O is a sub-domain of V. The initial state would be

$$\lambda^0_m(\sigma) = (0, \llbracket \mathtt{b} \rrbracket(\sigma))$$

and a transition relation could be defined by

$$\begin{array}{l} (i, o){\downarrow}^\sigma_m \quad \text{if } \sharp\llbracket \mathtt{d} \rrbracket(\sigma) > i, \\ (i, o) \longrightarrow^\sigma_a (i+1, o), (\llbracket \mathtt{d} \rrbracket(\sigma)(i), b) \quad \text{if } \sharp\llbracket \mathtt{d} \rrbracket(\sigma) \leq i \end{array}$$

where $\llbracket - \rrbracket$ is a function evaluating an expression in a state and $\sharp$ denotes the cardinality function.

However, the evaluation of the actual arguments may require many steps which then has to be reflected in the semantic domain Λ_a. Analogously, the evaluation of the target expression may not be atomic. We simply are not committed to any of these selections.

2.3 Semantics of Interactions

We now turn to the semantics of full interactions and their ordering constraints. We use a linear first-order temporal logic with temporal connectives $\Box$ (always), $\Diamond$ (eventually), and W (unless). The underlying state language consists of one flexible state variable σ from the semantic domain Σ, rigid variables, the semantic maps a_p and m_p as function symbols, and $\longrightarrow^{\sigma}_{a}$ as relation symbols (we also use the various abbreviations introduced above). The semantics of a specification in such a temporal logic is defined to be a transition system whose runs satisfy all formulae of the specification; for more details on this formalism cf. [8].

Let I be an interaction, M its set of messages, A the set of actions that is attached to M, and let $\alpha(m)$ denote the action of $m \in M$. The instantiation of the formula schemes (1–13) defined below according to I define a temporal logic specification of the semantics of I.

First, we embed the transition semantics for actions into temporal logic. Each occurrence of an action $a \in A$ is created in its initial state:

$$\forall p \in (\mathbb{N} \times \mathbb{N})^{+} \,.\, a_p(\sigma) = \bot \mathbin{\mathsf{W}} a_p(\sigma) = (0, \lambda^{0}_{a}(\sigma)) \ . \tag{1}$$

Each occurrence of an action $a \in A$ proceeds as given by its transition semantics:

$$\begin{aligned}
&(a_p(\sigma) = (i, \lambda) \Rightarrow (a_p(\sigma) = (i, \lambda) \mathbin{\mathsf{W}} \\
&\quad ((a_p(\sigma) = {\downarrow} \wedge \lambda{\downarrow}^{\sigma}_{a}) \vee \\
&\quad\; (a_p(\sigma) = (i, \lambda') \wedge (\lambda \longrightarrow^{\sigma}_{a} \lambda')) \vee \\
&\quad\; (a_p(\sigma) = (i+1, \lambda') \wedge (\lambda \longrightarrow^{\sigma}_{a} \lambda', \{\mu_1, \ldots, \mu_k\}) \wedge \\
&\quad\; \bigwedge\nolimits_{\substack{0 \le j \le k \\ m \in \alpha^{-1}(a)}} \ \alpha(m_{p.(i,j)}(\sigma)) = \mu_j \wedge \forall j > k \,.\, m_{p.(i,j)}(\sigma) = \bot))) \ .
\end{aligned} \tag{2}$$

Occurrences of actions that have terminated can not be reactivated:

$$(a_p(\sigma) = {\downarrow} \Rightarrow \Box a_p(\sigma) = {\downarrow}) \ .$$

Next, stimuli can only be created by appropriate actions:

$$m_{p.(i,j)}(\sigma) = \bot \mathbin{\mathsf{W}} (\alpha(m)_p(\sigma) = (i+1, \lambda) \wedge m_{p.(i,j)}(\sigma) \neq \bot) \ . \tag{3}$$

Stimuli do not change between sending and receiving:

$$((m_p(\sigma) = \mu \wedge \mu \neq \bot) \Rightarrow (m_p(\sigma) = \mu \mathbin{\mathsf{W}} m_p(\sigma) = {\downarrow})) \ . \tag{4}$$

Stimuli will be received sometime:

$$(m_p(\sigma) \neq \bot \Rightarrow \Diamond m_p(\sigma) = {\downarrow}) \ . \tag{5}$$

Stimuli that have been received can not be resent:

$$(m_p(\sigma) = {\downarrow} \Rightarrow \Box m_p(\sigma) = {\downarrow}) \ . \tag{6}$$

Finally, we treat the order constraints of the interaction I. If a message m is preceded by a message m'

$$(\alpha(m)_p(\sigma) = (0, \lambda) \Rightarrow \alpha(m')_p(\sigma) = \downarrow) \ , \tag{7}$$

saying that whenever the pth occurrence of message m's action is ready to be executed the preceding message m' must have terminated.

If message m is activated by message m'

$$(\alpha(m)_{p.(i,j)} = (0, \lambda) \Rightarrow m'_{p.(i,j)}(\sigma) = \downarrow) \ , \tag{8}$$

$$(m'_{p.(i,j)}(\sigma) = \downarrow \Rightarrow \Diamond\alpha(m)_{p.(i,j)} = (0, \lambda)) \ , \tag{9}$$

saying that when a stimulus with number i adhering to the pth occurrence of the action of message m' has been received, the $p.i$th occurrence of message m will be ready for execution some time later on, but that execution can not start prematurely.

If message r is the return message of message m

$$(r_{p.(i,j).(k,l)}(\sigma) = \downarrow \Rightarrow \alpha(m)_p(\sigma) = (i+1, \lambda)) \ , \tag{10}$$

saying that when a stimulus with number (k, l) adhering to the $p.(i, j)$th occurrence of message r's action has been received the pth activation of message m's action is (still) in state $i + 1$.

We additionally designate an initial state. Let N be the messages without an activator in I. Only occurrences of actions of messages in N may exist initially.

$$\forall m \in M \,.\, \forall p \in (\mathbb{N} \times \mathbb{N})^+ \setminus \{(0,0)\} \,.\, \alpha(m)_p(\sigma) = \bot \tag{11}$$

$$\forall m \in M \setminus N \,.\, \alpha(m)_{(0,0)}(\sigma) = \bot \tag{12}$$

$$\forall m \in N \,.\, \Diamond\alpha(m)_{(0,0)}(\sigma) = (0, \lambda^0_a(\sigma)) \tag{13}$$

The *semantics* of I is defined to be all runs (models) of the temporal logic specification yielded by instantiating formula schemes (1–13) according to I.

Example. For our running example as depicted in Fig. 1(c), denoting the action of message m by $\mathtt{a}m$,

$$\begin{aligned}
&(\mathtt{a2}_p(\sigma) = (0, \lambda) \Rightarrow \mathtt{a1}_p(\sigma) = \downarrow)\\
&(\mathtt{a1.1A}_{p.(i,j)} = (0, \lambda) \Rightarrow \mathtt{1}_{p.(i,j)}(\sigma) = \downarrow)\\
&(\mathtt{1}_{p.(i,j)}(\sigma) = \downarrow \Rightarrow \Diamond\mathtt{a1.1A}_{p.(i,j)} = (0, \lambda))\\
&(\mathtt{a1.1B}_{p.(i,j)} = (0, \lambda) \Rightarrow \mathtt{1}_{p.(i,j)}(\sigma) = \downarrow)\\
&(\mathtt{1}_{p.(i,j)}(\sigma) = \downarrow \Rightarrow \Diamond\mathtt{a1.1B}_{p.(i,j)} = (0, \lambda))\\
&(\mathtt{a1.1C}_{p.(i,j)} = (0, \lambda) \Rightarrow \mathtt{1}_{p.(i,j)}(\sigma) = \downarrow)\\
&(\mathtt{1}_{p.(i,j)}(\sigma) = \downarrow \Rightarrow \Diamond\mathtt{a1.1C}_{p.(i,j)} = (0, \lambda))
\end{aligned}$$

defines the ordering constraints for the messages `1`, `1.1A`, `1.1B`, and `1.1C` according to (7–9). A simple model construction will be discussed in the next section.

It may be noted that we assume that for shared actions several different stimuli complying to each of the corresponding messages occur. These may be comprised into only one stimulus complying to several messages and the shared action.

3 Partial Orders from Interactions

In order to provide evidence that our formalisation captures the intuitive semantics of interactions as described in the specification, we investigate the partial orders of stimuli that can be produced by executing an interaction. These partial orders are derived again directly from the abstract syntax. They may appear as an even more obvious approach to the semantics of interactions; however, it seems to be non-trivial to integrate them with a semantics of actions. We thus subsequently prove that our temporal logic semantics yields the same partial orders for terminating interactions.

More precisely, we assign a process, i.e. a set of (labelled) pomsets [11], to an interaction, with the labelling from the messages of the interaction. Such a process represents all possible unrollings or executions of the interaction assuming that each action of a message can be executed an arbitrary, but finite number of times; completion of a message, viz. of its action, will be indicated by special labels. The overall plan of the process construction is to assign to each activation nesting level of messages a set of processes in a bottom-up fashion and to insert the pomsets of these processes whenever the start message of a nesting level occurs in the previous nesting level.

The process is built from simple pomsets containing only a single atom m, denoting the possible occurrence of a stimulus complying to the message m, or $\overline{m}$, denoting the completion of the message's action. From these simple processes we proceed to more complex ones by the well-known (total) process operations of sequential composition (;), parallel composition ($\|$), sequential iteration (*), parallel repetition (†), and homomorphisms [11]; two additional partial operations for the treatment of synchronous and asynchronous actions are introduced. We define these operations for pomsets only; they are lifted to processes in the usual way:

Let p and q be pomsets such that q has a unique minimal element. Let $(X, \leq_X, \kappa)$ and $(Y, \leq_Y, \lambda)$ be representing partial orders for p and q, resp., such that X and Y are disjoint; let $m \in Y$ be the representation of the minimal element of q and let $C = \{x \in X \mid \kappa(x) = \lambda(m)\}$.

The *asynchronous insertion* of q in p, written $p \leftharpoonup q$, is given by the pomset represented by the partial order $(Z, \leq_Z, \mu)$ with: $Z = X \cup (Y \setminus \{m\}) \cdot C$ (where $N \cdot M$ denotes the M-fold disjoint sum of N); for every $z, z' \in Z$ define $z \leq_Z z'$ if either $z, z' \in X$ and $z \leq_X z'$, or $z, z' \in Y \times \{x\}$ for some $x \in C$ and $\pi_1 z \leq_Y \pi_1 z'$, or $z \in C$ and $z' \in Y \times \{z\}$; for every $z \in Z$ define $\mu(z) = \kappa(z)$ if $z \in X$ and $\mu(z) = \lambda(\pi_1 z)$ if $z \in Y \times C$.

The *synchronous insertion* of q in p, written $p \leftarrow q$, is given by the pomset represented by $(Z, \leq'_Z, \mu)$ with Z and μ as for the asynchronous insertion and $z \leq'_Z z'$ for $z, z' \in Z$ if either $z, z' \in X$ and $z \leq_X z'$, or $z, z' \in Y \times \{x\}$ for some $x \in C$ and $\pi_1 z \leq_Y \pi_1 z'$, or $z \in C$ and $z' \in Y \times \{z\}$, or $z \in Y \times \{z''\}$ for some $z'' \in C$ and $z'' \leq_X z'$.

Synchronous insertion is obviously a special case of homomorphism. Both synchronous and asynchronous insertion are associative; additionally

$$p \leftarrow q \leftarrow r = p \leftarrow r \leftarrow q \quad \text{and} \quad p \leftharpoonup q \leftharpoonup r = p \leftharpoonup r \leftharpoonup q$$

hold for pomsets p, q, and r such that all the synchronous and asynchronous insertions are defined, respectively.

3.1 Process Construction

To begin with, we construct processes for interactions without shared actions; the general case will be discussed shortly. Again we perceive such an interaction as a graph I with messages as its vertices and edges labelled either "activator" or "predecessor" which has to satisfy conditions (i–v) of Sect. 1. In order to abbreviate notation, we write $m \dashrightarrow m'$, if message m is the activator of m', and $m \twoheadrightarrow m'$, if m is a predecessor of m'. For the set of messages M of I fix a set $\overline{M}$, disjoint from M and a bijective function $\overline{\cdot} : M \to \overline{M}$.

We first describe how a process is assigned to each level of activation, i.e., a full subgraph of the interaction having as one vertex an activating message and as the additional vertices all the messages that it (immediately) activates. Each such level would be an interaction if requirement (v) would be dropped. Let C_N be the set of all graphs with vertices from $N \subseteq M$, edges labelled from $\{\dashrightarrow, \twoheadrightarrow\}$, fulfilling requirements (i–iv) above, and having exactly one activator vertex. Let P_N be the class of all processes with labels from $N \cup \overline{N}$. Define $\psi_N : C_N \to P_N$ as follows: Let $C = (N, R) \in C_N$, and let $c \in N$ be its unique activator. Define a pomset p represented by the partial order $(N \backslash \{c\}, \twoheadrightarrow, \mathrm{id}_{N \backslash \{c\}})$ and a homomorphism $\sigma : p \to P_N$ given by $\sigma(m) = m$ if the action of m is a return; $\sigma(m) = m^* \,; \overline{m}$ if the action of m is not a return and always sequential; and $\sigma(m) = m^{\dagger *} \,; \overline{m}$ otherwise. Then $\psi_N(C) = c \,; p\sigma$.

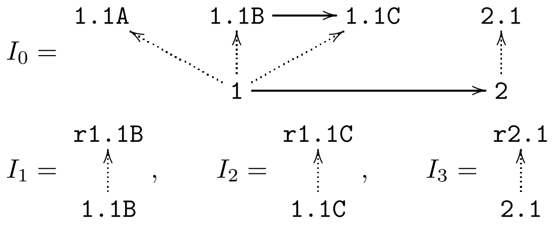
For the overall construction, let I_M be the set of all interactions with vertices M and define P_M as before. Define $\varphi : \prod_M I_M \to P_M$ mutually recursive as follows: Let $I = (M, R)$ be an interaction.

A. If $I = (M, R)$ such that there is no activator in M (i.e. there are no $c, m \in M$ with $c \dashrightarrow m$), then define a homomorphism $\sigma : M \to P_M$ by $\sigma(m) = m^* \,; \overline{m}$ if the action of m is always sequential and $\sigma = m^{\dagger *} ; \overline{m}$ if the action of m may be parallel; define a pomset p represented by the partial order $(M, \twoheadrightarrow, \mathrm{id}_M)$ with $\twoheadrightarrow$ the predecessor relation of I. Define $\varphi_M(I) = p\sigma$.

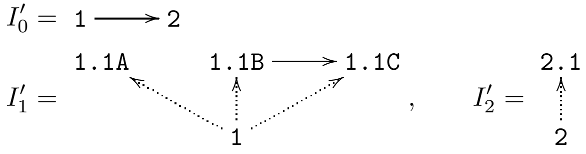
B. If $I = (M, R)$ such that there is at least one activator in M, then let $A = \{a_1, \ldots, a_k\}$ be the penultimate vertices of the activator forest in I (i.e. for every $a \in A$ there is an $m' \in M$ such that $a \dashrightarrow m'$ but there are no two vertices $m', m'' \in M$ such that $a \dashrightarrow m' \dashrightarrow m''$; and A is maximally so); for every $1 \leq i \leq k$ define $I_i = (M_i, R_i)$ as the full subgraph generated by $\{m \in M \mid a_i \dashrightarrow m\} \cup \{a_i\}$; further, define $I_0 = (M_0, R_0)$ as the full subgraph of I generated by $M \setminus \bigcup_{1 \leq i \leq k}(M_i \setminus \{a_i\})$; finally, define $P_0 = \varphi_{M_0}(I_0)$ and $P_i = P_{i-1} \leftarrow \psi_{M_i}(I_i)$ if the action of a_i is synchronous and $P_i = P_{i-1} \leftharpoonup \psi_{M_i}(I_i)$ if the action of a_i is asynchronous. Then $\varphi_M(I) = P_k$.

Let $I = (M, R)$ be an interaction and $\varphi : \prod_M I_M \to P_M$ defined as above. Then, $[\![I]\!] = \varphi_M(I)$ is the *corresponding process* of I.

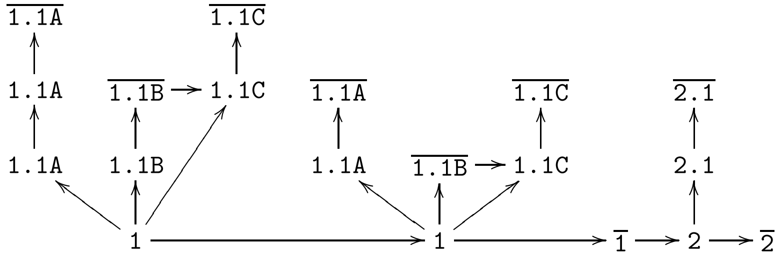
Example. We illustrate the construction of an interaction's corresponding process by our running example interaction I as depicted in Fig. 1(c). All actions in I are sequential. The graph I fulfils the conditions of (B) and we get



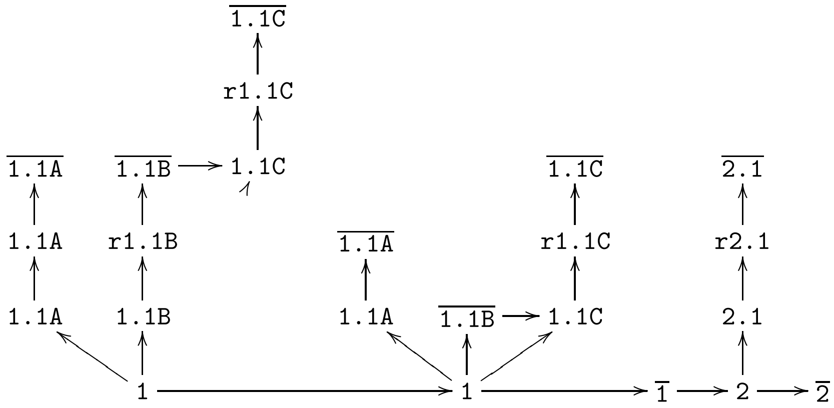
Omitting the indices of ψ and φ, we have $\psi(I_1) = \texttt{1.1B} \,;\, \texttt{r1.1B}$, $\psi(I_2) = \texttt{1.1C} \,;\, \texttt{r1.1C}$, and $\psi(I_3) = \texttt{2.1} \,;\, \texttt{r2.1}$. The graph I_0 again fulfils the conditions of (B):



Here, $\psi(I'_1) = \texttt{1} \,;(\texttt{1.1A}^* \,;\overline{\texttt{1.1A}} \,\|\, \texttt{1.1B}^* \,;\overline{\texttt{1.1B}};\texttt{1.1C}^* \,;\overline{\texttt{1.1C}})$, $\psi(I'_2) = \texttt{2};\texttt{2.1}^* \,;\overline{\texttt{2.1}}$. Now, by (A), $\varphi(I'_0) = \texttt{1}^* \,;\overline{\texttt{1}}\,;\texttt{2}^* \,;\overline{\texttt{2}}$, and thus $\varphi(I_0) = \varphi(I'_0) \leftharpoonup \psi(I'_1) \leftharpoonup \psi(I'_2)$; a typical pomset of this process is


$\overline{\texttt{1.1A}}$
$\overline{\texttt{1.1C}}$
1.1A
$\overline{\texttt{1.1B}}$
1.1C
$\overline{\texttt{1.1A}}$
$\overline{\texttt{1.1C}}$
$\overline{\texttt{2.1}}$
1.1A
1.1B
1.1A
$\overline{\texttt{1.1B}}$
1.1C
2.1
1
1
$\overline{\texttt{1}}$
2
$\overline{\texttt{2}}$

Finally, $\varphi(I) = \varphi(I_0) \leftarrow \psi(I_1) \leftarrow \psi(I_2) \leftarrow \psi(I_3)$; synchronous insertion into the pomset above yields


$\overline{\texttt{1.1C}}$
r1.1C
$\overline{\texttt{1.1A}}$
$\overline{\texttt{1.1B}}$
1.1C
$\overline{\texttt{1.1C}}$
$\overline{\texttt{2.1}}$
1.1A
r1.1B
$\overline{\texttt{1.1A}}$
r1.1C
r2.1
1.1A
1.1B
1.1A
$\overline{\texttt{1.1B}}$
1.1C
2.1
1
1
$\overline{\texttt{1}}$
2
$\overline{\texttt{2}}$

In the general case, actions may be shared by messages. The construction of a process for an interaction without shared actions can be lifted to this situation: Let $I = (M, R)$ be an interaction satisfying conditions (i–vi) of Sect. 1, viewed as an edge-labelled graph as above. Let $\alpha(m)$ denote the action of a message $m \in M$. Define an edge-labelled graph $I/\alpha = (M/\alpha, R/\alpha)$ with M/α the set of equivalence classes $[m]$ of messages quotiented by α, i.e., $[m] = [m']$ if $\alpha(m) = \alpha(m')$; and R/α as $v \dashrightarrow v'$ if $v = [m]$ and $v' = [m']$ and $m \dashrightarrow m'$, and $v \rightarrow v'$ if $v = [m]$ and $v' = [m']$ and $m \rightarrow m'$. By definition, I/α is the graph of an interaction. Furthermore, for $[m] = \{m_0, \ldots, m_{n-1}\}$, let $\sigma([m]) = m_0 \| \ldots \| m_{n-1}$ and $\sigma(\overline{[m]}) = \overline{m_0} \parallel \ldots \parallel \overline{m_{n-1}}$.

Then, $[\![I]\!] = [\![I/\alpha]\!]\sigma$ is the *corresponding process* of I.

3.2 From Runs to Pomsets

There is a precise relation between the formal semantics of an interaction I as given by all runs over a single state variable satisfying the temporal formulae derived from (1–13) in Sect. 2 and the process $[\![I]\!]$ according to the construction above. The pomsets in $[\![I]\!]$ only take into account the sending of messages and the termination of actions; we may additionally interpret an occurrence of a return stimulus in a pomset as the receipt of this stimulus. Conversely, a pomset can be assigned to a run of I, according to our semantics, by extracting every first occurrence of the termination of an action, the sending of a stimulus, or the receipt of a stimulus. Such a pomset, however, may show less concurrency than any pomset in $[\![I]\!]$ and the pomsets in $[\![I]\!]$ have to be augmented by additional, consistent, ordering in order to match. Additionally, since all partial orders in $[\![I]\!]$ are finite, we have to restrict our comparison to terminating runs, that is, runs showing an eventually stable state in which every stimulus (m_p) or action occurrence (a_p) is either undefined ($\perp$) or has terminated ($\downarrow$); note, that no temporal logic axiom unconditionally requires an action to terminate.

More precisely, for a terminating run $(\sigma_i)_{i\in\mathbb{N}}$ of I with messages M define

$$\begin{aligned} H_i = {} & \{(\overline{m}, p) \in \overline{M} \times (\mathbb{N} \times \mathbb{N})^+ \mid \alpha(m)_p(\sigma_i) = \downarrow\} \cup \\ & \{(m, p) \in M \times (\mathbb{N} \times \mathbb{N})^+ \mid m_p(\sigma_i) \in M_{\alpha(m)},\ \alpha(m) \text{ no return}\} \cup \\ & \{(m, p) \in M \times (\mathbb{N} \times \mathbb{N})^+ \mid m_p(\sigma_i) = \downarrow,\ \alpha(m) \text{ return}\} \end{aligned}$$

and let $\pi((\sigma_i)_{i\in\mathbb{N}})$ denote the pomset represented by $(H, \leq, \eta)$ with

$$\begin{aligned} & H = \bigcup\nolimits_{i\in\mathbb{N}} H_i\ , \\ & h \leq h' \text{ if } h = h' \text{ or } h \in H_k,\ h' \in H_l \setminus \bigcup\nolimits_{0\leq i<k} H_i,\ k < l\ , \\ & \eta((\overline{m}, p)) = \overline{m},\ \eta((m, p)) = m\ . \end{aligned}$$

It remains to show that these pomsets are augmentations of pomsets in $[\![I]\!]$, i.e., that for every run $(\sigma_i)_{i\in\mathbb{N}}$ there is a $\pi \in [\![I]\!]$ equipped with a bijective homomorphism from π to $\pi((\sigma_i)_{i\in\mathbb{N}})$.

For a proof sketch of this claim, the following observation is decisive: Let $(X, \leq, \lambda)$ be a representative of a pomset in $[\![I]\!]$. Then $x < x'$ only if: $\lambda(x) \notin \overline{M}$ and $\lambda(x) = \lambda(x')$; or $\lambda(x), \lambda(x') \notin \overline{M}$ and $\lambda(x)$ transitively precedes or activates $\lambda(x')$ in I; or $\lambda(x) \in \overline{M}$, and the message of $\lambda(x)$ precedes or activates the message of $\lambda(x')$ in I; or $\lambda(x) \in M$, $\lambda(x') \in \overline{M}$, and the action of $\lambda(x)$ is the return action for the action of the message of $\lambda(x')$. This is easily proven by the construction of $[\![I]\!]$. But these are exactly the orderings that are also at least required by the temporal formulae. Since $[\![I]\!]$ contains all pomsets with an arbitrary number of stimulus occurrences, the claim follows.

The correspondence between the pomset semantics and the temporal semantics may also be extended to non-terminating runs (in the sense explained above), if we drop the requirement that all messages have to be completed in the pomset semantics.

Conclusions and Future Work

We presented a formal semantics for UML interactions. This semantics is given by all runs satisfying certain temporal formulae that can be derived directly from the abstract syntax representation of an interaction; it tries to capture the requirements of the UML semantics specification as intuitively as possible. In particular, the semantics is parametric in both the notion of context of interactions and in a transition semantics for actions, that are under-specified by the UML semantics. Additionally, we investigated the relationship of these models to an event-based construction assigning pomsets over stimuli to interactions; it was shown that temporal runs correspond to augmentations of these pomsets.

It may be interesting to combine the construction of pomset models for interactions with the transition semantics for actions in the style of Cenciarelli et. al. [4]. Such a semantics could form an even more declarative alternative.

Our temporal logic semantics, however, already provides the necessary basis to study the important notion of composition of interactions. We considered sharing of actions as one possibility to combine different interactions in a common contextual collaboration, but much work remains to be done in this respect.

Finally, the contextual parametricity of our semantics allows its smooth integration with existing more general approaches to a comprehensive system model for UML, such as the SysLab [3] or pUML [5].

Acknowledgements. I profitted much from discussions with José Meseguer, Martin Wirsing, and Harald Störrle.

References

1. Egil P. Andersen. *Conceptual Modeling of Objects — A Role Modeling Approach.* PhD thesis, Universitetet i Oslo, 1997.
2. João Araújo. Formalizing Sequence Diagrams. In Luis F. Andrade, Ana Moreira, Akash R. Deshpande, and Stuart Kent, editors, *Proc. Wsh. Formalizing UML. Why? How?*, Vancouver, 1998.
3. Ruth Breu, Radu Grosu, Franz Huber, Bernhard Rumpe, and Wolfgang Schwerin. Systems, Views and Models of UML. In Axel Korthaus and Martin Schader, editors, *The Unified Modeling Language — Technical Aspects and Applications*, pages 93–108. Physica, Heidelberg, 1998.
4. Pietro Cenciarelli, Alexander Knapp, Bernhard Reus, and Martin Wirsing. An Event-Based Structural Operational Semantics of Multi-Threaded Java. In Jim Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *Lect. Notes Comp. Sci.*, pages 157–200. Springer, Berlin, 1999.
5. Robert France, Andrew Evans, Kevin Lano, and Bernhard Rumpe. The UML as a Formal Modeling Notation. *Comp. Stand. Interf.*, 19(7):325–334, 1998.
6. Thomas Gehrke, Ursula Goltz, and Heike Wehrheim. The Dynamic Models of UML: Towards a Semantics and Its Application in the Development Process. Technical Report 11/98, Universität Hildesheim, 1998.
7. Piotr Kosiuczenko and Martin Wirsing. Towards an Integration of Message Sequence Charts and Timed Maude. In Murat M. Tanik, Jiro Tanaka, Kiyoshi Itoh, Michael Goedicke, Wilhelm Rossak, Hartmut Ehrig, and Franz Kurfueß, editors, *Proc. 3rd Int. Conf. Integrated Design and Process Technology*, Berlin, 1998.
8. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems. Vol. 1: Specification.* Springer, New York–etc., 1992.
9. Object Management Group. OMG UML v1.3: Revisions and Recommendations — Appendix A: Issues Database Report. Technical report, Object Management Group, 1999. http://www.omg.org/docs/ad/99-06-11.pdf.
10. Object Management Group. Unified Modeling Language Specification, Version 1.3. Technical report, Object Management Group, 1999. http://www.omg.org/docs/ad/99-06-08.zip.
11. Vaughan Pratt. Modeling Concurrency with Partial Orders. *Int. J. Parallel Program.*, 15(1):33–71, 1986.
12. Martin Wirsing and Alexander Knapp. A Formal Approach to Object-Oriented Software Engineering. In José Meseguer, editor, *Proc. 1st Int. Wsh. Rewriting Logic and Its Applications*, volume 4 of *Electr. Notes Theo. Comp. Sci.*, pages 321–359. Elsevier, 1996.