Some questions concerning Interactions and StateMachines

Harald Störrle

Ludwig-Maximilians-Universität München
stoerrle@informatik.uni-muenchen.de

Abstract. In this paper, three questions concerning the semantics of dynamic UML are raised:

1. I Collaborations are to be runs of StateMachines, how can they be related?
2. If there is an ensemble of collaborating entities, each of which has a StateMachine defining its behavior, how do they communicate?
3. If ActionSequence is an Action, it should be atomic, but what does that mean in relation to RTC-steps?

While these questions have been discussed now and again among researchers, there seems to be no publication at all raising them, much less provide an answer. The recent "Action Semantics" avoids these issues except one interesting point: in stark contrast to the rest of UML, it seems to commit itself to a true concurrency semantics. We do not answer these questions either (of course), but hint at some ways out. A more complete answer is given in [3].

1 Motivation and approach

One of the core principles of the UML is the idea of multiple views on a system. However, inconsistencies may arise among overlapping (sets of) views. But rather than being just a nuisance, such inconsistencies allow to detect differences in points of view among developers. To be practical, however, one needs automated tool support to discover such inconsistencies, and thus formal semantics for the viewpoints one is interested in, particularly for the dynamic models.

Such a semantics should be based on the UML metamodel, rather than on the concrete syntax, i.e. the notations. We use Petri-nets as the semantic domain as (1) they provide true concurrency semantics and (2) allow to express both systems an their runs in the same formalism. See [3] for more details. However, this is immaterial to the present paper. Pursuing this approach, certain intriguing questions concerning the UML metamodel arise.

2 Intuitive semantics

The UML currently offers four different notations for representing dynamic models, and two groups of metaclasses these notations are mapped to. Sequence and

collaboration diagrams both map to Collaborations[1], state transition diagrams[2] map to StateMachine, and activity diagrams map to ActivityGraph, which is a subclass of StateMachine.

A StateMachine is a set of Transitions and a tree of StateVertexes, the root of which must be a State and is called top. Transitions correspond to arcs of the concrete syntax (!) of state transition diagrams. The intuitive concept of a transition is captured by the informal notion of "compound transition".

The operational semantics of StateMachine is defined by three separate entities: the state transition diagram proper, a context, and an event queue. The event queue is used to buffer Events that may be used as triggers, and that stores Events that are generated as effects (i.e. by an Action). The current state of the StateMachine is called *state configuration*. These three are interpreted together by the so called *run-to-completion* (RTC) semantics, which says *"that an event can only be dequeued and dispatched if the processing of the previous current event is fully completed"* (cf. [1, p. 2-149]). Semantically, this is called the interleaving assumption, and it means that executing a StateMachine always results in linear traces - not in partially ordered sets of Actions (or rather, their denotations). One can think of it as the definition of some sort of abstract machine: *"[...] the semantics are described in terms [...] of a hypothetical machine that implements the state machine specification."* (cf. [1, p. 2-143]), see Figure 1.

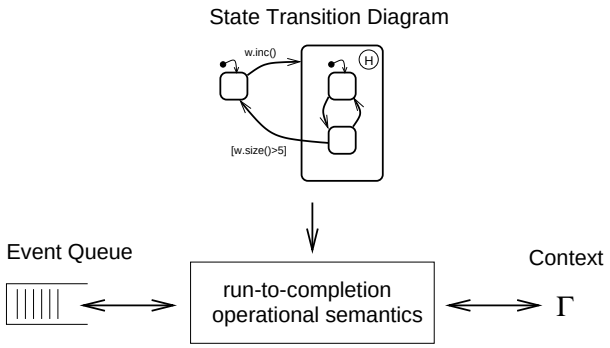

Fig. 1. The semantics of StateMachine (see [1, p. 2-143ff]).

A Collaboration owns (1) a context which is a collection of ClassifierRoles representing the static aspect of the Collaboration, and (2) a set of Interactions as the

[1] As a convention, we use sans-serif font for all UML metaclasses.

[2] The UML uses the term state chart diagram. However, this is a proper name for Harel's notation, and there are substantial differences to them, so we prefer to use the more traditional ad generic term state transition diagram.

dynamic aspect. Each of the Interactions represents an individual run. Without loss of generality, we will assume in this paper, that each Collaboration has exactly one Interaction. An Interaction consists of a set of Messages, that is partially ordered by the predecessors of the Messages. A Message also has activators, which will be ignored here. Virtue to the partial ordering of Messages, interaction diagrams are capable of expressing runs of truly concurrent systems.

3 Interactions as runs

One of the most important applications of interaction diagrams is the specification of sample runs of a system. If the complete behavior of such a system is described by a state transition diagram, one may want to ask whether a given interaction diagram really presents a sample run of a given state transition diagram. To answer such a question, the abstract syntax (i.e. the metamodel-structures representing these diagrams) must be translated to appropriate formal domains, that allow the formalization of the notion "is a run of".

Consider the following example (see Figure 2). A system is composed of two active Classes A and B (left, top), each with a StateMachine (going by the same name) describing their behavior (left, bottom). The interaction diagram (middle) might be a sample run of the system (in this example, it actually is). We assume for the time being, that the overall behavior of the system may be described by a StateMachine that has the StateMachines of its constituents as concurrent regions (right).

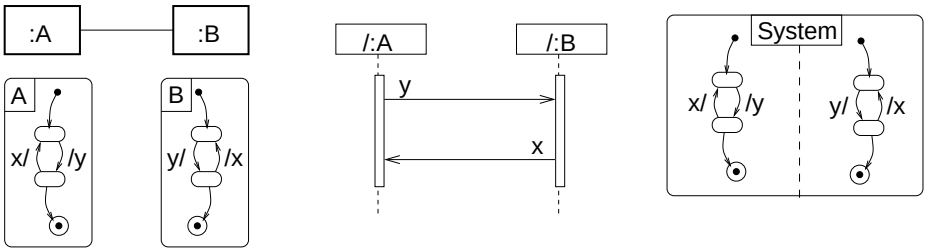

Fig. 2. A system composed of two active Classes with their StateMachines (left), an Interaction (middle), and a StateMachine representing what one would expect intuitively to be the behavior of the overall system (right).

When interpreting an Interaction as a run of a StateMachine, there has to be some relationship between them. In Figure 2, we glossed over this problem by using the same symbols as trigger, effect, and action, i.e. as Event and Action. But the Transitions of a StateMachine have Events as triggers and Actions as effects, while

Interactions exchange Messages between roles, and Message only has an Action and no Event. So, triggers have no general correlation in Message. Only for one special case it is possible to relate Action and Event, see Figure 3.[3]

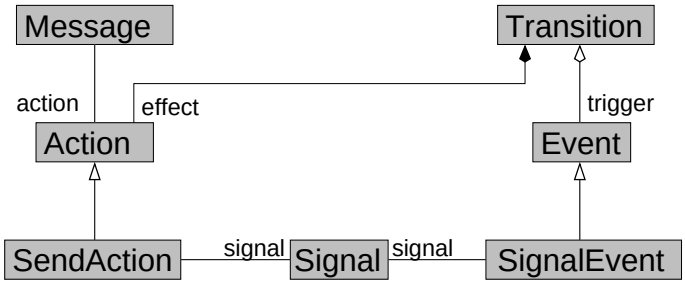

Fig. 3. The relationships between the metaclasses Event, Action, and Signal in the UML metamodel.

By exchanging Signals, Interactions and StateMachines can be related satisfactorily, but this only works for SendAction and SignalEvents. All other Actions and Events that may occur in either of the diagrams must be rendered invisible in a formal semantics, i.e. they must be hidden. Otherwise, many pairs of Interaction and StateMachines would not match, although they should, intuitively. This interpretation makes also sense when seen from another angle: Events that are not the result of an Action (a TimeEvent, say), must be generated outside the context as there is no sender for them *within* the system.

Among all the other orphan issues in UML, this one is particularly painful, as it disables us from providing effective, standard-compliant tool support for many kinds of Action and Event.

4 Concurrent StateMachines

So far, we have considered only individual StateMachines (the StateMachine System in Figure 2, right). But what happens if two systems run concurrently, each of which is described by its own StateMachine (Figure 2, left)? Each and every StateMachine has its *own* abstract machine, with its own event queue, its own event processor, executing its own RTC-steps – concurrently, without synchronization. Events are generated and consumed concurrently by the two StateMachines.

[3] There are interpretations about other correlations via Stimulus, too, but such speculations are not supported by the current version of the standard.

While the context may be shared by a common, enclosing NameSpace, this is *not* the case for event queues. Thus, we speak of an *event space* that encapsulates each StateMachine, individually. Such a concept is missing in the UML. It is not clear, how two such event spaces might communicate, and thus, it is not clear, how two StateMachines might communicate. In other words, it is not true that the system of Figure 2 (right) represents the behavior of our example system.

So, if a semantics is to be able to represent systems with several collaborating StateMachines, is has to be a true concurrency semantics. Obviously, we actually do want to describe true concurrency in systems, particularly at the architecture level. It is not clear how this can be accommodated in the current UML semantics for StateMachines, which seems so entirely devoted to interleaving semantics.

5 An ActionSequence is an Action

Intuitively, the denotation of an ActionSequence should be the same as that of the respective sequence of Actions: suppose, that Actions are translated into elements of an alphabet Σ. One would expect that ActionSequence is mapped to an element from Σ^*. Assuming that a and b are "simple" Actions with denotations a and b, and that c is an ActionSequence with a;b as its action, then both the sequence of a and b, and c should be semantically identical. Put another way, one would expect a certain correspondence between the semantic and the syntactic concatenation operators, namely $[\![\mathsf{c}]\!] = [\![\mathsf{a};\mathsf{b}]\!] = [\![\mathsf{a}]\!];[\![\mathsf{b}]\!]$. Then, for instance, the StateMachines X and Y of Figure 4 would not be distinguishable semantically.

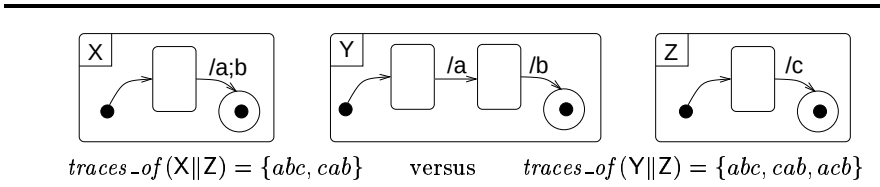

Fig. 4. Let $\mathcal{L}$ denote the set of all terminal linear traces.

The UML standard demands the opposite however: the action of an ActionSequence is a *"sequence of Actions performed sequentially as an atomic unit"* (cf. [1, p. 2-87]). So, when composing the StateMachines X and Y with Z, the resulting systems have different sets of linear traces – i.e. they are distinguishable, even with the weakest possible behavioral semantics. Using Σ^* as the domain for Actions does not solve the problem as $(\Sigma^*)^* = \Sigma^*$. Simply dropping the atomicity constraint for ActionSequence does not solve it either, but rather lead to a conflict with the interleaving assumption underlying the RTC-semantics of StateMachines.

6 Conclusion

6.1 Related work

The subjects raised in this paper have been discussed by various people informally for some time. However, there are no publications that we know of which actually pose these questions (such as papers on the previous UML conferences). The standard itself all but skirts the issues raised. There are some comments on Stimulus, but they are as yet not conclusive.

The recent response to OMG RFP ad/98-11-01 (aka. the Action semantics proposal) deals only with Actions, but leaves out all their applications: *"we have delayed showing how the actions proposed in this submission are used in collaborations."* (cf. [2, p. 12]), *"the definition of the semantics of state machine is outside the scope of the action semantics"* (cf. [2, p. 9]). At least, it does commit itself to a true concurrency semantics for StateMachine (called "relativistic time" there) and distributed state, which is of course absolutely in line with our own observations and proposals, and is a considerable improvement over the previous state of utter undefinedness (see [2, p. 8-9].

6.2 Discussion

Intuitively, interaction diagrams can be interpreted as runs of state transition diagrams. To formalize such a situation, we need semantics for Interaction and StateMachine, the metaclasses representing these diagrams. StateMachines and Interactions may be related only by the Signals of SendAction and SignalEvent. All other Actions and Events have to be hidden in a semantics.

The semantics of StateMachine is defined as an individual abstract machine for each StateMachine. The standard does not specify, how two StateMachines may interact. We use the notion of (disjoint) event spaces to capture this phenomenon. The interleaving assumption underlying the RTC-semantics described in the standard is unrealistic, particularly for distributed systems.

The only decent way out seems to be to drop the restrictive RTC-semantics in favor of a formally coherent semantics based on true concurrency, as it seems to be advocated in [2]. A step towards such a semantics is presented in the forthcoming PhD thesis of the author, see [3].

Acknowledgements Thanks go to Alexander Knapp for stimulating discussions on Stimulus, and to an anonymous referee for suggesting improvements.

References

1. OMG. OMG Unified Modeling Language Specification (version 1.3). Technical report, Object Management Group, 1998. Available at uml.shl.com.
2. OMG. Response to OMG RFP ad/98-11-01: Action Semantics for the UML, August 2000. Available at http://cgi.omg.org/cgi-bin/doc?ad/00-08-03.
3. Harald Störrle. *Models of Software Architecture. Design and Analysis with UML and Petri-nets.* PhD thesis, Ludwig-Maximilians-Universität München, Institut für Informatik, November 2000. to be published.