Extendable and modifiable operational semantics for UML 2.0 sequence diagrams

Mass Soldal Lund and Ketil Stølen
University of Oslo, Norway
SINTEF ICT, Norway
{msl,kst}@sintef.no

Abstract

UML sequence diagrams is a specification language that has proved itself to be of great value in system development. When put to applications such as simulation, testing and other kinds of automated analysis there is a need for formal semantics. Such methods of analysis are by nature operational, and this motivates formalizing an operational semantics. We present an operational semantics for UML sequence diagrams, which we believe gives a solid starting point for developing methods for automated analysis. The operational semantics can be proved to be sound and complete with resect to to a denotational semantics for the same language. It handles negative behavior as well as potential and mandatory choice. We are not aware of any other operational semantics of this strength.

UML sequence diagrams [5] and their predecessor Message Sequence Charts (MSC) [4] are specification languages that have proved themselves to be of great practical value in system development. When sequence diagrams are used to get a better understanding of the system through modeling, as system documentation or as a means of communication between stakeholders of the system, it is important that the precise meaning of the diagrams are understood; in other words, there is need for a well-defined semantics. Sequence diagrams may also be put to further applications, such as simulation, testing and other kinds of automated analysis. This further increase the need for a formalized semantics; not only must the people that make and read diagrams have a common understanding of their meaning, but also the makers of methods and tools for analyzing the diagrams must share this understanding.

Methods of analysis like simulation and testing are in their nature operational; they are used for investigating what will happen when the system is executing. When developing techniques for such analysis, we not only need to understand the precise meaning of a specification, we also need to understand precisely the executions that are specified. This motivates formalizing an operational semantics. We present an operational semantics for UML sequence diagrams, which we believe gives a solid starting point for developing such methods for analysis.

Sequence diagrams is a graphical specification language defined in the UML 2.0 standard [5][1] for specifying interaction between communicating objects represented by *lifelines*. The standard defines the graphical notation, but also an

[1]In the UML standard, *Interaction* is used as the common name for diagrams specifying

abstract syntax for the diagrams. Hence the language has a well-defined syntax. The semantics of sequence diagrams provided by the standard, however, is informal and defined by the means of natural language. Most notably, this is a trace based semantics: The semantics of a diagram is defined as a pair (p, n) of sets of valid and invalid traces characterized by the diagram. p and n does not need to by disjoint and the union of p and n need not exhaust the universe of traces.

In [2,3] a denotational semantics for sequence diagrams is formalized, called the STAIRS semantics. The STAIRS semantics is trace based and uses an extended version of the basic semantic model from the UML standard. Instead of a single pair (p, n) of valid and invalid traces, the semantic model of STAIRS is a set of pairs $\{(p_1, n_1), (p_2, n_2), \ldots, (p_m, n_m)\}$. A pair (p_i, n_i) is referred to as *interaction obligation*. The word obligation is used in order to emphasize that an implementation of a specification is required to fulfill every interaction obligation of the specification. Our operational semantics can be proved to be sound and complete with resect to this denotational semantics.

The operational semantics is defined by the means of a combination of two transition systems, which we refer to as an *execution system* and a *projection system*. The execution system is a transition system

$$[\_, \_] \in \mathcal{B} \times \mathcal{D} \tag{1}$$

where $\mathcal{B}$ represents the set of all states of the communication medium and $\mathcal{D}$ the set of all syntactically correct sequence diagrams. The execution system handles the syntactical representation of the sequence diagram and the communication medium, and is defined so that it allows for variations in the latter.

The projection system is a transition system

$$\Pi(\_, \_, \_) \in \mathbb{P}(\mathcal{L}) \times \mathcal{B} \times \mathcal{D} \tag{2}$$

where $\mathbb{P}(\mathcal{L})$ is the powerset of the set of all lifelines. The projection system is used for finding enabled events at each stage of the execution and is defined recursively in a way allowing fairness between the lifelines in the sequence diagram. This system makes sure the partial order of events characterized by the diagram is preserved and handles global choices, i.e., choices involving several lifelines.

These two systems work together in such a way that for each step in the execution, the execution system updates the projection system by passing on the current state of the communication medium, and the projection system updates the execution system by returning the state of the diagram after the execution of the event.

We also formalize a meta level that encloses the execution system. This meta level is necessary for distinguishing valid from invalid traces, and for distinguishing between traces of different interaction obligations. Further it may be used for defining different meta strategies that guide the execution. Examples of this may be generating all traces or a specific number of random traces, generating traces for a white box view or for a black box view. We have also, with the use of a suitable meta-level, implemented the the test generation algorithm of [6] on top of the operational semantics for the fragment of valid behavior.

interaction by sending and receiving of messages. Sequence diagrams are then one kind of Interaction

We are not aware of any other operational semantics for sequence diagrams as defined in UML 2.0. Several approaches approaches of defining semantics for MSCs have been made, but with certain shortcomings. What these approaches have in common is a lack of modifiability and extensibility, e.g., with respect to the communication model, and a lack of possibility and freedom in defining and formalizing the meta level. A further characteristic of these approaches is that they all require a transformation from the textual syntax of MSC into the formalism applied in the approach.

Our operational semantics for UML 2.0 sequence diagrams is simple and is defined with extensibility and variation in mind. It does not involve any translation or transformation of the diagrams into other formalisms, which makes it easy to use and understand. It is sound and complete with respect to a reasonable denotational formalization of the UML standard, and have a formalized meta level for defining execution strategies.

The operational semantics has been implemented with use of the rewrite language Maude [1]. This implementation forms the basis of analysis tool currently under development.

References

[1] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *Maude Manual (Version 2.1.1)*. SRI International, Menlo Park, April 2005.

[2] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 2005. To appear.

[3] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Why timed sequence diagrams require three-event semantics. In *Scenarios: Models, transformations and tools. International Workshop, Dagstuhl Castle, Germany, September 2003. Revised selected papers*, number 3466 in Lecture Notes in Computer Science, pages 1–25. Springer-Verlag, 2005.

[4] ITU-T. *Message Sequence Chart (MSC), ITU-T Recommendation Z.120 (11/1999)*, 1999.

[5] OMG. *Unified Modeling Language: Superstructure, version 2.0, Final Adopted Specification*. Object Management Group, 2004. OMG Document: ptc/2004-10-02.

[6] Jan Tretmans. Testing techniques. Reader, Univeriteit Twente, 2002.