# Strengthening the Semantics of UML Collaboration Diagrams*

Reiko Heckel and Stefan Sauer

University of Paderborn, Dept. of Mathematics and Computer Science
D-33095 Paderborn, Germany
reiko|sauer@uni-paderborn.de

## 1 Motivation

A serious weakness of the UML is the lack of appropriate means for specifying the semantics of operations in a class diagram and the interpretation of actions on a statechart diagram. As required in the corresponding *request for proposals* [11], such specification should describe the pre- and postconditions of operations and actions, their effect on the current state, as well as the calls or signals that are sent during their execution.

The semantics of *actions*, like call or send actions, has to be the same in all models, that is, it should be specified once and for all on the meta level. As *operations* and their interpretation differ in every model, they have to be specified on the level of individual models. In both cases, it is desirable to use UML to specify the semantics: In the case of operations, it has to be given by application developers, i.e., users of the UML, who should not be forced to learn yet another notation. And by specifying the semantics of actions by UML diagrams on the meta level, people without a strong background in formal methods, like tool developers, teachers, and advanced users, could benefit from the semantics specification.

With this motivation, in [8] a meta-modeling approach to the semantics of call actions in statechart diagrams is proposed which is based on collaboration diagrams. In a collaboration diagram, a *collaboration* specifying a graph of objects and links provides the context of an *interaction* describing the flow of messages and signals. The integration of structure and communication makes these diagrams suitable for specifying the semantics of operations and actions. However, the semantic weakness of collaboration diagrams, in particular concerning the collaboration part, limits their expressiveness and prevents their use for precise specifications. In fact, according to the UML specification [12], a collaboration is just the context of an interaction. It does not entail any structural requirements beside the obvious one that objects and links have to be present as soon as they are involved in an interaction.

In this note, we sketch how, without changing the syntax, the semantics of collaboration diagrams can be strengthened (and formalized) in order to allow for the specification of pre- and postconditions and structural effects thus providing a powerful means for action semantics on both the model level and the meta model level.

* Research partially supported by the ESPRIT Working Group APPLIGRAPH.

## 2 Collaborations as Graph Transformations

A collaboration diagram is based on a collaboration, i.e., a graph of objects, links, and attribute values conforming to a class diagram. The relation between class and instance diagrams can be formalized by the notion of *typed graphs* [4]. On this basis, a powerful interpretation of collaborations can be given in terms of rule-based *graph transformations* (see, e.g., [14, 6, 7] for a collection of surveys and [1] for an introductory text). In fact, using the constraints {new} and {destroyed}, a collaboration can specify the manipulation of object graphs. Thus, a collaboration on the specification level represents a graph transformation *rule* while collaborations on the instance level represent individual transformations.

A *graph transformation rule* $r = L \to R$ consists of a pair of graphs $L, R$ such that the union $L \cup R$ is defined. (This ensures that, e.g., edges which appear in both $L$ and $R$ are connected to the same vertices in both graphs.) The precondition $L$ contains all objects and links which have to be present before the operation, i.e., all elements of the collaboration diagram except for those marked as {new}. Analogously, the postcondition $R$ contains all elements not marked as {destroyed}. The {transient} constraint does not occur because the graph transformation specified by a rule is supposed to be atomic, i.e., there are no intermediate states with additional objects.

A *graph transformation* $G \overset{r(o)}{\Longrightarrow} H$ from a pre-state $G$ to a post-state $H$ is given by a subgraph isomorphism $o : L \cup R \to G \cup H$, called *occurrence*, such that

- $o(L) \subseteq G$ and $o(R) \subseteq H$, i.e., the left-hand side of the rule is embedded in the pre-state and the right-hand side into the post-state
- $o(L \setminus R) = G \setminus H$ and $o(R \setminus L) = H \setminus G$, i.e., that part of $G$ is {destroyed} which is matched by elements of $L$ not belonging to $R$ and, symmetrically, that part of $H$ is {new} which is matched by elements new in $R$.

Notice that, in order to be applicable to an object graph $G$, a graph transformation rule requires the presence of the objects and links specified in its left-hand side $L$. This provides us with powerful means to specify, through pattern matching, application conditions for operations. In the context of collaboration diagrams, this means that all objects and links of the diagram not marked as {new} have to be present before the start of the interaction.

Based on the rigorous interpretation of collaborations as graph transformation rules, a visual programming language is developed in [9] using collaboration diagrams for expressing the semantics of operations. However, graph transformations as described above do not provide a concept of interaction. Rather, a transformation step can be seen as an abstraction of a complex scenario to its pre and post-state. Next, we shall elaborate on this concept by enriching it with the history of the process that caused the state transformation.

## 3 Collaboration Diagrams as Graph Processes

A sequential model of a transformation process is given by a sequences $\tau$ of transformation steps, called a *trace*. A trace can be seen as an expanded rep-

resentation of a collaboration diagram on the instance level specifying a single, sequential scenario. For this purpose, the visible effects of the operations called within the diagram are specified by means of rules. Signals are represented as objects of class Signal, i.e., they are part of the object graph so that they can be put in queues and processed asynchronously. Thus, a trace represents a linear order of operation calls and send actions.

However, the natural interpretation of an interaction, suggested by the UML standard [12] and related approaches like message sequence charts [10], is a *partial* order of actions. In order to represent this partial order, we abstract from the concrete ordering of steps in a sequence. The structure obtained in this way, called *graph process* [4], records only the causal dependencies between the steps by means of the occurrences of the rules, while the information about the scheduling of independent steps is lost. Conceptually, a graph process $\rho$ is a set of rules $r = L \to R$ which are embedded in a *core graph* $C$. In order to make sure that a sequential scheduling exists, the existence of a linear order on $\rho$ is required which turns the process $\rho$ into a trace $\tau$. The core graph can be thought of as the union of all the graphs in $\tau$ containing all objects and links of the initial and the final state as well as transient elements which exist only in intermediate states.

From the point of view of collaboration diagrams, the core graph $C$ is the object graph of the collaboration where $G \setminus H$ contains the objects to be {destroyed}, $H \setminus G$ are the {new} objects, and $C \setminus (G \cup H)$ represents {transient} elements of the diagram. The occurrences of rules in $\rho$ represent method calls from within the diagram, send actions of signals, as well as local operations.

Graph processes have evolved as a generalization of processes of place-transition Petri nets [13] with similar aims and results (see [2] for a survey). In particular, the causal dependencies between the transformations in a process $\rho$ are represented by a partial order $\leq_\rho$ which can be derived from the process by analyzing the overlapping of the occurrences of the rules in $C$. This partial order, which is defined both on rules and on objects and links, represents the partial order on messages and signals required by the semantics of interactions [12].

The model described so far allows to represent collaboration diagrams both on the instance and on the specification level (as long as they do not contain control structures such as conditionals or loops). It remains to answer the question how these two levels are related. Therefore, in the next section, we will sketch how diagrams on the specification level can be used as graphical deduction rules in order to derive diagrams on the instance level representing computations.

## 4 A Proof-Theoretic Interpretation

A specification-level diagram, represented as a process $\rho$ with initial state $G$ and final state $H$, expresses the following conditional statement. A situation matching the pattern $G$ can be transformed as specified by $H$ provided that the set of actions $\rho$ can be performed. In order to stress this interpretation, a

process shall be denoted as a graphical deduction rule in the SOS style (cf. [3])

$$[\rho] \quad \frac{\{L_i \overset{op_i(x_i)}{\longrightarrow} R_i\}_{i\in I}}{G \overset{op(x)}{\longrightarrow} H}$$

where the $L_i \overset{op_i(x_i)}{\longrightarrow} R_i$ represent the premises of the rule and $G \overset{op(x)}{\longrightarrow} H$ represents the conclusion.[1]

A proof using such rules represents a computation, i.e., a collaboration diagram on the instance level. The main operation for building such proofs corresponds to the invocation of an operation. It matches (an instance of) the $k$th premise of rule $\rho$ with the conclusion of another rule

$$[\rho'] \quad \frac{\{L'_j \overset{op'_j(x'_j)}{\longrightarrow} R'_j\}_{j\in J}}{G' \overset{op'(x')}{\longrightarrow} H'}$$

where $op_k = op'$, thus forming a new rule

$$[\rho \circ_k \rho'] \quad \frac{\{L_i \overset{op_i(x_i)}{\longrightarrow} R_i\}_{i\in I\setminus\{k\}}, \{L'_j \overset{op'_j(x'_j)}{\longrightarrow} R'_j\}_{j\in J}}{G'' \overset{op(x)}{\longrightarrow} H''}$$

The conclusion $G'' \overset{op(x)}{\longrightarrow} H''$ is obtained as an instance of $G \overset{op(x)}{\longrightarrow} H$ by adding the new pre- and postconditions and effects induced by the subprocess $\rho'$. For the collaboration diagrams, this means to substitute the call $op_k(x_k)$ within the calling diagram $\rho$ by the collaboration diagram $\rho'$ specifying the implementation of $op_k$. A second, more basic operation on diagrams is the sequential composition $\rho; \rho'$ which is defined if the post-state $H$ of $\rho$ is equal to the pre-state $G'$ of $\rho'$.

In this way, a calculus of collaboration diagrams can be built which allows to combine simple specification level diagrams to bigger diagrams on the instance level. By recording the structure of the proof, e.g., using a proof term like $[\rho \circ_k \rho']$ above, we are able to represent collaboration diagrams with more than one level of method invocation.

## 5 Conclusion

In this note, we have proposed a semantics for collaboration diagrams based on graph transformation rules and graph processes which makes them a powerful tool for the precise specification of operations and actions. Graph processes provide a truly concurrent semantics to collaboration diagrams which can be helpful for analyzing the concurrency properties of operations in terms of the associated causal dependencies. In particular, the semantics of signals (implemented as particular objects) is compatible with that of asynchronous messages in a message sequence chart [10] given as partial order of send and receive actions (cf. [5] for details).

[1] If used to specify an operation *op* of class $c^0$, a rule $L \to R$ is equipped with a label $op(x)$ where $x = x^0, \ldots, x^n$ with $x^l \in L \cup R$ of class $c^l$ are the formal parameters of the operation. The *self* object $x^0$ of class $c^0$ is the first parameter of the operation.

The synchronous semantics of operation calls is captured by the substitution of the called diagram for the call in the calling diagram, which is formally described as the composition of two deduction rules over graph transformations.

The proof-theoretic interpretation of collaboration diagrams provides a basis for implementation, e.g., in a theorem prover, logic programming or (conditional) rewriting system. This is particularly important if collaboration diagrams are used for dynamic meta modeling [8] in order to analyze, test, and verify the semantics specification.

## References


1. M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Plump, A. Schürr, and G. Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, 34:1–54, 1999. `http://www.uni-paderborn.de/cs/ag-engels/Papers/AndriesSCP99.pdf`.
2. P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent semantics of algebraic graph transformation. In Ehrig et al. [7], pages 107 – 188.
3. A. Corradini, R. Heckel, and U. Montanari. Graphical operational semantics. In A. Corradini and R. Heckel, editors, *Proc. ICALP2000 Workshop on Graph Transformation and Visual Modelling Techniques, Geneva, Switzerland*, Geneva, July 2000. Carleton Scientific. `http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/CorradiniGTVMT00.pdf`.
4. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26(3,4):241–266, 1996.
5. R. Depke, R. Heckel, and J. M. Küster. Modeling agent-based systems with graph transformation. Technical Report No. 213, University of Paderborn, Dep. of Comp. Sci., August 2000. `http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/Depke00TR.pdf`.
6. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*. World Scientific, 1999.
7. H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency and Distribution*. World Scientific, 1999.
8. G. Engels, J.H. Hausmann, R. Heckel, and St. Sauer. Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In *Proc. UML 2000*, LNCS. Springer-Verlag, 2000. To appear, `http://www.uni-paderborn.de/cs/ag-engels/Papers/2000/EngelsUML00.pdf`.
9. T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story diagrams: A new graph transformation language based on UML and Java. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT'98), Paderborn, November 1998*, volume 1764 of *LNCS*. Springer-Verlag, 2000.
10. ITU-T, Geneva. *Recommendation Z.120: Message Sequence Chart (MSC)*, 1993. See also: *http://www.win.tue.nl/win/cs/fm/sjouke/msc.html*.
11. Object Management Group. Action semantics for the UML, November 1998. `http://www.omg.org/pub/docs/ad/98-11-01.pdf`.
12. Object Management Group. UML specification version 1.3, June 1999. `http://www.omg.org`.
13. W. Reisig and U. Goltz. The nonsequential behaviour of Petri nets. *Information and Control*, 57(2,3), 1983.
14. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*. World Scientific, 1997.