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

Abstract. Collaboration diagrams are strengthened by interpreting collaborations as visual queries for specifying pre and postconditions of operations. The conceptual idea is formalized by means of graph transformation rules and graph processes. A proof-theoretic interpretation is given which accounts for the composition of diagrams.

Keywords: collaboration diagrams, graph transformation, action semantics

1 Introduction and Motivating Example

A weakness of the UML is the lack of appropriate means for specifying the semantics of operations [13]. Such specification should describe the pre- and postconditions of operations, their effect on the current state, as well as the calls or signals that are sent during their execution. A solution recently proposed is the *action specification language* [1] (ASL) which allows a declarative style of specification based, e.g., on SQL-like queries and a pre/post-conditions for specifying operations.

However, given that the success of object-oriented modeling is largely due to the use of diagrams, it may be questioned if the majority of UML users can be motivated to learn yet another textual language. Moreover, we believe that the weakness described in [13] is not so much due to lack of appropriate diagrammatic notation, but due to the lack of precise semantics.

In this paper, we explore the possibility to specify the semantics of operations by means of UML collaboration diagrams. In a collaboration diagram, a *collaboration* representing a pattern of objects and links provides the context of an *interaction* describing the flow of messages. This integration of structure and behavior provides the basis for specifying the semantics of operations.

As an example, consider the collaboration diagram in the top of Fig. 1 specifying the implementation of operation processOrder. Collaboration diagrams with simple (unnested) sequence numbers are sufficient for this purpose since each diagram corresponds to the body of one operation. In fact, diagrams of this kind may be used for generating method implementations in Java [8], i.e., they can be seen as visual representations of programs.

* Research partially supported by the ESPRIT Working Group APPLIGRAPH.

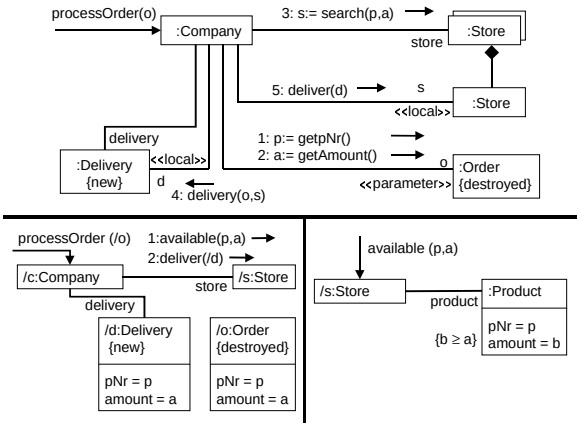

Fig. 1. An implementation-oriented collaboration diagram (top), its declarative presentation (bottom left), and a visual query operation (bottom right).

However, the semantic weakness of collaboration diagrams, in particular concerning the collaboration part, limits their expressiveness. In fact, according to the UML specification [12], the collaboration is just the context of the 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. This enforces a low-level style of specification as witnessed in the example by the use of get functions for attributes and the modeling of queries by multi-objects, composition, and search functions. We claim that, leaving out implementation details, the same operation can be specified by the diagram in the lower left of Fig. 1 where collaborations are seen as a visual query and update language on object graphs. As a second example, in the lower right of the same figure the specification of the predicate available is shown which succeeds if the :Store object matching /s is connected to a :Product object with the required product number p and an amount b greater than a.

In the following sections, we sketch how this strengthening of collaboration diagrams can be formalized by means of graph transformation rules and graph processes.

2 Collaborations as Graph Transformations

A collaboration on specification level is a graph of classifier roles and association roles which specifies a particular view of the classes and associations of a class diagram as well as a pattern for objects and links on the instance level. Such a pattern can be used as a query by requiring the existence of a *matching*, i.e., a structure-preserving mapping of classifier roles to objects, mathematically described as a subgraph isomorphism. On this basis, an interpretation of collaborations as query and update language for object structures can be given in terms

of rule-based *graph transformations* (see, e.g., [2] 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 can be used to visualize individual transformations.

A *graph transformation rule* $r = L \rightarrow R$ 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.) Consider the rule in Fig. 2 representing the collaboration of processOrder in the lower left of Fig. 1. The precondition L contains all objects and links which have to be present before the operation, i.e., all elements of the diagram except for /d:Delivery which is marked as {new}. Analogously, the postcondition R contains all elements except for /o:Order which is marked as {destroyed}. (Note that the {transient} constraint does not (yet) occur because the graph transformation specified by a rule is supposed to be atomic, i.e., there are no intermediate states with additional objects.)

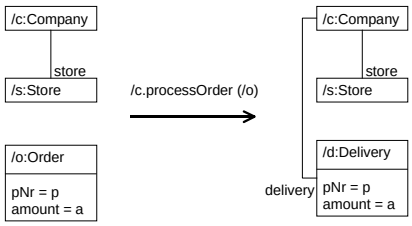

Fig. 2. Pre- and postcondition of operation processOrder.

Formally, 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 $o : L \cup R \rightarrow G \cup H$, called *occurrence*, such that $o(L) \subseteq G$ and $o(R) \subseteq H$ (i.e., the precondition of the rule is matched by the pre-state and the postcondition by the post-state), $o(L \setminus R) = G \setminus H$ and $o(R \setminus L) = H \setminus G$ (i.e., all objects of G are {destroyed} that match classifier roles of L not belonging to R and, symmetrically, all objects of H are {new} that match classifier roles in R not belonging to L). Thus, a collaboration with constraints {new} and {destroyed} allows the visual specification of pre- and postconditions and effects on the current state.

Based on this interpretation of collaborations as graph transformation rules, a visual programming language has been developed [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

An interaction represents the causality and concurrency between messages by means of a partial order, called *precedence relation* [12]. To represent such a partial order, a solution which is popular for its simplicity is to build all possible linearizations, i.e., to model concurrency by non-determinism. If both concurrency and non-determinism are of interest, this interleaving approach to interactions (which is taken, e.g., in [14]) might be considered too abstract as it hides the difference between the two concepts. (A situation where both concepts are important in their own right can be easily imagined because, first, many of today's software systems are concurrent and distributed, and second, non-determinism is an important means of abstraction in a model where we may not want to specify the full control flow of the implementation.) A semantics of interactions based on partial orders is developed in [11].

The key to partial order semantics is to abstract, in an individual run, from the ordering of operations which are not causally dependent, i.e., which appear in this order only by accident or because of the strategy of a particular scheduler. In the theory of graph transformation, this abstraction is realized by the construction of a *graph process* [5]. The graph process for the collaboration diagram in the lower left of Fig. 1 is shown in Fig. 3. The graph of the collaboration in the center provides the context for the rules above and below which model the actions specified within the collaboration diagram: The two rules in the top represent the two call actions where the auxiliary round vertex named 1 results from the sequence number 1: of the call to available. It ensures that the second operation is called only after the first is completed. The rules in the bottom result from the destroy and create actions in the diagram.

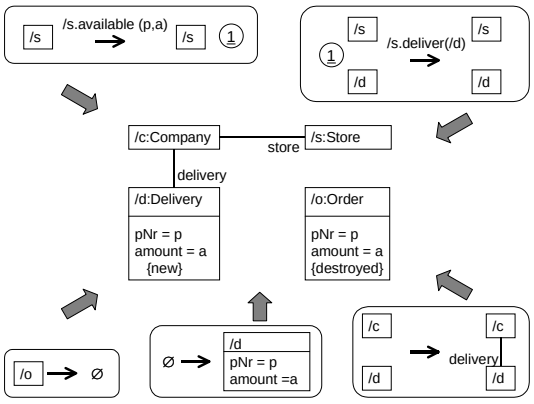

Fig. 3. Graph process for the collaboration diagram of operation processOrder.

Beside the sequence numbers specifying the precedence relation, the causality of actions is prescribed by the data dependencies, e.g., if an action creates an

object which is used by another one. This is precisely reflected in the process where the causal order is determined by the overlapping of the occurrences of the left- and right-hand sides of the rules in the graph of the collaboration. In our example, both the creation of the delivery link and the invocation of deliver depend on the creation of the object playing the role of /d. In addition, deliver depends on the completion of available due to the vertex 1 which occurs both in the left-hand side of deliver and in the right-hand side of available.

Graph processes have evolved as a generalization of processes of place-transition Petri nets [15] with similar aims and results (see [3] for a survey). In particular, the definition of graph process [5] entails that the causal relation is acyclic and free of conflicts (in the sense that every object may only be created and destroyed once) and that a rule can only be applied to an object after it is created and before it is destroyed.

The model described so far allows to represent collaboration diagrams on the specification level (as long as they do not contain control structures such as conditionals or loops). In order to represent actual computations, a notion of composition of diagrams is required which reflects, e.g., the invocation of operation available from within processOrder. 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 collaboration diagram specifying an operation like processOrder in the bottom of Fig. 1, expresses the following conditional statement. A situation matching the precondition of the diagram (given by all classifier and association roles not marked as {new}) can be transformed as specified by the postcondition (given by all classifier and association roles not marked as {destroyed}) provided that the call actions available and deliver can be performed. In order to stress this interpretation, the collaboration diagram can be denoted as a graphical deduction rule in the SOS style (cf. [4])

$$\frac{L_1 \overset{/\mathsf{s.available(p,a)}}{\longrightarrow} R_1, L_2 \overset{/\mathsf{s.deliver(p,a)}}{\longrightarrow} R_2}{L \overset{/\mathsf{c.processOrder(p,a)}}{\longrightarrow} R}$$

where $L_1 \to R_1$ and $L_2 \to R_2$ are the two rules in the top of Fig. 3 and $L \to R$ is the rule in Fig. 2. An analogous presentation of the collaboration diagram for available in Fig. 1 on the right leads to an axiom (a deduction rule without premise).

$$\frac{}{L' \overset{/\mathsf{s.available(p,a)}}{\longrightarrow} R'}$$

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, e.g., the first premise of the

rule for processOrder with the conclusion of the rule for available thus forming the proof tree[1]

$$\frac{\overline{L' \overset{/\mathsf{s.available(p,a)}}{\longrightarrow} R'}, L_2 \overset{/\mathsf{s.deliver(p,a)}}{\longrightarrow} R_2}{L'' \overset{/\mathsf{c.processOrder(p,a)}}{\longrightarrow} R''}$$

The conclusion $L'' \overset{/\mathsf{c.processOrder(p,a)}}{\longrightarrow} R''$ is obtained from $L \overset{/\mathsf{c.processOrder(p,a)}}{\longrightarrow} R$ by adding the new pre- and postconditions and effects induced by the subprocess $L' \overset{/\mathsf{s.available(p,a)}}{\longrightarrow} R'$. For the collaboration diagrams, this means to substitute the call to available in the bottom left diagram of Fig. 1 by the collaboration diagram in the bottom right of the same figure specifying the implementation of available. The resulting diagram on the instance level is shown in Fig. 4.

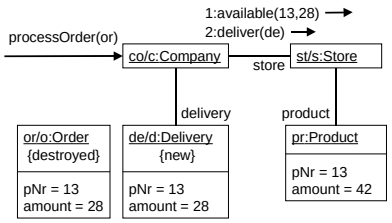

Fig. 4. Collaboration diagram on the instance level.

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 in a proof tree like above, we are able to represent collaboration diagrams with more than one level of method invocation.

5 Conclusion

In this paper, we have proposed a strengthening of collaboration diagrams which makes them suitable for the precise specification of operations. A formal interpretation of this extended semantics in terms of graph transformation rules and graph processes is provided which accounts for concurrency within interactions. An interpretation of collaboration diagrams (or graph processes) as deduction rules allows to model the invocation of operations as composition of rules. This proof-theoretic interpretation provides a basis for implementation in a theorem prover or logic programming system which is particularly important if collaboration diagrams are used for dynamic meta modeling [7] (which was, in fact, the original motivation of this work) in order to test and verify the semantics

[1] Since no specification for deliver is given, this branch of the proof tree remains unresolved.

specification. An alternative to a logic-based implementation is the translation into an executable specification language like ASL [1] which could be defined in analogy to the code generation from implementation-oriented collaboration diagrams in Java [8].

Finally, note that our formalization can also be applied to "ordinary" collaboration diagrams which do not employ the collaboration pattern as a query. We leave as future work the modeling of asynchronous communication and of more elaborate control structures.

References

1. Action Semantics Consortium. Precise action semantics for the Unified Modelling Language, August 2000. `http://www.kc.com/as_site/`.
2. 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.
3. P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent semantics of algebraic graph transformation. In Ehrig et al. [6], pages 107–188.
4. 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, July 2000.
5. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26(3,4):241–266, 1996.
6. 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.
7. 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.
8. G. Engels, R. Hücking, St. Sauer, and A. Wagner. UML collaboration diagrams and their transformation to Java. In France and Rumpe [10], pages 473–488.
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. TAGT'98*, volume 1764 of *LNCS*. Springer-Verlag, 2000.
10. R. France and B. Rumpe, editors. *Proc. UML'99 – Beyond the Standard*, volume 1723 of *LNCS*. Springer-Verlag, 1999.
11. A. Knapp. A formal semantics of UML interactions. In France and Rumpe [10], pages 116–130.
12. OMG. UML specification version 1.3, June 1999. `http://www.omg.org`.
13. OMG. Action semantics for the UML - request for proposals, November 1998. `http://www.omg.org/pub/docs/ad/98-11-01.pdf`.
14. G. Övergaard. A formal approach to collaborations in the unified modeling language. In France and Rumpe [10], pages 99–115.
15. W. Reisig and U. Goltz. The nonsequential behaviour of Petri nets. *Information and Control*, 57(2,3), 1983.