Proc. ICGT Workshop Graph Transformation and Visual Modeling Techniques (GT-VMT'2002), Paolo Bottoni, Mark Minas (Eds.), Electronic Notes in Theoretical Computer Science (ENTCS), Elsevier (October 2002).

Towards an Integrated Graph Based Semantics for UML

Martin Gogolla [a,1] Paul Ziemann [a,2] Sabine Kuske [a,3]

[a] *Department of Computer Science*
University of Bremen
Bremen, Germany

Abstract

Recently, we proposed an integrated formal semantics based on graph transformation for central aspects of UML class, object and state diagrams. In this paper, we explain the basic ideas of that approach and show how two more UML diagram types, sequence and collaboration diagrams, can be captured. For UML models consisting of a class diagram and particular state diagrams, a graph transformation system can be defined. Its graphs are associated with system states and its rules with operations in the class diagram and transitions in the state diagrams. Sequence and collaboration diagrams then characterize sequences of operation applications and therefore sequences of transformation rule applications. Thus valid sequence and collaboration diagrams correspond to derivations induced by the graph transformation system. Proceeding this way, it can be checked for example whether such an operation application sequence may be applied in a specific system state.

1 Introduction

In software development, analysis and design are important tasks. The Unified Modeling Language (UML) is a visual language supporting the software engineer using object-oriented techniques in these phases by providing diagram types for modeling various aspects of a system [BRJ98,RJB98,FS97]. UML integrates previously existing languages including Booch [Boo94], OMT [RBP+91], and OOSE [JCJO92]. The language was accepted as an industrial standard by the Object Management Group (OMG) and is specified semi-formally in [OMG01]. A lot of research has been done in recent years to formalize parts of UML. However, defining a formal semantics for the UML

1 Email: gogolla@informatik.uni-bremen.de
2 Email: ziemann@informatik.uni-bremen.de
3 Email: kuske@informatik.uni-bremen.de

UML notion	Notion in the graph transformation approach
UML class diagram	set of system states where each system state is a graph AND a set of graph transformation rules as semantics for operations
UML object diagram	system state
UML state diagram	graph transformation rules transforming system states into system states
UML sequence diagram	derivation in the defined graph transformation system
UML collaboration diagram	derivation in the defined graph transformation system

Table 1
UML and Graph Transformation Notions

as a whole is very difficult due to the vast extent of the UML. In particular, it is difficult not to restrict the generality of UML within such a formalization.

A UML model can describe structure (i.e., possible states of the system to be modeled) and behavior (i.e., state transitions or state sequences due to applications of operations). In order to formalize UML, it has to be stated, how a system state exactly looks like and a translation from UML models to sets of state sequences has to be given.

One recent such formalization approach is presented in [KGKK02], where a translation of a UML class diagram together with UML statechart diagrams into a graph transformation system is described. The graphs of the graph transformation system are called *system states*. A system state is characterized here by a set of objects with attribute values and links among each other. Additionally, the objects possess an object state and an event queue. These graphs include UML object diagrams, so an integrated semantics for central aspects of UML class, object, and state diagrams is given. The rules of the transformation system correspond to user defined operations in the classes and transitions in the statechart diagrams modifying the system state. The set of possible derivations of the transformation system is the formalization of the possible state sequences and the possible operation appplications the UML model specifies informally.

In this paper, we show that sequence and collaboration diagrams can easily be integrated into the approach [KGKK02] mentioned above. Altogether we obtain an operational formal semantics based on graph transformation for a large part of UML. Table 1 shows the notions from UML we use and the corresponding notions in the world of graph transformations.

The structure of the paper is as follows: In Sect. 2, the basics of graph transformation that are needed to understand our graph transformation approach are briefly reviewed. In Sect. 3 and 4, the approach of [KGKK02] is demonstrated by giving a concrete UML example model and the corresponding graph transformation system. Section 5 then describes the relationship between sequence resp. collaboration diagrams and derivations of the graph transformation system. It is shown how these derivations can help to check whether the model is adequate, for example, for a check whether a given message sequence is applicable in a system state or not. Section 6 shortly touches related work. We conclude in Sect. 7 with a summary and a discussion of current shortcomings that have to be faced in future work.

2 Graph Transformation

Graph transformation [Roz97,EEKR99,EKMR99] is a generalization of Chomsky grammars to graphs. The simplest form of a graph transformation system (which is the one we need) consists a set of graph transformation rules together with an initial graph.

A *graph* is a set of (labeled) nodes and (labeled) edges connecting nodes.

A *graph transformation rule* mainly consists of a graph on the left-hand side and a graph on the right-hand side which have a common part. In order to apply a rule to a graph, a match of the left-hand side to the graph has to be found. Then the matched part of the graph is removed and the right-hand side is pasted instead where the common part constitutes the connection to the rest of the graph. We follow the double pushout approach to graph transformation; that means that (1) it is not allowed that there is an element in the matched part with two inverse images of the left-hand side of the rule with one of them being present in the right-hand side and one not (identification condition) and (2) removing the matched part of the graph must not result in dangling edges (dangling condition).

In this paper, rules consist of one system state on the left-hand side and one on the right-hand side. The initial graph is often the empty graph.

3 Example UML Model

The UML diagrams shown in Fig. 1 and 2 model an office scenario with bosses, secretaries, tapes (used to record letter dictations on it), letters (electronic documents on an electronic device), printouts (of letters), and printers (producing printouts). The class diagram in Fig. 1 shows the corresponding classes with their attributes, operations and associations. Fig. 2 displays statechart diagrams for the classes *Boss*, *Secretary*, and *Printer*. In the approach proposed in [KGKK02], only call events and call actions are considered, so we chose appropriate statecharts.

The office process modeled by these diagrams is as follows: A boss takes a dictation of a letter on tape, then gives it to her or his secretary for typing it. The secretary calls the printer to print the letter. The boss reads the printout and then either signs it and tells the secretary to mail the letter or asks the secretary to adjust it. After adjusting, the letter is printed by the printer and read by the boss again.

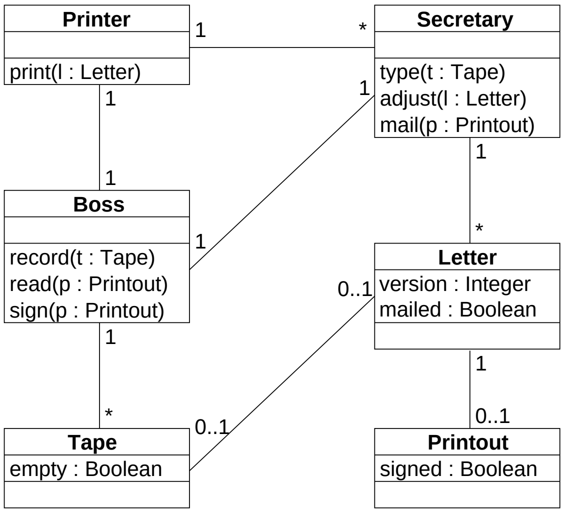


Fig. 1. Class Diagram

The approach in [KGKK02] requires that the effect of operations in classes are given by pairs of object diagrams, i.e. a graph transformation rule. The first (left) object diagram describes (part of) the system state before execution of the operation, and the second (right) describes the changes the execution effects. We only specify this pair for the operation `record(t)` in class `boss` in Fig. 3.

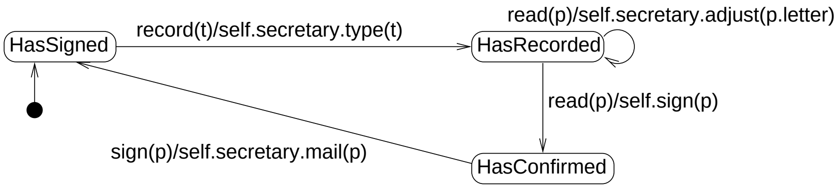
The effect of `record(t)` is that the value of the attribute `empty` of the tape `t` is changed from `true` to `false`. Thus, the operation is not applicable if the tape's attribute `empty` has the value `true`.

In the following section, we will describe the graph transformation system that corresponds to this model.

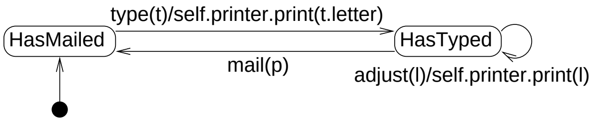
4 System States and Integrated Rules

A system state, as used in [KGKK02], is represented by an object diagram which is extended with object states and event queues. Figure 4 shows a system state graph for our office example.

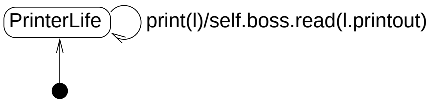
The object diagram part specifies the system's current structure, i.e., the existing objects with their current attribute values and links. Addionally,



(a) Boss statechart diagram



(b) Secretary statechart diagram



(c) Printer statechart diagram

Fig. 2. Statechart diagrams for *Boss*, *Secretary*, and *Printer*

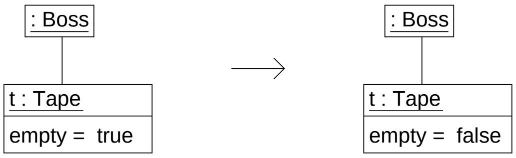


Fig. 3. Pair of object diagrams specifying the effect of `record(t)`

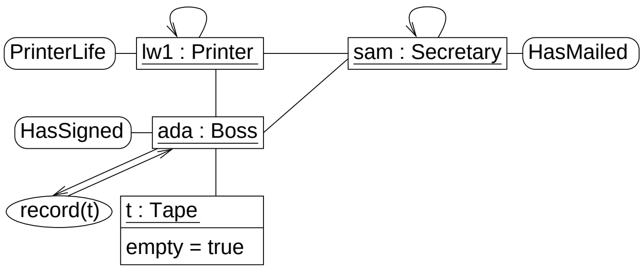


Fig. 4. System state graph

there is a state attached to each object if the class has a statechart diagram. States are depicted by rounded rectangles which are connected to an object by an edge and labeled with the name of a state occurring in the corresponding statechart diagram. Event queues of objects are visualized by ellipses connected by arrows. The object points to the event that is to be dispatched next, this event points to the next event and so on. The last event in the queue points back to the object.

The system state is modified during runtime by operation executions. The formal counterpart of an operation execution is the application of a graph transformation rule to the system state graph. The rules for an operation can be obtained by combining the object diagram pair specifying the effect of the operation with information from the statechart. In Fig. 5, Fig. 6, and Fig. 7, the so-called integrated rules for the operations of *Boss*, *Secretary*, and *Printer* are given.

We explain how to obtain these rules by elaborating on the operation `Boss::record(t)`, the first rule in Fig. 5. As can be seen in Fig. 3, the operation's effect on the structure is the change of `t.empty` from `true` to `false`. The left/right object diagram in Fig. 3 forms the base of the left-hand/right-hand side of the rule. Since the rule shall be applicable only if there is a corresponding call event in the front of the queue of the *Boss* object, the *Boss* object on the left-hand side points to a `record(t)` event.

We now take a look at the statechart in Fig. 2(a). According to this, a *Boss* object only reacts to a `record(t)` event while being in state *HasSigned*, so we attach this state to the *Boss* object on the left-hand side of the rule. The reaction to the event is to call the operation `type(t)` on the boss's secretary. The rule accommodates this by having a *Secretary* object linked to the *Boss* object both on left-hand and right-hand side. On the left-hand side, the last event in the secretary's queue is shown, where X is a placeholder for an arbitrary event (the same holds for Y in the boss's event queue). On the right-hand side, a `type(t)` event is placed behind that event X. The `record(t)` event in the front of the queue of the boss is deleted in the right-hand side, so that Y becomes the event for the Boss next to be dispatched. Finally, since the transition in the Boss statechart leads from state *HasSigned* to state *HasRecorded*, the *Boss* object is connected to the state *HasRecorded* in the right-hand side.

In addition to the rules for user defined operations, there exist rules for creating and deleting objects of any class. However, creation and deletion of objects could be done directly in a rule of a user defined operation as well.

5 Integration of Sequence and Collaboration Diagrams

UML sequence and collaboration diagrams represent sections of a system run. They show objects and a sequence of operations which are successively applied. The sequence diagram in Fig. 8 shows a section of a typical system run of the office example. The collaboration diagram in Fig. 9 contains the same information presented in a slightly different form. The sequence diagram emphasizes *time* aspects by a message ordering from top to bottom, whereas the collaboration diagram emphasizes *space* aspects by eplicitly showing the links

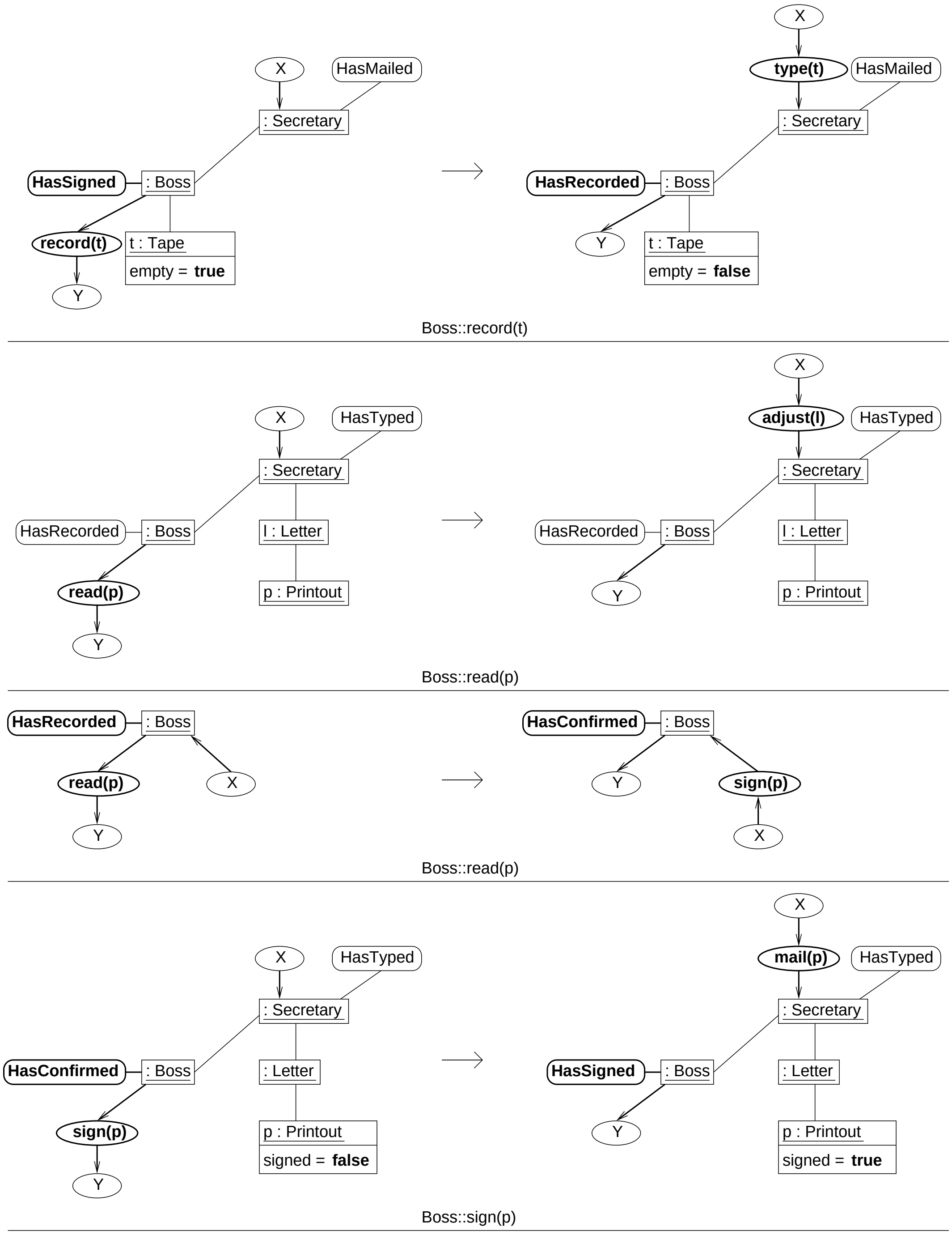


Fig. 5. Integrated Rules for *Boss* operations

between the objects (and expressing the message sequence by a numbering system).

The *User* object is the instance getting the ball rolling. Such an object can be present in sequence and collaboration diagrams without having a corresponding class in the class diagram. Here the user sends a message to the Boss ada to record a tape. Following that, ada sends a message to her Secretary

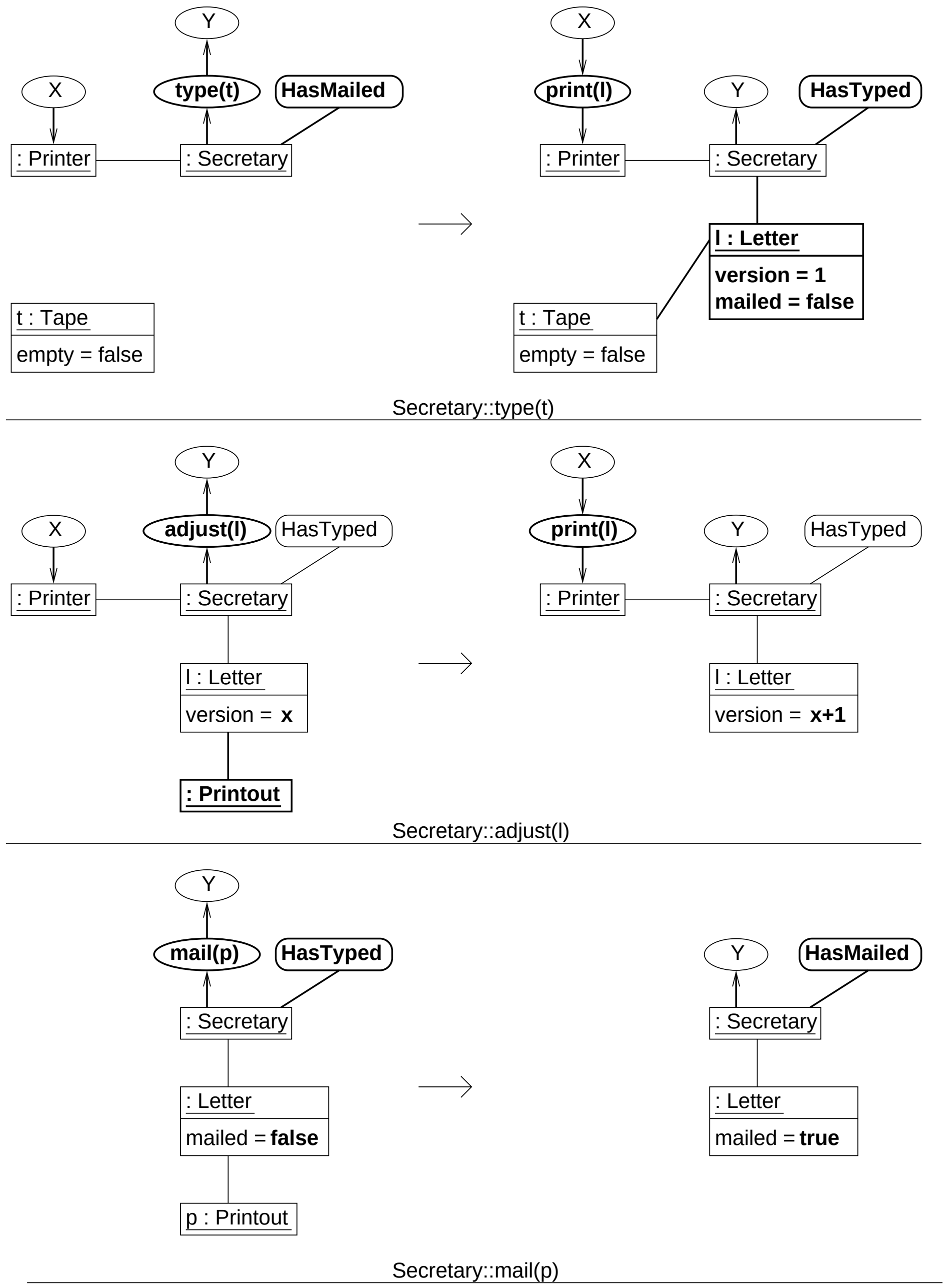


Fig. 6. Integrated rules for *Secretary* operations

sam to type that tape, and so on.

A central fact in our approach is that the sequence of operation calls depicted in the sequence and collaboration diagram (`record(t)`, `type(t)`, `print(t.letter)`, ...) corresponds to a sequence of applications of graph transformation rules.

However, without knowing the integrated diagram to start with, it is not possible to map a message sequence to a specific derivation. Therefore, the points of interest here are (1) whether such a derivation can be found at all and

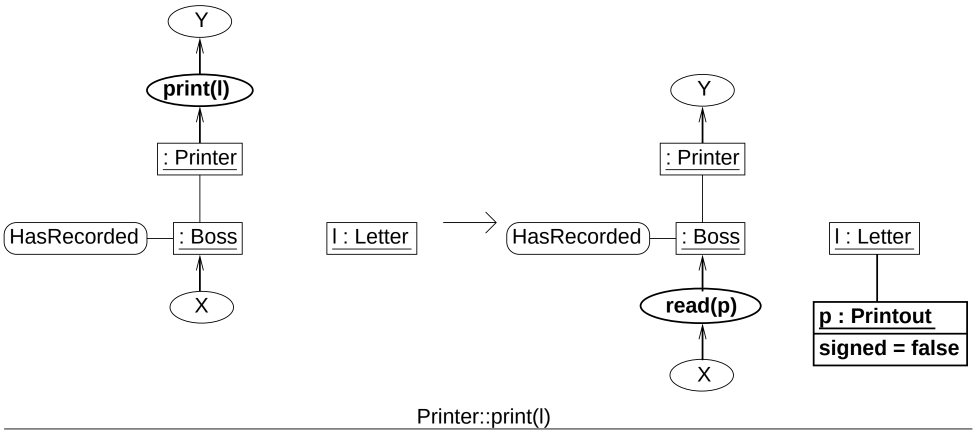


Fig. 7. Integrated rules for *Printer* operations

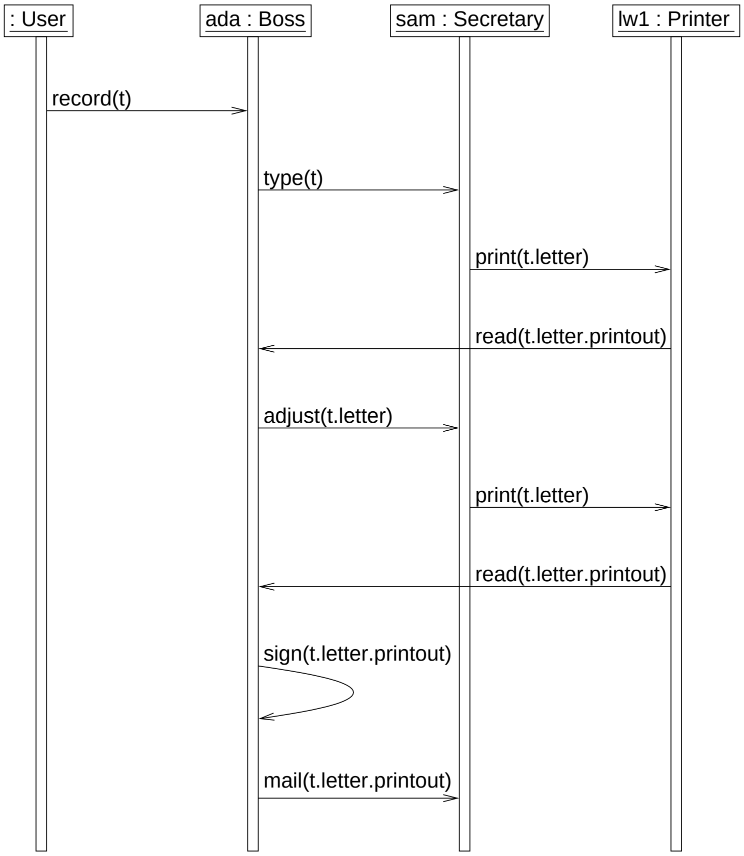


Fig. 8. Sequence diagram

(2) whether the sequence is applicable in a given system state. While the first question can not be answered without testing all valid system states as starting point, the latter is more feasible. The modeler can check by example whether the specified sequence is applicable in states where it should be applicable. And the modeller can test whether the specified sequence is not applicable in states where it should not be. Thus the formalization of UML diagrams by

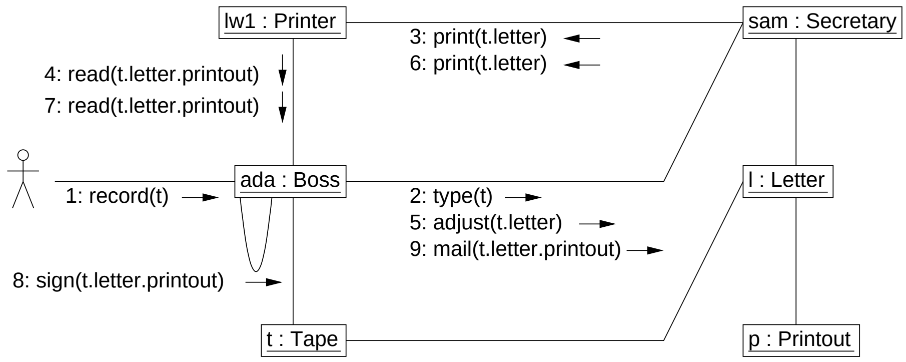


Fig. 9. Collaboration diagram

graph transformation gives feedback to the modeller about the applicabilty of the specified message sequence.

In the system state depicted in Fig. 10, the sequence is not applicable for two reasons. The only event the diagram shows is `record(t)`, so the rule for `record(t)` is the only one that could possibly be applied. However, the attribute `empty` of tape `t` has not the value `empty=true`. But even if it had, the sequence would not be applicable because the secretary is in state *HasTyped* in which she does not react to `type` events. This seems to be reasonably and therefore confirms the correctness of the sequence resp. collaboration diagram.

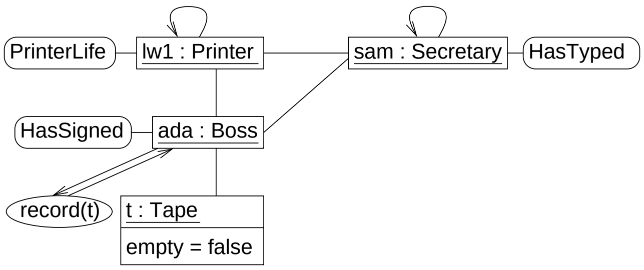


Fig. 10. System state not being a start state for the sequence in Fig. 8 and 9

Figure 11 and 12 show a derivation starting in the system state at the top of Fig. 11. In fact, that state is the minimal state where the message sequence is applicable. In the top state of Fig. 11, the rule for `record(t)` can be applied and all the other rules corresponding to the messages in the sequence resp. collaboration diagram can subsequently be executed as well. If the modeler rates this state sequence as reasonable, this would reinforce the modeler's belief in the correctness of her or his model. Otherwise, either the sequence resp. collaboration diagram or the model (consisting of the class diagram, the statechart diagrams, and the operation semantics given by graph transforrmation rules) has to be changed.

Thus derivations in the graph transformation approach correspond to UML sequence resp. UML collaboration diagrams. These derivations are the formal counterpart of the diagrams and can help to confirm or to decrease the modeler's belief in the model.

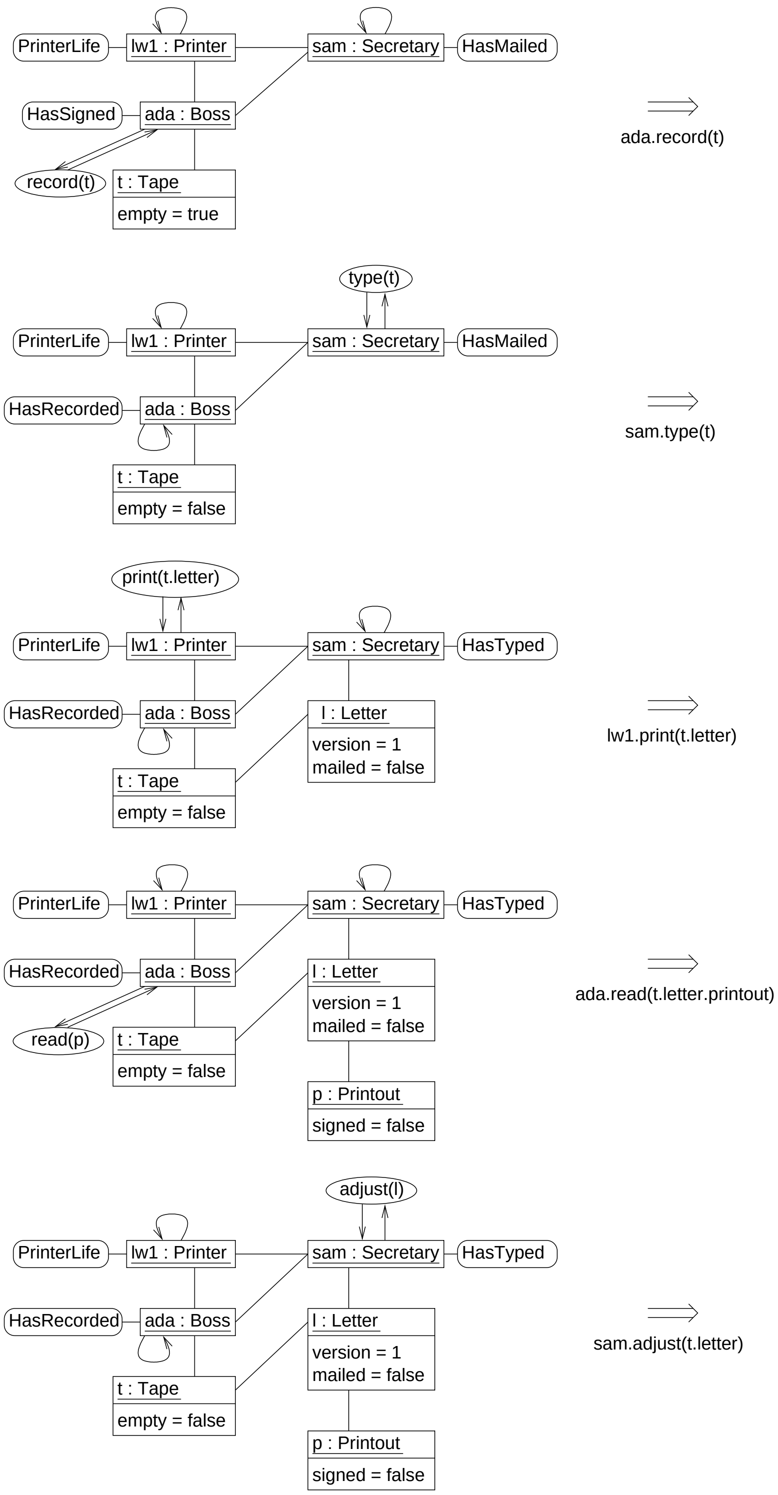


Fig. 11. Derivation (Part 1)

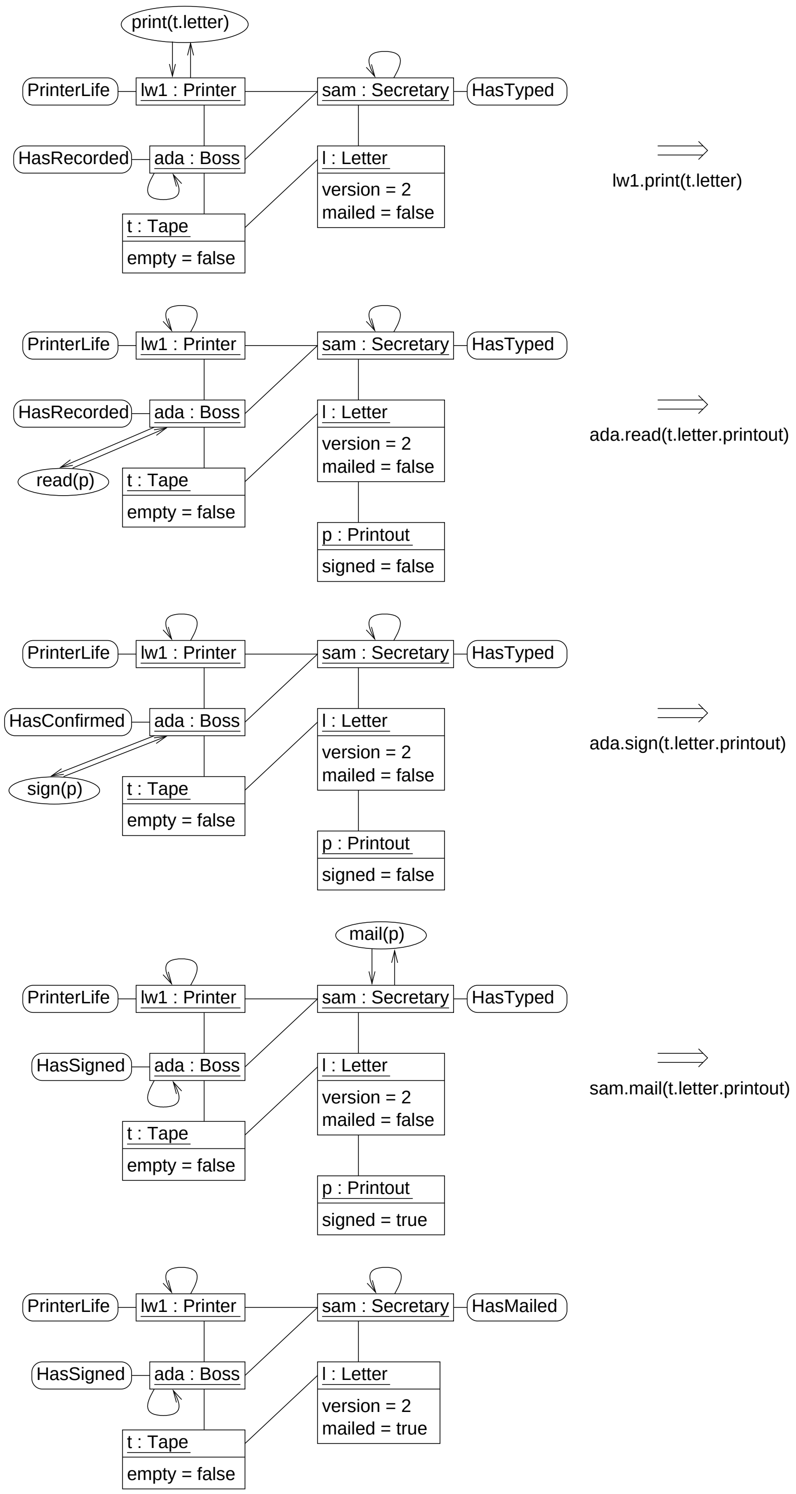


Fig. 12. Derivation (Part 2)

6 Related Work

Recently, many efforts have been made to formalize the semantics of UML. Here we only mention a selection of those contributions that relate to our work in that they use graph transformation.

In [GPP98] statechart diagrams are transformed into graphs to show the intended semantics. [GR99] gives transformation rules to transform class diagrams into simpler ones. [Gog00] continues this approach by transforming several UML features to a UML core. In [Sch99] process models for UML are studied on the basis of graph transformations. [Kus01] describes the execution semantics of statechart diagrams in a formal way. [KGKK02] is the contribution we built upon in this paper. It proposes an integrated semantics for class, object and statechart diagrams. In [TE00] consistency analysis between class and sequence diagrams based on attributed typed graphs and their graph transformation is described. [EHHS00] proposes dynamic meta modeling as an approach to describe the operational semantics of behavioral UML diagrams in terms of graph transformation rules.

7 Summary and Conclusions

In this paper, we have extended the approach of [KGKK02], which gives an integrated semantics for UML class, object, and statechart diagrams based on graph transformation. We have demonstrated that approach by modeling an example office scenario with UML class and statechart diagrams and giving a corresponding graph transformation system. Then we have shown how sequence and collaboration diagrams can be integrated into that approach. These diagrams specify sequences of operation applications and therefore determine the order of transformation rule applications. Together with a system state to start with, a sequence resp. collaboration diagram conforms to a specific derivation in the graph transformation system. The modeler can check whether a given system state is a possible starting point for an operation sequence in order to verify the correctness of his model.

Altogether, we now have an impression how to give an operational formal semantics for a central part of UML. However, in further research, this approach has to be elaborated by giving a formal translation from UML models to graph transformation systems. When doing so, several questions will probably arise:

(i) How do we cope with under-specification? It is desirable that a UML model can be translated into a graph transformation system even if important information is missing, such as semantics of operations in classes.

(ii) So far, the approach requires the semantics of operations given as graph transformation rules with object diagrams as graphs. Is it possible to get along with OCL pre- and post conditions instead?

(iii) The example in this paper (and the one in [KGKK02]) uses only basic features of UML. In particular, in statecharts we only use asynchronous call events and call actions. To which extent can UML be formalized in this approach and where are the limits? For example, what about composite states and different kinds of events in statecharts and features like conditional branches in sequence diagrams?

However, we are confident that the graph transformation approach meets many demands for a precise UML semantics and that further research is therefore a worthwhile task.

References

[Boo94] Grady Booch. *Object-Oriented Analysis and Design with Applications.* Benjamin/Cummings, 1994.

[BRJ98] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1998.

[EEKR99] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools.* World Scientific, Singapore, 1999.

[EHHS00] Gregor Engels, Jan Hendrik Hausmann, Reiko Heckel, and Stefan Sauer. Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In Andy Evans, Stuart Kent, and Bran Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings*, volume 1939 of *LNCS*, pages 323–337. Springer, 2000.

[EKMR99] Hartmut Ehrig, Hans-Jörg Kreowski, Ugo Montanari, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools.* World Scientific, Singapore, 1999.

[FS97] Martin Fowler and Kendall Scott. *UML Distilled: Applying the Standard Object Modeling Language.* Addison-Wesley, 1997.

[Gog00] Martin Gogolla. Graph Transformations on the UML Metamodel. In Jose D.P. Rolim, Andrei Z. Broder, Andrea Corradini, Roberto Gorrieri, Reiko Heckel, Juraj Hromkovic, Ugo Vaccaro, and Joe B. Wells, editors, *Proc. ICALP Workshop Graph Transformations and Visual Modeling Techniques (GVMT'2000)*, pages 359–371. Carleton Scientific, Waterloo, Ontario, Canada, 2000.

[GPP98] Martin Gogolla and Francesco Parisi-Presicce. State Diagrams in UML - A Formal Semantics using Graph Transformation. In Manfred Broy, Derek Coleman, Tom Maibaum, and Bernhard Rumpe, editors, *Proc. ICSE'98 Workshop on Precise Semantics of Modeling Techniques (PSMT'98)*, pages 55–72. Technical University of Munich, Technical Report TUM-I9803, 1998.

[GR99] Martin Gogolla and Mark Richters. Transformation rules for UML class diagrams. In Jean Bézivin and Pierre-Alain Muller, editors, *The Unified Modeling Language, UML'98 - Beyond the Notation. First International Workshop, Mulhouse, France, June 1998, Selected Papers*, volume 1618 of *LNCS*, pages 92–106. Springer, 1999.

[JCJO92] Ivar Jacobsen, Magnus Christerson, Patrik Jonsson, and Gunnar Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, 1992.

[KGKK02] Sabine Kuske, Martin Gogolla, Ralf Kollmann, and Hans-Jörg Kreowski. An Integrated Semantics for UML Class, Object, and State Diagrams based on Graph Transformation. In Michael Butler and Kaisa Sere, editors, *3rd Int. Conf. Integrated Formal Methods (IFM'02)*. Springer, Berlin, LNCS, 2002.

[Kus01] Sabine Kuske. A formal semantics of UML state machines based on structured graph transformation. In Martin Gogolla and Cris Kobryn, editors, *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference, Toronto, Canada, October 2001, Proceedings*, volume 2185 of *LNCS*, pages 241–256. Springer, 2001.

[OMG01] OMG, editor. *OMG Unified Modeling Language Specification, Version 1.4, September 2001.* Object Management Group, Inc., Framingham, Mass., Internet: `http://www.omg.org`, 2001.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs (NJ), 1991.

[RJB98] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1998.

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations.* World Scientific, Singapore, 1997.

[Sch99] Ansgar Schleicher. Formalizing UML-based process models using graph transformations. In *AGTIVE*, volume 1779 of *LNCS*, pages 341–357. Springer, 1999.

[TE00] A. Tsiolakis and H. Ehrig. Consistency analysis of UML class and sequence diagrams using attributed graph grammars. In H. Ehrig and G. Taentzer, editors, *Proc. of Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems, Berlin, March 2000*, 2000. Technical Report no. 2000/2, Technical University of Berlin.