In: Proc. ICALP Workshop Graph Transformations and Visual Modeling Techniques,
Andrea Corradini and Reiko Heckel, Eds., Carleton Scientific, Waterloo, Canada, 2000.

Graph Transformations on the UML Metamodel

Martin Gogolla
University of Bremen
Computer Science Department

Abstract

The Unified Modeling Language UML is a rich language so that it seems not feasible to give a single semantics for the complete language in one step. In order to define the semantics, we propose to translate as many UML language features as possible into UML itself. The part of UML which cannot be treated in this way is called the UML core. The semantics of the non-core language features is given by a translation into the core, whereas a different semantics (for example a set-theoretic one or a semantics on the basis of graphs) must be given to the UML core. It turns out that for the first step, namely the translation of UML language features into the UML core, graph transformations are an ideal language for expressing the required manipulations and the UML metamodel gives the necessary fundamental vocabulary and notions.

1 Introduction

The Unified Modeling Language UML [OMG99, RJB98] has been widely accepted as a standard for modeling and documenting software systems. UML comprises a number of diagram forms used to describe particular aspects of software artifacts. The diagram forms can be divided depending on whether they are intended to describe structural or behavioral aspects. From a fundamentals point of view, a central ingredient of UML is the Object Constraint Language OCL [OMG99, WK98, WK99], a language which is close to first order predicate calculus, but which claims to be easy to be applied by the "average business or system modeler".

The area of graph grammars and graph transformation [Roz97, EEKR99] is a well-established field especially in theoretical computer science. Today it seems to be mature so that it can be applied in practical computer science. Because UML is a diagrammatic language, it seems reasonable and promising to apply techniques developed in the graph grammar and graph transformation field to UML as this has been done already [SW98, EHSW99]. This paper therefore describes equivalence transformations on UML diagrams by means of graph transformations.

The structure of the rest of this paper is as follows. In Section 2 we give a brief summary of the UML metamodel and illustrate its basic ideas by means of an example. Section 3 explains the fundamentals of graph transformations on the UML metamodel by means of two examples - one taken from class diagrams and the other from statechart diagrams - and gives a formal representation in terms of double pushout rules. Section 4 explains further application areas within UML for graph transformations like the ones from Section 3. Section 5 states some concluding remarks.

2 The UML Metamodel

The basic idea of the UML metamodel is to give a systematic and rigorous description of the syntax and the semantics of the UML. The UML metamodel uses a subset of the UML, namely class diagrams, in order to do so.

2.1 Brief Summary

The UML metamodel explains the various UML modeling concepts, roughly speaking, diagram by diagram, but certain common features of different diagrams are factored out and are described in advance. Each description is structured in four sections: (1) Class diagrams, (2) explanations for the class diagram features, (3) well-formedness rules formulated in OCL, and (4) informal description of the semantics of the features.

1. Class diagrams, where classes and relationships are displayed, specify an abstract syntax of the various UML modeling concepts. This syntax description with class diagrams plays the same role a classical context-free grammar plays for a classical programming language.

   Example: Figure 1 shows an excerpt from the UML Metamodel Core Package. The diagram is already a merge of two class diagrams (Figure 2-5 and Figure 2-6) from the UML documentation and only shows classes, attributes and relationships needed for the example to follow. Rectangles represent classes, connecting lines symbolize associations, and arrows with open triangles show generalization hierarchies. This class diagram says, for example, that an Attribute is a StructuralFeature, which itself is a Feature being a ModelElement, and an Association is related to at least two AssociationEnds (role name connection with multiplicity 2..*) which in turn are related to a single Classifier (role name type with multiplicity 1).

2. For the specified class diagrams also verbal descriptions of classes, attributes, and associations are given.

   Example: The class Association in Fig. 1 is described in the UML documentation [OMG99, p. 2-19] by the following text.

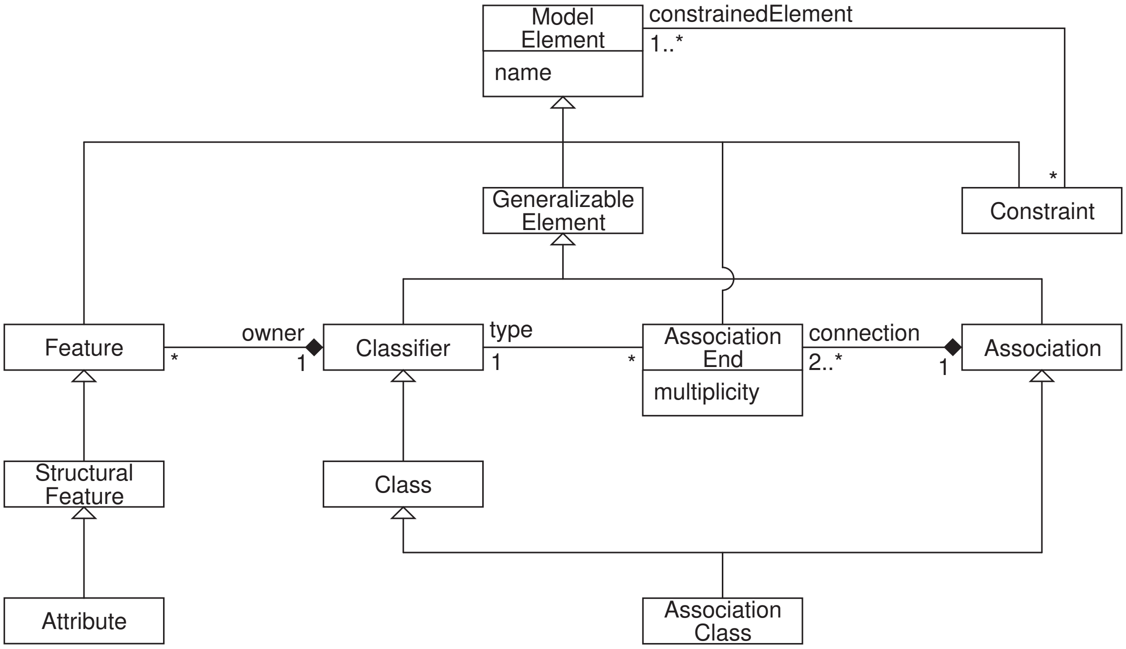


Figure 1: Excerpt from the UML metamodel core package

> *An association defines a semantic relationship between classifiers. The instances of an association are a set of tuples relating instances of the classifiers. Each tuple value may appear at most once.*

Example: Apart from the classes itself, the attributes and associations of the classes are explained as well. For instance, for the class Association, the attribute name (inherited from ModelElement) and the association end with the role name connection is described in [OMG99, p. 2-20] as follows.

> *name: The name of the Association which, in combination with its associated Classifiers, must be unique within the enclosing namespace (usually a Package).*
>
> *connection: An Association consists of at least two AssociationEnds, each of which represents a connection of the association to a Classifier. Each AssociationEnd specifies a set of properties that must be fulfilled for the relationship to be valid. The bulk of the structure of an Association is defined by its AssociationEnds.*

3. Well-formedness rules given as OCL constraints then restrict the syntactical possibilities specified by the class diagrams. These constraints roughly correspond to context-sensitive conditions in classical language descriptions.

   Example: A condition like "The AssociationEnds must have a unique name within the Association" is formulated in [OMG99, p. 2-44] with OCL as follows (the constraint is formulated within the context of the class Association, and therefore the term self refers to an arbitrary object of the class Association).

```
self.connection->forAll(ae1, ae2 |
  ae1.name=ae2.name implies ae1=ae2)
```

This condition represents a requirement which cannot be described by a context-free grammar and which is classically formulated as a context-sensitive condition (like the requirement that variable names within one block are unique).

4. Finally, verbal descriptions of the semantics of the diagram features are presented.

   Example: The concept of an association and its semantics is explained in the UML metamodel [OMG99, p. 2-57] as follows (only an initial segment of the description is shown).

   > *An association declares a connection (link) between instances of the associated classifiers (e.g., classes). It consists of at least two association ends, each specifying a connected classifier and a set of properties which must be fulfilled for the relationship to be valid. The multiplicity property of an association end specifies how many instances of the classifier at a given end (the one bearing the multiplicity value) may be associated with a single instance of the classifier at the other end. A multiplicity is a range of nonnegative integers.*

2.2 Example

Let us illustrate the ideas presented above with an example. In Fig. 2 an example for an application class diagram is given. By means of the UML metamodel it is now possible to present this class diagram as an instantiation of the UML metamodel class diagram in Fig. 1. This results in an object diagram shown in Fig. 3. Each concept of the application class diagram in Fig. 2 becomes a formal object in the object diagram. For example, the Job association class becomes an object of class AssociationClass, and the Manages association on Job becomes an object of class Association connected to the AssociationClass Job by the two AssociationEnd objects worker and boss.

As we have seen in Fig. 3, the UML metamodel gives means to represent a UML diagram formally as an object diagram belonging to the UML metamodel. What is important for the following considerations, is that such an object diagram can also be simply regarded as a graph. Doing this, the world of graph transformations opens the possibility to manipulate such graphs, i.e., to manipulate UML diagrams.

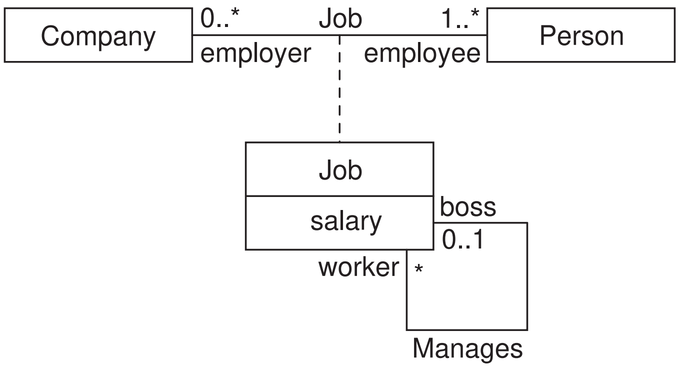


Figure 2: Example for a UML class diagram

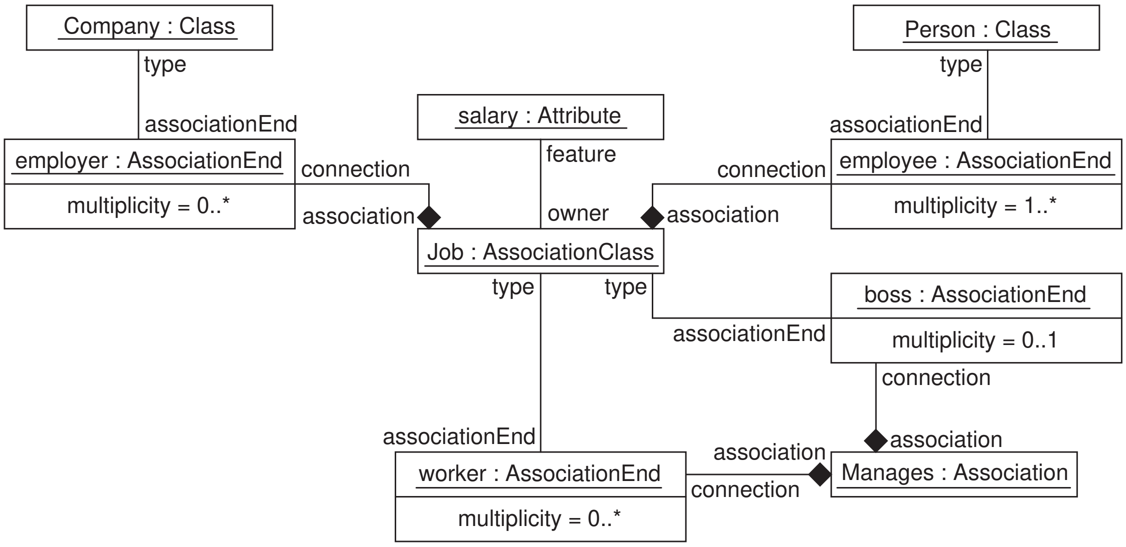


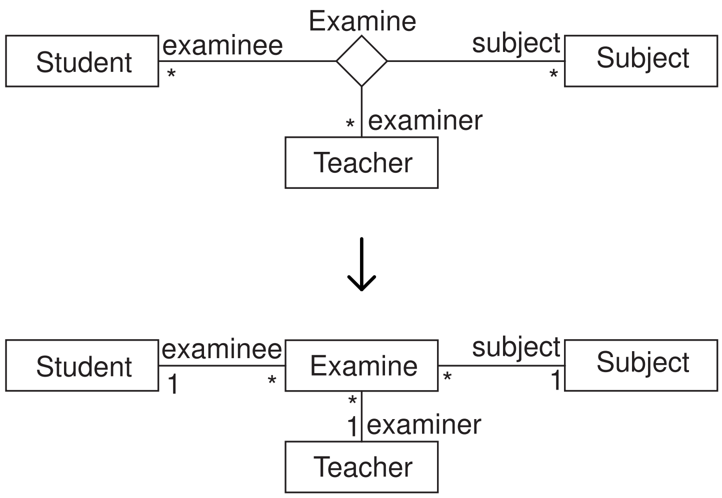
Figure 3: Representation of Fig. 2 as a UML metamodel object diagram

3 Fundamentals

This section explains the basic idea of our approach by means of two examples. The examples formulate equivalence rules for UML diagrams. UML is such a voluminous language that it seems not feasible (and not necessary) to give a single semantics for the complete language in one step [AEF+99]. Instead we propose to translate as many language features as possible into UML itself. We call the remaining part of UML the UML core. Thus the semantics of the non-core language features is given by a translation into the core, whereas a different semantics (for example a set-theoretic one or also a semantics on the basis of graphs) must be given to the UML core. It turns out that for the first step, namely the translation into the UML core, graph transformations seems to be an ideal language for expressing the required manipulations.

3.1 Informal Description

We discuss two examples here: one taken from class diagrams and the other from statechart diagrams. Both examples show how simple equivalence rules for UML diagrams can be formulated in terms of graph manipulation.




```
Examine.allInstances->forAll(e1,e2 |
  e1<>e2 implies (e1.examinee<>e2.examinee or
                  e1.examiner<>e2.examiner or
                  e1.subject<>e2.subject))
```

Figure 4: Transforming a ternary association into binary associations (example)

The rule stated in Fig. 4 takes a class diagram with a ternary association without restricting multiplicities and reformulates this class diagram equivalently by replacing the ternary association by a class and by replacing each branch of the ternary association by a single binary association with the indicated multiplicities. In addition, an OCL constraint must be given in order to assure that two different

Examine objects can be distinguished by their related Student, Teacher, or Subject objects.

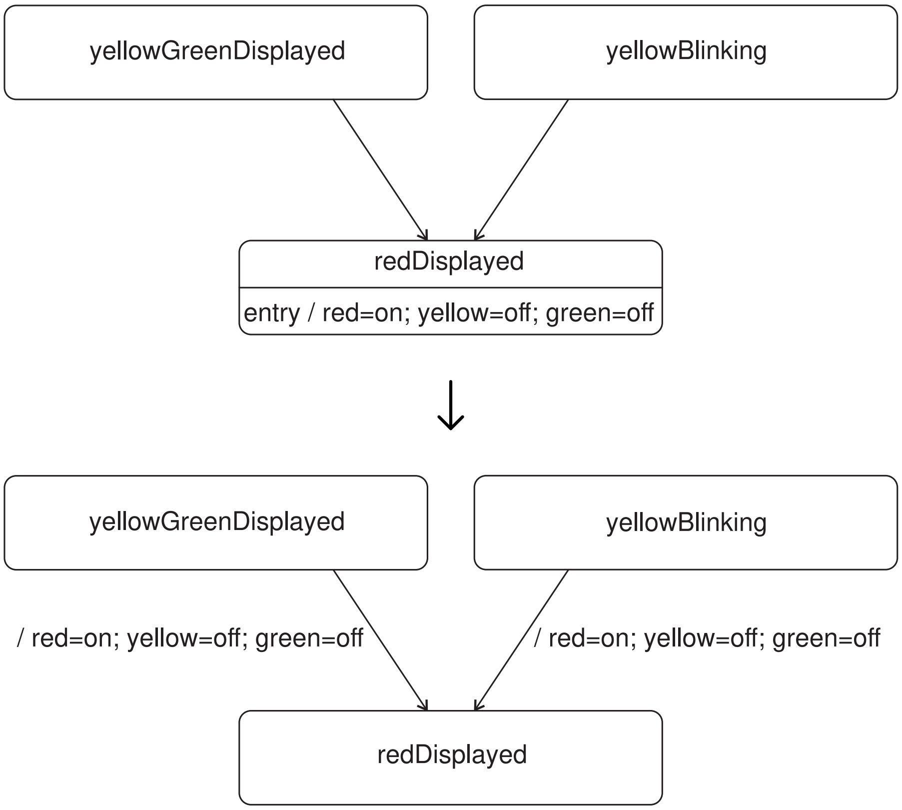


Figure 5: Transforming an entry action to incoming transitions (example)

The other example in Fig. 5 shows an excerpt from a statechart diagram for a traffic light system (it is an Italian traffic light because there is a transition from yellowGreenDisplayed to redDisplayed which different, for example, to Germany). Only two transitions to state redDisplayed are shown. As indicated by the keyword entry and the following action within state redDisplayed every time the state redDisplayed is entered the corresponding entry action is performed. In UML, actions can also be specified on transitions and therefore this situation can be expressed equivalently by the lower part of Fig. 5 where the actions are placed on the each incoming transition of state redDisplayed. Actions on incoming transitions (if such actions exist) must be executed before the inserted entry action.

As indicated in Fig. 6, this equivalence rule could also be formulated in a different way: Instead of changing the label of the transitions one could replace each transition by a new state and two transitions with the first transition having the former label and the second transition having a label with the action associated with the former entry action (in general, transitions in UML are labeled in the form e [g] / a with an event e, a guard g, and an action a). The language of graph transformations provides the propper vocabulary to formulate such a rule and opens a systematic way for a discussion about equivalence.

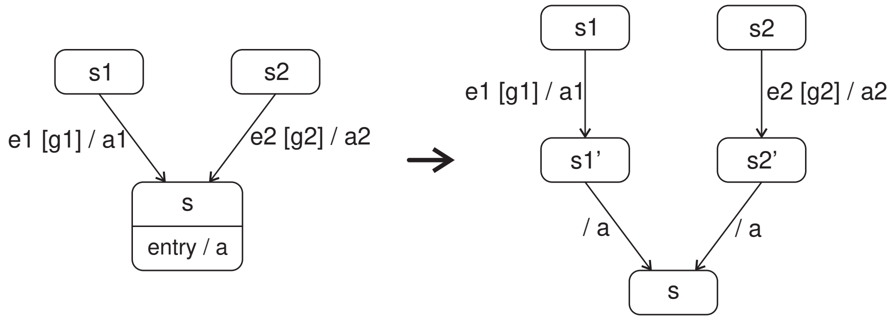


Figure 6: Alternative for transforming an entry action

3.2 Formal Representation

What we have done above in examples can be generalized and formulated as graph transformation rules on the UML metamodel. Figure 7 rephrases the example from Fig. 4 as a double pushout rule [Roz97] with Left side, Context part, and Right side. The Left side of the rule is based on Fig. 2-6 of the UML Semantics document and requires the existence of three classes A, B, and C, one ternary association R (as an application pre-condition of the complete rule, the OCL formula `R.connection->size=3` must be valid), and AssociationEnd objects with unconstrained multiplicity. The Right side transforms the Association object R into a Class object, introduces three new associations, connects them to the new class R, and places the required constraint on the class R.

The existence of the UML metamodel is a pre-requisite for the formulation of such rules because the UML metamodel delivers the required vocabulary. In the case of this rule, we took advantage of the concepts Class, Association, AssociationEnd, and Constraint. Such rules also rely on the fact that portions of the UML metamodel can be formulated with UML diagrams characterizing the situations one needs to describe.

One last remark concerning the types of the object R on the Left and Right side is needed: We have taken the freedom to change the type (resp. class) of the object R from Association to Class. We assume that the identity of the object is the same on the Left and Right side and that the information about its type (resp. class) is recorded with an extra edge and an extra node (for the type, resp. the class). In order to make the presented graph more convenient we have shown the type (resp. class) information together with the object's identity in a single node.

Analogously to the above line of thought, we have shown in Fig. 8 the general schema for transforming an entry action into incoming transitions of a state. The Left side is based on Fig. 2-21 of the UML Semantics document (the relevant part of Fig. 2-21 is given in Fig. 9) and displays the state with its entry action and an incoming transition with an effect action. The Right side associates an ActionSequence (abbreviated as ActionSeq in the rule) with the Transition object.

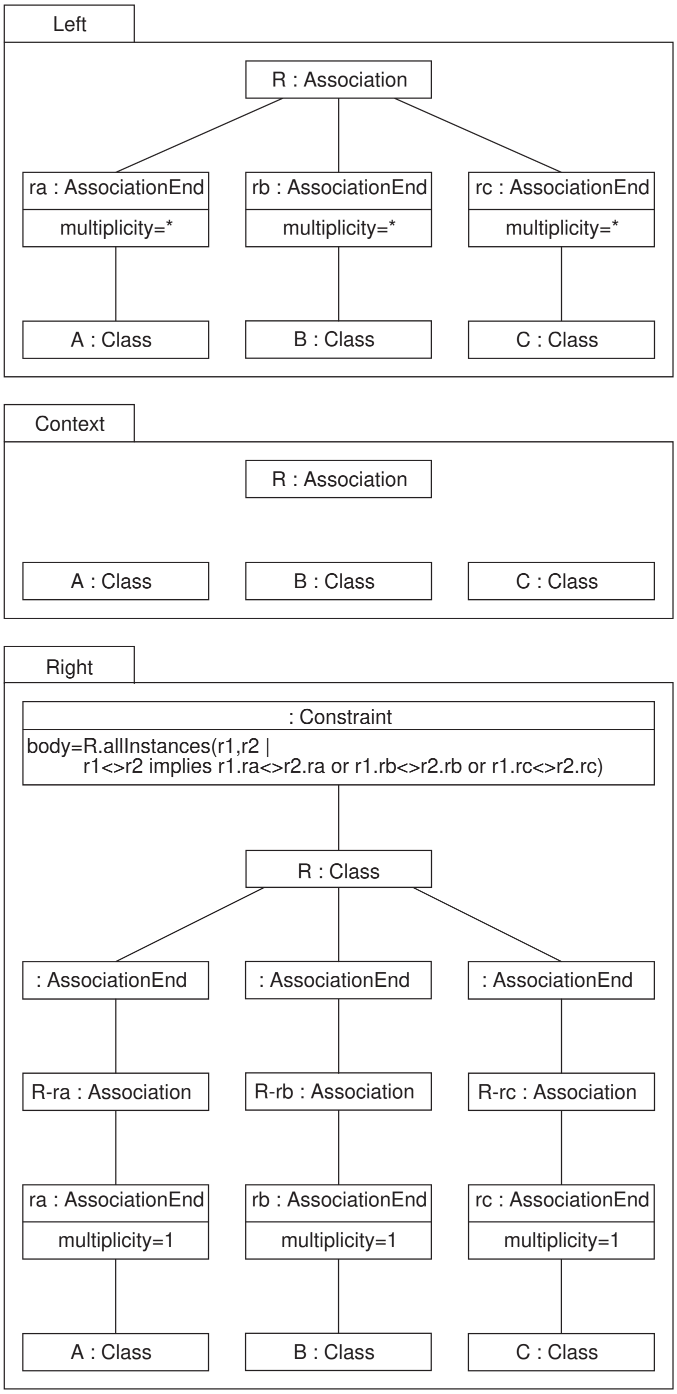


Figure 7: Transforming a ternary association into binary associations

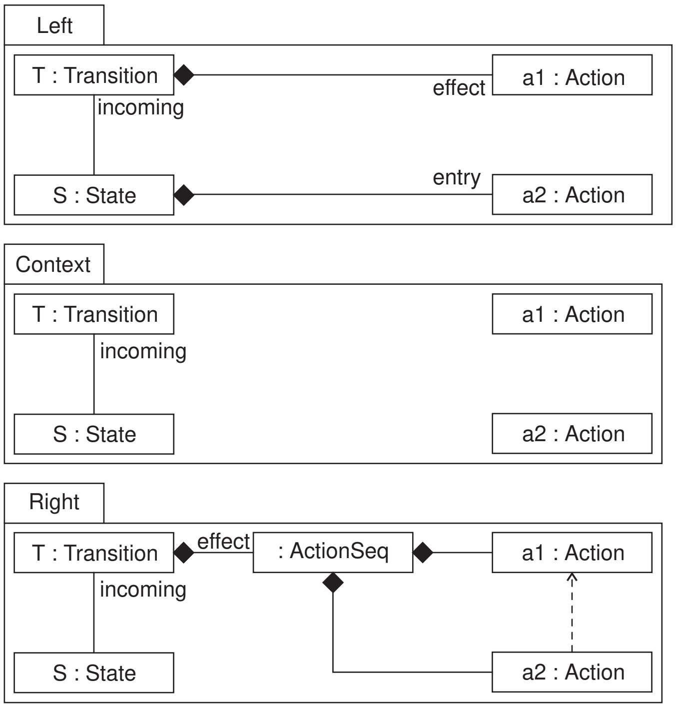


Figure 8: Transforming an entry action to incoming transitions

This ActionSequence consists of two actions, namely the former only effect action a1 followed by the former entry action a2 (concatenation of actions is denoted here by a dependency indicated by the dotted arrow). The rule as it now stands works for states with one incoming transition and must be modified appropriately for other cases.

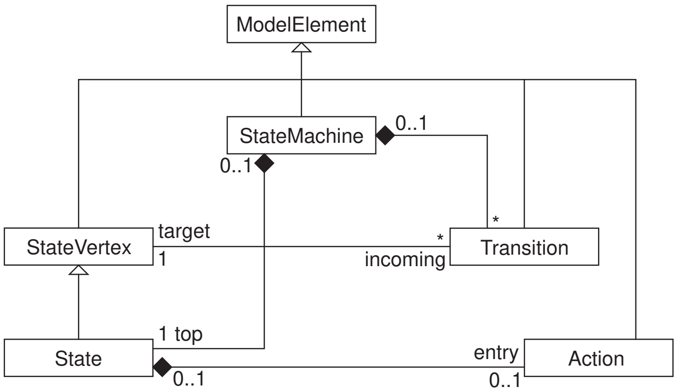


Figure 9: Excerpt from the UML metamodel state machines package

4 Further Application Areas within UML

Above we have given only two typical examples demonstrating the principle idea of the approach. Equivalence transformations formulated in the language of graphs can be employed for all UML diagram forms. We now shortly consider the two diagram forms which seem to be most important in the design phase: Class diagrams and statechart diagrams.

Class diagrams: In [GR99] we have already discussed many transformations for class diagrams on a conceptual level without explicitly formulating the corresponding rules for the UML metamodel. Multiplicities on binary and n-ary associations, aggregation and composition, n-ary association, qualifier, and (to a large extent) generalization can be reformulated by using binary associations and OCL conditions. The results indicate that with respect to class diagrams the UML core mainly consists of classes, binary associations, and OCL constraints.

Statechart diagrams: In [LB99] it is pointed out that entry and exit actions in states and sequential substates and parallel substates can equivalently be transformed into flat statechart diagrams. [GPP98] already transforms nested statechart diagrams and stub states into simple statechart diagrams on the basis of graph transformations. These results indicate that with respect to statechart diagrams the UML core mainly consists of flat statechart diagrams with simple states (no internal transitions are necessary) but with complex (event-guard-action) transitions. However, a close connection between statechart diagrams with its various forms of events (call, signal, change, and time events) and class diagrams is still missing.

So in both fields, namely class and statechart diagrams, results for transformations rules have already been presented. All these results have to be adjusted to the UML metamodel layer and have to be formulated as graph transformation rules.

5 Conclusion

We have studied transformation rules on the UML metamodel layer. The aim of these rules is to identify a minimal set of UML concepts, a UML core, necessary to express all UML language features. Due to the diagrammatical nature of UML, graph transformation seems to be a natural language for the formulation of such rules. An important ingredient of UML with respect to the UML core is the OCL which is used on various levels, in particular for (1) the syntax of UML, (2) the semantics of UML (translation to the UML core) and (3) arbitrary model specification. Questions concerning the semantics of UML and the UML core are subject to various research activities. For example, OCL is the focus of the UML Specification Environment USE developed at University of Bremen, and the UML-AID project at University of Bremen aims at applying graph transformations techniques for the semantics of UML in general.

Acknowledgements

Thanks to Ralf Kollmann, Hans-Jörg Kreowski, Sabine Kuske, Oliver Radfelder, and Mark Richters for discussions on the UML metamodel and transfomation rules. Mark Richters has also commented a draft version of this paper.

References

[AEF+99] Egidio Astesiano, Andy Evans, Robert France, Guy Geniloud, Martin Gogolla, Brian Henderson-Sellers, John Howse, Heinrich Hussmann, Shusaku Iida, Stuart Kent, Alain Le Guennec, Tom Mens, Richard Mitchell, Oliver Radfelder, Gianna Reggio, Mark Richters, Bernhard Rumpe, Perdita Stevens, Klaas van den Berg, Pim van den Broek, and Roel Wieringa. UML Semantics FAQ. In Ana Moreira and Serge Demeyer, editors, *ECOOP'99 Workshop Reader*, pages 33–56. Springer, Berlin, LNCS 1743, 1999.

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

[EHSW99] Gregor Engels, Roland Hücking, Stefan Sauer, and Annika Wagner. UML collaboration diagrams and their transformation to Java. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723 of *LNCS*. Springer, 1999.

[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 Bezivin and Pierre-Alain Muller, editors, *Proc. 1st Int. Workshop Unified Modeling Language (UML'98)*, pages 92–106. Springer, Berlin, LNCS 1618, 1999.

[LB99] Kevin Lano and Juan Bicarregui. Issues in the Semantics of UML. In S.J.H. Kent, A. Evans, and B. Rumpe, editors, *Proc. ECOOP'99 Workshop UML Semantics FAQ*. University of Brighton, 1999.

[OMG99] OMG, editor. *OMG Unified Modeling Language Specification (Version 1.3).* OMG, 1999. http://www.omg.org.

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

[Roz97] Gregorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation - Volume 1: Foundations.* World Scientific, 1997.

[SW98] Andy Schürr and Andreas Winter. Formal definition of UML's package concept. In Martin Schader and Axel Korthaus, editors, *The Unified Modeling Language – Technical Aspects and Applications*, pages 144–159. Physica-Verlag, Heidelberg, 1998.

[WK98] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Precise Modeling with UML.* Addison-Wesley, 1998.

[WK99] Jos Warmer and Anneke Kleppe. OCL: The constraint language of the UML. *Journal of Object-Oriented Programming*, may 1999.