Adding Visual Rules to Object-Oriented Modeling Techniques*

Gabriele Taentzer
TU Berlin, Germany, gabi@cs.tu-berlin.de

Abstract

The modeling of behavior in object-oriented systems usually relies on control flow diagrams and (extended) state automatas. These basic techniques do not well support the design in a declarative style. This style is useful for modeling integrity constraints, especially active constraints, on the data model, to perform event handling, to derived new attribute values and associations and to model strategies in business and engineering. In this contribution, rules are added as a declarative modeling technique for behavior. Roughly speaking, a rule consists of a condition and an action. If the condition is true, the action is performed. The concept of rules is integrated into an existing object-oriented modeling technique by formulating rules on object diagrams. A rule application may test and change a certain object structure. The application may be triggered automatically by a relevant change of this object structure, periodically or by an event occuring in some behavior diagram. This rule concept has an underlying formal semantics on the basis of graph transformation which may be exploited for analyzing a designed rule set concerning e.g. conflicts, dependencies, etc. The graphical notation of the rules follows the UML-notations as far as possible.

1 Introduction

The analysis and design of object-oriented systems highly relies on diagrammatic notations. (See e.g. OMT [RBP+91], OOD [Boo94], UML [UML97].) The methods and languages which are currently available nicely support the development of static system structures, like class and object structures, in class and object diagrams. For the behavior description different kinds of behavior diagrams are considered based on control flow diagrams and (extended) state automatas. These techniques support a procedural design. A technique which supports a more declarative style that states what should be done, and not how and when, is still missing.

Managing e.g. the maintenance of constraints concerning a certain data model, any dynamics on this model causes actions to ensure the satisfaction of the constraints over time. Each time the model is changed, all the constraints concerned with this change have to be checked again. Detected inconsistencies may be solved, i. e. the constraints may be active. Modeling this procedure by control flow diagrams and state machines would lead to a significant amount of additional control. The result are large diagrams difficult to understand and communicate. Omitting this part of control would lead to less clarity how

*Research partially supported by the German Research Council (DFG), the TMR network GETGRATS, and the ESPRIT Basic Research Working Group APPLIGRAPH and when constraints are proven. A similar situation occurs when a certain event can be triggered in parallel to the proper control flow. To be sure that the event is handled as early as possible lots of conditions and state handling is necessary.

These problems can be met by adding visual rules to the set of basic techniques for behavior description. Rules allow a more declarative design style stating what has to be done in a certain situation. Using rules, the developer is much less concerned with control flow, because the trigger concept for rules is predefined. Roughly speaking, a rule consists of a condition and an action. If the condition is true, the action is performed. In the following, rules are formulated on object diagrams. They may be used to model active constraints and event handling, to monitor certain model structures for events, to derived new attribute values and associations, to model strategies in business and engineering, etc.

The application of a rule may create or delete objects and links as well as compute new attribute values, thus, it may cause further rule applications. The applicability of a rule is tested when either the object structure was modified in a way relevant for the rule, some time event occurs that causes the rule application or a trigger for rule application may be added to some behavior diagram.

The application of object diagram rules has a formal semantics based on graph transformation. Object diagrams are graphs on objects and links. Graph rules are used to transform graphs. There are several approaches to graph transformation differing in the rule syntax and semantics as well as their formalization [Roz97], [Roz99]. In this contribution, we follow the algebraic approach to graph transformation. This approach is general enough to formulate object diagram rules in syntax and semantics and supports a rich set of rewriting properties. (See [CMR+97] for an introduction and overview on results.) The graphical notation of the rules follows the UML-notations as far as possible.

2 Object Diagram Rules

Object Diagrams In the object-oriented design the static structure of a system is modeled by class and object diagrams. A class diagram shows the system structure in principle, i.e. without any temporal information. An object diagram, however, shows instances compatible with a particular class diagram. A static object diagram presents a snapshot of the detailed state of a system at a point of time. A dynamic object diagram additionally contains messages and further annotations that allow to model a detailed state over a period of time, including the changes over time. Collaboration diagrams in UML are dynamic object diagrams. For the rule definition, we only need static object diagrams, because the temporal aspect is covered by the concept of rules. Therefore, we will omit the adjective "static" further on. An object diagram mainly consists of objects and links. In the following figures we use the UML-notation for objects and links and extend it by rule-specific issues. Note that unlike the UML-notation attribute values that are strings are written in quotation marks.

Rules To express constraints and certain kinds of dynamic behavior, rules on object diagrams are introduced. As usual, such a rule consists of a left and a right-hand side which are both object diagrams, here. Moreover, these object diagrams are interrelated. A rule is usually understand in a way that the left-hand side tests some inconsistency. The right-hand side may just contain some error message which may state some repair action,

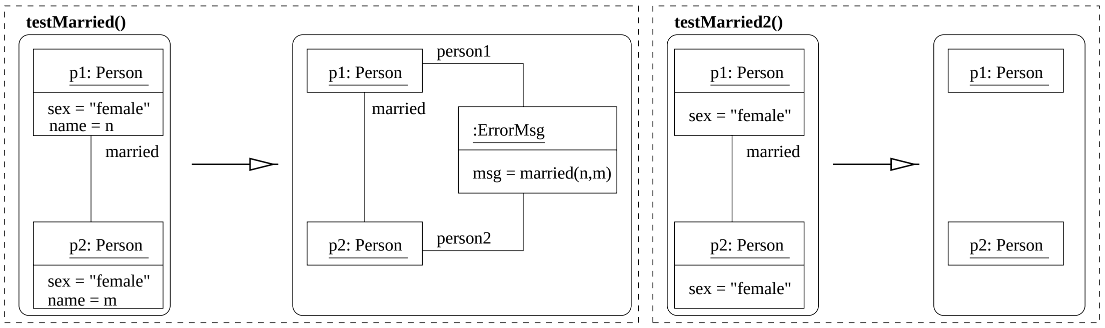

Figure 1. Sample rules formulating constraints

i.e. an active constraint is formulated. Furthermore, rules may be used to express event handling and business strategies. The left-hand side tests if a certain event or situation occurred and the right-hand side contains the corresponding reaction.

Interrelation between the object diagrams of a rule The interrelation between the left and the right-hand side of a rule is given by an intersection of the left and the right-hand side. This intersection is depicted by a partial mapping from the left to the right-hand side. This mapping is given by the same objects and links on both sides, i.e. objects with the same names are mapped to each other. For the application of a rule this means that objects and links occuring on the left and on the right are preserved by the rule application. All objects and links occuring only in the left-hand side are deleted and all objects and links occuring only in the right-hand side are newly created by the rule. See Figure 1 for sample rules. The left rule shows an inconsistency test checking if two women are married and prints out an error message. Please note that the objects “p1” and “p2” as well as the link between them are preserved, because they occur on both sides. Attributes which are just tested but not changed only occur on the left-hand side. The right rule shows an active constraint as a rule with the same test, but the reaction is different. It deletes the “married”-link to solve this inconsistent situation.

In Figure 2, further examples for rules are shown where a derived association and a derived attribute are computed.

Variables as attribute values The reader may have noticed that the left rule in Figure 1 contains two variables “n” and “m” as values of the attribute “name” of person “p1” and “p2”. Variables are depicted by strings without quotation marks. The scope of such a variable is the whole rule. Considering the left rule in Figure 1, both variables are used on the right-hand side to print out the names in the error message. Variables are instantiated by concrete values when the rule is applied. If a variable occurs more than once in the left-hand side the matching values have to be equal. In this way, variables are just shortcuts for longer expressions, such as “n” for “p1.name”, and some additional condition, e.g. adding a person “p3” with name “n” to the left-hand side would yield the additional condition “p1.name.equals(p3.name)”.

Attribute computations Moreover, it is possible to compute a new attribute value in the right-hand side of a rule. For this purpose, variables which are introduced in the left-hand side may be used. Reconsidering the left rule in Figure 1, the error message is

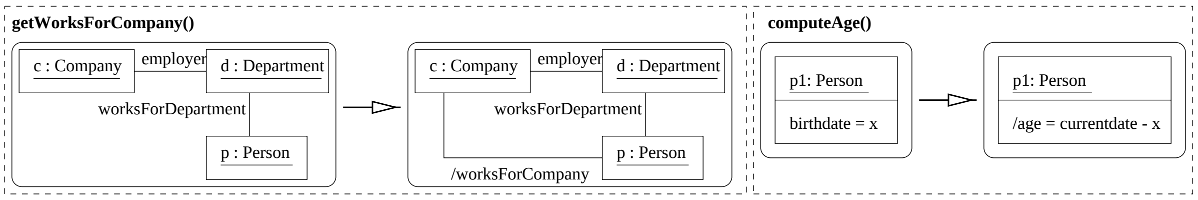

Figure 2. A sample rule for the update of a derived attribute

produced by calling a method "married" with two arguments being the variables occuring on the left-hand side. See the right rule of Figure 2 for another example of an attribute computation. This rule computes the new value of a derived attribute. Please note that the derived attribute does not have to occur on the left-hand side, because its value is not needed for testing. Derived attributes are indicated by a leading slash in UML.

Negative application conditions It is often not enough to test the existence of some objects and links as it is done by the left-hand side. Instead, the non-existence of some structure should lead to some reaction. A common constraint states that objects or links should be created if they are not already there. To indicate the non-existence of an object or link it is drawn dashed on the left-hand side.

The first diagram of Figure 3 shows a left-hand side of a rule that asks for a person without any child. Applying a rule with a negative application condition means that the positive part has to be found, but the negative extension cannot be matched, otherwise the rule is not applied. If a rule should be applied at most once, this can be achieved by formulating the newly created part also as negative application condition.

There are requirements that are not expressable by a single negative application condition, but by more than one. Consider e.g. the second diagram in Figure 3 which contains two negative application conditions. Each one is depicted in a dashed frame. Here, we look for a person which has no children *and* is not married. Combining both conditions in one (,i.e. depicting them in one dashed frame,) this would mean that we search for a person which has no children *or* is not married.

Comparing the combination of negative conditions with that of positive conditions we can state that conjunctions of positive conditions are expressed in one left-hand side of a rule. Disjunctions can be expressed by formulating several rules, each containing one alternative.

Moreover, it is possible to force that two objects in a rule should not be mapped to one and the same. Consider e.g. the third diagram in Figure 3 which is a test on different persons with the same names. More advanced application conditions allowing also predicate logic formulae on diagrams can be found in [HW95].

Attribute conditions Often it is not enough to restrict the matching structures for a rule, but also the matching attribute values may be restricted by additional conditions. An attribute condition may be any boolean expression which may contain rule variables.Moreover, object roles defined may be used. Consider e.g. the fourth diagram of Figure 3 which may be a left-hand side. It states that the age of a parent has to be greater than that of his or her child. An attribute condition should be free of side-effects, because it may be evaluated *many* times before the rule is applied.

The left rule in figure 4 describes the movement of a lift one floor up only if there is

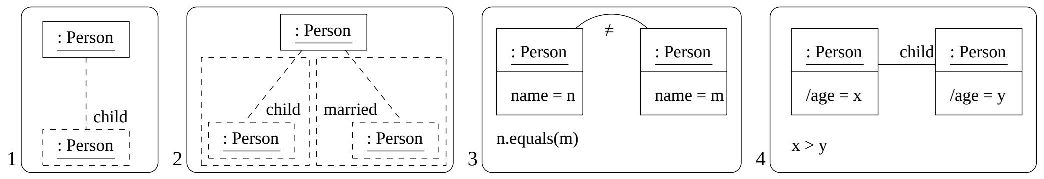

Figure 3. Left-hand sides with application conditions

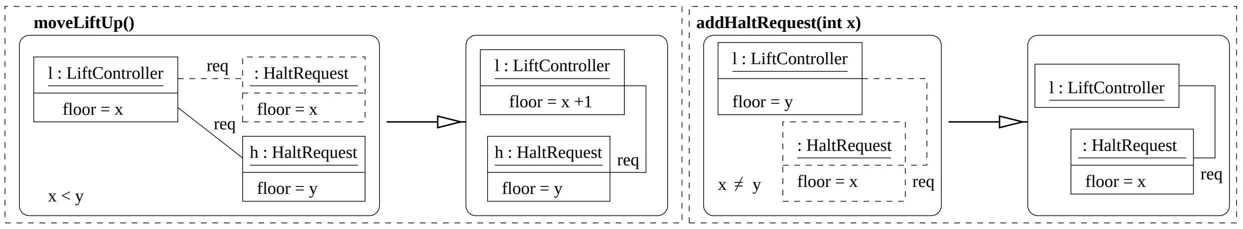

Figure 4. Sample rules for lift controlling

a halting request for an upper floor and there is not a halting request for the current floor. This rule has a negative application condition and an attribute condition. Here, the non-existence of a link together with an object is required, if its attribute has a certain value.

Rule parameters Another example for the combination of a negative application condition with an attribute condition is given by the right rule in Figure 4. This rule serves as a kind of "watch dog". If an external event occurs that creates a halting request for floor "x", this floor number is forwarded to the rule. The rule has an input parameter which is nothing else than a rule variable that is set from outside and not by matching. A new halting request for a floor "x" is added to a lift controller only if there is not already a halting request for this lift controller at this floor. This condition is expressed as a negative application condition. Moreover, there is an attribute condition stating that the halting request to be added has to be for another than the current floor where the lift is. Please note that the rule parameter is used in the negative application condition as well as in the attribute condition.

Rule diagrams Rules may be grouped into rule sets. These sets can be associated to classes, packages and components. A rule diagram shows this organization of rules. Consider e.g. the rule diagram on the left of Figure 5. Here, two rule sets where each rule is indicated by its name are organized in two different packages. Moreover, rule "computeAge" may also be associated to class "Person" (, depicted in an additional compartment below the methods). Two rules in package "RegisterOfPersons" are tagged with the property "automatic" which causes a test of applicability for a rule immediately when a relevant change of a matchable structure part occurs. A relevant change means the creation or deletion of an object or link as well as a new attribute value which may also occur in a match of the left-hand side of a rule. Then, the applicability of this rule is tested. Rules which are triggered by a time event are tested periodically. E.g. rule "computeAge" is triggered once a day to update the age if necessary. (Note that rule "testMarried2" does not occur in the rule diagram, because it is meant as alternative to rule "testMarried".)

Non-automatic rules, e.g. the rules in package "LiftController" have to wait for an event

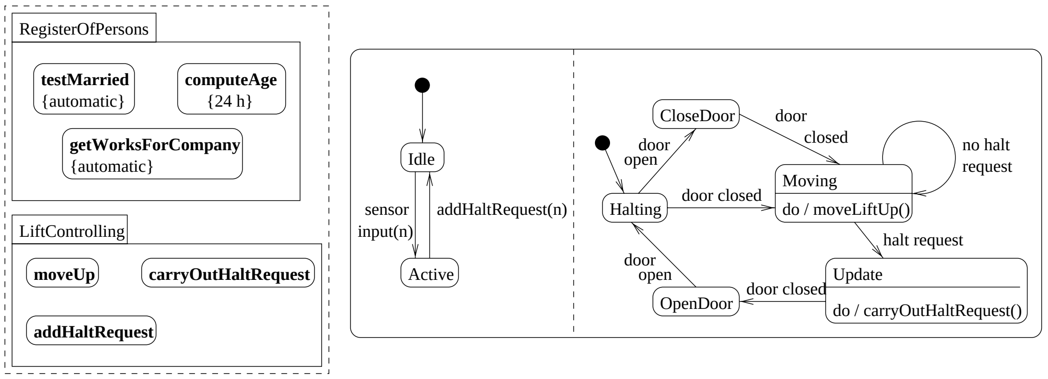

Figure 5. Sample rule and state diagram triggering rule applications

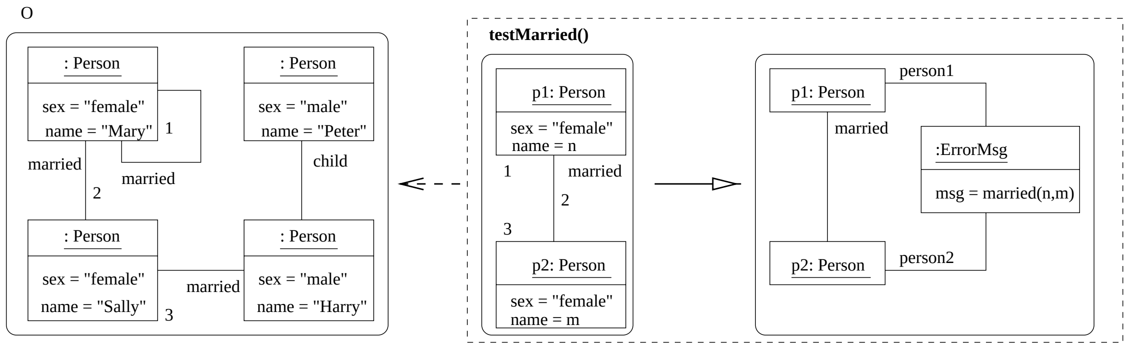

Figure 6. A sample matching of object diagrams

that triggers the test for applicability. Their application has to be triggered by an explicit event. This may be an event occuring in some behavior diagram. E.g it is meaningful that an external signal event causes a test of applicability for rule “addHaltRequest”. In case of applicability the rule is applied once. There may be a call event which causes the applicability test for rule “moveUp”. Consider the state diagram on the right of Figure 5 where rule “moveLiftUp” is called in state “Moving”. When the lift stopped on some floor rule “carryOutHaltRequest” is triggered to update the set of halt requests. (This rule which is not explicitly shown disconnects a halt request from its lift controller.)

3 Rule Semantics

Matching of object diagrams The matching of a rule's left-hand side L to an object diagram O is a mapping of all the objects and links of L onto those of O in a way that this mapping is compatible with the source and target relations of the links. Furthermore, the class names of interrelated objects have to be the same. The attribute values of corresponding objects have to be either equal or a variable in L is instantiated by the corresponding value in O. See Figure 6 for an example of a match of rule “testMarried” in Figure 1. An object or link of the left object diagram is mapped to an object or link in the right object diagram if they have equal numbers written aside. There is no matching to persons “p3” and “p4”possible, since they have a non-matching sex.

But, it is possible to match more than one object of L on one and the same object in O. E.g. person “x” and person “y” may be mapped to person “p1”. In this case, variables “n”

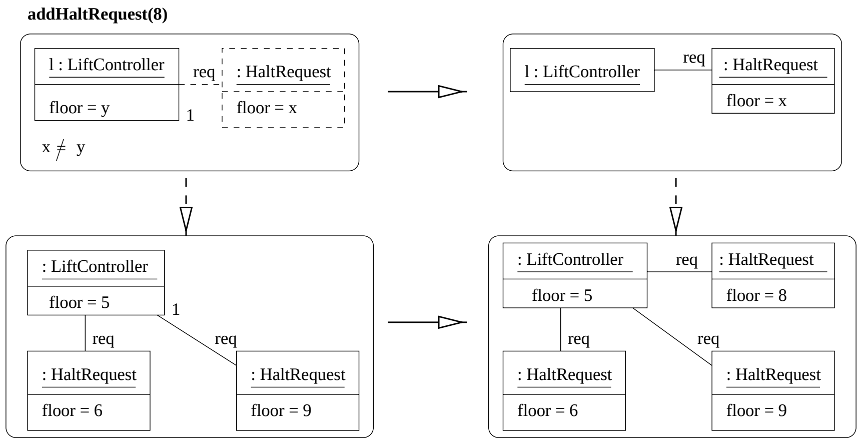

Figure 7. A sample application of the rule in Figure 4

and “m” are instantiated by the same name “Mary”. The link in between can be mapped to the reflexive link in O. This is also an inconsistency which can be discovered by this rule application.

To forbid such a non-injective matching see the third diagram of Figure 3. To force the matching onto two different objects the “person”-objects are connected by a special link indicating that these two are not allowed to be mapped on one. This left-hand side looks for two different persons with the same name.

Moreover, the negative application conditions have to be checked. This means that there is not any extension of the mapping found covering the additional objects and links of the negative condition. Considering the second diagram in Figure 3 the positive part of it can match with all persons in diagram O in Figure 6. The left negative application condition is true for persons “p1” to “p3” whereas the right condition holds only for “p3”. Therefore, only the matching with person “p3” fulfills all conditions.

Rule application Having matched a rule to an object diagram by mapping the positive part to a certain subpart of the diagram and checking all negative application conditions and attribute conditions, the rule can be applied to this diagram. This means that new objects and links may be created, old ones deleted and new values may be computed for attributes. See e.g. the rule application in Figure 7. Here, rule “addHaltRequest(8)” is applied to an object diagram showing two halting requests for two different floors. Both requests are linked to the lift controller. Applying the rule at the given match with the given parameter, a halting request for floor 8 is added to the lift controller, i.e. a “HaltRequest”- object is created and a “req”- link is added between the “LiftController”-objects and this new one. The “LiftController”-Object to which the new halting request is added is indicated by the match and the interrelation between the object diagrams of the rule. (There is only one possible match in this example.)

Consider rule “testMarried” and its matching to object diagram O in Figure 6 which is given in a way that persons “x” and “y” are both mapped to person “p1” and the link in between to the reflexive link at this person. In this case, the application of the rule would delete this reflexive link.

Please note that there is some implicit frame condition when applying a rule. This means that exactly those actions are performed which are stated in the rule, except for the implicit deletion of dangling links.

Non-determinism Considering again rule "testMarried" there are two possible matchings to object diagram O in Figure 6, the one indicated by numbers in Figure 6 and the match to person "p1" and its reflexive link. This means that the rule is applicable twice. There is not any restriction or priority which application should be performed first. Thus, rule application contains some kind of non-determinism. But looking closer to these rule applications we recognize that these applications do not interfere, i.e. one application does not prevent the other. There are two inconsistent situations and both have to - and can - be solved.

Another kind of non-determinism occurs with the selection of the rule to be applied. E.g. the adding of a halting request and moving the lift one floor up (Figure 4) are independent of each other when performed in a situation as depicted in the lower left diagram of Figure 7. This means that adding the halting request does not prevent the lift from moving up, and vice versa. Deleting a halting request for a certain floor is not independent of moving the lift up to the next floor. These actions are possible in this order, but not vice versa due to the negative application condition for moving up. One might imagine that it quickly becomes difficult to detect conflicting rules for a larger rule set. Here, the formal semantics of rule application based on graph transformation helps to analyze the rule set. The notion of critical pairs – known from term rewriting – has been extended to graph transformation [Plu93, LM95] and can be used to detect conflicting rule applications.

Triggering of a rule application The test for applicability of a rule may be triggered by implicit and explicit events. An implicit event is the creation or deletion of an object or link or a new attribute value. If there is a rule which matches partly with the changed object, link or attribute and this rule is an automatic one, a test for applicability of this rule is triggered. (The match includes negative application conditions and attribute conditions.) If the rule is applicable at a completed match (which includes the changed item), it is applied once. Even if afterwards, the rule is still applicable at the same match, i.e. nothing of the matched structure changed, it is not applied any more. See e.g. the rules "computeAge" and "getWorksForCompany" in Figures 2. These rules compute derived attributes and associations, but do not change the original structure. A repeated application of rule "getWorksForCompany" is not meaningful. Also rule "testMarried" should be applied only once for one and the same match to avoid cascades of error messages. Rule "testMarried2" would be applicable only once for a given match to correct an inconsistent situation. This is achieved by requesting the existence of structure which is deleted afterwards. Or the other way around, negative application conditions request the non-existence of that structure which should be added. The application of one rule may trigger further rule applications if the rule is more than a test, i.e. changes some structure. Rule "computeAge" is triggered periodically once a day, i.e. by a time event.

4 Conclusion and Related Work

To support the modeling of system dynamics in a declarative style, this contribution shows how visual rules can be added to an object-oriented modeling technique. Rules can be advantageously used to describe e.g. integrity constraints. Other than in [Ken97] it especially supports the formulation of active constraints on the data model. Rules provide a useful level of abstraction, allowing the designer to focus on important behavior. The inherent trigger concept for rules relieves the designer from the task to explicitly control the constraint check.

Moreover, rules can be used to model event handling, to derived new attribute values and associations and to model strategies in business and engineering. Especially for this last aspect, it is popular to use rules. Consider e.g. [BK97] which roughly explain the business rule approach. Rules are also very common in the active database community. Similar to rule concepts in this area also our rules follow the event-condition-action paradigm. But unlike those approaches, our rules do not need explicit trigger events, but may be triggered automatically in response of a relevant change. Moreover, our rule concept supports constraints and actions on several interrelated objects which is not done in many active database systems.

Because of its underlying formal semantics on the basis of graph transformation, this rule concept supports the analysis of a designed rule set concerning e.g. conflicts, dependencies, and further properties. As already mentioned above, a notion of critical pairs – known from term rewriting – has been extended to graph transformation [Plu93, LM95], can be used to detect conflicting rule applications. Logical formulas may be used to ensure additional properties of the rule set, e.g. to ensure that a certain object is created only if another is existing. If a rule set does not ensure the validity of a certain set of formulas, the rule set is automatically corrected ([HW95]). Visual editing of graph rules and their application are supported by the graph transformation machine AGG [TER99] which is implemented in Java. The visual layout of the rules is very similar to that one used in this contribution and thus, to UML. As a next step we intend to extend AGG by analysis tools as indicated above. To incorporate rules into existing OO-modeling techniques, AGG may be integrated into corresponding modeling tools.

In [JZ98], rules are embedded into so-called story flow diagrams which are also meant to complement existing object-oriented modeling techniques. The embedded rules describe basic data operations which are controlled by the surrounding story board, i.e. all rule applications have to be explicitly triggered by control flow.

Turning from modeling to programming, there are several approaches to incorporate rules into object-oriented programming languages. Most of the approaches are based on concepts of the rule-based language OPS5 [CW88]. There are approaches like Rete++ [Hal98] where rules are added to C++. Also R++ [LPSM97] incorporates rules into C++. Here, the kind of rules are not general, but restricted to so-called path-based rules. A path-based rule contains paths from a 'this'-object only, i.e. each rule is formulated within the local view of an object. The rule concepts of these integrations differ more or less from the one presented in this contribution.

Acknowledgments: Many thanks to Michael Rudolf for giving several valuable comments on a draft version of this paper.

References

[BK97] M. Barnes and D. Kelly. Play By The Rules. *Byte Magazin*, June 1997.

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

[CMR+97] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic Approaches to Graph Transformation Part I: Basic Concepts and Double Pushout Approach. In [Roz97].

[CW88] T. A. Cooper and N. Wogrin. *Rule-Based Programming with OPS5.* Morgan Kaufmann, San Mateo, California, 1988.

[Hal98] The Haley Enterprise. Seamless Integration of Rules and Objects Using the Rete Algorithm and C++. `http://www.haley.com`, 1998.

[HW95] R. Heckel and A. Wagner. Ensuring Consistency of Conditional Graph Grammars – A constructive Approach. *Proc. of SEGRAGRA'95 "Graph Rewriting and Computation", Electronic Notes of TCS*, 2, 1995. http://www.elsevier.nl/locate/entcs/volume2.html.

[JZ98] J.-H. Jahnke and A. Zündorf. Specification and Implementation of a Distributed Planning and Information System for Courses based on Story Driven Modelling. In *Proc. of the 9th Int. Workshop on software Specification and Design, Ise-Shima, Japan*, pages 77–86. IEEE Computer Society, 1998.

[Ken97] S. Kent. Constraint Diagrams: Visualising Invariants in Object Oriented Models. In *Proceedings of OOPSLA'97.* ACM Press, 1997.

[LM95] M. Löwe and J. Müller. Critical Pair Analysis in Single-Pushout Graph Rewriting. In G. V. Feruglio and F. R. Llompart (eds.), *Proc. Colloquium on Graph Transformation and its Application in Computer Science.* Technical Report B-19, Universitat de les Illes Balears, 1995.

[LPSM97] D. Litman, P. F. Patel-Schneider, and A. Mishra. Modeling Dynamic Collections of Interdependent Objects Using Path-Based Rules. In *Proc. of the 1997 ACM-SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, 1997.

[Plu93] D. Plump. Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence. In M. Sleep, M. Plasmeijer, and M. C. van Eekelen (eds.), *Term Graph Rewriting*, pages 201–214. Wiley, 1993.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, E. Eddy, and W. Lorenson. *Object-Oriented Modeling and Design.* Prentice Hall International, 1991.

[Roz97] G. Rozenberg (ed.). *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific, 1997.

[Roz99] G. Rozenberg (ed.). *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2: Applications, Languages and Tools.* World Scientific, 1999. to appear.

[TER99] G. Taentzer, C. Ermel, and M. Rudolf. AGG-Approach: Language and Tool Environment. In [Roz99], see also http://tfs.cs.tu-berlin.de/agg.

[UML97] Rational Software Cooperation. Unified Modeling Language. available via http://www.rational.com, 1997.