Electronic Communications of the EASST
Volume 26 (2010)

Manipulation of Graphs, Algebras and Pictures

Essays Dedicated to Hans-Jörg Kreowski
on the Occasion of His 60th Birthday

Lifting Parallel Graph Transformation Concepts to Model Transformation based on the Eclipse Modeling Framework

Enrico Biermann, Claudia Ermel and Gabriele Taentzer

19 pages

Guest Editors: Frank Drewes, Annegret Habel, Berthold Hoffmann, Detlef Plump
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Lifting Parallel Graph Transformation Concepts to Model Transformation based on the Eclipse Modeling Framework

Enrico Biermann[1], **Claudia Ermel**[1] **and Gabriele Taentzer**[2]

[1] Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany
enrico@cs.tu-berlin.de, claudia.ermel@tu-berlin.de

[2] Fachbereich Mathematik und Informatik
Philipps-Universität Marburg, Germany
taentzer@mathematik.uni-marburg.de

Abstract: Model transformation is one of the key concepts in model-driven software development. An increasingly popular technology to define modeling languages is provided by the Eclipse Modeling Framework (EMF). Several EMF model transformation approaches have been developed, focusing on different transformation aspects. This paper proposes parallel graph transformation introduced by Ehrig and Kreowski as a suitable framework for modeling EMF model transformations with multi-object structures. Multi-object structures at transformation rule level provide a convenient way to describe the transformation of structures with a variable number of recurring structures, dependent on concrete model instances. Parallel graph transformation means the simultaneous application of a set of transformation rules synchronized at the application of a kernel rule. We apply our extended EMF model transformation technique to model the simulation of statecharts with AND-states.

Keywords: graph transformation, model transformation, Eclipse, EMF

1 Introduction

Model-driven software development is considered as a promising paradigm in software engineering. Models are ideal means for abstraction and enable developers to master the increasing complexity of software systems. Model transformation, e.g. for behavior simulation or for performing model refactoring [1] is a key concept for model-driven software development.

The Eclipse Modelling Framework (EMF) [2] has evolved to one of the standard technologies to define modeling languages. EMF provides a modelling and code generation framework for Eclipse applications based on structured data models. The modelling approach is similar to that of MOF, actually EMF supports Essential MOF (EMOF) as part of the OMG MOF 2.0 specification [3].

EMF models can be manipulated by several approaches to rule-based model transformations. A transformation framework for EMF models which follows the concepts of algebraic graph transformation [4] as far as possible, is presented in [5, 6]. Although graph transformation is an expressive, graphical and formal means to describe computations on graphs, it has limitations.

For example, when describing the operational semantics of behavioral models, one often has the problem of modeling a variable number of parallel actions at different places in the same model. A simple example are transformations of some object structures occurring multiple times with the same properties (e.g. being contained in the same container, or referencing the same objects). We call such an object structure *multi-object structure* in this paper. One way to transform multi-object structures is the sequential application of rules such that we have to explicitly encode an iteration over all the actions to be performed. Usually, this is not the most natural nor efficient way to express the semantics. Thus, it is necessary to have a more powerful means to express parallel actions.

As main contribution of this paper, we propose the use of *amalgamated graph transformation* concepts, based on parallel graph transformation, originally proposed by Ehrig and Kreowski in [7] and extended to synchronized, overlapping rules in [8], to define EMF model transformations with multi-object structures. The essence of amalgamated graph transformation is that (possibly infinite) sets of rules which have a certain regularity, so-called *rule schemes*, can be described by a finite set of *multi-rules* modeling the elementary actions. For the description of such rule schemes the concept of amalgamating rules [9] is used in this paper to describe the application of multi-rules in an unknown context. The synchronization of rule applications is done along kernel rule applications which leads to a transformation step being maximally parallel in the following sense: an amalgamated rule, induced by a scheme, is constructed by a number of multi-rules being synchronized at the kernel rule. The number of multi-rules is determined by the number of different matches found such that they overlap in the match of the kernel rule. Hence, the transformation of multi-object structures can be described in a general way though the number of actually occurring objects in the instance model is variable.

Since EMF models are graphs with an additional containment hierarchy on object nodes, we lift the concept of amalgamated graph transformation to amalgamated EMF model transformations. In our previous paper [5] we showed that a restricted form of EMF model transformations can be well described by algebraic graph transformations being based on consistent transformation rules. This opens up the possibility to verify EMF model transformations using analysis techniques for graph transformation. In this paper, we prove that this consistency result can be lifted to amalgamated EMF model transformation.

We show the usefulness of amalgamated EMF model transformation by simulating the behavior of statecharts with AND-states which may have an arbitrary number of orthogonal components (called *regions* in UML state machines). For example, when the system enters an AND-state, it actually goes to the initial simple state in each region in parallel.

The paper is organized as follows. In Section 2, we introduce EMF models as typed, attributed graphs and present our running example, an EMF model for a simplified variant of statecharts with AND-states. Section 3 reviews the concepts of parallel graph transformation and lifts them to EMF transformations with multi-object structures. This section contains our main result on consistency of amalgamated EMF model transformations. Using EMF transformations with multi-object structures, we model a general simulator for statecharts with AND-states. Section 4 presents related research, and Section 5 ends with the conclusions and future work.

2 EMF Models as Typed, Attributed Graphs with Containment

The Eclipse Modeling Framework (EMF) [2] has evolved to one of the standard technologies to define modeling languages. EMF provides a modeling and code generation framework for Eclipse applications based on structured data models. The modeling approach is similar to that of MOF, actually EMF supports Essential MOF (EMOF) as part of the OMG MOF 2.0 specification. Containment relations, i.e. aggregations, define an ownership relation between objects. Thereby, they induce a tree structure in model instantiations.

In [5], we consider EMF instance models[1] as typed graphs with special containment edges. Typing is expressed by a type graph. It has some similarities to a meta-model, but does not contain multiplicities and other constraints. For simplicity, we consider type graphs without inheritance in this paper. For a complete definition of EMF model transformation based on type graphs with inheritance, see [5].

Since the containment concept plays a special role in EMF models, we distinguish a special kind of edge types defining containments in the type graph.

Definition 1 (Graph and graph morphism) A graph $G = (G_N, G_E, s_G, t_G)$ consists of a set G_N of nodes, a set G_E of edges, as well as source and target functions $s_G, t_G : G_E \to G_N$.

Given two graphs G and H, a pair of functions (f_N, f_E) with $f_N : G_N \to H_N$ and $f_E : G_E \to H_E$ forms a *graph morphism* $f : G \to H$, if it has the following properties:

1. $\forall e \in G_E : f_N \circ s_G(e) = s_H \circ f_E(e)$, with $s_G(e) \in G_N$, and
2. $\forall e \in G_E : f_N \circ t_G(e) = t_H \circ f_E(e)$, with $t_G(e) \in G_N$.

If f_N and f_E are inclusions, then G is called a subgraph of H, denoted by $G \subseteq H$.

Definition 2 (Type graph TG, containment relation $contains_{TG}$) A type graph $TG = (N, E, s, t)$ is a graph together with a set $C \subseteq E$ of *containment edges*. We define a containment relation[2]
$$contains_{TG} = \{(n,m) \in N \times N \mid \exists c \in C : s(c) = n \wedge t(c) = m\} \cup$$
$$\{(x,y) \in N \times N \mid \exists z \in N : (x\ contains_{TG}\ z \wedge z\ contains_{TG}\ y)\}$$
Based on $contains_{TG}$ we create a relation containing cyclic containments only: $cycle_{TG} = \{(x,y) \in contains_{TG} | (y,x) \in contains_{TG}\}$. Then a subset of containment edges, called *cycle-capable containment edges*, is defined whose instances might be part of containment cycles: $C_{Cycle} = \{c \in C \mid (s(c), t(c)) \in cycle_{TG}\}$.

Example 1 (EMF Model for statecharts with AND-States) *Fig. 1 shows the EMF model for statecharts with AND-states, where an arbitrary number of states may be grouped in orthogonal regions of the same AND-state.*

A State *may contains* Regions, *each of them containing* States *again. We attribute* States *by Boolean flags denoting whether they are initial or final states.* States *are connected by* Transitions *which are triggered by* Events*. For simulation, a* Current *object is needed which is linked to the currently active* States*. The* Current *object receives an* Event, *the first element of a queue*

[1] Note that the EMF community uses the terms “EMF model” for meta-model and “EMF instance model” for a model conforming to a meta-model.

[2] If there is no confusion, we use infix notation for $contains_{TG}$, e.g. $(x\ contains_{TG}\ y)$ instead of $(x,y) \in contains_{TG}$.

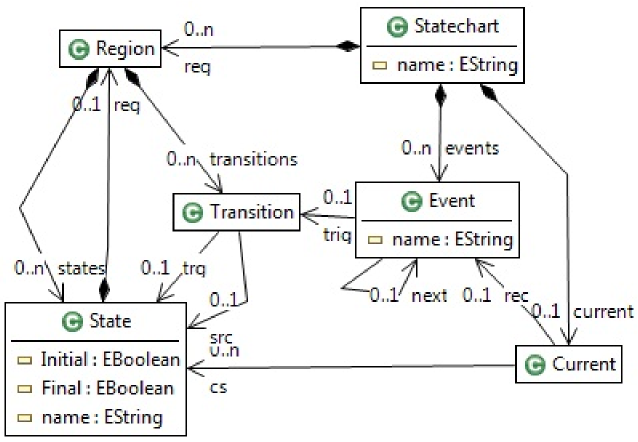

Figure 1: EMF model for statecharts with AND-states

*(*Events *linked by* next *links). The type graph with containment corresponding to the EMF model in Fig. 1 looks like the EMF model but has no multiplicities. We have six containment edge types (three of them have type* Statechart *as source, type* states *starts from type* Region *and type* reg *starts from type* State*). Types* states *and* reg *could lead to cycles in EMF instance models (corresponding to graphs typed over the type graph), because there could be theoretically a* Region r *which contains a* State *which transitively contains* Region r *again. Hence,* $C_{Cycle} =$ {states, reg} *is the set of cycle-capable containment edge types in this example.*

In *consistent* EMF instance graphs, each object node has at most one container and no containment cycles do occur. Graphs fulfilling these requirements are called *graphs with containment*. C-graphs can be related to each other by so-called C-graph morphisms. They are graph morphisms which preserve containment edges. Moreover, they have to be compatible with typing morphisms.

Although EMF instance models do not need to be rooted in general, this property is important for storing them, or more general, to define the model's extent.

Definition 3 (Graph with containment (C-graph)) A graph with containment, short C-graph, is a graph $G = (G_N, G_E, s_G, t_G)$ with a distinguished set of containment edges $G_C \subseteq G_E$. The containment edges induce the following binary relation $contains_G$ (the transitive closure of G_C):

- $contains_G = \{(x,y) \in G_N \times G_N \mid \exists e \in G_C : (s_G(e) = x \wedge t_G(e) = y)\} \cup$
  $\{(x,y) \in G_N \times G_N \mid \exists z \in G_N : (x \; contains_G \; z \wedge z \; contains_G \; y)\}$

All containment edges must fulfill the following properties (containment constraints):

- $e_1, e_2 \in G_C : t_G(e_1) = t_G(e_2) \;\Rightarrow\; e_1 = e_2$ (at most one container).
- $(x,x) \notin contains_G$ for all $x \in G_N$ (no containment cycles).

If G is typed over a type graph TG, there is a graph morphism $type\colon G \to TG$, called *typing morphism* which is consistent with containment, i.e. $\forall e \in G_C : type_{G_E}(e) \in TG_C$.

Please note that a type graph TG is not a C-graph in general (see e.g. our type graph for statecharts in Fig. 1 which has a containment cycle).

Definition 4 (Rooted graph) A C-graph G is called *rooted*, if there is a node $r \in G_N$, called *root node*, such that $\forall x \in G_N$ with $x \neq r : r\ contains_G\ x$.

Example 2 (Consistent EMF instance graph) *Fig. 2 shows a statechart with an AND-state. We model an ATM (automated teller machine) where the user can insert a bank card and, after the input of the correct pin, can draw a specified amount of cash from her or his bank account. The* display *region of the AND-state shows what is being displayed on screen, and, simultaneously, the* card-slot *component models whether the card slot is holding a bank card or not. The* enter *event triggers the transition before the AND-state to enter the AND-state. The* card-sensed *event happens if the sensor has sensed a user's bank card being put into the card slot. This event triggers two transitions in parallel. The next events (*pin-input, pin-ok *and* amount-input*) are local to the* display *region. The* end *event again triggers two transitions if the current state is any but the* welcome *state for the display region and* holding *for the card-slot region. Then, the final states are reached and the AND-state can be left if the* leave *event happens.*

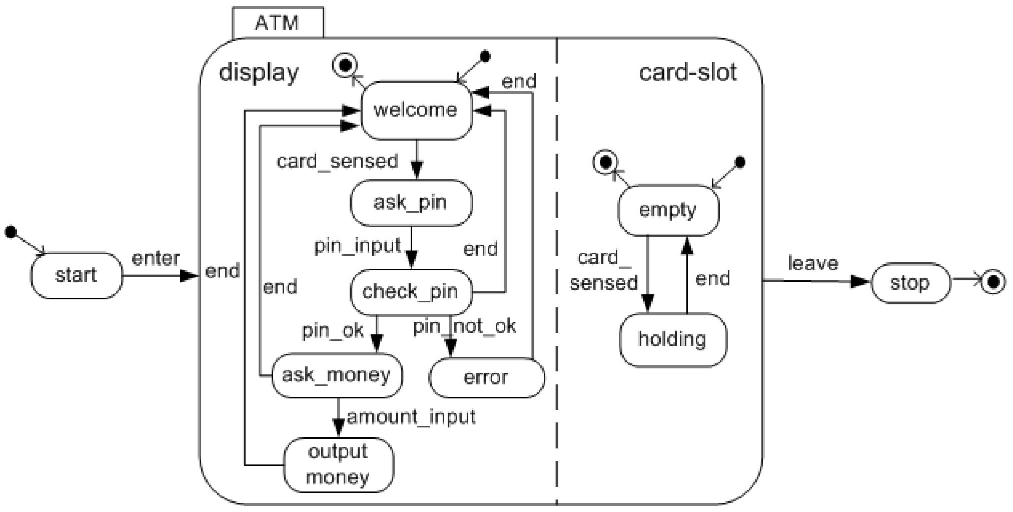

Figure 2: EMF instance graph: a statechart modelling an ATM

Fig. 3 shows the abstract syntax of the EMF instance graph corresponding to the ATM statechart in Fig. 2. This instance graph is typed over the type graph in Fig. 1. The initial state where we want to start the simulation is the start *state before the AND-state* ATM *is entered. The* Current *object points to the* start *state and is linked to an initial event queue consisting so far of the single event* enter *(the event needed to enter the AND-state) followed by the special event denoting the queue end. During the simulation, events may be added to the event queue such that the queue holds the events that should be processed during the simulation. For better readability, we write names which are not empty in quotation marks and put the name of a boolean attribute type (*Initial *or* Final*) if its value is true. Furthermore, we omitted the containment edges in Fig. 3 from the* Statechart *object named* ATM-SC *to all* Current *and* Event *objects, and from the* Region *objects to the corresponding* Transition *objects for better readability. Links being instances of containments are represented as containments for distinction from usual links.*

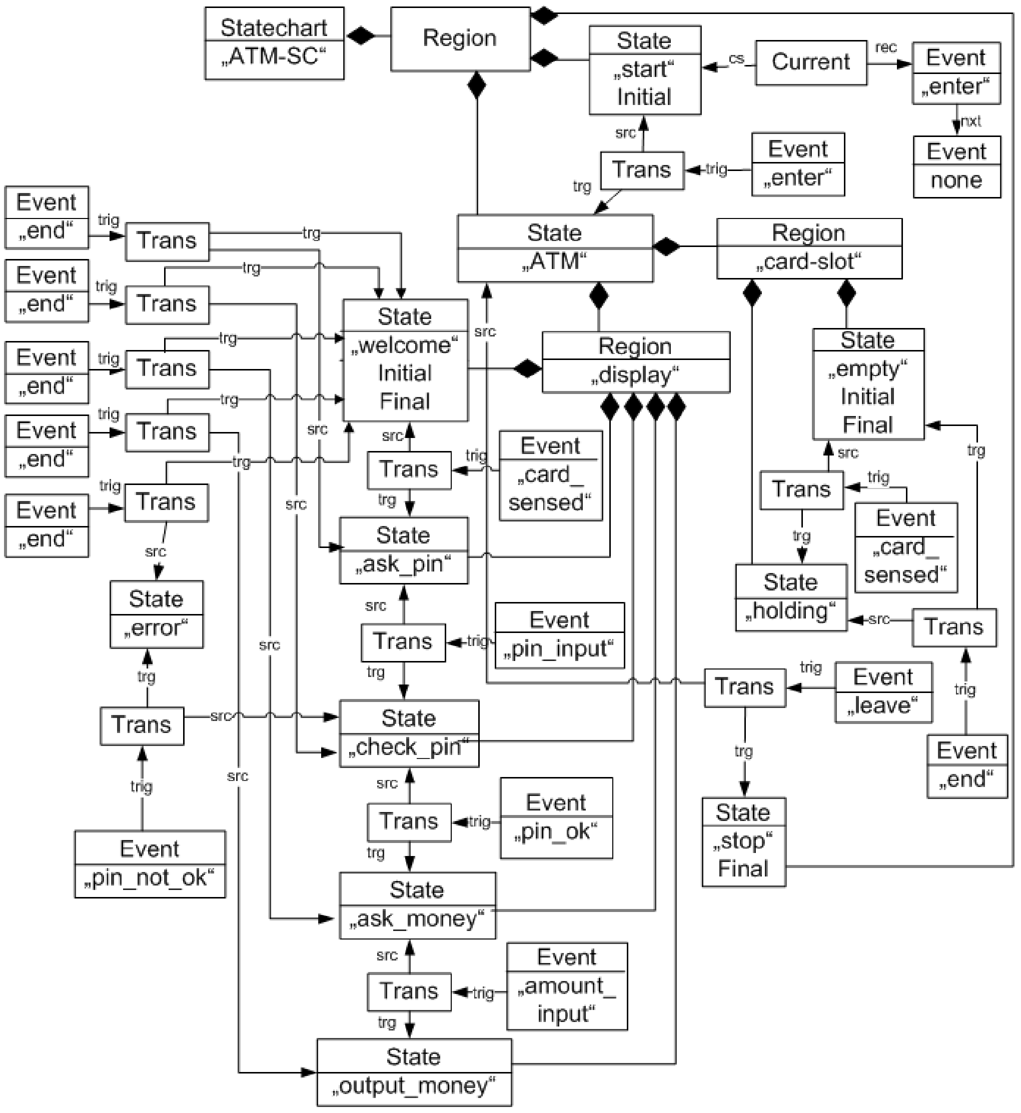

Figure 3: Abstract Syntax of the ATM statechart in Fig. 2 with *Current* pointer

The EMF instance graph in Fig. 3 is a C-graph since each object is contained in at most one container and there are no containment cycles. The C-graph is rooted, as the root node is the Statechart *object named* ATM-SC *which contains all objects transitively.*

3 EMF Model Transformations with Multi-Object Structures

EMF models can be manipulated by several approaches to rule-based model transformations. A transformation framework for EMF models which follows the concepts of algebraic graph transformation [4] as far as possible, is presented in [6]. But EMF model transformations do not always behave like algebraic graph transformation. The main reason is the difficulty to always satisfy the containment constraints of EMF. Hence, in our previous paper [5], we identify a kind of model transformation rules which lead to consistent EMF model graphs (i.e. fulfilling the containment constraints), if applied as normal graph transformation rules to consistent EMF model graphs. Thus, we identify a kind of EMF model transformations which behave like algebraic graph transformations. The advantage of this approach is that we provide a basis to apply the rich theory of algebraic graph transformation [4, 11, 12, 13] to EMF model transformations.

In Section 3.1, we shortly review the basic notions from [5]. Then in Section 3.2, we introduce amalgamated transformation, i.e. EMF model transformation with multi-object structures based on parallel graph transformation concepts and expand the capability of consistent EMF transformations by showing that the application of an amalgamated transformation rule to a consistent EMF model graph results in a consistent transformed EMF model graph again.

3.1 Consistent EMF Model Transformation Based on Graph Transformation

In the following, we define consistent transformation rules which restrict transformation rules such that their application to a consistent C-graph yields a C-graph again. If the C-graph is rooted in addition, the transformation result is also rooted.

Definition 5 (Transformation rule) A *transformation rule*, shortly *rule*, typed over a type graph TG is given by $r = (L \supseteq K \subseteq R,\ type,\ NAC)$, where L, K and R are C-graphs, $type$ is a triple of typing morphisms $type = (type_L\colon L \to TG,\ type_K\colon K \to TG,\ type_R\colon R \to TG)$, and NAC is a set of pairs $NAC_i = (N_i, type_{N_i}), i \in I$ with $L \subseteq N_i$ (and I being an index set), and $type_{N_i}\colon N_i \to TG$ a typing morphism, such that $type_{N_i} \supseteq type_L \supseteq type_K \subseteq type_R$ for all $i \in I$.

As a drawing convention, we omit K. All objects with equal numbers in L and R are also in K and are preserved when the rule is applied. A rule r can contain one or more negative application conditions (NACs) denoting situations which must not exist for the rule to be applicable. A NAC may be denoted partially not containing the whole left-hand side of its rule. It can be uniquely completed.

Definition 6 (Matching and application of transformation rules) Let $r = (L \supseteq K \subseteq R,\ type,\ NAC)$ be a transformation rule typed over TG, $(G, type_G)$ a typed C-graph with $type_G\colon G \to TG$ being a typing morphism, and $m\colon L \to G$ a graph morphism. Then m is a match with respect to r and $(G, type_G)$, if

1. m fulfills the so-called *dangling condition*, i.e. $\forall n \in L_N - K_N : \nexists e \in G_E - m_G(L_E)$ with $s_G(e) = m_N(n) \vee t_G(e) = m_N(n)$
2. m fulfills the *identification condition* for nodes, i.e. $\forall x_1, x_2 \in L_N$ with $m_N(x_1) = m_N(x_2)$: $x_1, x_2 \in K_N$ (analogously for edges)

3. m satisfies NAC, i.e. for each $nac_i = (N_i, type_{N_i}) \in NAC, i \in I$ there does not exist a graph morphism $o_i \colon N_i \to G$ such that $o_i|_L = m$.

Given a match m, rule r can be applied to G which means to replace the matched part $m(L)$ by the corresponding right-hand side R of the rule. By $G \overset{r,m}{\Longrightarrow} H$, we denote the *direct graph transformation* where rule r is applied to G at match m leading to the result graph H. The formal construction of a direct transformation is a double-pushout (DPO) which is shown in the diagram to the right with pushouts (PO_1) and (PO_2) in the category of (typed) graphs. Graph D is the intermediated graph after removing $m(L)$, and H is constructed as gluing of D and R along K (see [4]).

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(PO_1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(PO_2)"] & R \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

Example 3 (Transformation rule) *Rule* addEvent(e), *shown in Fig. 4, allows to add a new event of name* e *into the event queue. In this way, the events that should be processed during a simulation run, can be defined in the beginning of the simulation. Moreover, events can be inserted also while a simulation is running.*

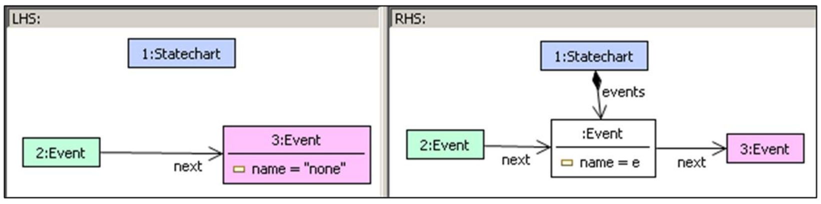

Figure 4: Rule *addEvent(e)* to insert Event *e* into the Event Queue

The application of rule r to C-graph G yields graph H which is not necessarily a C-graph. In the following we present sufficient conditions for rules such that their application to C-graphs result in C-graphs again. For that purpose, the form of allowed transformation rules has to be restricted such that nodes without container (except the root node) and containment cycles do not occur. Consistent transformation rules allow the following kinds of actions which change containments:

1. (*node creation*) Create a new object node and connect it immediately to its container, if there is one.

2. (*containment edge deletion*) Delete a containment edge together with its target object node or change the container of a preserved object node.

3. (*containment edge creation*) Create a containment edge with the target object node or change the container of an existing object node.

4. (*creation of cycle-capable containment edges*) For an object node contained via a cycle-capable containment edge, change its container only, if the old and the new container of the object node were already transitively related by containment. This pretty restrictively looking condition guarantees that containment cycles are not constructed.

Please note that an object node is always deleted with its containment relation, due to the dangling condition. Thus, we do not need an additional restriction for node deletion. In the following definition, we formalize all actions that preserve consistent containment relations which have been described above.

Definition 7 (Consistent transformation rule) Let $L'_C := L_C - K_C$, $R'_C := R_C - K_C$, $L'_N := L_N - K_N$ and $R'_N := R_N - K_N$. A transformation rule $p = (L \supseteq K \subseteq R,\ type,\ NAC)$ typed over TG is *consistent* wrt. containment if for each rule all the following constraints are satisfied:

1. (node creation) $\forall n \in R'_N$ with $type_R(n) = t_{TG}(c)$ for some $c \in TG_C$: $\exists e \in R'_C$ with $t_R(e) = n$,
2. (containment edge deletion) $\forall e \in L'_C$ with $t_L(e) = n$:
$$n \in L'_N \qquad \vee \qquad (n \in K_N \wedge \exists e' \in R'_C \text{ with } t_R(e') = n)$$
3. (containment edge creation) $\forall e \in R'_C$ with $t_R(e) = n$:
$$n \in R'_N \qquad \vee \qquad (n \in K_N \wedge \exists e' \in L'_C \text{ with } t_L(e') = n)$$
4. (creation of cycle-capable containment edges)
$\forall e \in R'_C$ which are cycle-capable and $n, m \in K_N$ with $s_R(e) = n \wedge t_R(e) = m$:
$\exists e' \in L'_C$ with $s_L(e') = o \wedge t_L(e') = m$:
$((o,n) \in contains_L \wedge (m,n) \notin contains_L) \vee (n,o) \in contains_L$

Please note that all conditions in Def. 7 have to be fulfilled to call transformation rules consistent. While conditions (1) - (3) have to be fulfilled for any kind of containment edges, condition (4) has to hold especially for cycle-capable containment edges. Note further that for condition (4), it is sufficient to inspect the containment in the rule's left-hand side. There cannot be a containment edge from the matched node m to n in C-graph G, because n would have two containers m and o then, and hence G would not be a C-graph.

Example 4 (Inconsistent transformation rule) *Consider the rule in the upper half of Fig. 5. Here, a state shall be moved from one region to another region. All rule graphs are C-graphs. The containment types are cycle-capable.*

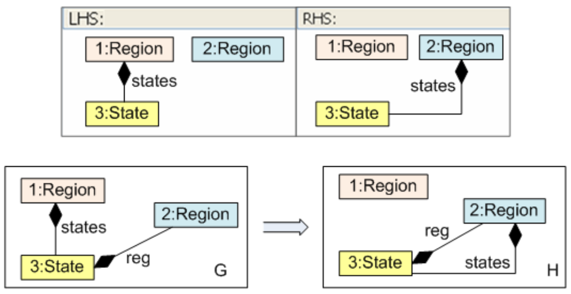

Figure 5: Application of an inconsistent rule leading to a containment cycle

The rule is not consistent because it violates condition (4): Its application can lead to a cycle as it is shown in the bottom of Fig. 5. When applied to G (which is a C-graph), the result is *graph H which is not a C-graph since it has a containment cycle. The reason for condition (4) is to prevent the introduction of cycles in the graph. However, a state still can be moved to another region (not being one of its own regions) using consistent rules without violating condition (4) by defining two rules: a first rule to move the state up the containment hierarchy, and a second rule to move it downwards into the destination superstate (see Fig. 6).*

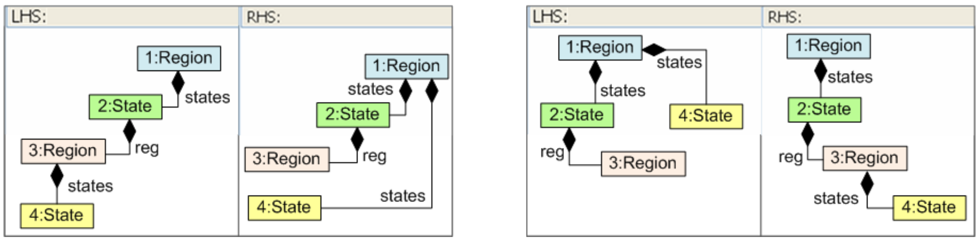

Figure 6: Consistent rules for moving states between regions

Example 5 (Consistent transformation rules) *Rule* addEvent(e) *presented in Example 3 is consistent, since for each created object its containment edge is created as well. Two further rules are depicted in Fig. 7 processing sequential transitions outside of AND-states.*

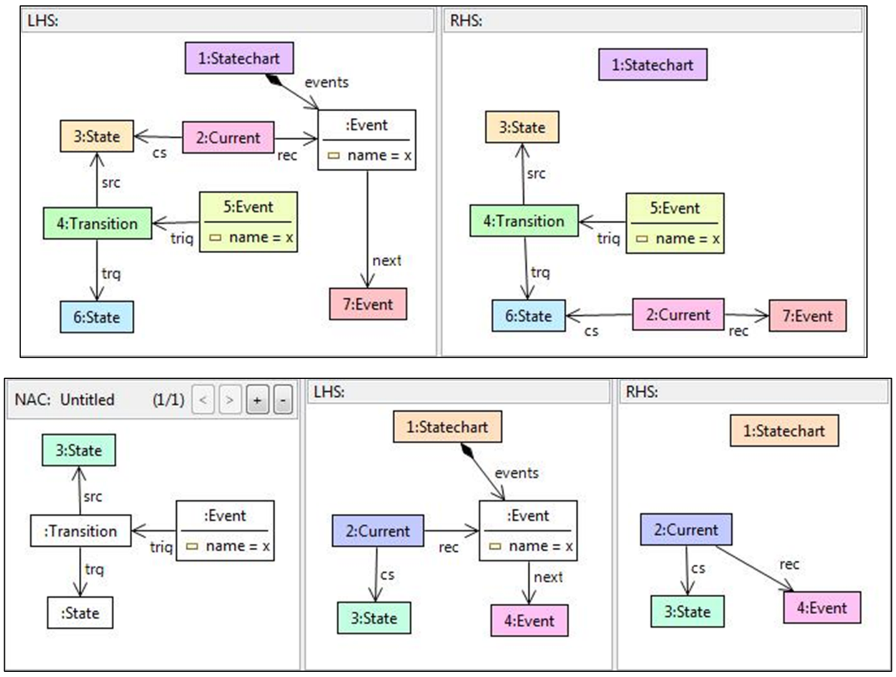

Figure 7: Rules *sequentialTransition* and *skipEvent*

Rule sequentialTransition *processes a transition in the current state which is triggered by the current event. This rule is consistent since the removed event node is deleted together with its containment edge. Rule* skipEvent *models the situation that no transition is triggered by* *the current event. In this case, the event is removed from the event queue, together with its containment edge.*

In our main theorems in [5], we show that the application of a consistent transformation rule to a consistent (rooted) EMF instance graph always results again in a consistent (rooted) EMF instance graph.

Theorem 1 (Consistent graph transformation step) *Let $r = (L \supseteq K \subseteq R, type, NAC)$ be a consistent transformation rule and $m : L \to G$ be a match to a C-graph G which is typed by $type_G : G \to TG$. Then, the result graph $(H, type_H)$ of direct transformation $(G, type_G) \overset{r,m}{\Longrightarrow} (H, type_H)$ is a C-graph.*

Proof. See [5]. $\square$

Theorem 2 (Rooted graph transformation step) *A consistent graph transformation step $(G, type_G) \overset{r,m}{\Longrightarrow} (H, type_H)$ leads to a* rooted *result graph H if graph G is rooted.*

Proof. See [5]. $\square$

3.2 Consistent EMF Model Transformations with Multi-Object Structures

In this section, we lift the essential concepts of parallel graph transformation [8] to EMF model transformation and also lift the consistency result for EMF model transformations from Section 3.1 to transformations with multi-object structures which we also call amalgamated EMF transformations.

Using parallel graph transformation, a system state modeled by a graph can be changed by several actions executed in parallel. Since graph transformation is rule-based without restrictive execution prescription, parallel graph transformation offers the possibility for massively parallel execution. The synchronization of parallel rule applications is described by common subrules, called *kernel rules*.

The simplest type of parallel actions is that of *independent actions*. If they operate on different objects they can clearly be executed in parallel. If they overlap just in reading actions on common objects, the situation does not change essentially. In graph transformation, this is reflected by a *parallel rule* which is a disjoint union of rules. The overlapping part, i.e. the objects which occur in the match of more than one rule, is handled implicitly by the match of the parallel rule. As the application of a parallel rule can model the parallel execution of independent actions only, it is equivalent to the application of the original rules in either order [7].

If actions are not independent of each other, they can still be applied in parallel if they can be synchronized by subactions. If two actions contain the deletion or the creation of the same node, this operation can be encapsulated in a separate action which is a common subaction of the original ones. A common subaction is modelled by the application of a *kernel rule* of all additional actions (modelled by *multi-rules*). The application of rules synchronized by kernel rules is then performed by gluing multi-rule instances at their kernel rules which leads to the corresponding *amalgamated rule*. The application of an amalgamated rule is called *amalgamated graph transformation*.

Formally, the synchronization possibilities of actions (multi-rule applications) are defined by an interaction scheme. For consistent amalgamated EMF model transformations (also called EMF model transformations with multi-object structures), we need consistent interaction schemes where all rules are consistent.

Definition 8 (Interaction Scheme) An *interaction scheme* $IS = (r_k, M)$ consists of rule r_k called *kernel rule* and a set $M = \{r_i | 1 \leq i \leq n\}$ of rules called *multi-rules* with $r_k \subseteq r_i$ for all $1 \leq i \leq n$.[3] All rules are typed over the same type graph. IS is *consistent*, if all rules are consistent.

In addition to the specification of multi-rules as well as their synchronization at a kernel rule, we must specify where and how often a set of multi-rules should be applied. The basic way to synchronize complex parallel operations is to specify a match of the kernel rule and to require that all multi-rules should be applied at *all possible matches* they have while overlapping with the kernel match (expressing massively parallel execution synchronized at one place). Please note that multi-rule matches may overlap in more than the kernel match. For further covering constructions see [8].

Definition 9 (Amalgamated transformation rule and its application)
Given an interaction scheme $IS = (r_k, \{r_i | i \in I\})$ and match m_k for the kernel rule r_K to C-graph G, IS is applied at m_K by constructing another interaction scheme $IS' = (r_k, \{r_j | 1 \leq j \leq n\})$ called *interaction scheme instance* of IS, with each r_j being a copy (a rule instance, i.e. a new rule with $L_i \cap L_j = L_k, K_i \cap K_j = K_k$, and $R_i \cap R_j = R_k$) of some r_i for $i \in I$. Each copy r_j of rule r_i is constructed by a different match $m_{ij} : L_i \to G$, i.e. for each two rule instances r_j, r_l for all $1 \leq j < l \leq n$ which are copies of the same r_i, we have that $m_j(L_j) \neq m_l(L_l)$.

There are maximal many rule instances r_j in the sense that each multi-rule match $m_i(L_i), i \in I$ corresponds to the match of one of its rule instances r_j:
$\forall m_i : L_i \to G \; \exists m_j : L_j \to G \; s.t. \; m_i(L_i) = m_j(L_j)$.

An *amalgamated transformation rule* $r_A = (L_A \supseteq K_A \subseteq R_A, type, NAC)$, shortly *amalgamated rule*, is a rule where the left-hand sides of all multi-rule instances in IS' are glued over the kernel left-hand side L_K yielding L_A. Similarly, K_A and R_A are constructed. NAC is the union of all NAC_j and NAC_k. $type$ is glued from the typing morphisms of all rule instances. Morphism m_A, called *amalgamated match* of r_A to G, is constructed by gluing all m_j which overlap at m_k.

Example 6 (Interaction scheme and amalgamated rule) *In Fig. 8, a sample interaction scheme $IS = \{r_k, \{r_1\}\}$ is shown in the upper left corner. The common sub-action (adding a loop to a object 1) is modeled by kernel rule r_k. We have only one multi-rule r_1 modeling that at each possible match object 2 shall be deleted together with its containment edge, and a new object shall be inserted such that it is contained in object 1. Both the kernel rule and the multi-rule are consistent and the kernel rule is part of the multi-rule. Given graph G, we have obviously three different matches from the multi-rule r_1 to G which overlap in the match of the kernel rule to G only. Hence, we have three multi-rule instances, each of them with a different match to G, and there are no more matches from r_1 to G. Gluing the multi-rule instances at their common kernel rule, we get the amalgamated rule r_A with respect to G, together with match $m_A : L_A \to G$.*

[3] $r_k \subseteq r_i$ is valid if $L_k \subseteq L_i$, $K_k \subseteq K_i$, and $R_k \subseteq R_i$.

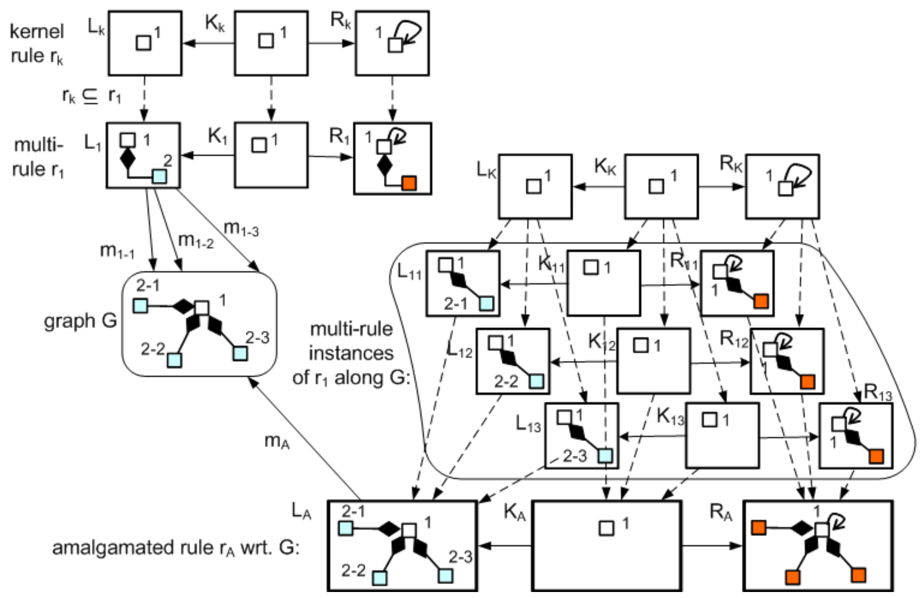

Figure 8: Construction of an amalgamated rule

We call a transformation *amalgamated* (or, alternatively, *transformation with multi-object structures*) if an amalgamated rule is applied. As with simple rules, an amalgamated rule can only be applied, if the amalgamated match satisfies the gluing condition. Note that a special interaction scheme consists of only one rule, i.e. a kernel rule, such that the interaction scheme is applied like a usual sequential rule.

Theorem 3 *The construction in Def. 9 yields a unique amalgamated rule up to isomorphism.*

Proof. Let $IS = (r_k, \{r_i | i \in I\})$ be an interaction scheme and m_k a match for the kernel rule r_K to C-graph G. First, we have to show that there is a unique interaction scheme instance $IS' = (r_k, \{r_j | 1 \leq j \leq n\})$ of IS, with r_j being a copy of some r_i with $i \in I$ such that $r_k \subseteq r_j$ for all $1 \leq j \leq n$. Since IS' is as large as we find matches of multi-rules, we know that the maximal interaction scheme instance comprises at least all rules of IS' since there is a match $m_j : L_j \to G$ with $m_{j|L_k} = m_k$ for each rule r_j with $1 \leq j \leq n$, according to Def. 9. IS' does not contain further rules, since there is not a further match m' of some rule r_i with $i \in I$ which is different from all matches m_j of its copies. Hence, IS' is the unique interaction scheme instance for IS and m_k.

The second part of the amalgamated rule construction is the gluing of all multi-rule instances along r_k. This gluing construction is unique and does not depend on copy constructions, since multi-rule instances overlap in kernel rules only, i.e. $L_j \cap L_l = L_k$, $K_j \cap K_l = K_k$, and $R_j \cap R_l = R_k$ for all $1 \leq j < l \leq n$. □

In order to show that EMF instance graphs resulting from amalgamated transformation are consistent (Theorem 4), we construct the amalgamated rule from a given consistent interaction scheme and show that this amalgamated rule is a consistent transformation rule. Afterwards, we can apply Theorem 1.

Theorem 4 *Let $IS = (r_k, \{r_j | 1 \leq j \leq n\})$ be a consistent interaction scheme instance and $m_k : L_k \to G$ a match from r_k to a C-graph G. Then, the amalgamated transformation rule r_A resulting from the construction acc. to Def. 9 is consistent.*

Proof.
Case $n = 0$: There is no match of any multi-rule. The amalgamated rule r_A is equal to the kernel rule r_k, which is consistent by assumption, since IS is consistent.

Case $n = 1$: There is one multi-rule instance of a multi-rule r_i. The amalgamated rule r_A is equal to r_i, thus it is consistent by assumption.

Case $n > 1$: We have to show that the amalgamated rule r_A satisfies all four consistency constraints for transformation rules according to Def. 7:

1. (node creation) To show: $\forall n \in R'_{A_N}$ with $type_R(n) = t_{TG}(c)$ for some $c \in TG_C$: $\exists e \in R'_{A_C}$ with $t_{R_A}(e) = n$.
   W.l.o.g. $n \in R'_{j_N}$: Then, there is a unique $e \in R'_{j_C}$ with $t_{R'_{j_C}}(e) = n$, since r_j is consistent. There cannot be another $e \in R'_{A_C}$ with $t_{R_A}(e) = n$, since the construction of the amalgamated rule instances results in an overlap of multi-rules in the kernel rule only. (Note that the amalgamated match may glue multi-rule matches outside of kernel match.)

2. (containment edge deletion) To show: $\forall e \in L'_{A_C}$ with $t_{L_A}(e) = n$:
   $$n \in L'_{A_N} \qquad \vee \qquad (n \in K_{A_N} \wedge \exists e' \in R'_{A_C} \text{ with } t_{R_A}(e') = n).$$
   W.l.o.g. $e \in L'_{j_C}$ with $t_{L_j}(e) = n$. Then, $n \in L'_{j_N} \vee (n \in K_{j_N} \wedge \exists e' \in R'_{j_C}$ with $t_{R_j}(e') = n)$, since r_j is consistent.

3. (containment edge creation) To show: $\forall e \in R'_{A_C}$ with $t_{R_A}(e) = n$:
   $$n \in R'_{A_N} \qquad \vee \qquad (n \in K_{A_N} \wedge \exists e' \in L'_{A_C} \text{ with } t_{L_A}(e') = n)$$
   W.l.o.g. $\forall e \in R'_{j_C}$ with $t_{R_j}(e) = n$. Then, $n \in R'_{j_N} \vee (n \in K_{j_N} \wedge \exists e' \in L'_{j_C}$ with $t_{L_j}(e') = n)$, since r_j is consistent.

4. (creation of cycle-capable containment edges)
   To show: $\forall e \in R'_{A_{C_{Cycle}}}$ with $s_{R_A}(e) = n \wedge t_{R_A}(e) = m : \exists e' \in L'_{A_C}$ with $s_{L_A}(e') = o \wedge t_{L_A}(e') = m : \quad ((o,n) \in contains_{L_A} \wedge (m,n) \notin contains_{L_A}) \vee (n,o) \in contains_{L_A}$.
   W.l.o.g. $e \in R'_{j_{C_{Cycle}}}$ with $s_{R_j}(e) = n \wedge t_{R_j}(e) = m$.
   Then, there is $e' \in L'_{j_C}$ with $s_{L_j}(e') = o \wedge t_{L_j}(e') = m :$
   $((o,n) \in contains_{L_j} \wedge (m,n) \notin contains_{L_j}) \vee (n,o) \in contains_{L_j}$.
   In addition, we have to show that there is no $(m,n) \in contains_{L_j}$ for some $l \neq j$. Since r_j and r_l overlap in r_K only, $m,n \in L'_{K_N} \subseteq L'_{j_N}$ and $(m,n) \notin contains_{L_j} \implies (m,n) \notin contains_{L_l}$. $\square$

Corollary 1 *Given a consistent interaction scheme $IS = (r_k, \{r_i | 1 \leq i \leq n\})$ and matches m_k and m_i to G for all $1 \leq i \leq n$. Then, if G is a C-graph, the result graph H after applying interaction scheme IS to G is a C-graph as well.*

Proof. Due to Theorem 4, the amalgamated rule constructed from IS is consistent. By Theorem 1, consistent rules preserve C-graphs. Hence, the result graph H is again a C-graph. □

Corollary 2 *Given a consistent interaction scheme IS like in Corollary 1. Then, if G is a* rooted *C-graph, the result graph H after applying the interaction scheme IS to G is a* rooted *C-graph as well.*

Proof. Due to Theorem 4, the amalgamated rule constructed from IS is consistent. By Theorems 1 and 2, we know that consistent rules preserve C-graphs and the rootedness of C-graphs. Hence, the result graph H is a rooted C-graph. □

Example 7 (Simulator for statecharts with AND-States) *In our statecharts variant, every region belonging to an AND-state has exactly one initial state and at least one final state. The intended semantics for our statecharts requires that if an AND-state is reached, the active states become the initial ones of each region. A transition is processed if its pre-state is active and its triggering event is the same as the event which is received by the* Current *object (the first event in the queue). Afterwards, the state(s) following the transition become(s) active, the event of the processed transition is removed from the queue, and the previously active state(s) (the pre-state(s) of the transition) is/are not active anymore. More than one transition are processed simultaneously if they belong to different regions of the same AND-state, if their pre-states are all active and if they are all triggered by the same event which is received by the* Current *object. All regions belonging to the same AND-state must have reached a final state before the AND-state can be left and the transition from the AND-state to the next state can be processed. For our simulator we use the* Current *object not only as object which receives the next event (and is linked to the event queue) but also as pointer to the current active states. Thus, our simulation rules model the relinking of the* Current *object to the next active states and the updating of the event queue.*

Note that in the following screenshots of interaction schemes we use an integrated notation, where we define the kernel rule and one multi-rule within one rule picture. This is possible since each of our interaction schemes consists of a kernel rule and one multi-rule only. We distinguish objects belonging to the multi-rule by drawing them as multi-objects (with indicated multiple boxes instead of simple rectangles). The kernel rule consists of all simple objects which are not drawn as multiple boxes. All arcs adjacent to multi-objects belong to the multi-rule only, but not to the kernel rule. All multi-objects together with their adjacent arcs in one multi-rule form a multi-object structure.

The upper part of Fig. 9 shows the interaction scheme enterRegion *which moves the* Current *pointer along a transition that connects a state to an AND-state. In this case, the* Current *pointer has not only to point to the AND-state afterwards but also to all initial states of all regions of the AND-state. Hence, the amalgamated rule consists of as many copies of the multi-rule as there are regions in the AND-state (provided that each component has exactly one initial state which has to be ensured by a suitable syntax grammar).*

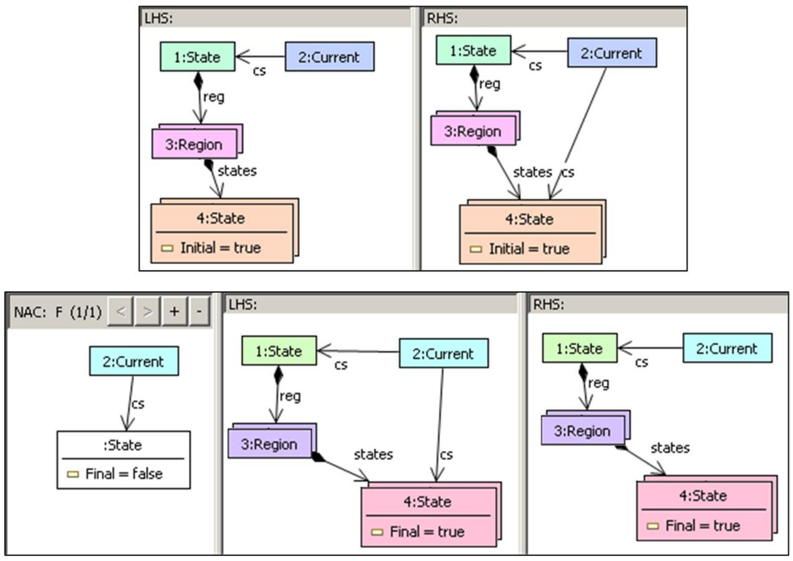

Figure 9: Interaction Schemes *enterRegion* and *leaveRegion*

Vice versa, when an AND-state is left, the Current *pointer has to be removed from all of its regions. This step is realized by the interaction scheme* leaveRegion *at the bottom of Fig. 9. The fact that the active states of all regions have to be* Final *is modelled by the NAC. The multi-rule models how all inner links from the* Current *pointer to the regions' final states are removed.*

A simultaneous transition is modelled by interaction scheme simultanTrans *in Fig. 10. Here, an arbitrary number of transitions in different regions of an AND-state are processed if triggered by the same event. In our ATM example this happens at different points of the simulation: When the AND-state is entered and the event* card-sensed *is happening, then the two first transitions of the two regions are processed simultaneously. Similarly, at any state of the display the user can abort the transaction: the* end *event triggers the return of the display region to state* welcome *and the return of the card-slot region to state* empty.

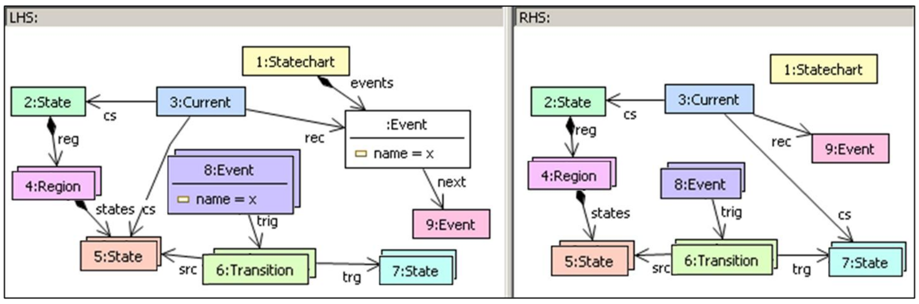

Figure 10: Interaction Scheme *simultanTrans*

The simultanTrans *interaction scheme is a good example for a concise way to model simultaneous transitions which are triggered by a single event. This would be quite difficult to model* *using simple rules. Note that this scheme is applicable also for sequential transition processing within an AND-state. Then there is only one copy of the multi-rule, similar to rule* sequentialTrans. *In the case that no transition leaving an active state is triggered by the current event, we have the situation that there is no copy of the multi-rule of* simultanTrans, *but the kernel rule can be applied anyway. This means that an event which does not trigger any transition inside of an AND-state simply is removed from the event queue. Again, this is similar to applying rule* skipEvent *with the difference that regions are used here.*

4 Related Work

There are two tool-based approaches known to us which also realize parallel graph transformation: AToM^3 and GROOVE, where AToM^3 supports the explicit definition of interaction schemes in different rule editors [14] and GROOVE implements rule amalgamation based on nested graph predicates [15]. A related conceptual approach aiming at transforming collections of similar subgraphs is presented in [16]. The main conceptual difference is that we amalgamate rule instances whereas the authors of [16] replace all collection operators (multi-object structures) in a rule by the mapped number of collection match copies. Similarly, a cloning operator is defined in [17] where cloned nodes correspond to multi-objects, but complete multi-object structures cannot be described. Moreover, the graph transformation tools PROGRES [18] and FuJaBA [19] feature so-called set nodes which are duplicated as often as necessary, but are not based on amalgamated graph transformation. None of the related approaches support the transformation of EMF models.

5 Conclusions and Future Work

This paper presented amalgamated EMF transformation as a valuable means for modelling and simulation. They extend the capabilities of EMF transformation based on simple graph transformation [5] by allowing parallel execution of synchronized EMF transformation rules. This is useful for e.g. specifying simulators for formalisms in which parallel actions can be performed. A concrete example of such a formalism are statecharts with AND states. It has been shown in the paper that an amalgamated transformation always leads to a consistent EMF instance model which satisfy the containment constraints of EMF.

In the future, we plan to apply the approach to other kinds of EMF model transformations, such as model refactorings where multi-object structures can be found frequently.

Amalgamated transformations of EMF models are currently implemented in the tool EMF Henshin (formerly called EMF Tiger [6]), a recently developed Eclipse plug-in supporting the specification and interpretation of EMF model transformations, based on graph transformation concepts. The goal of EMF Henshin is to provide the means to graphically define rule-based transformations on EMF models. Rule applications change EMF model instances in-place, i.e. an EMF instance model is modified directly, without being copied before. Moreover, control of rule applications by transformation units [20] is supported, as well as pre-definition of (parts of) the match. EMF Henshin currently consists of a *graphical editor* for visually defining EMF model transformation rules and an *interpreter* which executes EMF model transformation. In the near future, the translation of EMF transformation rules to AGG shall be supported to open up the possibility for verification of transformations.

Bibliography

[1] Mens, T., Tourwé, T.: A survey of software refactoring. Transactions on Software Engineering **30**(2) (February 2004) 126–139

[2] Eclipse Consortium: Eclipse Modeling Framework (EMF) – Version 2.4. (2008) http://www.eclipse.org/emf.

[3] Object Management Group: Meta Object Facility (MOF) Core Specification Version 2.0. http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF (2008)

[4] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theor. Comp. Science. Springer (2006)

[5] Biermann, E., Ermel, C., Taentzer, G.: Precise Semantics of EMF Model Transformations by Graph Transformation. In Proc. Conf. on Model Driven Engineering Languages and Systems (MoDELS'08). Vol. 5301 of LNCS., Springer (2008) 53–67

[6] Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., Weiss, E.: Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In Proc. Conf. on Model Driven Engineering Languages and Systems (MoDELS'06). Vol. 4199 of LNCS. Springer (2006) 425–439

[7] Ehrig, H., Kreowski, H.J.: Parallel graph grammars. In Lindenmayer, A., Rozenberg, G., eds.: Automata, Languages, Development. North Holland (1976) 425–447

[8] Taentzer, G.: Parallel and Distributed Graph Transformation: Formal Description and Application to Communication-Based Systems. PhD thesis, TU Berlin (1996)

[9] Böhm, P., Fonio, H.R., Habel, A.: Amalgamation of graph transformations: a synchronization mechanism. Journal of Computer and System Science **34** (1987) 377–408

[10] Tiger Project Team, Technische Universität Berlin: EMF Tiger (2009) http://tfs.cs.tu-berlin.de/emftrans.

[11] Ehrig, H., Kreowski, H.J., Montanari, U., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation. Vol 3: Concurrency, Parallelism and Distribution. World Scientific (1999)

[12] Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools. World Scientific (1999)

[13] Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformations, Vol. 1: Foundations. World Scientific (1997)

[14] de Lara, J., Ermel, C., Taentzer, G., Ehrig, K.: Parallel Graph Transformation for Model Simulation applied to Timed Transition Petri Nets. In: Proc. Graph Transformation and Visual Modelling Techniques (GTVMT) 2004. (2004)

[15] Rensink, A., Kuperus, J.H.: Repotting the geraniums: On nested graph transformation rules. In: Int. Workshop of Graph Transformation and Visual Modelling Techniques (GT-VMT'09). (2009)

[16] Grønmo, R., Krogdahl, S., Møller-Pedersen, B.: A collection operator for graph transformation. In: Int. Conf. on Model Transformation (ICMT'09). (2009)

[17] Hoffmann, B., Janssens, D., van Eetvelde, N.: Cloning and expanding graph transformation rules for refactoring. In: Int. Workshop on Graph and Model Transformation (GraMoT'05). Vol. 152 of ENTCS, Elsevier (2006) 53–67

[18] Schürr, A., Winter, A., Zündorf, A.: The PROGRES-approach: Language and environment. In [12].

[19] Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story diagrams: A new graph rewrite language based on the UML. In Proc. Workshop on Theory and Application of Graph Transformation. Vol. 1764 of LNCS, Springer (2000) 296–309

[20] Kreowski, H.-J. and Kuske, S.: Graph Transformation Units with Interleaving Semantics. Formal Aspects of Computing. Vol. 11, No. 6 (1999) 690–723