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

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

Enrico Biermann\textsuperscript{1}, Claudia Ermel\textsuperscript{1} and Gabriele Taentzer\textsuperscript{2}

\textsuperscript{1} Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany
enrico@cs.tu-berlin.de, claudia.ermel@tu-berlin.de

\textsuperscript{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 to be a suitable framework for modeling EMF model transformations with multi-object structures. Multi-object structures at transformation rule level provide a flexible way to describe the transformation of structures with a flexible number of recurring structures, dependent on concrete model instances. Parallel graph transformation means massively parallelizing the application of model transformation rules synchronized at 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. Since models are central artifacts in model-driven software development, the quality of generated software is directly dependent on the quality of models. Modifying models, i.e. for behavior simulation or for performing model refactoring [1]) is an important part of model 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
Lifting Parallel Graph Transformation Concepts to EMF Model Transformation

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 an arbitrary number of parallel actions at different places in the same model. A simple example are transformations of object structures of the same class occurring multiple times which all have the same properties (e.g. being contained in the same container, or referencing the same objects). We call such object structures multi-object structures 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 at kernel rules [9] is used in this paper to describe the application of multi-rules in an unknown context. The synchronization of rules along kernel rules forces a transformation step to be 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, transforming multi-object structures can be described in a general way though the number of actually occurring objects in the instance model is variable.

In order to respect the special restrictions of EMF models (imposed by the containment hierarchy), we lift the concept of amalgamated graph transformation to amalgamated EMF transformation by showing that the conditions from our previous paper [5] are sufficient to guarantee the consistency of amalgamated EMF model transformations.

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 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) 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 $s_G, t_G : G_E \to G_N$.

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

$$contains_{TG} = \{(n,m) \in N \times N \mid \exists c \in C : s(c) = n \land t(c) = m\} \cup$$

$$\{(x,y) \in N \times N \mid \exists z \in N : (x contains_{TG} z \land z contains_{TG} y)\}$$

Based on $contains_{TG}$ we create the equivalence relation $equiv_{contains} = r(contains_{TG}) - \{(x,y) \in contains_{TG} \mid (y,x) \notin contains_{TG}\}$.

with $r$ being the reflexive closure of $contains_{TG}$. Then we define $C_{Cycle} = \{c \in C : (s(c), t(c)) \in equiv_{contains}\}$ as a subset of cycle-capable containment edges that might be part of containment cycles.

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 (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

---

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 $(x,y) \in contains_{TG}$. 
contains a State which transitively contains Region \( r \) again. Hence, \( C_{\text{Cycle}} = \{ \text{State}, \text{Region} \} \) 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. 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 \( \text{contains} \) (the transitive closure of \( G_C \)):

\[
\text{contains} = \{(x, y) \in G_N \times G_N \mid \exists e \in G_C : s_G(e) = x \land t_G(e) = y\} \cup \\
\{(x, y) \in G_N \times G_N \mid \exists z \in G_N : (x \text{ contains}_G z \land z \text{ contains}_G y)\}
\]

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).
- \( (x, x) \notin \text{contains}_G \) for all \( x \in G_N \) (no containment cycles).

If \( G \) is typed over a type graph \( T_G \), there is a typing morphism \( \text{type} : G \to T_G \) which is consistent with containment, i.e. \( \forall e \in G_C : \text{type}_G(e) \in T_G \).

Please note that a type graph \( TG \) is no C-graph in general (see e.g. our type graph for state-charts 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 \( x \neq r : r \text{ contains}_G x \).

**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.

![Statechart diagram](image)

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.

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
Lifting Parallel Graph Transformation Concepts to EMF Model Transformation

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

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]. We then introduce amalgamated EMF model transformation, i.e. EMF model transformation with multi-object structures, based on parallel graph transformation concepts in Section 3.2 and expand the capability of consistent EMF transformations by showing that the application of an amalgamated EMF model 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 Concepts

In order to precisely define consistent graph rules, we have to define relations between typed C-graphs, so-called C-graph morphisms. They define mappings of nodes and edges respectively, such that they are compatible with typing, source and target functions (like typed graph morphisms) and especially preserve containment edge types.

**Definition 5** (C-graph morphism) Given two C-graphs $G$, $H$, a pair of functions $(f_N, f_E)$ with $f_N : G_N \rightarrow H_N$ and $f_E : G_E \rightarrow H_E$ forms a valid C-graph morphism $f : G \rightarrow H$, if it has the following properties:

- $f_N \circ s_G(e) = s_H \circ f_E(e)$, $f_N \circ t_G(e) = t_H \circ f_E(e)$, and
- $\forall e \in G_C \Rightarrow f_E(e) \in H_C$ (containment edges are preserved).

If $G$ and $H$ are typed over $T_G$, $f$ must be type compatible, i.e. $\text{type} G = \text{type} H \circ f$.

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

**Definition 6** (Graph rule) A graph rule typed over a type graph $T_G$ is given by $r = (L \supseteq K \subseteq R, \text{type}, NAC)$, where $L, K$ and $R$ are C-graphs, $\text{type}$ is a triple of typing morphisms $\text{type} = (\text{type}_L : L \rightarrow T_G, \text{type}_K : K \rightarrow T_G, \text{type}_R : R \rightarrow T_G)$, and $NAC$ is a set of pairs $NAC_i = (N_i, \text{type}_N_i), i \in \mathbb{N}$ with $L \subseteq N_i$, and $\text{type}_N_i : N_i \rightarrow T_G$ a typing morphism, such that $\text{type}_L \supseteq \text{type}_K \subseteq \text{type}_R$.

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 $p$ can contain one or more negative application conditions (NACs) denoting situations which must not exist for the rule to be applicable. Formally this is expressed by C-graphs $NAC_i$ such that $L \subseteq NAC_i$. A rule is applicable to a C-graph $G$ at a match $m : L \rightarrow G$ being a C-graph morphism if there is no injective C-graph morphism $n^i_i : NAC_i \rightarrow G$ such that $m \subseteq n_i^i$ for all $i \in I$. The application of rule $r$ to C-graph $G$ leads to the derivation of a 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. Formally, a derivation $G \xRightarrow{r} H$ is a DPO construction in the category of typed graphs and graph morphisms.

**Example 3** (Graph 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 also can be inserted while a simulation is running.

Now we define a special kind of graph transformation which formalizes a form of EMF model transformation leading always to EMF models consistent with typing and containment constraints. For that purpose, the form of allowed transformation rules has to be restricted. Consistent transformation rules allow the following kinds of actions which change containments:

1. Delete an object node with its containment relation.
2. Create a new object node and connect it immediately to its container.
3. Delete a containment edge together with its target object node or change the container of a preserved object node.

4. Create a containment edge with the target object node or change the container of an existing object node.

5. 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.

In the following definition, we formalize all actions that preserve consistent containment relations which have been described above.

**Definition 7** (Consistent graph 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 graph rule \( p = (L \supseteq K \subseteq R, \text{type, NAC}) \) is consistent wrt. containment if for each rule all of the following constraints are satisfied:

1. (node deletion) \( \forall n \in L_N' : \exists e \in L_C' \text{ with } t_{L_C'}(e) = n, \)
2. (node creation) \( \forall n \in R_N' : \exists e \in R_C' \text{ with } t_{R_C'}(e) = n, \)
3. (containment edge deletion) \( \forall e \in L_C' \text{ with } t_{L_C'}(e) = n : n \in L_N' \lor (n \in K_N \land \exists e' \in R_C' \text{ with } t_{R_C'}(e') = n) \)
4. (containment edge creation) \( \forall e \in R_C' \text{ with } t_{R_C'}(e) = n : n \in R_N' \lor (n \in K_N \land \exists e' \in L_C' \text{ with } t_{L_C'}(e') = n) \)
5. (creation of cycle-capable containment edges) \( \forall e \in R_C' \text{ which are cycle-capable and } n, m \in K_N \text{ with } s_{R}(e) = n \land t_{R}(e) = m : \exists e' \in L_C' \text{ with } s_{L}(e') = o \land t_{L}(e') = m \land ((o, n) \in \text{contains}_{L} \land (m, n) \notin \text{contains}_{L}) \lor (n, o) \in \text{contains}_{L} \)

Please note that all conditions in Def. 7 have to be fulfilled to call graph rules consistent. While conditions (1) - (4) have to be fulfilled for any kind of containment edges, condition (5) has to hold especially for cycle-capable containment edges. Note further that for condition (5) (creation of cycle-capable containment edges), 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 \( G \), because \( n \) would have two containers \( m \) and \( o \) then, and hence \( G \) would not be a C-graph.
**Example 4 (Consistent graph rules)** Rule `addEvent(e)` from Example 3 is a consistent graph rule since for each created object its containment edge is created as well. Two further rules are depicted in Fig. 5 which process sequential transitions outside of AND-states. Rule `sequential-Transition` 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. Again, the event is deleted together with its containment edge.

![Figure 5: Rules sequentialTransition and skipEvent](image)

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

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

*Proof.* See [5].

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

*Proof.* See [5].
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 transformations (also called EMF model transformations with multi-object structures), we need consistent interaction schemes where all rules are consistent.

**Definition 8** (Consistent Interaction Scheme) An interaction scheme \( I = (r^K, M, ke) \) consists of a kernel rule \( r^K \), a set \( M = \{r_i | 1 \leq i \leq n\} \) of rules called multi-rules, and a set \( ke \) of kernel rule embeddings \( ke_i : r^K \rightarrow r_i \) into the multi-rules (i.e. \( r^K \) is subrule of all multi-rules). \( I \) is consistent, if all rules are consistent.

**Example 5** An example of the construction of an amalgamated EMF transformation rule from an interaction scheme is given in Fig. 6.

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 (the blue) object 2 shall be deleted together with its containment edge, and a new (red) object shall be inserted such that it is contained in object 1. Both the kernel-rule and the multi-rule are consistent, and we have a subrule embedding from the kernel rule to the multi-rule given by the three C-graph morphisms \( L_K \rightarrow L_1, K_K \rightarrow K_1 \) and \( R_K \rightarrow R_1 \). Given graph \( G \), we have obviously three different matches.
from the multi-rule $r_1$ to $G$ which overlap in the match from the kernel rule to $G$. Hence, we have three multi-rule instances, each of them with a different match to $G$. Gluing the multi-rule instances at their common kernel rule, we get the amalgamated rule with respect to $G$, shown at the bottom of Fig. 6. The amalgamated rule contains the common action and, additionally, all actions from the multi-rules that do not overlap. Dashed arrows in Fig. 6 indicate rule embedding morphisms, embedding the kernel rule into the corresponding instances of the multi-rules, and the multi-rules into the amalgamated rule.

In addition to specifying how multi-rules should be synchronized, we must decide where and how often a set of multi-rules should be applied. The basic way to synchronize complex parallel operations is to require that a rule should be applied at all different matches it has in a given graph (expressing massively parallel execution). In this paper, we restrict the covering of $G$ (the image of all different matches from instances of multi-rules in $G$) to all different matches of multi-rules that overlap in the match of their common kernel rule and do not overlap anywhere else. For more complex covering constructions see [8].

**Definition 9 (Amalgamated EMF model transformation rule)**

Given a consistent interaction scheme $I = (r_K, \{r_i\}_{1 \leq i \leq n}, ke)$ with $(L_i - L_K) \cap (L_j - L_K) = \emptyset$ and a match $m_K$ for the kernel rule $r_K$. An amalgamated EMF model transformation rule $r_A = (L_A \leftarrow K_A \rightarrow R_A)$ is an EMF model transformation rule where as many copies $r_{i}\left| m_i \right.$ of multi-rules $r_i$ are joined as there are different matches $m_i : L_i \rightarrow G$ such that the copies $r_{i}\left| m_i \right.$ of $r_i$ in $r_A$ all overlap at kernel rule $r_K$ and the matches $m_i$ overlap at match $m_K$ only.
We speak of amalgamated EMF model transformation (or, alternatively, of EMF model transformation with multi-object structures) if the definition of the EMF model transformation is given by consistent interaction schemes. In this case, the application of an amalgamated EMF model transformation rule defines an EMF model transformation step. 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.

In order to show that EMF instance graphs resulting from amalgamated EMF model transformation are consistent (Theorem 3), we construct the amalgamated rule from a given consistent interaction scheme and show that this amalgamated rule is a consistent graph rule. Afterwards, we can apply Theorem 1.

**Theorem 3** Given a consistent interaction scheme $I = (r_K, \{r_i | 1 \leq i \leq n\}, k e)$ and matches $m_k$ and $m_i$ to $G$ for all $1 \leq i \leq n$. Then, if $G$ is a $C$-graph, the resulting amalgamated EMF model transformation rule is consistent.

**Proof.**

**Case $n = 0$:** No multi-rule is applied. The amalgamated rule $r_A$ is equal to the kernel rule $r_K$, which is consistent by assumption ($I$ is consistent).

**Case $n = 1$:** There is one application of a multi-rule. The amalgamated rule $r_K$ is equal to this multi-rule, thus it is consistent by assumption.

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

1. (node deletion) To show: $\forall n \in L_{A_N}': \exists e \in L_{A_C}'$ with $t_{L_{A_C}'}(e) = n$.
   W.l.o.g. $n \in L_{i_N}'$: Then, there is $e \in L_i'$ with $t_{L_i'}(e) = n$, since $r_i$ is consistent.

2. (node creation) To show: $\forall n \in R_{A_N}'$: $\exists e \in R_{A_C}'$ with $t_{R_{A_C}'}(e) = n$.
   W.l.o.g. $n \in R_{i_N}'$: Then, there is a unique $e \in R_{i_C}'$ with $t_{R_{i_C}'}(e) = n$, since $r_i$ is consistent.
   There cannot be another $e \in R_{A_C}'$ with $t_{R_A}(e) = n$, since the assumption allows an overlap of multi-rules in the kernel rule only. In this case, they would have to overlap in node $n$, too, which is not necessarily required here.

3. (containment edge deletion) To show: $\forall e \in L_{A_C}'$ with $t_{L_{A_C}'}(e) = n$:
   
   \[ n \in L_{A_N}' \lor (n \in K_{A_N} \land \exists e' \in R_{A_C}' \text{ with } t_{R_{A_C}'}(e') = n). \]
   W.l.o.g. $e \in L_{i_C}'$ with $t_{L_i}(e) = n$. Then, $n \in L_{i_N}' \lor (n \in K_{i_N} \land \exists e' \in R_{i_C}' \text{ with } t_{R_{i_C}'}(e') = n)$, since $r_i$ is consistent.

4. (containment edge creation) To show: $\forall e \in R_{A_C}'$ with $t_{R_{A_C}'}(e) = n$:
   
   \[ n \in R_{A_N}' \lor (n \in K_{A_N} \land \exists e' \in L_{A_C}' \text{ with } t_{L_{A_C}'}(e') = n) \]
   W.l.o.g. $\forall e \in R_{i_C}'$ with $t_{R_i}(e) = n$. Then, $n \in R_{i_N}' \lor (n \in K_{i_N} \land \exists e' \in L_{i_C}' \text{ with } t_{L_i}(e') = n)$, since $r_i$ is consistent.
the result graph 2, we know that consistent rules preserve C-graphs and the rootedness of C-graphs. Hence, due to Theorem 3, the amalgamated rule constructed from interaction scheme I to G is a C-graph as well.

Proof. Due to Theorem 3, the amalgamated rule constructed from interaction scheme I to G for all \(i \leq i \leq n\). Then, if G is a C-graph, the result graph H after applying interaction scheme I to G is a C-graph as well.

\[\text{Corollary 1} \quad \text{Given a consistent interaction scheme } I = (r_K, \{r_i|1 \leq i \leq n\}, ke) \text{ and matches } m_k \text{ and } m_i \text{ to G for all } 1 \leq i \leq n. \text{ Then, if G is a C-graph, the result graph } H \text{ after applying interaction scheme } I \text{ to G is a C-graph as well.} \]

Proof. Due to Theorem 3, the amalgamated rule constructed from I is consistent. By Theorem 1, consistent rules preserve C-graphs. Hence, the result graph H is again a C-graph.

\[\text{Corollary 2} \quad \text{Given a consistent interaction scheme } I \text{ like in Corollary 1. Then, if G is a rooted C-graph, the result graph H after applying interaction scheme I to G is a rooted C-graph as well.} \]

Proof. Due to Theorem 3, the amalgamated rule constructed from I 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.

\[\text{Example 6 (Simulator for statecharts with AND-States) \quad In our statecharts variant, every region belonging to an AND-state has exactly one initial state and at least one final state. The intended semantics for our statecharts requires that if an AND-state is reached, the active states become the initial ones of each region. A transition is processed if its pre-state is active and its triggering event is the same as the event which is received by the Current object (the first event in the queue). Afterwards, the state(s) following the transition become(s) active, the event of the processed transition is removed from the queue, and the previously active state(s) (the pre-state(s) of the transition) is/are not active anymore. More than one transition are processed simultaneously if they belong to different regions of the same AND-state, if their pre-states are all active and if they are all triggered by the same event which is received by the Current object. All regions belonging to the same AND-state must have reached a final state before the AND-state can be left and the transition from the AND-state to the next state can be processed. For our simulator we use the Current object not only as object which receives the next event (and is linked to the event queue) but also as pointer to the current active states. Thus, our simulation rules model the relinking of the Current object to the next active states and the updating of the event queue.} \]

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. 7 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 7: Interaction Schemes enterRegion and leaveRegion](image)

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. 7. 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. 8. 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.

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 sequential-Trans. 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: AToM³ and GROOVE, where AToM³ 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, Hoffmann et al. define a cloning operator 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 specifying simulators for formalisms in which parallel actions happen. This is the case for a great number of formalisms, such as statecharts with AND states. It has been shown in the paper that amalgamated EMF transformation always leads to consistent EMF instance models 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 are found frequently.

Amalgamated EMF transformation are currently being implemented in the tool EMF TIGER\cite{10, 6} (Transformation generation), a recently developed Eclipse plug-in supporting modeling and code generation for EMF model transformations, based on structured data models and graph transformation concepts. The goal of EMF Tiger is to provide the means to graphically define rule-based transformations on EMF models. Rule applications change an EMF model instance in-place, i.e. an EMF instance model is modified directly, without copying it before. Moreover, control of rule applications is supported by EMF TIGER, as well as pre-definition of (parts of) the match. EMF TIGER currently consists of a graphical editor for visually defining EMF model transformation rules, and a compiler, generating Java code from these transformation rules to be included into existing projects performing EMF model transformation. It also contains an interpreter which translates EMF transformation rules to AGG. This interpreter is useful for verification purposes.

Bibliography


