Electronic Communications of the EASST
Volume 7 (2007)

Proceedings of the
Sixth International Workshop on
Graph Transformation and Visual Modeling Techniques
(GT-VMT 2007)

Ensuring Containment Constraints in Graph-based Model Transformation Approaches

Christian Köhler, Holger Lewin, Gabriele Taentzer

12 pages

Guest Editors: Karsten Ehrig, Holger Giese
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Ensuring Containment Constraints in Graph-based Model Transformation Approaches

Christian Köhler[1], **Holger Lewin**[2], **Gabriele Taentzer**[3]

[1] christian.koehler@cwi.nl
Department of Software Engineering,
CWI Amsterdam, The Netherlands

[2] kinscher@cs.tu-berlin.de
Department of Software Engineering and Theoretical Computer Science
Technical University of Berlin, Germany

[3] taentzer@mathematik.uni-marburg.de
Department of Mathematics and Computer Science
Philipps-University Marburg, Germany

Abstract:

Within model driven software development, model transformation has become a key activity. A number of transformation approaches for metamodel-defined modeling languages have been developed in the past years and are going to be established in research and industry. None of these have made it to a standard yet. There is a demand for correct model transformation in various senses. Formal methods are helpful for showing correctness issues of model transformations. As one approach, graph transformation has been applied to the field of model transformation and is a perspective for achieving provable correct model transformations. We show in this paper, that *containment associations* as proposed by the OMG are an integral part of MOF-based languages and imply a couple of constraints which must be ensured in model transformation approaches. Based on a double-pushout approach to graph transformation, conditions are stated that ensure these containment constraints. This is an important step for achieving formal transformation semantics for modeling languages based on MOF, or specifically EMF.

Keywords: Model Transformation, Graph Transformation, MOF, EMF

1 Introduction

Model driven engineering is an emerging approach to software engineering aiming at fast development of high-quality software. The central entities in the terminology of model-driven software development are of course the *models*. Relations between these models are basically of two kinds: *instantiations* of so-called metamodels by models and *model transformations* between models typed by the same metamodel. A metamodel defines the common structure of all possible instantiations and therefore can be seen as some kind of language definition.

In this paper we consider a special kind of models, which are called *structural data models*. These models do not include any definition of behavior, but structural aspects only. Therefore, they are mainly used to specify domain specific languages. The two most popular modeling languages for structural data models are the *Meta Object Facility* (MOF) [OMG06] and the *Eclipse Modeling Framework* (EMF) [EMF]. While MOF is a slightly richer language, EMF comes with powerful code generation facilities. The key concepts in both languages are classes with inheritance and attribution and associations with multiplicity and containment properties. EMF can generate Java code which supports the creation, modification, storage, and loading of model instances. Moreover, it provides generators to support the editing of instance models.

It is possible to interpret these structures as graphs and define model transformations formally through graph transformations. Attribution of nodes and edges [EEPT06], node inheritance [EEPT05] and multiplicity constraints for nodes and edges [TR05] have already been studied in this area. An important property that has not been considered in this context are *containment* or *composite* associations.

Containment associations define an ownership relation between objects. Thereby, they induce a tree structure in model instantiations. In UML 2, ownership does not only occur in the form of composite associations between classes, but also at various other points, like components and their subcomponents or states and substates in state charts. In MOF and EMF, the tree structure induced by containment associations is further used to implement a mapping to XML, known as XMI (XML Metadata Interchange).

Containment always implies a number of constraints for model instantiations that must be ensured at run-time. The MOF specification [OMG06] states as semantical constraints for containment edges the following:

- *"An object may have at most one container."*
- *"Cyclic containment is invalid."*

As mentioned earlier, EMF provides full implementations of their models. These implementations always ensure these constraints. In fact, the MOF specification also says how such a constraint should be ensured, e.g.:

- *"If an object has an existing container and a new container is to be set, the object is removed from the old container before the new container is set."*

However, there is no suggestion how to avoid a cyclic containment of objects. Being a part of the MOF specification, these two properties are essential for valid models and therefore must be always ensured. This is especially the case when it comes to defining transformations for these models. A model transformation approach for MOF and EMF has to deal with these issues, i.e. must ensure that the result of a model transformation conforms to these constraints. We consider in this paper a graph transformation approach with formal semantics. The way of ensuring the containment constraints as stated in the MOF specification is not applicable in this context, since it would break the formal transformation semantics and therefore make existing results on termination and confluence of graph transformations invalid.

Therefore, we consider containment associations explicitly in the following and apply their properties in the context of a graph-based model transformation approach.

This paper is structured as follows: Section 2 gives a short introduction to model transformation by algebraic graph transformation. In Section 3, we consider graphs with containment relations and define the necessary containment conditions which have to be fulfilled by graph transformations. Section 4 gives a short description of related work in this area. At last, Section 5 contains a conclusion and refers to possible future work in the field of model checking and model transformation.

2 Model Transformation by Graph Transformation

Model transformations define relations between model instances. The most common scenario is a mapping from a source model to a target model. In this situation, an instance of the target model can be either completely constructed from scratch or updated incrementally. Transformations are usually defined through transformation rules. Basing model transformation on graph transformation, rules consist of one positive pattern, an arbitrary number of negative patterns and a description of the in-place modifications that should be performed. To apply a rule, it is necessary to either specify or automatically find a pattern match in the model instance that should be transformed. Thereafter, the existence of negative patterns has to be checked. If they are fulfilled, the in-place modifications can be applied.

In the graph transformation approach, patterns are given through graphs, while transformations of graphs are usually defined through pushout constructions. Here, we use the *double-pushout* approach (DPO) where a transformation rule is given by a span of injective graph homomorphisms $(L \xleftarrow{l} K \xrightarrow{r} R)$. Such a rule can be applied w.r.t. a given match m into a source model instance. See the following figure:

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "c"] \arrow[dr, phantom, "(PO)"] & R \arrow[d, "n"] \\
M & C \arrow[l, "g"] \arrow[r, "h"'] & N \\
Source \arrow[rr, dotted, "Transformation"'] & & Target
\end{tikzcd}$$

The input should be a graph corresponding to a valid model instance according to the MOF specification / EMF implementation. Therefore it must fulfill the containment constraints stated in the introduction. The aim is now to ensure that the result is also not violating the containment constraints. We achieve this goal by restricting the possible transformations. A rule application w.r.t. a given match is allowed, if the result does not violate the containment constraints. This is crucial for working with MOF and EMF models, especially for ensuring semantical properties like termination and confluence of transformations. For this purpose, we now introduce graphs with distinguished containment edges and use the double-pushout approach for defining transformations between them. Of course, other model constraints like multiplicities, have to be ensured, too, but are outside of the scope of this paper. For more details see [TR05].

3 Graphs with Containment Edges

Classes and associations can be interpreted as nodes and edges in a model graph. Accordingly, objects and links can be seen as nodes and edges in an instance graph. The 'instance of'-relation between these two graphs is achieved through a typing graph homomorphism, assigning to each object (each link) of an instance graph, a class (an association) in the corresponding model graph.

Definition 1 (Graph with containment edges) A graph with containment edges is a tuple $G = (V, E, C, source, target)$ consisting of a set of nodes V, a set of edges E, a distinguished set of containment edges $C \subseteq E$ and two functions $source, target : E \to V$ assigning a source and a target node to each edge. The containment edges induce the following transitive binary relation:

- $contains$[1] $= \{(x,y) \in V \times V \mid \exists e \in C : (source(e) = x \wedge target(e) = y)\} \cup \{(x,y) \in V \times V \mid \exists z \in V : (x \; contains \; z \wedge z \; contains \; y)\}$

The containment edges must have the following properties:

- $e_1, e_2 \in C : target(e_1) = target(e_2) \Rightarrow e_1 = e_2$ (at most one container).
- $(x,x) \notin contains$ for all $x \in V$ (no cycles).

This definition ensures that there are no containment cycles and that an object has at most one incoming containment edge, i.e. not more than one container. Accordingly, a homomorphism for graphs with containment edges is a usual graph homomorphism that preserves containment edges and their order properties.

Definition 2 (Homomorphism for graphs with containment edges) Given two graphs with containment edges G_1, G_2, a pair of functions (h_V, h_E) with $h_V : V_1 \to V_2$ and $h_E : E_1 \to E_2$ forms a valid homomorphism $h : G_1 \to G_2$ for graphs with containment edges, if it has the following property:

- $e \in C_1 \Rightarrow h_E(e) \in C_2$ (containment edges are preserved).

These two definitions induce the category of graphs with containment edges, which will be denoted as **CGraphs** in the following.

As already mentioned above, typing is accomplished by a graph homomorphism from an instance graph to a model graph. It is important to note that the constraints for containment edges as stated in Definition 1 apply to instance graphs only, not to model graphs. As shown in Figure 1, model graphs can have containment cycles, being the *substates* relation in this example. However, an instance of this model must not violate these constraints. Correspondingly, the constraints in the MOF specification apply to objects and links only, not to classes and associations.

Since we use the DPO approach to transform graphs with containment edges, it is necessary to show that the category of graphs with containment edges has pushouts[2] [EEPT06]. Pushouts

[1] If there is no confusion, we use infix notation for *contains*, e.g. $(x \; contains \; y)$ instead of $(x,y) \in contains$.

[2] Analogously for the category of graphs with containment edges typed over a fixed metamodel graph.

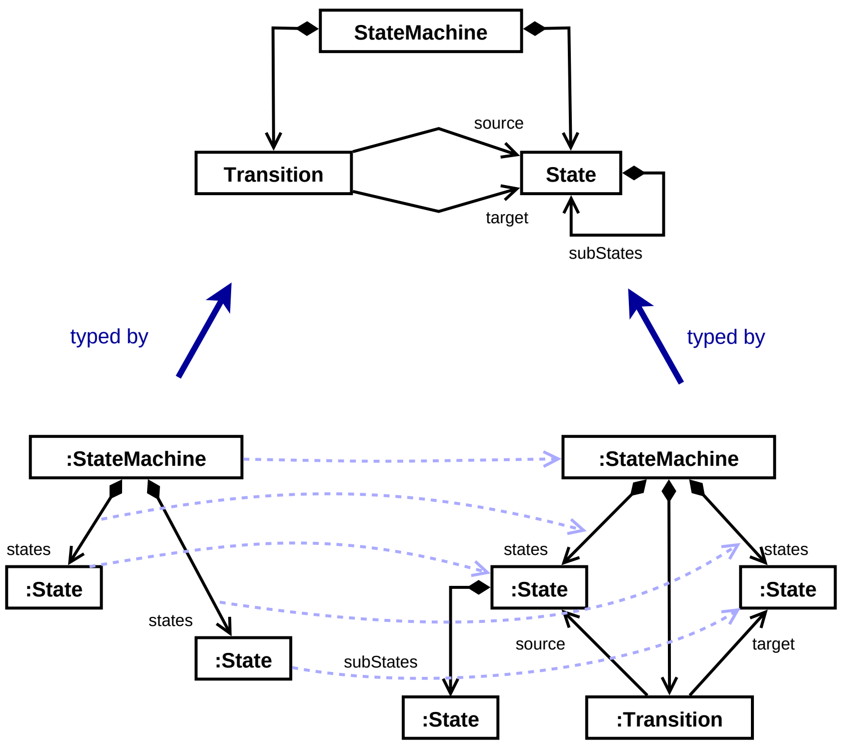

Figure 1: A model graph (top), two instance graphs (bottom) and a typed graph homomorphism.

of plain graphs are constructed as in **Set** (componentwise for nodes and edges). Given a span of graph homomorphisms $(G_1 \leftarrow G_0 \rightarrow G_2)$, the pushout result can be constructed by gluing the graphs G_1, G_2 over the interface graph G_0. Transformation rules in the DPO approach are spans of injective graph homomorphisms. Applying such a transformation rule consists of two steps. At first, a pushout complement graph and then a usual pushout graph has to be constructed. This construction is now considered for graphs with containment edges.

Containment edges can be seen as a conservative extension of plain graphs. A pushout of graphs with containment edges must also be a valid pushout of plain graphs, forgetting the containment properties of the edges. Given a span of valid graphs with containment edges, the question arises whether the pushout result as constructed for plain graphs also is a valid graph with containment edges. The pushout morphisms must be valid homomorphisms for graphs with containment edges respectively (preserve the containment edges).

As shown in Figure 2, the result graph does not necessarily have the designated order properties. Although graphs G_0, G_1 and G_2 are valid graphs with containment edges, the *'at most one container for each object'*-property in the pushout result G_3 is violated. The second example in Figure 3 shows, that there might also appear a containment cycle in the pushout result. This gives rise to state a condition under which a pushout in the category **Graphs** is also a valid graph with containment edges, i.e. has no containment cycles and each node has at most one container. Therefore, we introduce first the notions of so-called *newly contained points* and *cyclic contained points*.

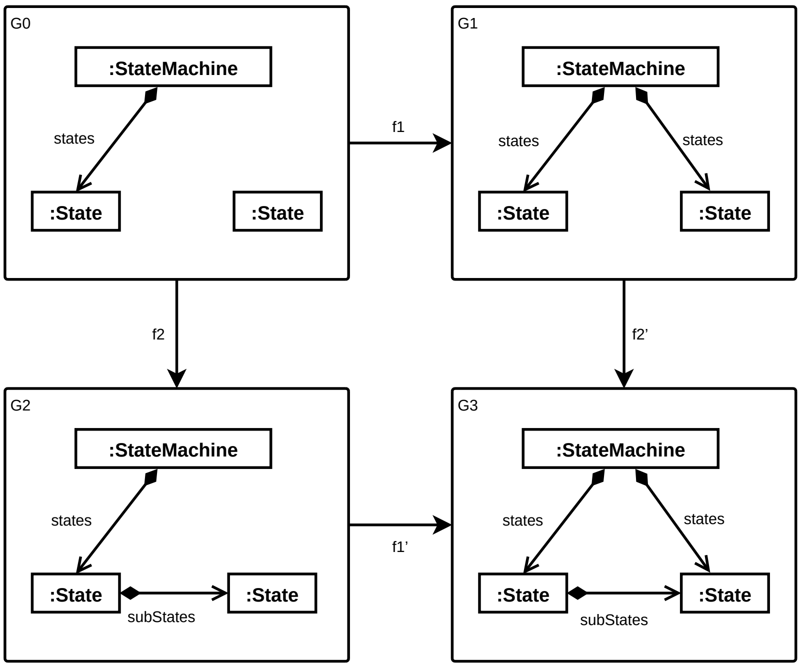

Figure 2: Invalid pushout of graphs with containment edges. A State node in the result graph G_3 has two containers.

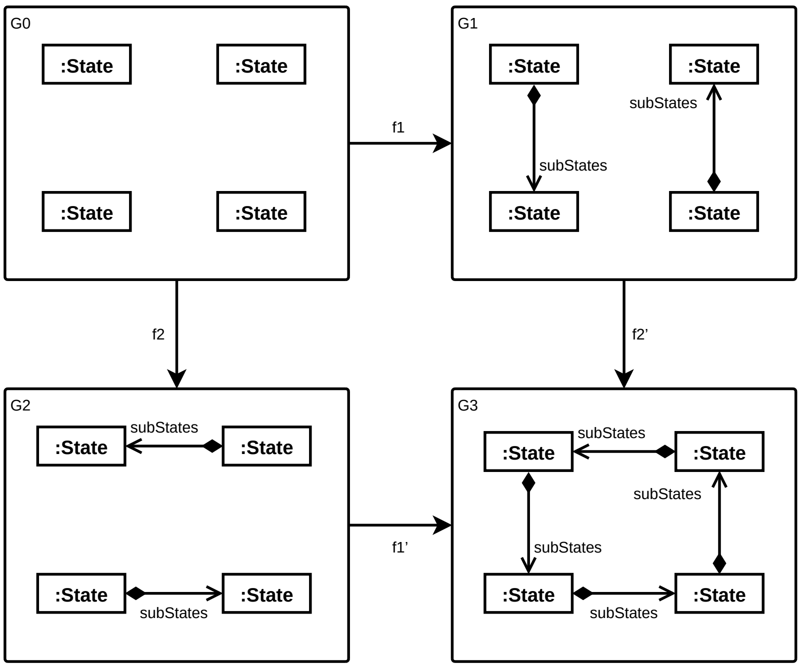

Figure 3: Invalid pushout of graphs with containment edges The pushout result G_3 has a containment cycle.

Definition 3 (Newly contained points) For a homomorphism between graphs with containment edges $h : G_0 \rightarrow G_1$, the set of *newly contained points* $NCP_h \subseteq V_0$ is defined as

- $NCP_h = \{x \in V_0 \mid \forall y \in V_0 : \ (y,x) \notin contains_0 \ \wedge \ \exists z \in V_1 : \ (z, h_V(x)) \in contains_1\}$[3]

Definition 4 (Cyclic contained points) For a span of homomorphisms between graphs with containment edges $(G_1 \xleftarrow{f_1} G_0 \xrightarrow{f_2} G_2)$, the set of *cyclic contained points* $CCP_{f_1,f_2} \subseteq V_1 \cup V_2$ is defined as

- $CCP_{f_1,f_2} = \{x \in V_1 \cup V_2 \mid ([x]_\equiv, [x]_\equiv) \in \ t(R) \ \}$

where t(R) is the transitive completion of $R = \{ \ ([x_1]_\equiv, [x_2]_\equiv) \mid (x_1, x_2) \in \ contains_1 \ \cup \ contains_2 \ \}$ and $[x]_\equiv$ denotes the equivalence class of an $x \ \in V_1 \cup V_2$ w.r.t. the equivalence relation $\equiv \, = t(s(r(\sim)))$, with $\sim \, = \ \{(f_1(x_0), f_2(x_0)) \mid x_0 \ \in V_0\}$.

Given a homomorphism $h : G_0 \rightarrow G_1$ of graphs with containments, NCP_h is the set of nodes in G_0 that do not have a container in G_0, but do have one in G_1. The definition of the set of cyclic contained points CCP_{f_1,f_2} is more complex. It is based on the equivalence relation $\equiv$ that states which nodes of G_1 and G_2 have a common source in the interface graph G_0 and which are therefore glued together in the pushout result G_3. The relation $t(R)$ is the transitive closure of the containment relations of G_1 and G_2, based on the equivalence classes induced by $\equiv$. Cyclic contained points are those nodes for which this relation is reflexive, i.e. $([x]_\equiv, [x]_\equiv) \in \ t(R)$. These nodes must form a containment cycle in the pushout result G_3 by construction.

These two definitions of *newly contained points* and *cyclic contained points* induce the condition under which a pushout of graphs with containment edges exists.

Definition 5 (Containment condition) Given a span of graphs with containment edges $(G_1 \xleftarrow{f_1} G_0 \xrightarrow{f_2} G_2)$, where f_1 and f_2 are valid homomorphisms for graphs with containment edges, the containment condition is stated as:

- $NCP_{f_1} \cap NCP_{f_2} = \emptyset$ and
- $CCP_{f_1,f_2} = \emptyset$

If the newly contained points of f_1 and f_2 are disjoint, as stated in the first condition, each node in the result graph has at most one container. For ensuring that there is no containment cyclic in G_3 the set of cyclic contained points must be empty.

Theorem 1 (Pushouts of graphs with containment edges) *Given a span of homomorphisms between graphs with containment edges $(G_1 \xleftarrow{f_1} G_0 \xrightarrow{f_2} G_2)$, the pushout result in the category of graphs $(G_1 \xrightarrow{f_2'} G_3 \xleftarrow{f_1'} G_2)$ forms also a valid pushout in* ***CGraphs****, if and only if the containment condition holds for the span $(G_1 \xleftarrow{f_1} G_0 \xrightarrow{f_2} G_2)$.*

[3] $contains_0$ and $contains_1$ are the induced containment relations of graphs G_0 and G_1 .

Proof. **At most one container.** Let $a \in NCP_{f_1} \cap NCP_{f_2}$, then:

- $f_1(a), f_2(a)$ have incoming containment edges $e_1 \in C_1, e_2 \in C_2$ that have no origin in G_0.
- $b := f'_2 \circ f_1(a) = f'_1 \circ f_2(a) \in V_3$. The images of a are glued together by construction.
- b has two incoming containment edges $f'_1(e_2) \neq f'_2(e_1)$.

Conversely, consider $b \in V_3$ with two incoming containment edges:

- Without loss of generality, the containment edges can be written as $f'_2(e_1)$ and $f'_1(e_2)$ with $e_1 \in C_1$ and $e_2 \in C_2$. If both had an origin in G_1 for instance, G_1 would already violate the containment properties.
- There is $a \in V_0$ with $f'_2 \circ f_1(a) = b = f'_1 \circ f_2(a)$. The node b must have origins in V_1, V_2 and finally $a \in V_0$ because there must be target nodes of e_1 and e_2 and these must have been glued together to the node $b \in V_3$.
- Node a has no incoming containment edge. Otherwise this edge would have images in G_1 and G_2 and there would be only one incoming containment edge for b in G_3. Therefore, a is a is a newly contained point, both of f_1 and f_2.

No cycles. The idea of the proof is the construction of a sequence of nodes $a_0, ..., a_n$ in $G_1 \cup G_2$ that corresponds to a containment cycle in the pushout graph G_3. Every cycle in G_3 consists of edges from G_1 and G_2. Since G_1 and G_2 are cycle free, the cycle in G_3 emerges from gluing of edges by identifying certain nodes. Nodes from $G_1 \cup G_2$ are identified, iff they are in the same equivalence class w.r.t. equivalence relation $\equiv$. So every containment cycle in G_3 corresponds to a sequence $a_0, .., a_n$ s.t. there is either a containment edge from a_i to a_{i+1} or a_i and a_{i+1} are identified in G_3. Furthermore a_0 and a_n are identified in G_3.

If there is a $x_3 \in V_3 : x_3 \; contains_3 \; x_3$, then exist $a_0, ..., a_n \in V_1 \cup V_2$ with:

- $i \in [0, n] : \exists \, (a_i, a_{i+1}) \in contains_1 \cup contains_2$. If there is a path of containment edges in G_3, the path consists of containment edges that have preimages in $E_1 \cup E_2$. We write this path as list of $a_i \in V_1 \cup V_2$. At least one of the a_i is container for a_{i+1}, since every containment path consists of at least one edge.
- $(a_i, a_{i+1}) \notin contains_1 \cup contains_2 \Rightarrow [a_i]_\equiv = [a_{i+1}]_\equiv$. If there is no containment edge from a_i to a_{i+1}, both are in the same equivalence class w.r.t. the pushout construction, i.e. they will be identified in G_3. These are the points where the subpaths from G_1 and G_2 are glued to the containment path in G_3.
- $[a_0]_\equiv = [a_n]_\equiv$. Without loss of generality, the path begins and ends at nodes which are glued.
- $f'_{1/2}(a_0) = f'_{1/2}(a_n) = x_3$, with $f'_{1/2}(a_i) = f'_1(a_i)$ if $a_i \in V_1$ and $f'_{1/2}(a_i) = f'_2(a_i)$ if $a_i \notin V_1$.

So in the transitive completion t(R) exists a tuple (r_1, r_2), $r_1 = [a_0]$, $r_2 = [a_n]$ with $r_1 = r_2$.

Conversely, if there is a pair $(r_1, r_2) \in t(R)$ with $r_1 = r_2$, there exist $a_0, \ldots, a_n \in V_1 \cup V_2$ with

- $i \in [0, n] : \exists\, (a_i, a_{i+1}) \in contains_1 \cup contains_2$. $(r_1, r_2) \in t(R)$ implies a containment path consisting of at least one containment edge of $E_1 \cup E_2$.
- $(a_i, a_{i+1}) \notin contains_1 \cup contains_2 \Rightarrow [a_i]_\equiv = [a_{i+1}]_\equiv$. If a_i does not contain a_{i+1}, they denote the points where the subpaths from G_1 and G_2 are glued in G_3.
- $[a_0]_\equiv = r_1 = r_2 = [a_n]_\equiv$.

So there is at least one containment edge in G_3 s.t.:

- $(f'_{1/2}(a_i), f'_{1/2}(a_{i+1})) \in contains_3$
- $(f'_{1/2}(a_i), f'_{1/2}(a_{i+1})) \notin contains_3 \Rightarrow f'_{1/2}(a_i) = f'_{1/2}(a_{i+1})$
- $f'_{1/2}(a_0) = f'_{1/2}(a_n)$

So there is a containment cycle in G_3: $(f'_{1/2}(a_0), f'_{1/2}(a_n)) \in contains_3$. $\square$

Using the DPO approach to graph transformation, we can state the conditions under which a match is valid for a given rule now. That means the result has a valid graph structure (no dangling edges) and all containment constraints are fulfilled. The first constraint is ensured by checking the *gluing condition* [EEPT06] before constructing the pushout complement. When constructing the pushout complement, edges and nodes may be deleted from the input graph. Containment constraints cannot get violated in this phase, because containment cycles or multiple containers for an object can only appear if containment edges are *added* to a valid model instance. After computing the pushout complement, the containment condition must be verified to construct the actual transformation result through a pushout of graphs with containment edges.

4 Related Work

Graphs with additional order structure are also discussed in [DSLO04], where the nodes and edges are labelled using partially ordered sets. However, this partial order is an additional feature and not induced by the graph structure. Therefore, one injective morphism in a span is already enough for ensuring the existence of a pushout in the category of these so called *Poset labelled graphs*. Moreover, these graphs are applied in the area of architectural design of software components.

Further, graphs with additional containment relations are also considered in the context of hierarchical graph transformation (see e.g. [BKK05]). Similarly, each graph item may have at most one container and the containment relation has to be acyclic. But in contrast to our approach, each node and edge (except of the root) must belong to a container in most kinds of hierarchical graphs. Thus, a typical hierarchical graph forms a special case of our graphs with containment edges which does not require that all graph items are contained and that a unique root does exist. For graphs with containment relations as well as for hierarchical graphs, graph transformation has to be defined accordingly to the kind of graphs considered. That means the pushout construction being the basic building block of algebraic graph transformation, is defined such that the result graph is an object of the corresponding category.

So called *ownership types* are an object oriented model for containment. An extensive introduction to ownership types can be found in [Cla01]. This approach is based on an extension of Abadi and Cardelli's object calculus with subtyping. A so called *containment invariant* is introduced to prove the soundness of ownership types systems.

5 Conclusion and Future Work

The extension of the graph-based approach to model transformation by graphs with containment edges better reflects transformations of MOF, EMF and UML 2 models. The containment condition stated above ensures that a transformation rule can only be applied to a model instance, if the result does not violate any containment constraints.

In the case of EMF, the implementations generated from a model ensure at run-time that these containment constraints are always satisfied. If EMF detects a violation of these constraints at a certain point, it deletes containment edges that produce the problem. This behavior breaks the formal semantics that was achieved through the graph transformation approach. Therefore, it is important to first check whether the result of a transformation step would be valid. Only if this is ensured, the rule can actually be applied. By that, we can avoid the problems of invalid model instances, ensuring formal transformation semantics.

Even though it can be argued, that this approach is conservative (since it restricts the possible applications of a transformation rule), it has the advantage of avoiding situations where an invalid model instance has to be repaired, by deleting edges for instance. Especially, the naive strategy of applying a transformation first, without any restrictions and repairing the model in the end might lead to unexpected results. Moreover, it is not obvious how to resolve problems like containment cycles in a canonical way.

The containment condition can be used in model transformation frameworks to ensure well-defined transformation semantics. The gluing condition together with the containment condition are the basis for semantical analysis of MOF/EMF model transformation, e.g. termination and confluence (critical pair analysis) [EEPT06].

Further work should lead to notions of consistency of model transformations which might not only be limited to basic model constraints, but also high-level properties of models and model transformations formulated in OCL for instance. A translation of OCL constraints into the language of graph transformation has been started in [WTEK06]. Consistency checks are probably going to play an important role in the field of model driven engineering in the future. A comprehensive theoretical foundation can lead to an improved tool support to check the quality of models and model transformations.

Bibliography

[BKK05] G. Busatto, H.-J. Kreowski, S. Kuske. Abstract Hierarchical Graph Transformation. *Mathematical Structures in Computer Science 15(4), pp. 773-819.*, 2005.

[BRST05] J. Bezivin, B. Rumpe, A. Schürr, L. Tratt. Model Transformation in Practice Workshop Announcement. 2005.
http://sosym.dcs.kcl.ac.uk/events/mtip/

[CH03] K. Czarnecki, S. Helsen. Classification of Model Transformation Approaches. *OOPSLA 03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

[Cla01] D. Clarke. Object Ownership and Containment. *PhD thesis, School of Computer Science and Engineering University of New South Wales*, 2001.
http://citeseer.ist.psu.edu/468103.html

[DSLO04] M. Denford, A. Solomon, J. Leaney, T. O'Neill. Architectural Abstraction as Transformation of Poset Labelled Graphs. *Journal of Universal Computer Science, vol. 10, no. 10, 1408-1428*, 2004.
http://www.jucs.org/jucs_10_10/architectural_abstraction_as_transformation/Denford_M.pdf

[EEL$^+$05] H. Ehrig, K. Ehrig, J. de Lara, G. Taentzer, D. Varró, S. Varró-Gyapay. Termination Criteria for Model Transformation. *Proc. Fundamental Approaches to Software Engineering (FASE), Lecture Notes in Computer Science, ISSN 0302-9743, Springer Verlag*, 2005.
http://www.springerlink.com/content/dkpvlnfgrn3k8xp7/

[EEPT05] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. Formal Integration of Inheritance with Typed Attributed Graph Transformation for Efficient VL Definition and Model Manipulation. *Proc. of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), IEEE Computer Society*, 2005.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation, EATCS Monographs, ISBN 3-540-31187-4, Springer Verlag*. 2006.
http://www.springer.com/3-540-31187-4

[EMF] Eclipse Modeling Framework (EMF) Homepage.
http://www.eclipse.of/emf

[EMT] Eclipse Model Transformation (EMT) Project Homepage at TU-Berlin.
http://tfs.cs.tu-berlin.de/emftrans

[Koe06] C. Koehler. A Visual Model Transformation Environment for the Eclipse Modeling Framework. Diploma thesis, Technical University of Berlin. 2006.
http://tfs.cs.tu-berlin.de/emftrans/papers/06-ChristianKoehler.pdf

[KS06] A. Königs, A. Schürr. Tool Integration with Triple Graph Grammars - A Survey. *Electronic Notes in Theoretical Computer Science 148, 113-150*, 2006.

[KSW04] J. M. Kuester, S. Sendall, M. Wahler. Comparing Two Model Transformation Approaches. *Proc. of OCL MDE*, 2004.

http://www.cs.kent.ac.uk/projects/ocl/oclmdewsuml04/papers/6-kuster_sendall_wahler.pdf

[OMG06] Object Management Group (OMG). Meta Object Facility (MOF) Core Specification Version 2.0. 2006.
http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF

[TEB+06a] G. Taentzer, K. Ehrig, E. Biermann, G. Kuhns, E. Weiss, C. Koehler. Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. *Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science 4199, Springer Verlag*, 2006.
http://www.springerlink.com/content/t681013811w30537/

[TEB+06b] G. Taentzer, K. Ehrig, E. Biermann, G. Kuhns, E. Weiss, C. Koehler. EMF Model Refactoring based of Graph Transformation Concepts. *To appear in the Electronic Communications of the EASST*, 2006.

[TEG+06] G. Taentzer, K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovszky, U. Prange, D. Varró, S. Varró-Gyapay. Model Transformation by Graph Transformation: A Comapartive Study. *Model Transformations in Practice Workshop, MoDELS 2005*, 2006.
http://sosym.dcs.kcl.ac.uk/events/mtip05/submissions/

[TR05] G. Taentzer, A. Rensink. Ensuring Structural Constraints in Graph-Based Models with Type Inheritance. *Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science 0302-9743, Springer Verlag*, 2005.
http://www.springerlink.com/content/98tey0jerhy7pryn

[WTEK06] J. Winkelmann, G. Taentzer, K. Ehrig, J. M. Küster. Translation of Restricted OCL Constraints into Graph Constraints for Generating Meta Model Instances by Graph Grammars. *To appear in GT-VMT 2006, Electronic Notes in Theoretical Computer Science*, 2006.