Electronic Communications of the EASST
Volume 42 (2011)

Proceedings of the 4th International Workshop on Multi-Paradigm Modeling (MPM 2010)

Towards Transformation Rule Composition

Mark Asztalos, Eugene Syriani, Manuel Wimmer and Marouane Kessentini

13 pages

Guest Editors: Vasco Amaral, Hans Vangheluwe, Cécile Hardebolle, Lazlo Lengyel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Towards Transformation Rule Composition

Mark Asztalos[1], Eugene Syriani[2], Manuel Wimmer[3] and Marouane Kessentini[4]

[1] Budapest University of Technology and Economics, Budapest 1111, Hungary
asztalos@aut.bme.hu
[2] McGill University, Montréal, Québec, Canada H3A 2A7
esyria@cs.mcgill.ca
[3] Vienna University of Technology, 1040 Wien, Austria
wimmer@big.tuwien.ac.at
[4] DIRO, Université de Montréal, Montréal, Québec, Canada H3T 1J4
kessentm@iro.umontreal.ca

Abstract: Many model transformation problems require different intermediate transformation steps. For example, platform-specific models (PSM) are often generated from platform-independent models (PIM) by chains of model transformations. This requires the presence of several intermediate meta-models between those of the PIM and the PSM. Thus, most of the effort is needed to define a transformation mechanism for each intermediate step. The solution proposed in this paper is to investigate whether it is possible to generate a single transformation from a chain of transformations, solely involving the initial PIM and final PSM meta-models. The presented work focuses on the composition of transformations at the rule level. We apply the automatic procedure for composing rules in the context of the evolution of the Enterprise Java Beans (EJB) language, transforming UML models into EJB 2.0 models and then to EJB 3.0 models.

Keywords: rule composition, transformation chain, transitive transformation

1 Introduction

Nowadays, software platforms evolve very rapidly. This is also true for modelling languages, which have to reflect the evolution of the underlying platforms. The evolution of a modelling language requires one to adapt its meta-model as well as any model transformation involving it. The task of adapting the transformations to the new version of the language can be very tedious and error prone, especially when this is done manually. Let us take the example scenario of generating platform-specific models (PSMs) from platform-independent models (PIMs). Due to the continuous evolution of the platform, while several versions of the platform-specific meta-model have to be employed, transformations between these meta-model versions are necessary for migrating the PSMs at version n to PSMs at version $n+1$. These transformations can be also reused within a model transformation chain for transforming a PIM over several intermediate meta-models into a PSM for the latest platform version. Over time, such transformation chains naturally become larger and larger, which has a negative impact on maintainability and execution performance.

The goal of this paper is to reduce the manual effort of shortening transformation chains by eliminating intermediate transformation steps. The presented work proposes to compose a chain of transformations into one transformation that does not involve any intermediary meta-model. In particular, this is done by computing the transitive transformation of two given transformations.

In Section 2, we first define the composition of transformations in general. Section 3 reduces the problem to the composition of rules by (1) elaborating on the criteria for composing graph transformation rules and (2) presenting an automatic procedure to compose such rules into one. In Section 4, we illustrate this approach in the context of the evolution of the Enterprise Java Beans (EJB) language, transforming UML models into EJB 2.0 models and then to EJB 3.0 models. Section 5 is dedicated to the related work and we conclude in Section 6.

2 Transformation Composition

In this section, we define a composition operator to precisely specify the meaning of a transformation composition. This operation is applied in the context of a chain of model transformations as defined below.

Definition 1 (Transformation chain)**.** Let $\mathbf{T_n} = \langle T_1, T_2, \ldots, T_n \rangle_{n \in \mathbb{N}}$ be an ordered sequence of transformations where each T_i defines a mapping from a meta-model $\mathfrak{M}_i$ to a different meta-model $\mathfrak{M}_{i+1}$. We denote such a *transformation chain* as $\mathfrak{M}_1 \xrightarrow{T_1} \mathfrak{M}_2 \xrightarrow{T_2} \mathfrak{M}_3 \xrightarrow{T_3} \ldots \xrightarrow{T_n} \mathfrak{M}_{n+1}$. Note that we enforce that all the meta-models involved in the chain $\mathbf{T_n}$ be different from one another, *i.e.,* each transformation must be *exogenous* [MV06].

Using the previous notation, we call $\mathfrak{M}_i$ the domain of T_i and $\mathfrak{M}_{i+1}$ its co-domain. The transformation is applied on a model m_i conforming to its meta-model $\mathfrak{M}_i$ and results in a new model $m_{i+1} = T_i(m_i)$ conforming to its meta-model $\mathfrak{M}_{i+1}$. Note that transformations, transformation rules, as well as the pre- and post-condition patterns of the rules are also considered as models conforming to their respective meta-models [KMS+10].

The presented approach assumes that each transformation in the chain is specified using algebraic graph transformation rules. The models involved are represented as graph objects in the category of typed attributed graphs as defined in [EEPT06]. In the remainder of the paper, a model m and its *element graph* G will be used interchangeably. The typed attributed graph G consists of a set of nodes $V(G)$ and edges $E(G)$, where each node conforms to a specific node type in a type graph (representing $\mathfrak{M}$, the meta-model of m) and can hold attribute values. We however require that graph edges be partitioned in two sets $E(G) = E_m(G) \cup \Lambda(G)$, distinguishing *trace edges* $\Lambda(G)$ from the edges $E_m(G)$ conforming to those defined in the type graph. A trace edge represents a traceability link connecting any two nodes regardless of their type. While a transformation is applied, traceability links are created such that any newly created element must have at least a traceability link[1].

Definition 2 (Transformation composition)**.** Let T_1 and T_2 be two consecutive transformations in a transformation chain such that $\mathfrak{M}_1 \xrightarrow{T_1} \mathfrak{M}_2 \xrightarrow{T_2} \mathfrak{M}_3$. We denote $T' = T_2 \bullet T_1$ the *composed*

[1] Traceability links can be created implicitly such as in [JK06]. Otherwise, their creation must be explicitly specified in the rules.

transformation of T_1 with T_2, following the composition operator $\bullet$ which satisfies the *sequence*, *elimination*, and *transitivity* criteria as defined below.

We describe the application criteria of the composition operator given an arbitrary input model m_1 for T_1, $m_2 = T_1(m_1)$, and $m_3 = T_2(m_2)$, where m_1, m_2, and m_3 conform to $\mathfrak{M}_1, \mathfrak{M}_2$, and $\mathfrak{M}_3$ respectively. We denote $m' = T_2 \bullet T_1(m_1)$ be the resulting model after the composition. In the case where traceability links are created explicitly in the rules, $\hat{m}$ represents the graph model isomorphic to m without any trace edge.

Sequence There shall exist three injective graph morphisms $(seq_i)_{i=(1,2,3)}$ that must be defined as: $seq_1 : m_1 \to m'$, $seq_2 : \hat{m_3} - \hat{m_2} - \hat{m_1} \to m'$, and $seq_3 : \hat{m'} \to m_3$. seq_1 ensures that the input model is preserved. seq_2 ensures that all the elements from $\mathfrak{M}_3$ produced by T_2 are present in m'. seq_3 ensures that m' contains no other elements than those found in m_3.

Elimination There should not be any morphism $elem : m_2 - \hat{m_1} \to m'$. That is, m' shall not contain any occurrence of an element from $\mathfrak{M}_2$. Moreover, no traceability links involving elements from $\mathfrak{M}_2$ shall be present.

Transitivity We denote by λ_{ij} a traceability link (trace edge) between an element from m_i and an element from m_j. The following predicate must hold: $\exists \lambda_{12} \in \Lambda(m_3) \wedge \exists \lambda_{23} \in \Lambda(m_3) \Rightarrow \exists \lambda_{13} \in \Lambda(m')$. This ensures the transitive closure of traceability links, *i.e.,* for any instance element of $\mathfrak{M}_2$ in m_3, if it is connected through trace edges to both an instance element of $\mathfrak{M}_1$ *and* an instance element of $\mathfrak{M}_3$, then m' must have a trace edge between the latter two instance elements.

The sequence criterion ensures soundness and completeness of the composition operator. The elimination criterion ensures that the resulting transformation is independent from any intermediate meta-model. Finally, the transitivity criterion ensures that traceability links correctly map the source and target model elements of the composed transformation T'.

The following generalizes the transformation composition definition to an arbitrary number of transformations.

Definition 3 (Transformation chain composition)**.** Given the chain $\mathbf{T_n} = \langle T_1, T_2, \ldots, T_n \rangle_{n \in \mathbb{N}}$, the composed transformation of $\mathbf{T_n}$ is a transformation $T' = T_n \bullet (T_{n-1} \bullet \ldots (T_3 \bullet (T_2 \bullet T_1)) \ldots)$. This can be written in short $T' = T_n \bullet T_{n-1} \bullet \ldots \bullet T_3 \bullet T_2 \bullet T_1$.

3 Rule Composition

The task of composing two arbitrary transformations is a very complex problem. That is because the choice of which rule from one transformation to compose with a rule from the other transformation often depends on the domain of application. For the scope of this paper, we concentrate on applying the composition operation on two graph transformation rules. In this section, we provide a procedure for composing two individual rules into a single one such that the sequence, elimination, and transitivity criteria are satisfied.

3.1 Criteria for Rule Composability

In the following, we assume that rewriting rules or productions are defined as presented in [EEPT06]. This means that a rule $p = (L \leftarrow K \rightarrow R)$ consists of three objects in the category of typed attributed graphs: the left hand side (L), the interface K, and the right hand side (R) objects respectively. In this paper, we assume that each transformation transforms an instance of one metamodel into an instance of another, therefore, the objects L, K, and R may contain elements from both the source and the target metamodel of the current transformation.

To apply the composition operator on two individual rules, we assume that each of the transformations involved consists of a single rule for sake of completeness: $T_1 = \{r_1\}$ and $T_2 = \{r_2\}$. The procedure assumes that the rules r_1 and r_2 are monotonically increasing, *i.e.,* they can only create new elements and/or modify attribute values. Moreover, all traceability links created during the application of T_1 and T_2 shall be preserved. The output of the composition procedure is a new transformation $T_3 = T_2 \bullet T_1 = \{r_2\} \bullet \{r_1\} = \{r_3\}$ consisting of a single rule. The following proposition specifies the necessary condition for the composition procedure to satisfy Definition 2.

Proposition 1 (Composability condition)**.** Two rules $r_1 = L_1 \leftarrow K_1 \rightarrow R_1$ and $r_2 = L_2 \leftarrow K_2 \rightarrow R_2$ satisfy the composability condition if there exists a partial morphism $n : L_2 \rightarrow R_1$ such that:

- the domain of n is a subgraph of L_2, which consists of all the elements that is from $\mathfrak{M}_2$,
- the co-domain of n is a subgraph of R_1 consisting of elements only from $\mathfrak{M}_2$,
- the mapping from the domain to the co-domain of n is a total injective morphism.

The formal definition of the traditional composition of two sequential rewriting rules is described in [EEPT06], this composition is called the E-concurrent production. The definition states that given two rules p_1 and p_2, then they can be composed into a new rule $p = (L, K, R)$. Informally, the p is composed along a new graph object E, which is produced by jointly surjective morphisms from R_1—the right-hand side (RHS) of p_1—and L_2—the left-hand side (LHS) of p_2. The application of the new rule is equal with the sequential application of the two original rules. However, there are often more than one possible compositions of the rules, because of the non-determinism of the matches.

To satisfy the elimination and transitivity criteria of Definition 2, the sub-procedure in Algorithm 1 is required:

Algorithm 1 `eliminate` (m)

1: **for all** $\lambda_{12}, \lambda_{23} \in \Lambda(m)$ **do**
2: **if** $trg(\lambda_{12}) = src(\lambda_{23})$ **then**
3: create λ_{13} such that $src(\lambda_{13}) = src(\lambda_{12})$ and $trg(\lambda_{13}) = trg(\lambda_{23})$
4: $\Lambda(m) \leftarrow \Lambda(m) \cup \{\lambda_{13}\} - \{\lambda_{12}, \lambda_{23}\}$
5: $V(m) \leftarrow V(m) - \{trg(\lambda_{12})\}$
6: **end if**
7: **end for**
8: **for all** $\lambda_{12} \in \Lambda(m)$ **do**
9: $\Lambda(m) \leftarrow \Lambda(m) - \{\lambda_{12}\}$
10: $V(m) \leftarrow V(m) - \{trg(\lambda_{12})\}$
11: **end for**

Given a model m, the elimination procedure performs two runs over the trace edges in m. In the first run (lines 1 to 7), it first looks for a trace edge λ_{12} linking an element conforming to $\mathfrak{M}_1$, say e_1, to an element conforming to $\mathfrak{M}_2$, say e_2 and another trace edge λ_{23} linking e_2 to an element conforming to $\mathfrak{M}_3$, say e_3. It then creates the transitive trace edge λ_{13}, removes the two other traceability edges as well as e_2. In the second run, the elimination procedure looks for all remaining trace links involving $\mathfrak{M}_1$ and $\mathfrak{M}_2$ elements and removes them from m. Note that there cannot be any trace edge in the form λ_{23} remaining after the first run, since any element from $\mathfrak{M}_2$ must be linked to an element from $\mathfrak{M}_1$ by construction. Therefore after the elimination procedure terminates, the only remaining trace edges in m link elements from $\mathfrak{M}_1$ to elements from $\mathfrak{M}_3$.

3.2 Composition Procedure

Let $r_1 = L_1 \leftarrow K_1 \rightarrow R_1$ and $r_2 = L_2 \leftarrow K_2 \rightarrow R_2$ be two rules that satisfy the *composability condition* of Proposition 1. We want to produce the composite rule r_3 such that $\{r_3\} = \{r_2\} \bullet \{r_1\}$ as defined in Section 2.

Algorithm 2 `compose` (r_1, r_2)

1: compute the E-based composition (L_3, K_3, R_3) of r_1 and r_2 such that $E = R_1$
2: $K_3 \leftarrow R_1$
3: $L_3 \leftarrow$ `eliminate` (L_1)
4: $R \leftarrow \phi$
5: $r_2' \leftarrow r_2$ extended with R_2 as a NAC, if not present
6: **repeat**
7: $\quad R_3 \leftarrow$ apply r_2' exhaustively on E
8: $\quad$ `eliminate` (R_3)
9: $\quad R \leftarrow R \cup \{(L_3, K_3, R_3)\}$
10: **until** all application sequences of r_2' have been exhausted on E
11: **return** R

Algorithm 2 produces the set of all possible compositions of r_1 and r_2. r_2 is extended with a negative application condition (NAC) corresponding to its RHS. This ensures that r_2 is only applied once on every match found in E. It is worth noting that there can be different R_3's even if r_2' is applied exhaustively on E, if the order of application affects the result.

Before analyzing the algorithm, we demonstrate its operation on the composition of two simple rules R_1 and R_2 presented in Figure 1. Let R_2' be rule R_2 extended by the NAC which consists of the RHS of R_2. By Algorithm 2, the E graph is RHS of R_1. If we modify R_1 by applying R_2' once on its RHS, we get rule R. However, we apply R_2' exhaustively, therefore, RHS of R is modified again, which results in rule R'. Note that there are no other possible matches, because of the NAC in R_2'. The next step is the application of the elimination algorithm, which performs the transitive closure on the trace edges. RHS of R' is eliminated, which results in rule R''.

The lemmas below validate the composition procedure. Lemma 1 ensures that the procedure will output all possible composed rules r_3 and Lemma 2 ensures its correctness.

Lemma 1. If r_1 and r_2 satisfy the *composability condition*, then `compose` (r_1, r_2) outputs all compositions of r_1 and r_2 such that the exhaustive application of `compose` (r_1, r_2) is equivalent to the composition of r_1 and r_2 using the composition operator of Definition 2.

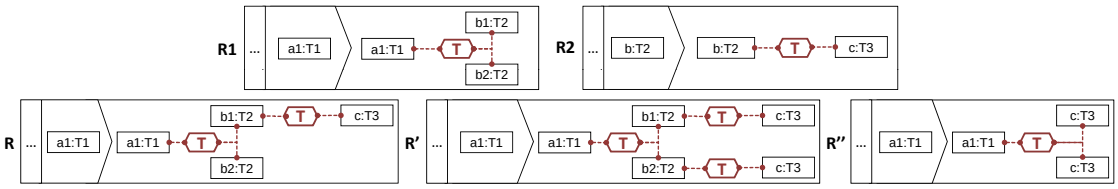

Figure 1: Example for rule composition.

Proof. Assume that there is a possible E-based composition $r = L \leftarrow K \rightarrow R$ such that the E-graph $E \neq R_1$. This implies that $\exists e \in E : e \notin R_1$ where e can be any type of element in the graph. E is produced by jointly surjective morphisms from R_1 and L_2; thus $e \in L_2$. Moreover, e is an element conforming to $\mathfrak{M}_2$ as it is the domain of r_2. However $e \notin R_1$, which implies that $e \in L$, according to the definition of the E-concurrent production. But L cannot contain elements from $\mathfrak{M}_2$ because if it did, the input model would contain elements conforming to $\mathfrak{M}_2$, which is a contradiction.

□

Lemma 2. The result of the composition procedure $\{r_3\} = \{r_2\} \bullet \{r_1\}$ satisfies Definition 2.

Proof. Assume that a model m_1 is processed by the transformations T_1 and T_2 through a possible traditional E-based composition r'_3 of the rules r_1 and r_2. Let r_3 be a rule computed by applying the elimination procedure on the LHS, RHS, and interface graph of r'_3. Let $T_3 = \{r_3\}$, $m_2 = T_1(m_1)$, $m_3 = T_2(m_2)$, and $m' = T_3(m_1)$. We shall now prove that T_3 satisfies the sequence, elimination, and transitivity criteria.

- *Sequence Criterion:* $\exists seq_1 : m_1 \rightarrow m'$, because $L_3 = K_3$ and hence the input model m_1 is not modified. $\exists seq_2 : \hat{m_3} - \hat{m_2} - \hat{m_1} \rightarrow m'$ as no elements from r'_3 have been deleted during the elimination that was performed to produce r_3. Moreover, $\exists seq_3 : \hat{m'} \rightarrow m_3$ since R_3 contains elements conforming to $\mathfrak{M}_3$ because of the exhaustive application of r'_2.
- *Elimination Criterion:* m' does not contain any element from $\mathfrak{M}_2$ since applying the elimination procedure on r'_3 ensures that all elements from the intermediate meta-model are removed from it.
- The *Transitivity Criterion* is also satisfied because the elimination procedure generates all the traceability links required by the condition.

□

When NACs come into play in r_1 or r_2, we distinguish the following case:

- If there is a NAC in r_1 and it corresponds to R_1, then we extend each composite rule r_3 with a NAC corresponding to R_3.
- If there is a NAC in r_2 and it corresponds to R_2, then it is taken into account when applying r_2 to E.
- Any other NAC is not considered in the presented procedure.

4 Application

We now apply the composition approach presented in Section 3 in the following scenario. A company has developed a transformation T_1 for transforming UML class diagrams to Enterprise Java Beans (EJB) 2.0. However, after some time, the company decided to use EJB 3.0 due to several simplifications of the new version of the standard. Thus, they developed a transformation T_2 for migrating existing EJB 2.0 models to EJB 3.0 models. However, to support the generation of new EJB 3.0 models from UML class diagrams, they would have to implement a dedicated transformation T_3, if applying the transformation chain $\langle T_1, T_2 \rangle$ is undesired. Reasons for this may be related to performance issues for ensuring rapid generation of EJB 3.0 models. Also, direct traceability between UML models and EJB 3.0 models is desired since EJB 2.0 instances would become obsolete.

4.1 Involved Artefacts

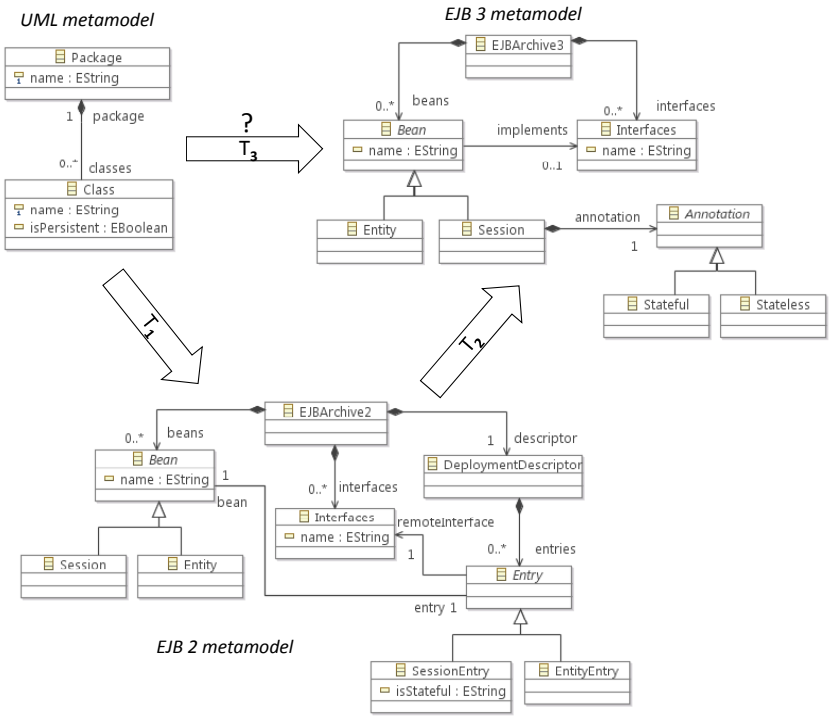

Figure 2: Meta-models of the case study.

A simplified version of the meta-models and transformation rules for this scenario are illustrated in Figures 2 and 3 respectively. T_1 transforms `Packages` into `EJBArchives` and `Classes` into either `SessionBeans` or `EntityBeans`, depending on the `isPersistent` attribute, as well as into `Interfaces`. Furthermore for each `Bean`, an `Entry` in the `DeploymentDescriptor` has to be generated. The `DeploymentDescriptor` concept is no longer used in EJB 3.0, because no additional XML configuration files for `Beans` are required. Instead, a light-weight approach for configuring `Beans` directly in the Java code through `Annotations`

is supported by EJB 3.0. Note that given the semantics of this migration, all rules of the transformations are applied exhaustively.

The transformations T_1 and T_2 have been implemented in *ATL* [JK06] and subsequently transformed into graph transformation rules based on *EMF Tiger*[2] [BET08]. To adhere to the behaviour of *ATL*, the resulting graph transformation rules have the following properties which also comply to the criteria for rule composition:

- *Matchable Elements*: *ATL* is designed as a model-to-model transformation language meaning that the target model is completely rebuilt from the source model. Thus, the only elements that can be matched by a rule are elements of the source model and elements of the target model already created by previous rule applications. The latter are only accessible via trace edges.
- *Creation and Deletion of Elements*: In *ATL* the source model is considered as read-only, thus elements of this model may not be altered. Furthermore, elements of the target model are created by executing the transformation, but once created, they can no longer be deleted by the transformation.
- *Trace Model*: For each rule execution, a trace element is generated linking all matched source elements to all generated target elements. Other transformation rules can build on this trace information, *e.g.,* for adding links to already created target elements.
- *Unique Matching*: Each transformation rule can only match once for a given set of elements. Thus, to ensure this behaviour in the graph transformation rules, each rule comprises a NAC corresponding to the RHS of the rule[3].

4.2 Composing the Transformations

We now apply the composition procedure to our example by composing the rules of T_1 with those of T_2. Since the composition procedure is applied on individual rules, we have implemented a program in Java that first detects which combinations of rules from T_1 can be composed with rules from T_2, based on Proposition 1. The iteration over the rules of T_2 follows the order shown in the upper left of Figure 3. However, this may lead to several possible valid combinations of rules. The user then selects the most appropriate combination according to his knowledge of UML class diagrams and EJB. Then, the composition procedure is applied on these two rules. The result, *i.e.,* the transformation T_3, is shown at the bottom of Figure 3.

Composing $T_2 : R_1$. $T_2 : R_1$ is composable with $T_1 : R_1$, $T_1 : R_2$, and with $T_1 : R_3$ according to the composability condition. However, due to the fact that $T_1 : R_2$ and $T_1 : R_3$ both contain a subgraph of the LHS of $T_2 : R_1$ in both their LHS and RHS, $T_1 : R_1$ seems to be more appropriate for composition. The reason is that $T_1 : R_1$ actually generates the input elements for $T_2 : R_1$ in contrast to the other two rules which only check for the existence of these elements. The composite rule $T_3 : R_1$ is constructed by composing $T_1 : R_1$ and $T_2 : R_1$ as follows. The LHS of $T_3 : R_1$ remains the same as the one for $T_1 : R_1$. Then to create the RHS of $T_3 : R_1$, the composition procedure connects an `EJBArchive3` element to the `Package` element of T_1 :

[2] Other graph transformation frameworks explicitly representing transformations as models are applicable as well.
[3] Please note that due to space limitations, the NACs are not shown in Figure 3.

R_1 via a trace edge. Then the elimination procedure removes both the `EJBArchive2` and

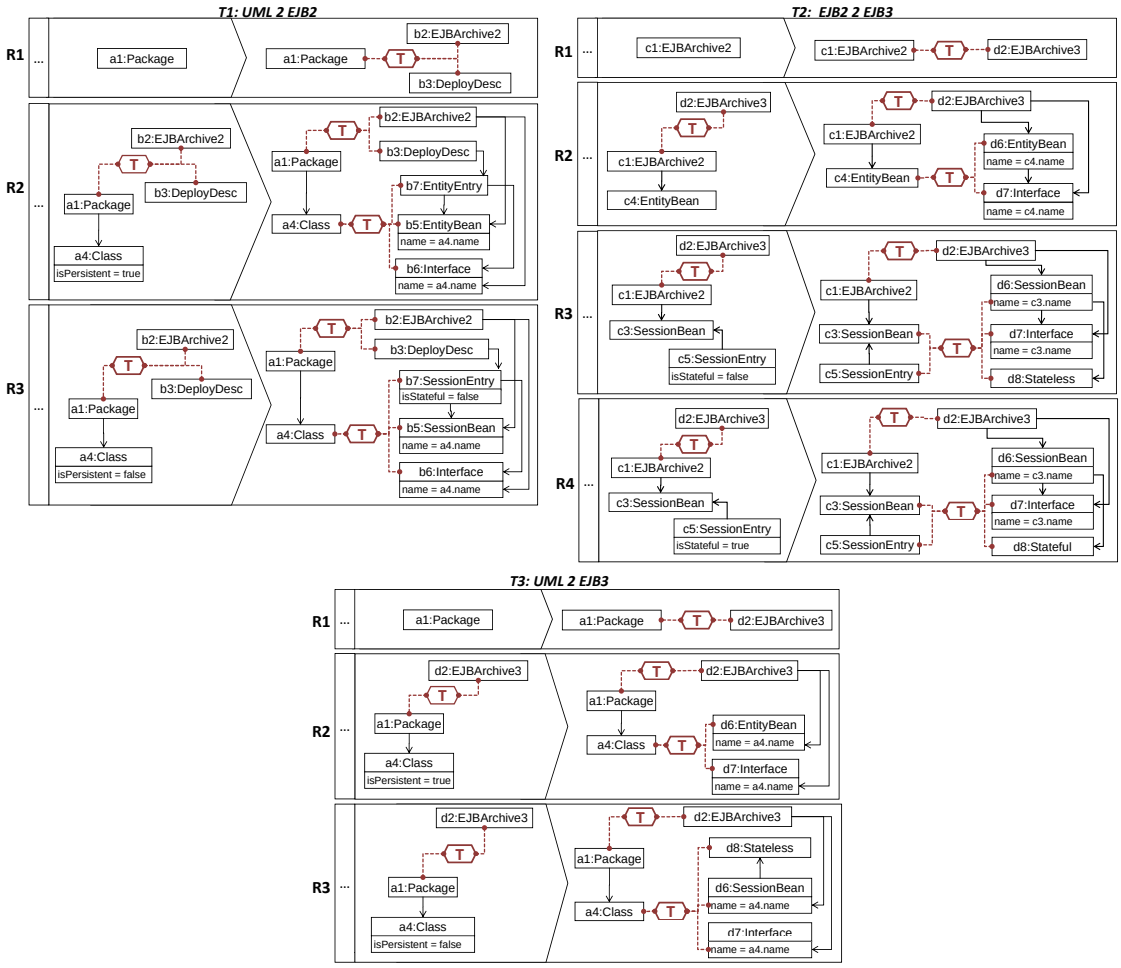

Figure 3: Transformations of the case study.

`DeployDesc` elements from the result. Finally a trace edge connecting the `Package` element to the `EJBArchive3` is created. $T_3 : R_1$ also comprises a NAC corresponding to its RHS since $T_1 : R_1$ did have a NAC corresponding to its own RHS. For computing this NAC, we are currently not using a composition procedure. Instead we just copy the elements of the RHS into the NAC to ensure the aforementioned unique rule matching.

Composing $T_2 : R_2$. $T_2 : R_2$ is only composable with $T_1 : R_2$ as it is the only rule of T_1 that has a RHS matchable by the LHS of $T_2 : R_2$. The two rules are thus composed in the same way as described in the previous case. In addition, we now have to compose not only the graph patterns but also the attribute value computations. For example, consider the assignment `name = c4.name` in element `d6:EntityBean` of the RHS of $T_2 : R_2$. It cannot be copied as is

since the assignment refers to an element of the EJB 2.0 meta-model. In this example, we only have simple value assignments without using more complex functions. For setting the attribute values in the composed transformation rule, we have to find out for each attribute value assignment in T_2, how the value is actually computed in T_1. In our example, we can easily find out that the name attribute of the element `c4` in $T_2 : R_2$ is actually calculated by using the `name` attribute value of the element `a4`. Thus, only this assignment has to be used in the composed transformation rule. Finally, the elimination procedure applied on the LHS of $T_3 : R_2$ not only deletes the `DeployDesc` element from the RHS of $T_3 : R_2$ (as in the previous case), but also from the LHS of $T_3 : R_2$.

Composing $T_2 : R_3$. $T_2 : R_3$ is only composable with $T_1 : R_3$. In this case, in addition to composing the nodes and edges of the pattern, we also consider the attribute value condition `isStateful = false` of the LHS of $T_2 : R_3$. However, the rest of the composition is analogous to the previous case.

Composing $T_2 : R_4$. $T_2 : R_4$ is not composable with any rule of T_1.

4.3 Implementation

The presented composition procedure allows to compose T_1 and T_2 nearly automatically. The transformation T_3 can be entirely produced with the help of some heuristics to further filter out meaningful composition possibilities (*e.g.,* reasoning about if a rule generates the elements or uses them only as context, as discussed in the first composition). Furthermore, some specific extensions such as attribute value assignments as well as constraints are necessary in the future to allow for a higher automation degree.

We have implemented the composition procedure on top of *EMF Tiger*. The user chooses two transformations to compose. If they are composable, the procedure outputs the composite rule. In the case where there are more than one possibility, the user can interactively select the most appropriate composition. The implementation relies on a higher-order transformation implemented in Java. The first step is the generation of templates out of the LHS of the rules from T_2. These templates are then matched against the RHS of the rules from T_1. This match model is the basis for further composition computations. In a second step, the rules of T_1 are rewritten according to the presented composition procedure. In addition, we have implemented the mentioned heuristic for filtering the composition possibilities and support simple attribute value assignments. After the composition computation has finished, the resulting transformation is serialized as T_3 expressed again as an *EMF Tiger* transformation.

5 Related Work

In this section, we outline how others have investigated in transformation composition: in graph transformation theory, in model-driven engineering, and more widely for model management in the field of data engineering.

5.1 Composition of Algebraic Graph Transformations

As mentioned in Section 3.2, a formal definition for the composition of two graph transformation rules was already proposed in [EEPT06], by creating the so-called E-concurrent rule. However, the authors do not explicitly precise how this rule is constructed. In the current paper, we propose a systematic algorithm to (1) detect if two rules are composable and (2) explicitly give the steps on how to construct the E-concurrent rule. Also, the scope of the definitions and algorithms of this paper are directly applicable in model-driven frameworks.

5.2 Composition of Model Transformations

In the latest years, the sequential composition of model transformations has been an active research field. Several approaches for modelling transformation chains [Old05, VBH+06, FABJ09, RRL+09] have been proposed. Most of them are based on UML Activity Diagrams which orchestrate several transformations to achieve a larger goal. However, none of these approaches tries to compute new transformations out of existing transformations as done in this paper.

In [PGPB08], the authors present an approach for composing rules within one transformation: the so called *internal composition*. For example, considering a transformation from UML class diagrams to Java, two rules can be composed when they both transform UML classes to Java classes with different mapping details. In [Wag08], Wagelaar presents sophisticated internal composition techniques for *ATL* and *QVT* [Obj08] in order to improve the design of model transformations. Since these approaches focus on internal composition only, they do not discuss the computation of the transitive transformation from two given transformations.

In [BHE09], the compositionality of model transformations is addressed. By compositionality the authors do not mean *sequential composition* as meant in this paper, but they are interested in the *spatial composition* when mapping a model to its semantic domain. Compositionality is guaranteed by a transformation T if the execution of T produces a set of semantic expressions (instances of the semantic domain) such that their composition represents the semantics of the whole model.

In summary, to our best knowledge no comparable approach to ours exists in the field of model-driven engineering for composing two transformations into the transitive transformation.

5.3 Composition in Model Management

In the area of data engineering, model management [BM07] has gained much interest during the last decade. Model management stands for the idea of dealing with evolution in data engineering by using models (*i.e.,* schemas and mappings between them) and operators for producing new models out of existing ones. They define schema operators, such as *diff* and *merge*, as well as mapping operators, such as *inverse* and *compose*. The goal of the *compose* operator is similar to our model transformation composition approach. However, its realization is quite different (cf. [BGMN08] and [YP05]). First, in data engineering, only relational and hierarchical schemas are considered in contrast to object-oriented meta-models, which are the basis for the composition approach of this paper. Second, in data engineering, pre-defined relational operators (*e.g., project*, *select*, and *join*) are used for describing mappings between schemas. In contrast, our approach is built on graph transformations, which is a significantly different paradigm for describing mappings between object-oriented meta-models.

6 Conclusion

In this paper, we provide a mechanism for composing individual rules from a transformation chain. This composition allows for the creation of a new transformation involving only the initial and target meta-models. Although some assumptions must be made on the syntax of rules, the composition procedure is general enough in the sense that it is independent from the input model. The presented approach is based on the syntactic composition of the rules. Extending the procedure to the transformation level requires to take into account the semantics of the chain of transformations.

The main benefits of our approach are: (1) it is possible to reduce the complexity of transformation chains by eliminating unnecessary transformation steps, (2) if there is traceability from m_1 to m_2 and from m_2 to m_3, we are able to provide traceability from m_1 to m_3, and (3) our approach seems to be perfectly suited in metamodel evolution scenarios where the target metamodel evolves. If there is already an instance migration transformation from the initial target metamodel version to the new target metamodel version, this migration transformation may be composed with the transformation between the source and the initial target metamodel in order to ensure transformation co-evolution.

As this is a first attempt on composing chains of transformations, a number of open issues still remain. In the presented example, we have only considered the core part of *ATL* which is comparable to the core of other model-to-model transformation approaches, such as *QVT-Relations* [Obj08]. In particular, we did not focus on transformations requiring an explicit rule scheduler (*e.g.,* with a control flow). Also, several other features of *ATL* should be supported, such as OCL queries and *called rules* (rules that are not automatically executed by the transformation engine but that have to be explicitly invoked in the transformation). Furthermore, our example only considers simple attribute value assignments in the rules. However, before considering more complex attribute manipulations in the composition, one should first think of how to map them to graph transformations in order to provide a theoretical basis for extending the composition procedure. Moreover, dealing with arbitrary OCL expressions when composing transformations is challenging and should certainly form a composition topic on its own. Finally, we have to provide tool support for transforming the composed transformations, expressed as graph transformations, back to ATL transformations. In this context, we have intend to migrate our current prototype to a bi-directional model transformation formalism.

Acknowledgements

We would like to thank all the participants of the 2010 Computer-Aided Multi-Paradigm Modelling workshop (CAMPaM) for their useful feedback.

Bibliography

[BET08] E. Biermann, C. Ermel, G. Taentzer. Precise Semantics of EMF Model Transformations by Graph Transformation. In *International Conference on Model Driven Engineering Languages and Systems*. LNCS 5301, pp. 53–67. Springer, 2008.

[BGMN08] P. A. Bernstein, T. J. Green, S. Melnik, A. Nash. Implementing mapping composition. *VLDB J.* 17(2):333–353, 2008.

[BHE09] D. Bisztray, R. Heckel, H. Ehrig. Compositionality of Model Transformations. *Electronic Notes in Theoretical Computer Science* 236:5–19, 2009.

[BM07] P. A. Bernstein, S. Melnik. Model management 2.0: manipulating richer mappings. In *International Conference on Management of Data*. Pp. 1–12. ACM, 2007.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS. Springer-Verlag, 2006.

[FABJ09] M. D. D. Fabro, P. Albert, J. Bézivin, F. Jouault. Achieving Rule Interoperability Using Chains of Model Transformations. In *International Conference on Theory and Practice of Model Transformations*. LNCS 5563, pp. 249–259. Springer, 2009.

[JK06] F. Jouault, I. Kurtev. Transforming Models with ATL. In *Model Transformation in Practice Workshop*. LNCS 3844, pp. 128–138. Springer, 2006.

[KMS+10] T. Kühne, G. Mezei, E. Syriani, H. Vangheluwe, M. Wimmer. Explicit Transformation Modeling. In *MoDELS 2009 Workshops*. LNCS 6002, pp. 240–255. Springer, 2010.

[MV06] T. Mens, P. Van Gorp. A Taxonomy of Model Transformation. In *GraMoT'05*. ENTCS 152, pp. 125–142. Tallinn (Estonia), March 2006.

[Obj08] Object Management Group. Meta Object Facility 2.0 Query/View/Transformation Specification. April 2008.

[Old05] J. Oldevik. Transformation Composition Modelling Framework. In *International Conference on Distributed Applications and Interoperable Systems*. LNCS 3543, pp. 108–114. Springer, 2005.

[PGPB08] C. Pons, R. Giandini, G. Perez, G. Baum. An Algebraic Approach for Composing Model Transformations in QVT. In *International Workshop on Software Language Engineering*. 2008.

[RRL+09] J. E. Rivera, D. Ruiz-Gonzalez, F. Lopez-Romero, J. Bautista, A. Vallecillo. Orchestrating ATL Model Transformations. In *MtATL Workshop*. Pp. 34–46. 2009.

[VBH+06] B. Vanhooff, S. V. Baelen, A. Hovsepyan, W. Joosen, Y. Berbers. Towards a Transformation Chain Modeling Language. In *International Workshop on Embedded Computer Systems*. LNCS 4017, pp. 39–48. Springer, 2006.

[Wag08] D. Wagelaar. Composition Techniques for Rule-Based Model Transformation Languages. In *International Conference on Theory and Practice of Model Transformations*. LNCS 5063, pp. 152–167. Springer, 2008.

[YP05] C. Yu, L. Popa. Semantic Adaptation of Schema Mappings when Schemas Evolve. In *Int. Conference on Very Large Data Bases*. Pp. 1006–1017. ACM, 2005.