# Pattern-based Model-to-Model Transformation

Juan de Lara[1] and Esther Guerra[2]

[1] Universidad Autónoma de Madrid (Spain), `jdelara@uam.es`
[2] Universidad Carlos III de Madrid (Spain), `eguerra@inf.uc3m.es`

**Abstract.** We present a new, high-level approach for the specification of model-to-model transformations based on declarative patterns. These are (atomic or composite) constraints on triple graphs declaring the allowed or forbidden relationships between source and target models. In this way, a transformation is defined by specifying a set of triple graph constraints that should be satisfied by the result of the transformation.

The description of the transformation is then compiled into lower-level operational mechanisms to perform forward or backward transformations, as well as to establish mappings between two existent models. In this paper we study one of such mechanisms based on the generation of operational triple graph grammar rules. Moreover, we exploit deduction techniques at the specification level in order to generate more specialized constraints (preserving the specification semantics) reflecting pattern dependencies, from which additional rules can be derived.

## 1  Introduction

Model-Driven Development (MDD) is a software engineering paradigm where models are the core asset. They are used to specify, simulate, test, verify and generate code for the application to be built. Most of these activities include the specification and execution of model transformations, some of them between different languages. The transformation of a model conformant to a meta-model into another one conformant to a different meta-model is called model-to-model (M2M) transformation, and is the topic of this paper.

There are two main approaches to M2M transformation: *operational* and *declarative*. The first one is based on rules or instructions that explicitly state how and when the elements of the target model should be created starting from the elements of the source one. In declarative approaches, a description of the mappings between the source and target models is provided. This description states the relation that should hold between two models rather than how to create and link their elements. Declarative approaches are higher-level than operational ones since they form a compact description of a set of (operational) rules. In addition, they are inherently bidirectional because they do not specify any causality. Thus, they bring together in a single specification forward (i.e. source-to-target) and backward (i.e. target-to-source) transformations.

The state-of-the-art on declarative M2M transformation notations includes a handful of languages (see Section 5). However, sometimes they lack a formal

foundation and analysis techniques able to prove properties of the transformation [1]. In other cases, specifications are not fully declarative and may require a control mechanism or defining a causality between existing elements and those to be created in a given relation [2, 3], introducing some degree of operationality.

In this paper, we propose a purely declarative, formal approach to M2M transformation based on *triple patterns* to express the relations between source and target models. These are similar to graph constraints [4] but for triple graphs, made of two graphs related through an intermediate one. Patterns can specify positive (the relation they declare must hold) or negative information (the relation must not hold) and can be constrained by positive and negative restrictions. This high-level specification is compiled into lower-level mechanisms based on triple graph grammar operational rules [3] to achieve forward and backward transformations, as well as to relate two existing models. The compilation is performed in two steps. First, we use deduction rules to derive additional patterns that reflect dependencies and refine existing patterns with negative restrictions. Then, a rule for the chosen transformation direction is derived from each pattern.

The advantages of our technique are the following. First, it is purely declarative, based on patterns and constraints. This contrasts with other declarative approaches (such as Triple Graph Grammars (TGGs) [2, 3, 5]) where a causality has to be given between the existing elements and the ones that have to be created. As we exploit interactions between patterns, these dependencies are automatically derived. Second, it has a formal foundation that allows the study of the M2M transformation specification, in both declarative (i.e. patterns) and operational (i.e. derived rules) formats. Finally, we have devised deduction techniques, able to derive semantic information from the very patterns. For example, having a positive pattern demanding a certain structure and a negative one forbidding its duplication allows generating two rules: one creating the structure if it is not present, and another one reusing it if it already exists.

**Paper Organization**. Section 2 introduces triple patterns and Section 3 presents their deduction rules. Then, Section 4 shows how to derive the operational rules. Section 5 presents related work. Finally, Section 6 ends with the conclusions.

## 2 Specifying Transformations: Triple Patterns

This section introduces the different kinds of triple patterns, their satisfiability and the characteristics of the underlying operational mechanisms. These concepts rely on the notion of triple graph, which we introduce first.

Triple graphs are made of two graphs related through an intermediate one. We can use any graph model for these three graphs, from standard unattributed graphs $(V; E; s, t \colon E \to V)$ to more complex attributed graphs (e.g. E-graphs [4]).

**Def. 1 (Triple Graph)** *A triple graph* $TrG = (G_s, G_c, G_t, cs \colon V_{G_c} \to V_{G_s}, ct \colon V_{G_c} \to V_{G_t})$ *is made of two graphs* $G_s$ *and* $G_t$ *called source and target, related through the nodes of the correspondence graph* $G_c$.

Nodes in the correspondence graph $G_c$ have morphisms to nodes of the source and target graphs. If $\exists m \in V_{G_c}$ s.t. $x \xleftarrow{cs} m \xmapsto{ct} y$ we write $x \, rel \, y$. Other kinds of mappings could be used as well, for example the simpler one in [2], where the correspondence functions are graph morphisms or the more complex one in [6] where the correspondence functions can relate edges or be undefined. We use the notation $TrG|_x$ (for $x \in \{s, t, c\}$) to refer to the $G_x$ component of $TrG$, and write $\langle G_s, G_t \rangle$ for a triple graph with source and target graphs $G_s$ and $G_t$, and $\langle G_s, \emptyset, G_t \rangle$ for a triple graph with empty correspondence.

Next, we define triple graph morphisms.

**Def. 2 (Triple Graph Morphism)** *A triple graph morphism $f = (f_s, f_c, f_t) : TrG^1 \to TrG^2$ is made of three graph morphisms $f_x : TrG^1|_x \to TrG^2|_x$ (with $x = \{s, c, t\}$), where $f_s|_V \circ cs^1 = cs^2 \circ f_c|_V$ and $f_t|_V \circ ct^1 = ct^2 \circ f_c|_V$*

Source and target graphs can be typed by a type graph, or more in general by a *meta-model*, which includes inheritance [7]. In the latter case, we use the term *model* instead of graph. Given meta-model $MM$, $L(MM)$ refers to the set of all valid models conformant to (typed by) it. Similarly, we use the notion of meta-model triple [6] for the typing of triple graphs.

Triple patterns are similar to graph constraints [4, 8], but defined on triple graphs. We use them to describe the allowed and forbidden relationships between source and target models. We consider both simple and composite patterns.

**Def. 3 (Pattern)** *Given triple injective morphism $C \xrightarrow{q} Q$ and sets $N_{Pre} = \{Q \xrightarrow{c_i} C_i\}_{i \in Pre}$, $N_{Post} = \{Q \xrightarrow{c_j} C_j\}_{j \in Post}$ of negative pre- and post-conditions:*

- $\bigwedge_{i \in Pre} \overleftarrow{N}(C_i) \Rightarrow P(Q) \bigwedge_{j \in Post} \overrightarrow{N}(C_j)$ *is a simple pattern (S-Pattern).*
- $\bigwedge_{i \in Pre} \overleftarrow{N}(C_i) \wedge \overleftarrow{P}(C) \Rightarrow P(Q) \bigwedge_{j \in Post} \overrightarrow{N}(C_j)$ *is a composite pattern (C-Pattern).*
- $\overrightarrow{N}(C_j)$ *is a negative pattern (N-Pattern).*

**Remark.** The notation $\overleftarrow{P}(\cdot)$, $\overleftarrow{N}(\cdot)$ and $\overrightarrow{N}(\cdot)$ is just syntactic sugar to indicate a positive pre-condition, a negative pre-condition or a negative post-condition.

An S-Pattern is made of a positive graph Q restricted by negative pre- and post-conditions (*Pre* and *Post* sets). Intuitively, $Q$ should be present in triple graph $TrG$ whenever no negative pre-condition $C_i$ is found; and if $Q$ is found, then no occurrence of the negative post-conditions should be found. That is, while pre-conditions express restrictions for the pattern $Q$ to occur, post-conditions describe forbidden graphs. A C-Pattern is an S-Pattern with an additional positive pre-condition graph $C$. Thus an S-Pattern is a C-Pattern with $C$ and $q$ empty. Finally, an N-Pattern is a C-Pattern where $C$ and $Q$ are empty and there is only one negative post-condition, forbidden to occur.

**Def. 4 (M2M Specification)** *A M2M specification $S = \bigwedge_{i \in I} P_i$ is a conjunction of patterns, where each $P_i$ can be simple, composite or negative.*

**Remark.** For technical reasons, we assume that initially in a specification only N-patterns have negative post-conditions. This is not a restriction, as any post-condition can be expressed as an N-pattern. In fact, a M2M specification is usually made of just N- and S-patterns, from which we automatically derive C-patterns with positive pre-conditions encoding pattern dependencies, and transform N-patterns into post-conditions for the other patterns (see Section 3).

**Example.** Fig. 1 shows some patterns in an example M2M specification, inspired by the class to relational database transformation [1]. S-Pattern `C-T` states that a $C$ node (a class) that is not connected to another one (i.e. it does not have a parent) should be related to a $T$ (table). S-Pattern `A-Co` states that a $C$ node connected to an $A$ (an attribute), should be related to a $T$ with a $Co$ (column). Differently from TGGs, we don't need to specify here a positive pre-condition stating that a relation between a $C$ and a $T$ should already exist. This dependency is detected by the deduction rules we present in Section 3. S-Pattern `A-Co2` specifies that in the case of two $C$ nodes connected through an $R$ (a directed relation), the associated $T$ node of the source $C$ should have as foreign key ($F$ node) an attribute of the target class. Finally, N-Pattern `notDupF` forbids two $F$s between two $T$s.
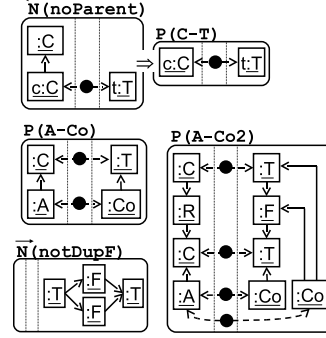


**Fig. 1.** M2M Specification.

Next we define the satisfaction of a pattern. As S- and N-Patterns are special cases of C-Patterns, it is enough to formulate C-Pattern satisfaction.

**Def. 5 (Pattern Satisfaction)** *Triple graph $TrG$ satisfies $CP = [\bigwedge_{i \in Pre} \overleftarrow{N}(C_i) \wedge \overleftarrow{P}(C) \Rightarrow P(Q) \bigwedge_{j \in Post} \overrightarrow{N}(C_j)]$, written $TrG \models CP$, iff:*

- *$CP$ is* forward satisfiable, $TrG \models_F CP$: $[\forall m^s \colon P_s \rightarrow TrG$ s.t. $(\forall i \in Pre$ s.t. $N_i^s \not\cong P_s,\ \nexists n_i^s \colon N_i^s \rightarrow TrG$ with $m^s = n_i^s \circ a_i^s)$, $\exists m \colon Q \rightarrow TrG$ with $m \circ q^s = m^s$, s.t. $\forall j \in Post\ \nexists n_j \colon C_j \rightarrow TrG$ with $m = n_j \circ c_j]$,
- *and $CP$ is* backwards satisfiable, $TrG \models_B CP$: $[\forall m^t \colon P_t \rightarrow TrG$ s.t. $(\forall i \in Pre$ s.t. $N_i^t \not\cong P_t,\ \nexists n_i^t \colon N_i^t \rightarrow TrG$ with $m^t = n_i^t \circ a_i^t)$, $\exists m \colon Q \rightarrow TrG$ with $m \circ q^t = m^t$, s.t. $\forall j \in Post\ \nexists n_j \colon C_j \rightarrow TrG$ with $m = n_j \circ c_j]$,

*with $P_x = C +_{C|_x} Q|_x$, $N_i^x = C +_{C|_x} C_i|_x$ and $N_i^x \xleftarrow{a_i^x} P_x \xrightarrow{q^x} Q$ $(x \in \{s,t\})$, see the left of Fig. 2. $C +_{C|_x} Q|_x$ is the pushout object of $C$ and $Q|_x$ through $C|_x$.*

**Remark**. Morphisms $q^x \colon P_x \rightarrow Q$ $(x = \{s,t\})$ uniquely exist due to the universal pushout property (as $C|_x \hookrightarrow C \xrightarrow{q} Q = C|_x \xrightarrow{q_x} Q|_x \hookrightarrow Q$). For the same reason, $a_i^x \colon P_x \rightarrow N_i^x$ uniquely exist (as $C|_x \hookrightarrow C \xrightarrow{e_i^s} N_i^x = C|_x \xrightarrow{q_x} Q|_x \xrightarrow{c_i|_x} C_i|_x \xrightarrow{d_i} N_i^x$). Moreover, $b_x^i = c_i \circ q_x$.∎

C-Patterns have a universal quantification, therefore we split them into two directed constraints. For this purpose we demand that in forward satisfaction,
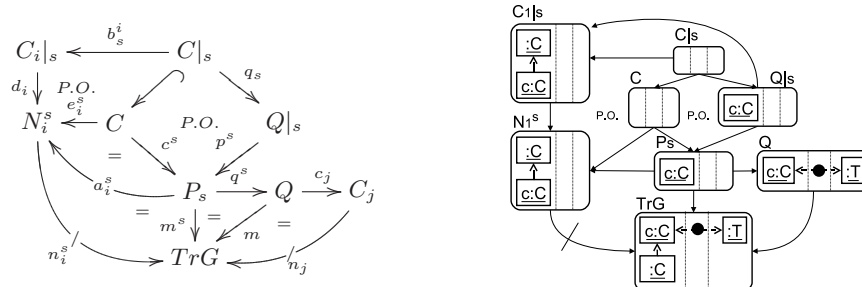
**Fig. 2.** Forward Satisfaction of Pattern (left). Forward Satisfaction Example (right).

for each occurrence of $P_s = Q|_s +_{C|_s} C = \langle Q|_s, C|_c, C|_t \rangle$ satisfying the negative pre-conditions, an occurrence of $Q$ must be found satisfying the negative post-conditions, see the left of Fig. 2. A positive pattern graph $Q$ is satisfied either because no $m^s$ is found (*vacuous satisfaction*), because $m^s$ and some negative pre-conditions are found (*negative satisfaction*), or because $m^s$ and $m$ are found and the negative pre- and post-conditions are not found (*positive satisfaction*). Note that if the resulting directed negative pre-condition $N_i^x$ is isomorphic to $P_x$, then it is not taken into account. This is needed as many pre-conditions express a restriction in either source or target but not on both. In addition to forward satisfaction, similar conditions are demanded for the target graph (backwards satisfiability). A graph satisfies specification $S$ if it satisfies all its patterns.

**Example.** The right of Fig. 2 shows the forward satisfaction of S-Pattern C-T by a triple graph. We have that $TrG \models_F C - T$ as there are two occurrences of $m^s$, the first one is shown in the figure (upper node $C$ in $TrG$) and is positively satisfied, while the second (lower $C$) is negatively satisfied. We also have $TrG \models_B C - T$, as there is just one $m^t$, positively satisfied. Thus, $TrG \models C - T$.

Please note that the specification does not explicitly state if a class with a parent should be connected with a table or not. An additional pattern could describe such situation. The forward operational mechanism presented in Section 4 does not add such table, as it minimally enforces the specification.

Starting from a specification $S$, lower level operational mechanisms are derived to perform forward ($\overrightarrow{S}$) and backward transformations ($\overleftarrow{S}$), as well as to relate two existing models ($\overleftrightarrow{S}$, omitted for space constraints, see [9]).

**Def. 6 (Operational Mechanisms)** *Specification $S$ has the following associated operational transformations:*

- **Forward:** *A function $\overrightarrow{S}: V_S(MM_S) \to TrG$ with domain $V_S(MM_S) = \{M_s \in L(MM_s) | \exists \langle M_s, X \rangle \models S\}$ s.t. $\forall M_s \in V_S(MM_S)$ $[\overrightarrow{S}(M_s) \models S] \wedge [\overrightarrow{S}(M_s)|_s \cong M_s]$.*
- **Backwards:** *A function $\overleftarrow{S}: V_T(MM_T) \to TrG$ with domain $V_T(MM_T) = \{M_t \in L(MM_t) | \exists \langle X, M_t \rangle \models S\}$ s.t. $\forall M_t \in V_T(MM_T)$ $[\overleftarrow{S}(M_t) \models S] \wedge [\overleftarrow{S}(M_t)|_t \cong M_t]$.*

The previous definitions are similar to the concept of *correct transformation* given in [10], but in addition we forbid modifying the source (resp. target) model in forward (resp. backwards) transformations.

## 3   Deduction and Annotation Mechanisms for Patterns

Next we present the deduction rules we use to: (i) generate new patterns that take dependencies into account, which guide the order of pattern enforcement by the operational mechanism; (ii) enrich S- and C-Patterns with pre- and post-conditions derived from other patterns; and (iii) deduce positive information from N-Patterns. We use two operations: *deduction*, which infers new patterns, and *annotation*, which makes dependencies among patterns explicit.

For example, from the specification in Fig. 1, the deduction rules generate a new pattern to reflect the dependency between `C-T` and `A-Co` (to take into account whether a pair $(C, T)$ is already related, before relating a pair $(A, Co)$). The deduction rules also add negative post-conditions derived from the `notDupF` N-pattern to the rest of patterns, and produce new patterns that reuse part of `notDupF` so that duplication of $F$ objects is not possible.

Most deduction rules are based on the maximal intersection of two triple graphs, called *maximal intersection object* (MIO), which is defined next.

**Def. 7 (MIO)** *Given triple graphs $TrG_1$ and $TrG_2$, a maximal intersection (MI) is given by a span of injective morphisms $(TrG_1 \xleftarrow{m_1} M \xrightarrow{m_2} TrG_2)$, s.t. $M \not\cong \emptyset \wedge \not\exists\, M' \not\cong M$ with $(TrG_1 \xleftarrow{m'_1} M' \xrightarrow{m'_2} TrG_2)$ and $m_{12} \colon M \to M'$ injective s.t. the diagram to the left of Fig. 3 commutes. Object $M$ is called MIO.*

MIOs are not unique, as the example to the right of Fig. 3 shows: $M_1$ and $M_2$ are both MIOs, but not $M_3$ as $M_1$ is bigger. The set of all MIs (resp. MIOs) of $TrG_1$ and $TrG_2$ is denoted by $MI(TrG_1, TrG_2)$ (resp. $MIO(TrG_1, TrG_2)$).
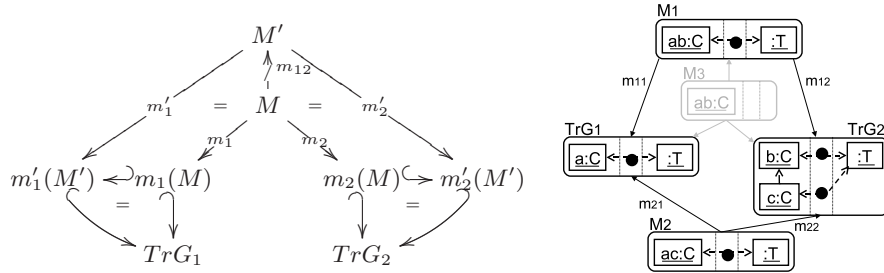


**Fig. 3.** Conditions for MIO (left). Example (right).

Patterns in a specification may have dependencies inducing a certain order of enforcement by the operational mechanism. We make such dependencies explicit by annotating patterns with additional graphs, related to the positive graph $Q$.

Dependencies are calculated by the intersection of two patterns, and can be interpreted as restrictions that must not hold when the pattern is enforced.

**Def. 8 (Annotated Pattern)** *An annotated pattern $(P, \{n_k\colon D_k \to Q\}_{k \in K})$ contains a pattern $P$ and a set of dependencies $D_k$ to $P$'s positive graph $Q$.*

Before presenting the deduction rules, we define an operation called *pre-condition weakening* (PW), which tests whether the positive graph of a C-Pattern is included in another one, and then adds the negative pre-conditions from the former to the latter. We show it here in its simplest form for S-Patterns, see [9] for the complete definition.

**Def. 9 (PW)** *PW on $[\bigwedge_{i \in Pre^1} \overleftarrow{N}(C_i^1) \Rightarrow P(Q^1)] \wedge [P(Q^2)]$ with $Q^1 \hookrightarrow Q^2$ results in $[\bigwedge_{i \in Pre^1} \overleftarrow{N}(C_i^1) \Rightarrow P(Q^1)] \wedge [\bigwedge_{i \in Pre^1} \overleftarrow{N}(C_i^1 +_{Q^1} Q^2) \Rightarrow P(Q^2)]$*

**Remark.** The specification resulting from PW is not equivalent to the original one. The second pattern is added negative pre-conditions, so that it is satisfiable by more graphs, namely by those in which $\exists n^i\colon C_i^1 +_{Q^1} Q^2 \to TrG$ (injective), as then $Q^2$ is not forced to occur. However, we use this operation to make coherent a specification: as an occurrence of the second pattern implies an occurrence of the first, by adding the negative pre-conditions we ensure that a positive satisfaction of the second implies a positive satisfaction of the first.

**Example.** As S-Pattern `C-T` is included in `A-Co`, PW adds the negative restriction $\overleftarrow{N}(noParent)$ from the former to the latter. The resulting pattern is shown to the right of Fig. 4 (second row, to the left).

Next, we show some deduction rules that preserve the specification semantics. We only show the simplest forms of them (for S-Patterns), see [9] for additional definitions and proofs. We first present the deduction rule for two S-Patterns called *S-Deduction* and its annotation mechanism $SA(\_,\_)$. S-Deduction creates a new pattern handling an intersection of two S-Patterns, while the annotation mechanism adds such intersections as dependencies to the two original patterns.

**Prop. 1 (S-Deduction)** *From $\bigwedge_{k \in \{1,2\}}[\bigwedge_{i \in Pre^k} \overleftarrow{N}(C_i^k) \Rightarrow P(Q^k)]$, we deduce the new patterns $\bigwedge_{M \in MIO(Q^1,Q^2)}[\bigwedge_{i \in Pre^1 \cup Pre^2} \overleftarrow{N}(C_i') \wedge \overleftarrow{P}(M) \Rightarrow P(Q^1 +_M Q^2)]$, where the $C_i'$ are calculated as shown to the left of Fig. 4.*

**Def. 10 (S-Annotation)** *Given two annotated S-Patterns $(P_i, D_i)$ with positive graphs $Q^i\colon SA((P_1, D_1), (P_2, D_2)) = \{(P_i, D_i \bigcup_{M \in MIO(Q^1,Q^2)}\{M \to Q^i\})\}_{i=1,2} \bigcup_{M \in MIO(Q^1,Q^2)}\{(SD(P_1, P_2, M), \emptyset)\}$, with $SD(P_1, P_2, M)$ the resulting pattern from applying S-Deduction using $M$.*

**Example.** The right of Fig. 4 shows an example of S-Annotation, where the newly generated pattern (bottom right) considers the fact that the relation demanded by pattern `C-T` may already exist. The added dependencies ($D_1$) ensure that the first and second patterns will only be enforced by the operational mechanisms when no occurrence of $D_1$ is found. As we will see later, this makes the
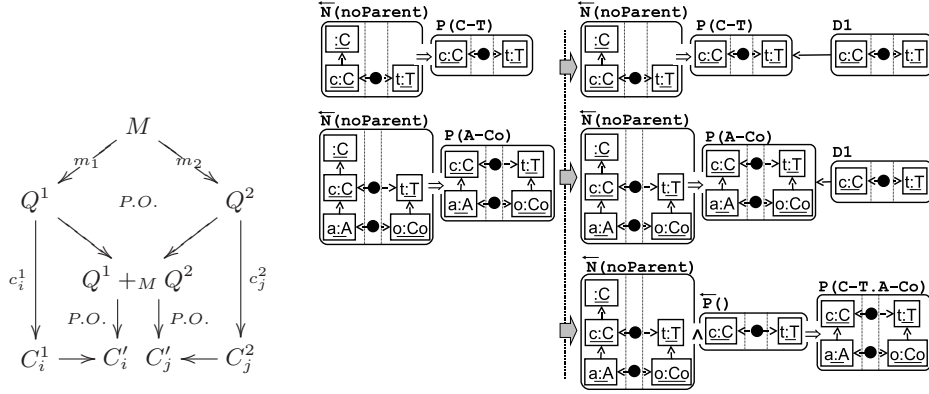
**Fig. 4.** Negative Pre-Conditions in S-Deduction (left). S-Annotation Example (right).

TGG operational rules generated for the first two patterns mutually exclusive with the one of the third, as well as confluent. Moreover, the rule for the third pattern will be able to reuse the structure created by the rule of the first.

Next deduction rule is used to take into consideration the interaction of N-patterns, which express global negative constraints, with other patterns.

**Prop. 2 (N-Deduction)** $[\bigwedge_{i\in Pre} \overleftarrow{N}(C_i) \Rightarrow P(Q) \bigwedge_{j\in Post} \overrightarrow{N}(C_j)] \wedge [\overrightarrow{N}(C_N)]$
*is equivalent to* $[\bigwedge_{i\in Pre} \overleftarrow{N}(C_i) \Rightarrow P(Q) \bigwedge_{j\in Post} \overrightarrow{N}(C_j) \bigwedge_{C_r\in RS} \overrightarrow{N}(C_r)] \wedge [\overrightarrow{N}(C_N)]$
*with* $RS = \{r^n \colon Q \to C_r\}_{C_r \in PO(MI(Q,C_N))}$ *and* $PO(MI(Q,C_N))$ *is the set of pushout objects of all spans in* $MI(Q,C_N)$.

*Proof(Sketch).* We have related $C_N$ in all possible (maximal) ways with $Q$, which is given by the pushout of each span in $MI(Q,C_N)$. This is similar to the procedure to convert a graph constraint into a post-condition [4, 8]. ∎

**Remark.** Removing $\overrightarrow{N}(C_N)$ does not yield an equivalent specification, as e.g. a graph with no occurrence of $Q$ is allowed to have an occurrence of $C_N$. Note however that we will delete N-Patterns when generating the TGG operational rules, as these by construction cannot generate any forbidden pattern.
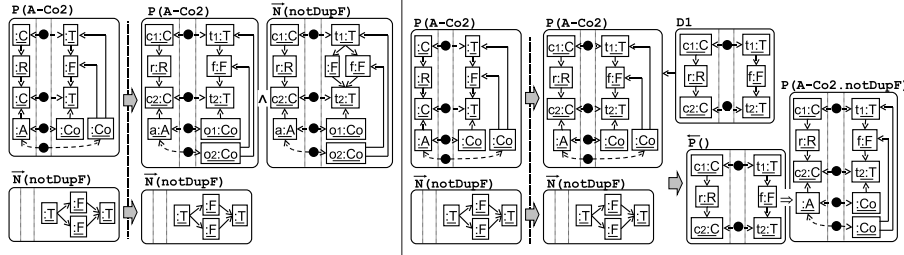


**Fig. 5.** N-Deduction Example (left). NP-Deduction and Annotation Example (right).

**Example.** The left of Fig. 5 shows how N-Pattern `notDupF` induces a negative constraint on S-Pattern `A-Co2`, resulting in the S-Pattern to its right. There

are two isomorphic MIOs (both made of two $T$s and one $F$) resulting in two isomorphic negative constraints, so that one is eliminated.

The following deduction rule detects N-Patterns that forbid a repetition of structures and generates a positive pattern that reuses such structure. First, we define the completion of a triple graph $M$ with respect to a graph $T$ such that $M \hookrightarrow T$. The completion adds to $M|_t$ all elements that are related to elements of $M|_s$ and belong to $T - M$, and similar for source elements. In addition, completion includes all unrelated elements of $T$.

**Def. 11 (Completion)** $C(M,T) = G$ *iff* $G$ *is the smallest graph s.t.* $M \hookrightarrow G \hookrightarrow T \land (\forall n \in V_{G|_s}, \nexists m \in V_{T|_t} - V_{G|_t} s.t. \ n \ rel \ m) \land (\forall x \in V_{G|_t}, \nexists y \in V_{T|_s} - V_{G|_s} s.t. \ y \ rel \ x) \land (\nexists z \in (V_{T|_s} \cup V_{T|_t}) - (V_{G|_s} \cup V_{G|_t}) \ s.t. \ z \ is \ unrelated \ ). \ G \ also contains all edges of $T$ with source and target in nodes of $G$.*
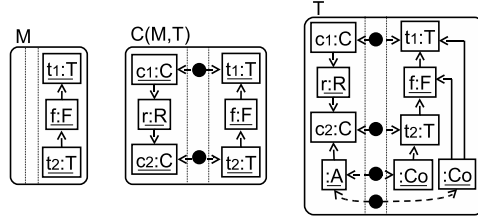


**Fig. 6.** Example of Completion.

**Example.** Fig. 6 shows an example of completion, where graph `M` is completed with respect to graph `T`, yielding graph `C(M, T)`. Note that $M \hookrightarrow C(M,T) \hookrightarrow T$.

**Prop. 3 (NP-Deduction)** $[\bigwedge_{i \in Pre} \overleftarrow{N}(C_i) \Rightarrow P(Q)] \land [\overrightarrow{N}(S)]$, *with $S$ the pushout of two isomorphic graphs $S_1 \cong S_2$ and $S_1 \in MIO(Q,S)$, is equivalent to* $[\bigwedge_{i \in Pre} \overleftarrow{N}(C_i) \Rightarrow P(Q)] \land [\overrightarrow{N}(S)] \land [\bigwedge_{i \in Pre} \overleftarrow{N}(C_i) \land \overleftarrow{P}(C(S_1,Q)) \Rightarrow P(Q)]$.

*Proof.* $C(S_1,Q) \hookrightarrow Q$, thus $[\overleftarrow{P}(C(S_1,Q)) \Rightarrow P(Q)]$ is subsumed by $P(Q)$. ∎

The NP-Deduction rule has an associated annotation rule $NP(\_,\_)$, which adds a dependency to the S-Pattern equal to the positive pre-condition of the newly generated pattern (see details in [9]).

**Example.** The right of Fig. 5 shows the derivation of C-Pattern `A-Co2.notDupF` from `A-Co2` and `notDupF`. The latter is made of the pushout of two isomorphic graphs made of two $T$s and one $F$, which belongs to `MIO(A-Co2, NotDupF)`. The completion of one of the isomorphic graphs with respect to `A-Co2` is the pre-condition graph $\overleftarrow{P}()$ of `A-Co2.notDupF`. The newly generated pattern reuses two $T$s and one $F$ so that the rule to be generated from it will not produce the situation forbidden by `notDupF`. The annotation procedure adds a dependency to `A-Co2` so that the generated rule will be mutually exclusive with the one for the deduced pattern.

## 4 Generating the Operational Rules

Next we show the generation of operational TGG rules from a specification. Compilation into other formalisms is also possible, e.g. to a constraint satisfaction problem [11]. We first present the structure of non-deleting TGG rules.

**Def. 12 (Non-Deleting Oper. TGG Rule)** *A TGG rule* $r = (L \xrightarrow{l} R, pre = \{n_i \colon L \to N_L^i\}_{i \in I}, post = \{n_j \colon R \to N_R^j\}_{j \in J})$ *is made of an injective morphism $l$ of triple graphs, and sets pre and post of negative pre- and post-conditions.*

Next we show how to generate a TGG rule given an annotated C-Pattern. The main idea is to use $P_s = C +_{C|s} Q|_s = \langle Q|_s, C|_c, C|_t \rangle$ as the LHS (for the forward rule) and $Q$ as the RHS. The negative pre- and post-conditions of the C-Pattern are used as negative pre- and post-conditions of the rule. Note the similarities with the satisfiability of patterns (Def. 5 and Fig. 2). The rule's RHS is used as a negative pre-condition so that satisfiability is enforced only once. Finally, dependencies are converted into negative pre-conditions.

**Def. 13 (Derived TGG Rule)** *Given annotated pattern $T = (\bigwedge_{i \in Pre} \overleftarrow{N}(C_i) \wedge \overleftarrow{P}(C) \Rightarrow P(Q) \bigwedge_{j \in Post} \overrightarrow{N}(C_j), D = \{n^k \colon D_k \to Q\}_{k \in K})$, the following TGG operational rules are derived:*

- **Forward.** $\overrightarrow{r_T} : (L = \langle Q|_s, C|_c, C|_t \rangle \xrightarrow{(id, q_c, q_t)} R = Q, pre = \{L \xrightarrow{n} R\} \cup \{a_i^s \colon L \to N_i^s | L \not\cong N_i^s\}_{i \in Pre} \cup \{s^k \colon L \to S^k\}_{k \in K}, post = \{n_j \colon R \to C_j\}_{j \in Post})$.
- **Backwards.** $\overleftarrow{r_T} : (L = \langle C|_s, C|_c, Q|_t \rangle \xrightarrow{(q|_s, q|_c, id)} R = Q, pre = \{L \xrightarrow{n} R\} \cup \{a_i^t \colon L \to N_i^t | L \not\cong N_i^t\}_{i \in Pre} \cup \{s^k \colon L \to S^k\}_{k \in K}, post = \{n_j \colon R \to C_j\}_{j \in Post})$.

*where $N_i^x \cong C_i|_x +_{C|_x} C$, and $a_i^x \colon L \to N_i^x$ is uniquely determined (see Fig. 2, where $P_x = L$). $S^k$ is the left-extension of $D_k$, see left of Fig. 7, where $n^k \circ b^k = r \circ l^k$ and $d^k \circ b^k = s^k \circ l^k$ are pullback and pushout squares respectively.*
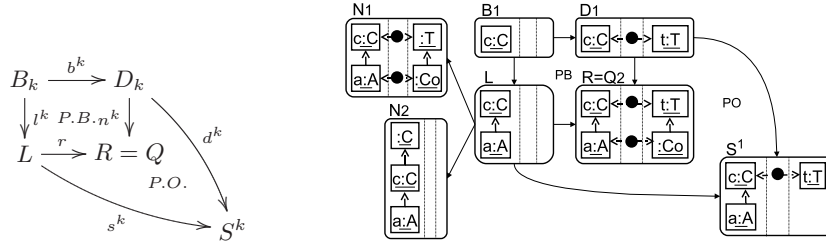


**Fig. 7.** Left Extension of $D_k \to Q$ (left). Generated Forward Rule A-Co (right).

**Example.** The right of Fig. 7 shows the generated forward rule from the annotated pattern A-Co of Fig. 4. Note how the NAC $S^1$ forbids applying the rule if

the node $C$ has an associated $T$. In this case, the rule generated from the derived pattern `C-T.A-Co` in Fig. 4 would be applicable (see rule `C-T.A-Co` in Fig. 8).

Before generating the rules we use the deduction and annotation mechanisms on the initial M2M pattern specification in order to transform N-patterns into negative post-conditions of the other patterns, generate patterns that take into consideration the satisfaction of other patterns, and identify dependencies between them. As stated before, we assume that the initial specification does not include patterns with both a positive graph and a negative post-condition.

**Def. 14 (Generation of Operational TGG Rules)** *Given specification S:*

1. *Use PW (Def. 9) on all possible patterns.*
2. *Use S-Annotation (Def. 10) for each pair of S-Patterns.*
3. *Use NP-Annotation on all possible patterns, initial and derived.*
4. *Use N-Deduction (Prop. 2) on all possible patterns and eliminate N-Patterns.*
5. *Take each derived pattern, and add to it all dependencies of the patterns it was derived from. Do not add such dependencies if they are included in the positive pre-condition of the derived pattern, as the pattern would be useless.*
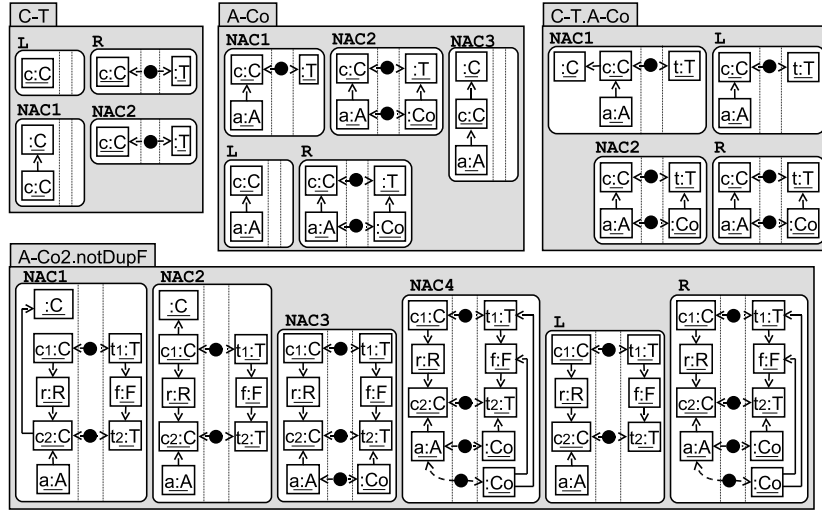6. *Generate an operational TGG rule for each causal pattern (Def. 12).*



**Fig. 8.** Some of the Generated Forward Operational Rules.

**Example.** Fig. 8 shows some of the generated forward rules. Rule `C-T` is generated from pattern `C-T`. NAC1 results from a pre-condition, while NAC2 is equal to the RHS. Rule `A-Co` results from pattern `A-Co`. NAC3 comes from the PW operation with pattern `C-T`, NAC2 is equal to RHS, and NAC1 is derived from a dependency when making S-Deduction with `C-T`. Rule `C-T.A-Co` is generated from a pattern derived from `C-T` and `A-Co` through S-Deduction. Its first

NAC comes from a dependency induced by their source patterns. Finally, rule `A-Co.notDupF` results from NP-Deduction (see the right of Fig. 5), where NAC1 and NAC2 come from pre-conditions of the patterns from which it is derived, and NAC3 comes from a dependency. These two last rules have some additional NACs (not shown), stemming from N-Deduction with pattern `notDupF`. The procedure generates a total of 10 rules (see [9]).

### 4.1 Correctness of the Operational Mechanisms

Now we show the correctness of the generated rules, focussing on forward rules as a similar reasoning holds for the backwards case. The generated rules: (i) produce models satisfying the specification, (ii) are confluent, (iii) terminate, and (iv) transform each source model for which there is a correct target model. We give an intuition, see [9] for more details.

(i) follows from the construction of the TGG rules. Their LHS is $\langle Q|_s, C|_c, C|_t \rangle = C +_{C|_s} Q|_s = P_s$, which is the *base* graph from which forward satisfaction is checked (see Fig. 2). As $R = Q$, morphism $m \colon Q \to TrG$ exists after the application of the rule. The rule negative pre- and post-conditions are derived from the negative pre- and post-conditions of the pattern. Thus, the rule can be applied iff the base morphism $m^s$ exists and the negative pre- and post-conditions of the pattern are satisfied. The additional NAC $\cong$ R makes the rule enforce the pattern once. As initially all forbidden graphs are expressed as N-Patterns, and we have performed N-Deduction, no rule can produce a forbidden result. Since we start with an empty target graph, backwards satisfaction is also obtained.

(ii) follows because S-annotation adds dependencies (which are transformed into NACs) to the initial patterns, and these are appropriately propagated to their derived patterns in step 5 of the rule generation process. Note, however, that initially we may have patterns included in others: $[P(Q_1)] \wedge [P(Q_2)]$ with $Q_1 \hookrightarrow Q_2$. In this case S-Deduction generates $[P(Q_1)] \wedge [P(Q_2)] \wedge [P(M) \Rightarrow P(Q_1 +_M Q_2)]$ (assuming just one MIO), from which we generate three rules. There is a conflict between the first two rules (i.e. a critical pair). However in a situation where both the first and the second are applicable (e.g. if we have $Q_2|_s \hookrightarrow TrG$), applying the first and the third is equivalent to applying the second. Besides, we cannot apply the first and the second due to the generated NACs. Thus, even in this case, rules are confluent (see [9]). **Example.** Consider the rules for the patterns `C-T` and `A-Co` and the and their derived pattern (`C-T.A-Co` see Fig. 8). Assume a situation where both `C-T` and `A-Co` are applicable. If `C-T` is applied first, then `A-Co` is disabled, but `C-T.A-Co` can be applied. If `A-Co` is applied first, then no other rule is applicable. However, in both cases we reach the same result.

(iii) follows from the fact that (a) each rule has its RHS as a NAC, therefore it can only be applied once for each initial match in the source model; and (b) a forward rule only changes the target model.

(iv) cannot be achieved for arbitrary M2M specifications. We restrict to what we call *Injective Positive Specifications*, which contain enough positive patterns to produce the operational TGG rules. Next definition introduces the forward case (FIP), the backwards one is similar.

**Def. 15 (FIP Spec.)** *Specification $S = \bigwedge_{i=1..n} T_i$ is FIP, iff $\forall M_s \in L(MM_s)$ s.t. $\exists TrG = \langle M_s, X \rangle \models S, \exists k_i \in \mathbb{N}, \exists P_s^i \leftarrow S_{uv}^{ij} \rightarrow P_s^j$ with $P_s^m = C^m +_{C^m|_s} Q^m|_s$, $u = \{1..k_i\}$, $v = \{1..k_j\}$ and $S_{uv}^{ij} \not\cong P_s^i$ if $i = j$, s.t. $G$ is the colimit of the diagram to the left of Fig. 9 (with all arrows injective) with $G \hookrightarrow TrG$ and $G|_t \cong TrG|_t$.*
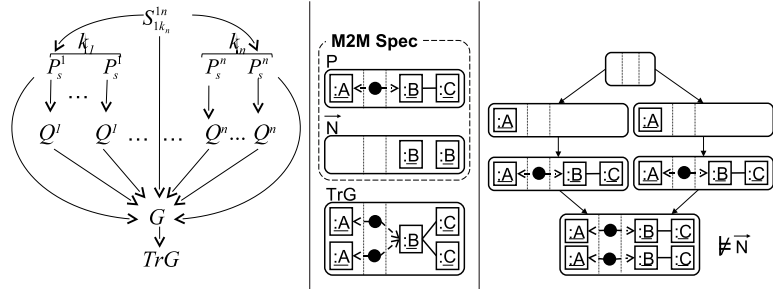


**Fig. 9.** Condition for FIP (left). Non-FIP Specification (center). Invalid Graph (right).

**Remark.** The definition considers $k_i$ occurrences of each pattern $T_i$. Two occurrences of patterns $T_i$ and $T_j$ can overlap, and this is modelled by $S_{uv}^{ij}$. We forbid $P_s^i$ be the overlap of two occurrences of the same pattern $Q^i$, as the operational mechanism minimally enforces each pattern (i.e. rules have a NAC equal to the RHS). We have made a simplification in the diagram, but each occurrence of $T_i$ should satisfy its negative pre- and post-conditions.

**Example.** Consider the specification in the center of Fig. 9, and assume no deduction is performed. There is a valid triple graph $TrG$ with two $As$ in its source, but the rules generated without deduction cannot create such graph, as they would produce two $Bs$.

NP-Deduction can turn some non-FIP specifications into FIP, because it creates a new pattern that reuses an already created structure. The right of Fig. 9 shows that if NP-Deduction is not applied, we cannot handle a graph with two $As$. Fig. 10 shows that after NP-Deduction the resulting pattern can handle such graph as it reuses a $B$ and is applied twice. It is up to future work to determine further deduction rules to cover additional non FIP-specifications.
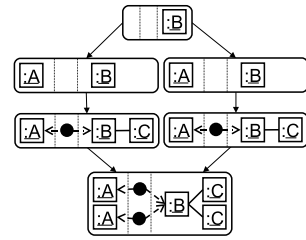


**Fig. 10.** FIP Specification.

# 5   Related Work

Some declarative approaches to M2M transformation use a textual syntax, e.g. PMT [12] or Tefkat [13]. These two particular notations are uni-directional, whereas we generate forward and backward transformations.

Among the visual declarative approaches, a prominent example is the QVT-relational language [1]. The relations may include *when* and *where* clauses that identify pre- and post-conditions and can refer to other relations. From this specification, executable QVT-core is generated that performs forward/backward transformations. This approach is similar to ours, but we compile our patterns to TGG rules, allowing the analysis of the generated transformation [4]. Besides, we can analyse at a higher-level (i.e. pattern level) as our patterns have a formal foundation [9]. Moreover, we automatically detect pattern dependencies and perform pattern inference. In QVT-relations dependencies must be made explicit in the *when* and *where* clauses, and there is no equivalent to our N-Patterns. An attempt to formalize QVT-core is found in [14].

In [15], transformations are expressed through positive patterns that rely on OCL constraints, but no operational mechanism is given. In BOTL [16], the mapping rules use a UML-based notation that allows reasoning about applicability or meta-model conformance. We can reason both at the specification and operational levels.

TGGs [3] formalize the synchronized evolution of two graphs through declarative rules. From this specification, low-level operational TGG rules are derived to perform forward and backward transformations, as well as to relate two existing graphs. We also generate these operational rules from our patterns. However, whereas in declarative TGG rules dependencies must be made explicit (i.e. we must say which elements should exist and which ones are created), in our patterns this information is derived. For instance, in TGGs, a rule like pattern `C-T.A-Co` has to be specified, it is not enough to give `C-T` and `A-Co`.

Although inspired by TGGs, our patterns are a different approach: patterns specify relations, not rules. Similar to graph constraints [8, 17], a M2M specification by patterns describes a language of valid triple graphs. Moreover, TGGs have some limitations. First, they allow neither specifying negative information, nor deriving positive information from negative one (like NP-Deduction). In [5], the lack of negation is alleviated by assigning execution priorities to rules. However, this is insufficient to simulate general application conditions, it has an operational nature, and implies knowing the rule generation mechanism and execution engine. Second, a control mechanism is needed to guide the execution of the operational rules, such as priorities [5] or their coupling to editing rules [2]. One can see TGGs as a subset of our approach, where a TGG rule is a pattern of the form $\overleftarrow{P}(L) \Rightarrow P(R)$ without negative conditions or deduction techniques.

In [18], an algorithm is given for the derivation of declarative TGGs from example pairs of models. Interestingly, the user does not have to specify the correspondence nodes in these pairs. The employed techniques resemble our use of MIOs, but our patterns are richer, allowing negative pre- and post-conditions, and our theory supports further derivation techniques (e.g. NP-Deduction).

With respect to graphical patterns, in [17] a logic of constraints is proposed, in which constraints are existentially satisfied, while ours are universal. Moreover, we provide deduction techniques specially tailored for M2M specifications and triple patterns. In [19] we presented a simpler notion of pattern and used it to extend normal rules to synchronous TGGs. We applied it to the synchronization of the concrete and abstract syntax of visual models. The patterns were restricted to work with positive information, and the execution of the derived rules was associated to editing rules (like traditional TGGs). Here we present a new concept of pattern, which allows expressing negative conditions, introduce deduction rules and present a new algorithm for TGG rule derivation that is suitable for M2M transformation and does not need a normal rule to start with.

## 6   Conclusions and Future Work

In this paper we have presented a new formal approach to declarative M2M transformation. Relations between source and target models are expressed as different kinds of patterns, from which operational TGG rules are derived implementing forward/backwards transformations and taking into account pattern interactions. This is done by deduction mechanisms that detect interdependencies and produce new patterns that reuse structures created by other patterns. This is one of the strengths of the present work: pattern dependencies are automatically calculated and not explicitly given by the designer such as with QVT and TGGs.

We have already identified analysis properties, both at the specification (e.g. language covering, pattern conflicts) and operational levels (e.g. hippocratic transformations [10]). We have omitted them by lack of space (see [9]).

Although we generate operational TGG rules from a pattern specification, other target formalisms could be used as well (e.g. OCL, Alloy). In fact, one of our next goals is expressing a specification in terms of a constraint satisfaction problem, in the lines of [11]. This would eliminate some problems of the compilation into rules, such as the restriction to handle FIP specifications only. Note that with the theory presented so far we can handle attributes, but not attribute conditions or computations. Our aim is to use OCL and the analysis techniques we proposed in [11].

It would be interesting to extend the set of derived operational rules to handle incremental synchronization and change propagation. More complex patterns able to deal with recursion or having parameters are also under consideration. Finally, we aim to formalize a part of QVT using this technique.

## References

1. QVT. `http://www.omg.org/docs/ptc/05-11-01.pdf` (2005)

2. Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information preserving bidirectional model transformations. In: FASE'07. Volume 4422 of LNCS., Springer (2007) 72–86
3. Schürr, A.: Specification of graph translators with triple graph grammars. In: WG'94. Volume 903 of LNCS., Springer (1994) 151–163
4. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of algebraic graph transformation. Springer-Verlag (2006)
5. Königs, A.: Model transformation with Triple Graph Grammars. In: MTiP'05. (2005)
6. Guerra, E., de Lara, J.: Event-driven grammars: Relating abstract and concrete levels of visual languages. SoSyM, special section on ICGT'04 (2007) 317–347
7. de Lara, J., Bardohl, R., Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Attributed graph transformation with node type inheritance. TCS **376**(3) (2007) 139–163
8. Heckel, R., Wagner, A.: Ensuring consistency of conditional graph rewriting - a constructive approach. ENTCS **2** (1995)
9. de Lara, J., Guerra, E.: Pattern-based model-to-model transformation: Long version. arXiv:0804.4745v1 [cs.SE], `http://arxiv.org/abs/0805.4745v1` (2008)
10. Stevens, P.: Bidirectional model transformations in QVT: Semantic issues and open questions. In: MoDELS'07. Volume 4735 of LNCS., Springer (2007) 1–15
11. Cabot, J., Clarisó, R., Guerra, E., de Lara, J.: Analysing graph transformation rules through OCL. In: To appear in Proc. ICMT'08. LNCS, Springer (2008)
12. Tratt, L.: A change propagating model transformation language. JOT **7**(3) (2008) 107–126
13. Lawley, M., Steel, J.: Practical declarative model transformation with Tefkat. In: MoDELS Satellite Events. Volume 3844 of LNCS., Springer (2005) 139–150
14. Greenyer, J.: A study of model transformation technologies: Reconciling TGGs with QVT. Master's thesis, University of Paderborn (2006)
15. Akehurst, D.H., Kent, S.: A relational approach to defining transformations in a metamodel. In: UML'02. Volume 2460 of LNCS., Springer (2002) 243–258
16. Braun, P., Marschall, F.: Transforming object oriented models with BOTL. ENTCS **72**(3) (2003)
17. Orejas, F., Ehrig, H., Prange, U.: A logic of graph constraints. In: FASE'08. Volume 4961 of LNCS., Springer (2008) 179–198
18. Kindler, E., Wagner, R.: Triple graph grammars: Concepts, extensions, implementations and application scenarios. Tech. Rep. TR-RI-07-284, U. Paderborn (2007)
19. de Lara, J., Guerra, E., Bottoni, P.: Triple patterns: Compact specifications for the generation of operational triple graph grammar rules. In: GT-VMT'07. Volume 6 of Electronic Communications of the EASST. (2007)