Inter-model Consistency Checking Using Triple Graph Grammars and Linear Optimization Techniques

Erhan Leblebici[1(✉)], Anthony Anjorin[2], and Andy Schürr[1]

[1] Technische Universität Darmstadt, Darmstadt, Germany
{erhan.leblebici,andy.schuerr}@es.tu-darmstadt.de
[2] Universität Paderborn, Paderborn, Germany
anthony.anjorin@uni-paderborn.de

Abstract. An important task in *Model-Driven Engineering* (MDE) is to *check consistency* between two *concurrently developed yet related* models. Practical approaches to consistency checking, however, are scarce in MDE. *Triple Graph Grammars* (TGGs) are a rule-based technique to describe the consistency of two models together with correspondences. While TGGs seem promising for consistency checking with their precise consistency notion and explicit traceability information, the substantial search space involved in determining the "optimal" set of rule applications in a consistency check has arguably prevented mature tool support so far. In this paper, we close this gap by combining TGGs with *linear optimization* techniques. We formulate decisions between single rule applications of a consistency check as integer inequalities, which serve as input for an optimization problem used to detect maximum consistent portions of two models. To demonstrate our approach, we provide an experimental evaluation of the tool support made feasible by this formalization.

Keywords: Consistency check · Traceability · Linear optimization

1 Introduction and Motivation

Models are used in Model-Driven Engineering (MDE) to represent abstractions of a system with respect to a certain perspective. In a typical MDE process, especially when different disciplines are involved, there are often models containing related information but maintained by different engineers concurrently giving rise to *consistency* challenges. A crucial task in MDE is thus to perform a *consistency check*, i.e., to determine if, or to what extent, two models are consistent, before applying any consistency restoration. We discuss in this paper consistency checking with *Triple Graph Grammars* (TGGs) [25], a rule-based language for specifying a consistency relation between two modeling languages.

The basic idea of TGGs is to specify a set of rules (a grammar) describing how consistent model pairs are constructed together with a *correspondence* model

M. Huisman and J. Rubin (Eds.): FASE 2017, LNCS 10202, pp. 191–207, 2017.
DOI: 10.1007/978-3-662-54494-5_11

representing explicit traceability information. Given such a specification and two models, the goal of a consistency check is to determine whether the models can be constructed by the grammar and, if so, to create a respective correspondence model. If the model pair is not completely consistent, we propose to determine a partial correspondence model referencing consistent subparts of the models.

Establishing consistency checking with TGGs is crucial as practical solutions to consistency checking are currently scarce in MDE. QVT-R [22] (in particular its *checkonly mode*) is the only available standard for consistency checking in MDE. The QVT-R implementation candidate Medini QVT [20], however, is able to check consistency only if one of the models is generated by the tool itself via model transformation and auxiliary traces are already available. Consistency checking for models developed *concurrently* (in independent environments by different developers) where traces are not available beforehand is not addressed so far. Our goal is to tackle this general consistency challenge in concurrent MDE activities by clearing the last obstacles for the applicability of TGGs.

The pioneer work for consistency checking with TGGs is [4] which derives consistency checking rules from a TGG. How to conclude consistency (or inconsistency) of two models with these rules, however, remains open due to the substantial state space regarding decisions among possible rule applications. Finding the *best* partial correspondence model between two inconsistent models (e.g., relating as many elements as possible) is consequently also an open issue. We close this gap by formulating a *linear optimization problem* for choices among rules, and discuss the respective tool support made feasible by this novel formalization. While we discuss relating a maximum number of model elements as a general objective fort the optimization problem, our approach can be extended with custom objectives reflecting case-specific policies for handling inconsistency (e.g., covering as many elements as possible of a certain type, model, or property).

As a running example, we consider consistency between Java code and UML class diagrams throughout the paper. Note that many UML tools generate Java code from UML class diagrams (or vice versa) in a consistent way but no practical solution exists to check consistency between these artifacts if they are developed concurrently (similar to the shortcomings of QVT-R implementations as discussed above). The excerpt we focus on in our running example is a one-to-one mapping between Java and UML classes, methods, and parameters but already reveals the complexity of consistency checking. The challenging part of our case study arises from *overloaded methods*: Determining the corresponding pairs of methods belonging to the same class and sharing the same name can require a careful decision making. Consider, for example, the consistent Java and UML class pair in Fig. 1. The dashed lines represent correct decisions of corresponding `remove` methods (and consequently corresponding parameters), while the dotted lines represent wrong decisions. In fact, such local decisions while relating two models are not specific to this example and wrong decisions can be chosen by a

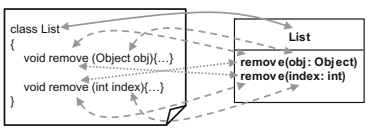


Fig. 1. A consistent model pair

TGG-based consistency check. In this case, our consistent model pair would be identified erroneously as inconsistent (due to incompatible parameters of mistakenly corresponding methods). Our experiments with HenshinTGG [8], the only TGG tool we are aware of with consistency checking support, showed that consistency checking indeed fails in cases where such decisions are necessary.

Our approach considers alternative steps of a consistency check and uses logical dependencies between single steps to calculate a correct subset. This corresponds to creating all lines together in Fig. 1, solving a suitable optimization problem to maximize the number of related elements, and eliminating the dotted lines in retrospect. Intuitively, the dashed and dotted lines in Fig. 1 are alternatives where the dashed ones relate a larger number of elements.

After reviewing basic TGG theory in Sect. 2, we formalize in Sect. 3 choices between alternative decisions in a consistency check as integer inequalities. Our basic formal result in Theorem 1 states that any choice satisfying these inequalities leads to some consistent portions of models. Subsequently, we state a sufficient (Corollary 1) as well as a sufficient and necessary (Corollary 2) condition for consistency by maximizing these portions. Section 4 evaluates our tool support. Section 5 discusses related work, and Sect. 6 concludes the paper.

2 Preliminaries

In line with the algebraic formalization of graph grammars [6], we represent *models* as *graphs*. We then introduce *triples* of graphs (Fig. 2) as we shall be dealing with *source*, *target*, and *correspondence* models (denoted with S, T, or C prefix, respectively). The notion of triple graphs provides a precise means for describing correspondences as graph patterns that are amenable to mature graph transformation tools. We provide our formalization without type and attribute information in graphs for brevity. The formalization can be extended compatibly to attributed typed graphs with inheritance according to [6].

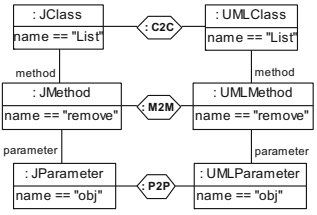


Fig. 2. A triple graph

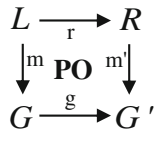
Definition 1 (Graph, Triple Graph). *A* graph $G = (V, E, s, t)$ *consists of a set* V *of vertices, a set* E *of edges, and two functions* $s, t : E \to V$ *assigning to each edge a source and target vertex, respectively.* $\mathsf{elements}(G)$ *denotes the union* $V \cup E$ *where each* $e \in \mathsf{elements}(G)$ *is an* element *of* G*. A* graph morphism $f : G \to G'$*, with* $G' = (V', E', s', t')$*, is a pair of functions* $f_V : V \to V'$*,* $f_E : E \to E'$ *such that* $f_V \circ s = s' \circ f_E \wedge f_V \circ t = t' \circ f_E$*.* f *is a* monomorphism *iff* f_V *and* f_E *are injective.*

A triple graph $G = G_S \overset{\gamma_S}{\leftarrow} G_C \overset{\gamma_T}{\rightarrow} G_T$ *consists of graphs* G_S*,* G_C*,* G_T*, and graph morphisms* $\gamma_S : G_C \to G_S$ *and* $\gamma_T : G_C \to G_T$*.* $\mathsf{elements}(G)$ *denotes the union* $\mathsf{elements}(G_S) \cup \mathsf{elements}(G_C) \cup \mathsf{elements}(G_T)$*. A* triple morphism $f : G \to G'$ *with* $G' = G'_S \overset{\gamma'_S}{\leftarrow} G'_C \overset{\gamma'_T}{\rightarrow} G'_T$*, is a triple* $f = (f_S, f_C, f_T)$ *of graph morphisms*

where $f_X : G_X \to G'_X$ *and* $X \in \{S, C, T\}$, $f_S \circ \gamma_S = \gamma'_S \circ f_C$ *and* $f_T \circ \gamma_T = \gamma'_T \circ f_C$. *f is a triple monomorphism iff* f_S, f_C, *and* f_T *are monomorphisms.*

A TGG comprises monotonic (i.e., non-deleting) *triple rules* that generate and thus define the language of consistent source and target graphs.

Definition 2 (Triple Rule and Derivation). *A* triple rule $r : L \to R$ *is a triple monomorphism. A* direct derivation *via a triple rule r, denoted as* $G \overset{r@m}{\Longrightarrow} G'$, *is constructed, as depicted to the right, by a pushout over r and a triple monomorphism* $m : L \to G$ *where m is called* match. *A* derivation $D : G \overset{r_1@m_1}{\Longrightarrow} G_1 \overset{r_2@m_2}{\Longrightarrow} \ldots \overset{r_n@m_n}{\Longrightarrow} G_n$ *(short* $D : G \overset{*}{\Rightarrow} G_n$*) is a sequence of direct derivations. We refer to the set* $\mathcal{D} = \{d_1, \ldots, d_n\}$ *of direct derivations included in D as* the underlying set *of D.*



Example 1. Figure 3 depicts four TGG rules for our running example where created elements of a rule (i.e., elements in R but not in L) are depicted green with a ++-markup. Context elements (L) are depicted black. Triple rule r_1 creates a Java class and a UML class together with a correspondence. Triple rule r_2 does the same with additional inheritance links on both sides. Triple rule r_3 creates a corresponding pair of Java and UML methods, while triple rule r_4 creates parameters. The attribute constraints (e.g., *jc.name == uc.name* in r_1) enforce name equality of corresponding classes, methods, and parameters.

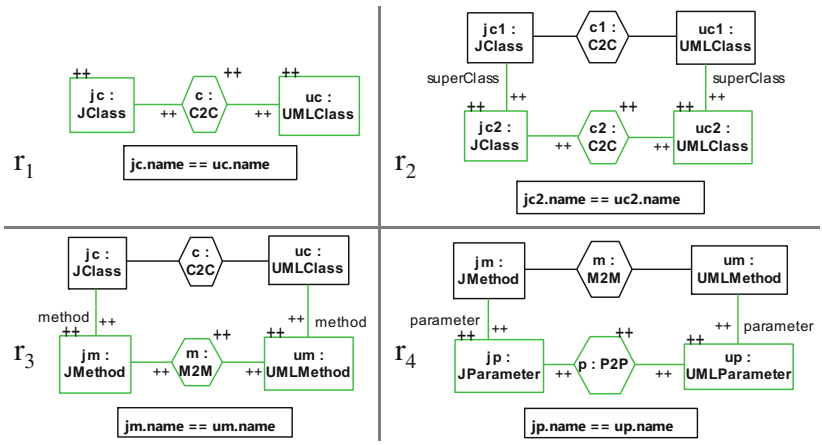


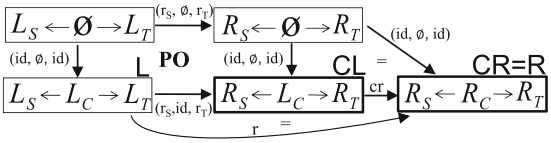
Fig. 3. TGG rules describing how consistent models are constructed

Definition 3 (Triple Graph Grammar and Consistency). *A* triple graph grammar $TGG : \mathcal{R}$ *consists of a set* $\mathcal{R}$ *of triple rules. The generated language*

$\mathcal{L}(TGG)$ is defined as follows: $\mathcal{L}(TGG) = \{G_\emptyset\} \cup \{G \mid \exists D : G_\emptyset \xRightarrow{r_1@m_1} G_1 \xRightarrow{r_2@m_2} \ldots \xRightarrow{r_n@m_n} G_n = G\}$, *where* $G_\emptyset$ *is the empty triple graph and,* $\forall i \in \{1, \ldots, n\}$, $r_i \in \mathcal{R}$. *A source graph* G_S *and a target graph* G_T *are* consistent *with respect to TGG iff* $\exists G \in \mathcal{L}(TGG)$ *with* $G = G_S \leftarrow G_C \rightarrow G_T$.

Finally, we define *consistency rules* derived from the original triple rules. They *mark* source and target elements that would be created by the original TGG rules. This way, it can be determined whether a given pair of source and target graphs can be constructed by applying the original triple rules of a TGG.

Definition 4 (Consistency Rule and Marking Elements). *Given a triple rule* $r : L \rightarrow R$ *with* $L = L_S \leftarrow L_C \rightarrow L_T$ *and* $R = R_S \leftarrow R_C \rightarrow R_T$, *the respective* consistency rule $cr : CL \rightarrow CR$ *is constructed, as depicted to the right, such that* CL *is a pushout of* L *and* $R_S \leftarrow \emptyset \rightarrow R_T$ *over* $L_S \leftarrow \emptyset \rightarrow L_T$, *and* $CR = R$ ($cr : CL \rightarrow CR$ *is induced as the universal property of the pushout). An element* $e \in \mathit{elements}(R_S) \cup \mathit{elements}(R_T)$ *is referred to as a* marking element *of* cr *iff* $\nexists e' \in \mathit{elements}(L_S) \cup \mathit{elements}(L_T)$ *with* $r_S(e') = e$ *or* $r_T(e') = e$.



Example 2. The consistency rule cr_4 derived from the original triple rule r_4 is depicted to the right together with its marking elements. Intuitively, a consistency rule *marks* exactly those source and target elements that are created by the original triple rule (++-markup is replaced by a gray checked box on the source and target side), and creates the same correspondences. Consistency rules cr_1, cr_2, and cr_3 for the respective triple rules r_1, r_2, and r_3 are derived analogously.

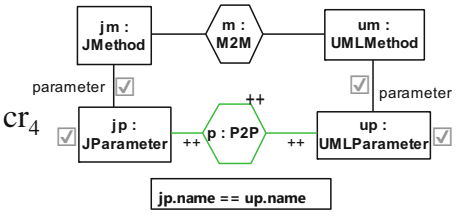


Fig. 4. Consistency rule cr_4 derived from r_4 in Fig. 3

3 Choices Between Markings as an Optimization Problem

Our goal in this section is to check consistency for a given model pair G_S and G_T with respect to a TGG, i.e., to find a triple graph $G'_S \leftarrow G_C \rightarrow G'_T \in \mathcal{L}(TGG)$ where G'_S and G'_T refer to the consistent portions of G_S and G_T, respectively ($G'_S = G_S$ and $G'_T = G_T$ if G_S and G_T are consistent). Direct derivations via consistency rules represent the single steps of such a consistency check. Markings *simulate* the creation of G_S and G_T by the original triple rules and correspondences (G_C) are created in the process serving as traceability information. As we have discussed in Sect. 1, however, this process can result in wrong markings and correspondence creations if it is not suitably controlled.

In the following, we consider derivations with consistency rules that possibly mark model elements multiple times and thus represent a superset of correct markings. We consider each direct derivation of such a derivation as an integer between 0-1 and formulate integer inequalities for exclusion and implication dependencies between direct derivations which were discussed in previous work [17]. In sum, we combine two techniques: Graph pattern matching (via consistency rules) is performed on triple graphs and logical constraints over matched patterns are solved. While the first reduces the search space via structural patterns (as compared to purely constraint-based solutions such as [18,19]), the latter leads to a final choice between matchings.

Moreover, we handle the logical constraints as an *optimization problem* to address consistency and inconsistency in a unified manner. This allows us to use an *objective function* that governs the process to find a best choice among collected direct derivations, which is especially crucial in case of inconsistency. In this paper, we only focus on maximizing the number of related elements as the objective while our approach can be extended by further custom objectives reflecting case-specific consistency policies (e.g., marking as many UML elements as possible while marking Java elements is not of uppermost priority). The main idea is depicted schematically in Fig. 5 based on our exemplary model pair.

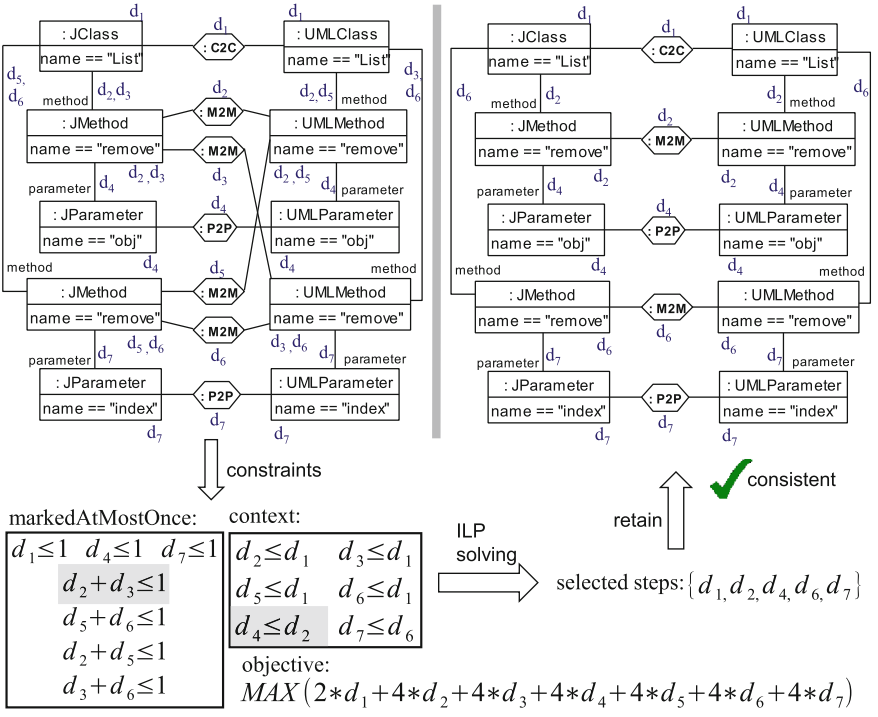


Fig. 5. A schematic overview of our approach with consistent models

In the upper left part of Fig. 5, a derivation of seven direct derivations $\{d_1, \ldots, d_7\}$ with consistency rules marks the source and target model elements. Every source and target model element is annotated with its marking direct derivations. Similarly, each correspondence is annotated with its creating direct derivation. Without taking a decision, for instance, all overloaded remove methods are marked twice due to multiple options. Sets of *constraints* then state logical dependencies between direct derivations. For example, both d_2 and d_3 mark the same remove method on the Java side as alternatives and thus cannot be chosen together, leading to $d_2 + d_3 \leq 1$ (highlighted with a gray shading in Fig. 5). Furthermore, d_2 creates a correspondence and marks source and target elements used by d_4 as context to mark obj parameters. Hence, d_4 can only be chosen if d_2 is chosen (leading to $d_4 \leq d_2$). Finally, an *objective function* maximizes the number of marked elements while satisfying the inequalities (each direct derivation is weighted with the number of its marked elements). This forms a linear optimization problem and can be appropriately handled with *Integer Linear Programming* (ILP) techniques in practice (in fact, a special case of ILP with 0-1 integers). The model pair in Fig. 5 is identified to be consistent as the outcome of the optimization problem marks each model element exactly once (as they would be if created by the original TGG rules).

To formalize this idea, we first define sets of *marked, required,* and *created* elements of a direct derivation, which are decisive for formulating constraints.

Definition 5 (Marked, Required, and Created Elements). *For a direct derivation* $d : G \overset{cr@cm}{\Longrightarrow} G'$ *via a consistency rule* $cr : CL \to CR$ *with* $G = G_S \leftarrow G_C \to G_T$ *and* $G' = G_S \leftarrow G'_C \to G_T$*, we define the following sets:*

- $\mathit{marks}(d) = \{e \in \mathit{elements}(G_S) \cup \mathit{elements}(G_T) \mid \exists e' \in \mathit{elements}(CL)$ *with* $cm(e') = e$ *where* e' *is a marking element of* $cr\}$
- $\mathit{requiresSrcTrg}(d) = \{e \in \mathit{elements}(G_S) \cup \mathit{elements}(G_T) \mid \exists e' \in \mathit{elements}(CL)$ *with* $cm(e') = e$ *where* e' *is not a marking element of* $cr\}$
- $\mathit{requiresCorr}(d) = \{e \in \mathit{elements}(G_C) \mid \exists e' \in \mathit{elements}(CL)$ *with* $cm(e') = e\}$
- $\mathit{creates}(d) = \mathit{elements}(G'_C) \setminus \mathit{elements}(G_C)$.

Given a model pair $G_0 = G_S \leftarrow \emptyset \to G_T$, a derivation constraint is a set of integer inequalities representing exclusions and implications between direct derivations collected in a consistency check process starting from G_0.

Definition 6 (Constraints for Consistency Check Derivations). *Given a triple graph* $G_0 : G_S \leftarrow \emptyset \to G_T$*, let* $D : G_0 \overset{*}{\Rightarrow} G_n$ *be a derivation via consistency rules with the underlying set* $\mathcal{D}$ *of direct derivations. For each direct derivation* $d_1, \ldots, d_n \in \mathcal{D}$*, we define respective integer variables* $\delta_1, \ldots, \delta_n$ *with* $0 \leq \delta_1, \ldots, \delta_n \leq 1$*. A* constraint $\mathcal{C}$ *for* D *is a conjunction of linear inequalities which involve* $\delta_1, \ldots, \delta_n$*. A set* $\mathcal{D}' \subseteq \mathcal{D}$ *fulfills* $\mathcal{C}$*, denoted as* $\mathcal{D}' \vdash \mathcal{C}$*, iff* $\mathcal{C}$ *is satisfied for variable assignments* $\delta_i = 1$ *if* $d_i \in \mathcal{D}'$ *and* $\delta_i = 0$ *if* $d_i \notin \mathcal{D}'$*.*

Our first constraint $\mathsf{markedAtMostOnce}(G_0)$ requires that each source and target element of a model pair G_0 be marked at most once, i.e., a choice between

alternative markings of the same element(s) is enforced. As a result of a consistency check, an element can either remain unmarked (due to inconsistency) or it can be marked once. Definition 7 introduces the sum of alternative markings of the same element and Definition 8 restricts it to 0–1 as a constraint.

Definition 7 (Sum of Alternative Markings for an Element). *Given a triple graph $G_0 = G_S \leftarrow \emptyset \rightarrow G_T$, let $D : G_0 \overset{*}{\Rightarrow} G_n$ be a derivation via consistency rules with the underlying set $\mathcal{D}$ of direct derivations. For each element $e \in \mathsf{elements}(G_0)$, let $\mathcal{E} = \{d \in \mathcal{D} \mid e \in \mathsf{marks}(d)\}$. The integer $\mathsf{markersSum}(e)$ denotes the sum of variables for each $d \in \mathcal{E}$ as follows:*
If $\mathcal{E} = \emptyset$, $\mathsf{markersSum}(e) = 0$. If $\mathcal{E} = \{d_1\}$, $\mathsf{markersSum}(e) = \delta_1$.
If $\mathcal{E} = \{d_1, \ldots, d_n\}$, $\mathsf{markersSum}(e) = \delta_1 + \ldots + \delta_n$.

Definition 8 (Constraint 1: Marking Each Element at Most Once). *Given a triple graph $G_0 = G_S \leftarrow \emptyset \rightarrow G_T$, let $D : G_0 \overset{*}{\Rightarrow} G_n$ be a derivation via consistency rules with the underlying set $\mathcal{D}$ of direct derivations. The constraint $\mathsf{markedAtMostOnce}(G_0)$ denotes $\bigwedge_{e \in \mathsf{elements}(G_0)} \mathsf{markersSum}(e) \leq 1$.*

The next constraint $\mathsf{context}(D)$ defines dependencies as implications between direct derivations due to their required context: A direct derivation is either not chosen, or its required source and target elements must be marked and its required correspondences must be created by some other chosen direct derivations. This is necessary as each chosen marking should be traced back to a derivation by the original TGG rules, where the context must always be provided.

Definition 9 (Constraint 2: Providing Context for Markings). *Given a triple graph $G_0 = G_S \leftarrow \emptyset \rightarrow G_T$, let $D : G_0 \overset{*}{\Rightarrow} G_n$ be a derivation via consistency rules with the underlying set $\mathcal{D}$ of direct derivations. For each direct derivation $d_i \in D$, we define the following constraints:*
$\mathsf{contextSrcTrg}(d_i) = \bigwedge_{e \in \mathsf{requiresSrcTrg}(d_i)} \delta_i \leq \mathsf{markersSum}(e)$, $\mathsf{contextCorr}(d_i) = \bigwedge_{d_j \in \mathcal{D}, \mathsf{requiresCorr}(d_i) \cap \mathsf{creates}(d_j) \neq \emptyset} \delta_i \leq \delta_j$,
The constraint $\mathsf{context}(D)$ denotes $\bigwedge_{d_i \in \mathcal{D}} \mathsf{contextSrcTrg}(d_i) \wedge \mathsf{contextCorr}(d_i)$.

Example 3. In Fig. 5, the constraints $\mathsf{markedAtMostOnce}(G_0)$ and $\mathsf{context}(D)$ are depicted (after some logical simplifications) for our example.

The constraint $\mathsf{context}(D)$ ensures that the context for each chosen direct derivation is supplied but *cycles* must still be avoided. Intuitively, two chosen direct derivations may not provide context for each other (also not transitively) as such derivations cannot be sequenced in terms of the underlying TGG.

Definition 10 (Cyclic Markings). *Let $D : G_0 \overset{*}{\Rightarrow} G_n$ be a derivation via consistency rules with the underlying set $\mathcal{D}$ of direct derivations. We define a relation $\rhd \subseteq \mathcal{D} \times \mathcal{D}$ between two direct derivations $d_i, d_j \in \mathcal{D}$ as follows: $d_i \rhd d_j$ iff $\mathsf{requiresSrcTrg}(d_i) \cap \mathsf{marks}(d_j) \neq \emptyset$ or $\mathsf{requiresCorr}(d_i) \cap \mathsf{creates}(d_j) \neq \emptyset$. A sequence $cy \subseteq \mathcal{D}$ with $cy = \{d_1, \ldots, d_n\}$ of direct derivations is a* cycle *iff $d_1 \rhd \ldots \rhd d_n \rhd d_1$.*

Definition 11 (Constraint 3: Eliminating Cycles). *Given a triple graph* $G_0 = G_S \leftarrow \emptyset \rightarrow G_T$*, let* $D : G_0 \overset{*}{\Rightarrow} G_n$ *be a derivation via consistency rules with the underlying set* $\mathcal{D}$ *of direct derivations and let* $\mathcal{CY}$ *be the set of all cycles* $cy \subseteq \mathcal{D}$*. We define a constraint* acyclic(D) *as follows:*

$$\mathsf{acyclic}(D) = \bigwedge_{\substack{cy \in \mathcal{CY}, \\ cy = \{d_1, \ldots, d_n\}}} \delta_1 + \ldots + \delta_n < |cy| \text{ where } |cy| \text{ is the cardinality of } cy.$$

Example 4. The derivation depicted in Fig. 5 exhibits no cycles. In Fig. 6, however, two direct derivations (d_2 and d_3, both via the consistency rule cr_2 derived from r_2) mark each others required elements ($d_2 \triangleright d_3$ and $d_3 \triangleright d_2$).

Given two classes List and Queue with cyclic inheritance relation on both sides, d_2 marks the Queue classes and requires List classes, and conversely for d_3. Although d_2 and d_3 mark the model pair entirely (without being alternatives to each other for any element), they cannot be chosen together as they cannot be sequenced in terms of the original grammar. In fact, these models are inconsistent (they just exhibit the same type of inconsistency on both sides) as our TGG (in particular the triple rule r_2) cannot create cyclic inheritance relations.

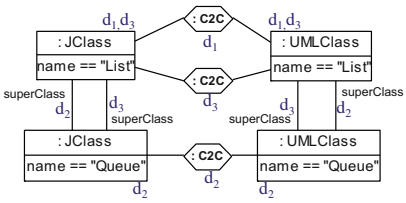


Fig. 6. Cyclic markings of d_2 and d_3

Our constraints so far enforce that (i) each element is marked at most once, (ii) chosen direct derivations completely satisfy their context with other direct derivations, and (iii) direct derivations do not provide context in a cyclic manner to each other. Theorem 1 in the following states that, given a model pair $G_0 = G_S \leftarrow \emptyset \rightarrow G_T$ and a derivation D via consistency rules, each subset of direct derivations in D satisfying these constraints leads to a triple graph representing a consistent portion of G_S and G_T. The consistent triple graph consists of elements marked and created by the chosen subset of direct derivations.

Theorem 1 (Consistent Portions of Source and Target Graphs). *Given a TGG* $: (TG, \mathcal{R})$ *with the set* $\mathcal{CR}$ *of respective consistency rules and a triple graph* $G_0 = G_S \leftarrow \emptyset \rightarrow G_T$*, let* $D : G_0 \overset{*}{\Rightarrow} G_n$ *be a derivation via rules in* $\mathcal{CR}$ *with the underlying set* $\mathcal{D}$ *of direct derivations. For any set* $\mathcal{D}' \subseteq \mathcal{D}$ *with* $\mathcal{D}' \vdash$ markedAtMostOnce$(G_0) \wedge$ context$(D) \wedge$ acyclic(D)*, we get a triple graph* $G' = G'_S \leftarrow G'_C \rightarrow G'_T$ *such that* $G' \in \mathcal{L}(TGG)$*,* elements$(G'_S) \subseteq$ elements(G_S)*,* elements$(G'_T) \subseteq$ elements(G_T)*, and* elements$(G') = \bigcup_{d \in \mathcal{D}'}$ (marks$(d) \cup$ creates(d))*.*

Proof. For each direct derivation d in $\mathcal{D}'$, the required source and target elements are marked and the required correspondences are created by some other direct derivations in $\mathcal{D}'$ (context(D)). Furthermore, direct derivations in $\mathcal{D}'$ provide context for each other in an acyclic manner (acyclic(D)). Hence, all direct derivations in $\mathcal{D}'$ can be sequenced to a derivation D' via rules in $\mathcal{CR}$. Marked

and created elements of each direct derivation in D' are equal to created elements by the respective original triple rule (cf. consistency rule construction in Definition 4). Consequently, the union of marked and created elements of D' leads to a triple graph $G' = G'_S \leftarrow G'_C \rightarrow G'_T \in \mathcal{L}(TGG)$. Moreover, G'_S and G'_T are composed by picking each element of G_S and G_T at most once as $\mathsf{markedAtMostOnce}(G_0)$ holds, i.e., we get $\mathsf{elements}(G'_S) \subseteq \mathsf{elements}(G_S)$ and $\mathsf{elements}(G'_T) \subseteq \mathsf{elements}(G_T)$. □

In practice, when applying Theorem 1, we employ an ILP solver together with an objective maximizing the number of marked elements as depicted in Fig. 5. Consistency of two models can be concluded if the maximally marked portions in Theorem 1 are equal to the entire models.

Corollary 1 (A Sufficient Condition for Consistency). *Given a TGG* $: (TG, \mathcal{R})$ *with the set* $\mathcal{CR}$ *of respective consistency rules and a triple graph* $G_0 : G_S \leftarrow \emptyset \rightarrow G_T$*, let* $D : G_0 \overset{*}{\Rightarrow} G_n$ *be a derivation via rules in* $\mathcal{CR}$ *with the underlying set* $\mathcal{D}$ *of direct derivations.* G_S *and* G_T *are consistent if a set* $\mathcal{D}' \subseteq \mathcal{D}$ *exists with* $\mathcal{D}' \vdash \mathsf{markedAtMostOnce}(G_0) \wedge \mathsf{context}(D) \wedge \mathsf{acyclic}(D)$ *and* $\bigcup_{d \in \mathcal{D}'} \mathsf{marks}(d) = \mathsf{elements}(G_0)$.

Proof. This is a special case of Theorem 1 where elements marked and created by direct derivations in $\mathcal{D}'$ result in $G_S \leftarrow G_C \rightarrow G_T \in \mathcal{L}(TGG)$. □

Example 5. In Fig. 5, two models G_S and G_T are given together with a derivation that marks some model elements multiple times. A subset of direct derivations satisfying the constraints is then determined leading to a triple graph $G_S \leftarrow G_C \rightarrow G_T$, i.e., G_S and G_T are marked entirely.

Corollary 1 is a sufficient condition for consistency and already useful to conclude consistency from arbitrarily collected markings. If no subset of markings in a derivation D is found that satisfies constraints and marks all elements, however, it is unclear if the models are really inconsistent or if there are some further markings that were not collected in D. We thus characterize *final* derivations with consistency rules providing *all possible markings*, and lift our result to a sufficient and necessary condition for consistency. We restrict ourselves in the following to TGGs whose consistency rules mark at least one element (called *progressive* TGGs). Consistency rules that only create correspondences but do not contribute any markings are excluded as it is unclear how often to apply such rules for collecting a complete set of markings. This restriction does not have any significant consequence in practice according to our experience, and is fulfilled by all industrial and academic case studies we have worked on so far.

Definition 12 (Progressive TGG). *A TGG* $: (TG, \mathcal{R})$ *with the set* $\mathcal{CR}$ *of respective consistency rules is* progressive *iff each* $cr \in \mathcal{CR}$ *has at least one marking element.*

Definition 13 (Final Derivations with Consistency Rules). *Given a progressive TGG* $: (TG, \mathcal{R})$ *with the set* $\mathcal{CR}$ *of respective consistency rules and a triple graph* $G_0 : G_S \leftarrow \emptyset \rightarrow G_T$*, let* $D : G_0 \overset{*}{\Rightarrow} G_n$ *be a derivation via rules in* $\mathcal{CR}$ *with the underlying set* $\mathcal{D}$ *of direct derivations. D is* final *iff* $\forall d_{n+1} : G_n \overset{cr_{n+1}@cm_{n+1}}{\Longrightarrow} G_{n+1}$ *with* $cr_{n+1} \in \mathcal{CR}$, $\exists d_i : G_{i-1} \overset{cr_i@cm_i}{\Longrightarrow} G_i$ *where* $d_i \in \mathcal{D}$, $cr_i = cr_{n+1}$, *and* $cm_i = cm_{n+1}$.

Remark 1. An interesting issue is the existence of a final derivation for a given *TGG* and a model pair. In some cases, the search for a final derivation does not terminate when consistency rules create new matches for each other in a cyclic manner (e.g., in case of a cyclic inheritance in our example as depicted in Fig. 6, direct derivations via cr_2 continuously create new correspondences and thus new matches for each other). The problem is similar to the termination problem of graph grammars which is in general undecidable [23]. In practice, such cycles can either be detected and aborted at runtime, or additional restrictions for the model pair or for the TGG can be imposed. For example, a TGG can be specified in the style of a Layered Graph Grammar [5,24] whose termination with distinct matches is shown in [5], or models can be constrained to avoid cyclic matches (cyclic inheritance must be prohibited in our concrete case). We leave it to future work to explore a restricted yet sufficiently expressive class of TGGs (statically) guaranteeing the existence of a final derivation.

A final derivation provides all possible markings. In this case, inconsistency can be concluded if a subset of direct derivations satisfying our constraints and marking all elements does not exist. Corollary 2 in the following thus extends our result from Corollary 1 to a sufficient and necessary condition for final derivations via consistency rules of progressive TGGs.

Corollary 2 (A Sufficient and Necessary Condition for Consistency). *Given a progressive TGG* $: (TG, \mathcal{R})$ *with the set* $\mathcal{CR}$ *of respective consistency rules, and a triple graph* $G_0 : G_S \leftarrow \emptyset \rightarrow G_T$*, let* $D : G_0 \overset{*}{\Rightarrow} G_n$ *be a final derivation via rules in* $\mathcal{CR}$ *with the underlying set* $\mathcal{D}$ *of direct derivations.* G_S *and* G_T *are consistent iff a set* $\mathcal{D}' \subseteq \mathcal{D}$ *exists with* $\mathcal{D}' \vdash \mathsf{markedAtMostOnce}(G_0) \wedge \mathsf{context}(D) \wedge \mathsf{acyclic}(D)$ *and* $\bigcup_{d \in \mathcal{D}'} \mathsf{marks}(d) = \mathsf{elements}(G_0)$.

Proof. If $\mathcal{D}'$ exists, the same arguments as in Corollary 1 apply to conclude consistency of G_S and G_T. We show in the following that inconsistency can be concluded if $\mathcal{D}'$ does not exist: *TGG* is progressive and D is final. Hence, there does not exist any further direct derivation via consistency rules that contribute new markings with a different match to D. If $\mathcal{D}'$ does not exist, there does not exist any derivation D' via consistency rules whose marked and created elements compose a triple graph $G_S \leftarrow G_C \rightarrow G_T \in \mathcal{L}(TGG)$. As a result of the consistency rule construction (Definition 4), furthermore, for each derivation via original triple rules in $\mathcal{R}$ there exists a unique derivation via consistency rules in $\mathcal{CR}$. Thus, the absence of D' leads to the absence of a derivation via triple rules in $\mathcal{R}$, i.e., G_S and G_T cannot be constructed together by the grammar. □

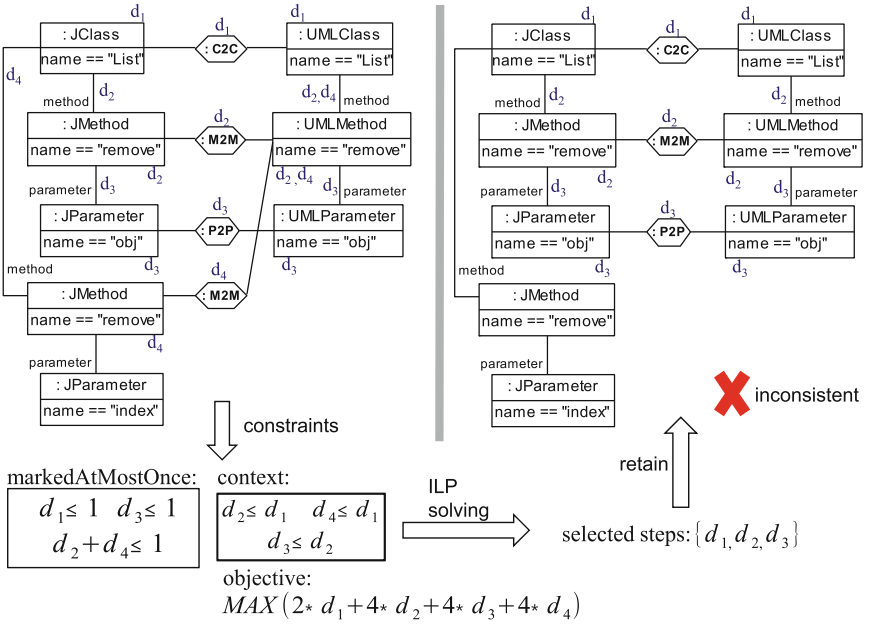


Fig. 7. A further example with inconsistent models

Example 6. Figure 7 shows an example where inconsistency of two models is concluded (we have less methods on the UML side). In the upper left part, we have a final derivation consisting of four direct derivations where d_2 and d_4 mark the same remove method on the UML side. The upper right part depicts a subset of direct derivations satisfying our constraints with the maximum number of marked elements (d_2 is preferred over d_4 in order to mark obj parameters with d_3). Having still unmarked elements on the Java side, however, the models are identified to be inconsistent. Nevertheless, the retained markings and correspondences refer to the maximum consistent portions of these models.

4 Experimental Evaluation

Our goal in this section is to evaluate the applicability of our tool support for consistency checking with regard to performance. To this end, we state the following two research questions to be investigated with our experiments:

RQ1: Are consistency checks by combining TGGs and linear optimization applicable to real-world model pairs?

RQ2: How is the scalability of our implementation affected by different factors including model size and numbers of collected/chosen marking steps?

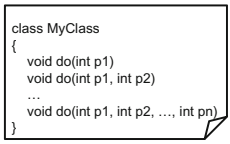
Evaluation set-up. We approach both research questions with an extended version of our running example. We extracted Java and UML model pairs from

real and synthetically generated software projects using the MoDisco tool [21] and performed consistency checks using our TGG tool. Our TGG tool collects alternative markings between two models and utilizes an ILP solver, namely Gurobi [13], for a decision in retrospect (we chose Gurobi due to its performance, available academic licence, and Java API). The TGG in our experiments has 17 rules and relates packages, types, attributes, methods, and parameters on both sides. Method bodies in Java models are ignored as they do not have any counterpart in UML models. In all cases, the only inconsistency detected with our TGG was the primitive type string in UML models (which is not primitive in Java). We repeated our measurements 15 times with Intel i5@3.30 GHz, Windows 7 (64 bit), Java 8, Eclipse Neon, and 15 GB memory, and show the median.

Evaluation results and discussion. The upper part of Table 1 shows measurement results with four real software projects with diverse sizes. The number of marked source elements is generally larger than the number of marked target elements as Java models represent the same information with more vertices and edges as compared to UML models. Moreover, there is always a difference between the number of all marking steps and the number of chosen marking steps (as a result of the optimization problem) due to alternative markings of overloaded methods as we have exemplified throughout the paper. Especially the project modisco.java makes intensive usage of method overloading and is thus the most noticeable one among our real software projects with respect to this difference (ca. 3.8 K of 30 K marking steps are chosen). In all experiments with real software projects, ILP solving requires under 1 s while collecting all markings requires between 5 s and 2.5 min depending on the model size. Removing eliminated markings and correspondences has negligible runtime.

Table 1. Measurement results with real and synthetically generated software projects

	Model size		ILP problem size		Runtime			
	# src. elts.	# trg. elts.	# all marking steps	# chosen steps	Collect markings (sec)	ILP solving (sec)	Retain chosen (sec)	Total (sec)
tgg.core	8,484	5,594	2,007	1,919	**5.29**	**0.11**	**0.01**	**5.39**
modisco.java	16,705	11,279	29,977	3,791	**15.28**	**0.94**	**0.12**	**16.34**
eclipse.graphiti	33,778	21,778	8,819	7,271	**63.20**	**0.36**	**0.03**	**63.60**
eclipse.compare	53,391	31,912	11,670	10,700	**143.63**	**0.42**	**0.03**	**144.09**
Synthetic ($n = 25$)	2,300	1,081	6,162	362	**1.70**	**0.51**	**0.05**	**2.26**
Synthetic ($n = 50$)	8,950	4,006	45,437	1,337	**6.18**	**7.64**	**0.31**	**14.13**
Synthetic ($n = 75$)	19,975	8,806	149,087	2,937	**23.53**	**53.50**	**2.64**	**79.665**
Synthetic ($n = 100$)	35,375	15,481	348,362	5,162	**67.51**	**201.33**	**13.89**	**282.73**
Synthetic ($n = 125$)	55,150	24,031	674,512	8,012	**164.70**	**674.08**	**51.91**	**890.69**



In order to explore the limits of the ILP solver (which is not challenged by real models), we furthermore generated synthetic projects consisting of a class with n overloaded methods including 1 to n parameters. We used the same naming convention for all parameters as depicted to the right. With this strategy, we get n^2 possible markings for methods where n of them must be chosen. For parameters, the number of all markings is given by $\sum_{i=1}^{n} i^2$ and the number of chosen ones by $\sum_{i=1}^{n} i$. In all cases, there are 12 further alternativeless markings for (primitive) types and packages. Measurement results in the lower part of Table 1 show that collecting markings requires similar runtime as in real models of similar size, while ILP solving requires more than 3 min for $n = 100$ (ca. 5 K of 348 K markings are chosen) and more than 11 min for $n = 125$ (ca. 8K of 674 K markings are chosen). This time, removing eliminated markings has also observable runtime (52 s) in the largest case.

In line with our results, we conclude the following for **RQ1** and **RQ2:**

RQ1: Our approach is applicable to realistic models, terminating in the order of only a few minutes for large models with up to 50 K elements. ILP solving can easily cope with our specific type of constraints and objectives if the number of alternative markings is not exceptionally large. Collecting all markings, however, is currently the limiting factor for applicability to larger models.

RQ2: Scalability of collecting all markings strictly depends on the model size but not necessarily on the number of collected markings. This step shows similar runtime behaviour for real and synthetic projects although much more markings are collected in the latter. Apparently, searching for markings between large models (*pattern matching*) is the most costly operation which we also confirmed by profiling. Conversely, scalability of ILP solving has a strict dependency on the number of collected markings (as they form together the optimization problem).

Finally, we believe to have come up with a consistency checking approach which (i) is already applicable to realistic models in its current form, (ii) has reasonable runtime even for corner cases with lots of alternative markings, and (iii) has potential for improvements, especially with respect to pattern matching.

Threats to validity. *External validity* is our primary concern as generalizability of our results requires further non-trivial case studies. We argue, nevertheless, that our synthetically generated models address ultimately challenging cases for their sizes. Furthermore, expectations and research interests of the authors may be a threat to *conclusion validity*. We thus used real-world and randomly chosen models to make experiments unpredictable and carefully utilized profiling tools to draw conclusions on the scalability of individual components.

5 Related Work

We consider two groups of related work: (i) consistency checking approaches in MDE and (ii) MDE-related applications of *optimization techniques*.

Consistency checking approaches. QVT-R [22] proposed by OMG is the only current standard for consistency checking and describes consistency as a set of relations between two models. Seminal contributions to QVT-R, however, primarily address its ambiguous semantics due to a missing formalization in the standard. In [26], a game-theoretic approach is proposed to define semantics of consistency checking with QVT-R, later extended by recursive relations in [1]. In this setting, consistency checking is a game between a verifier and a refuter whose interest is to satisfy or to contradict relations, respectively. In [12], QVT-R is translated to graph constraints using similar formal foundations as for TGGs. Due to the nature of QVT-R, however, consistency must be designed in two directions in these formalisms (there is a forward and backward consistency check) and direction-agnostic traceability information is not provided [26]. An interesting approach is proposed in [18,19] defining QVT-R semantics as a constraint solving problem. While constraint solving is employed for entire models in this case, our approach is in contrast rule-based and formulates only decisions between rule applications as constraints. Our constraints are thus more compact and manageable for state-of-the-art solvers. This claim is supported by the order of processable model sizes of our approach (currently up to 50 K elements) as compared to experimental results in [19] (hundreds of elements). It is, nonetheless, crucial to establish benchmarks for a direct comparison of different approaches. Considering recent work on TGGs, reusing existing markings and correspondences from former runs is proposed in [7,14] when relating two models. Decision making for remaining parts, however, is still open and can be tackled with our approach. Combining our approach with [7,14] could yield performance gains via incrementality and is thus important future work.

Optimization techniques in MDE. We observe a close relation between our work and [10] which combines search-based optimization techniques with model transformation. Given a set of rules, input model(s), and an objective, the idea is to calculate an "optimal" sequence of rule applications via search-based algorithms. Interestingly, this can reverse the complexity distribution of our approach: While we invest substantial effort in rule applications (collecting all markings) and solve a rather simple optimization problem (at least in case of realistic models) in retrospect, more effort is put into optimization in [10] and necessary rule applications are determined in advance. Different MDE tasks are addressed with search-based optimization including change detection [9] and refactoring [11]. Further investigation is needed to understand to what extent the same methodologies are applicable to our goals. Other applications of optimization techniques in MDE include bidirectional model transformation [2] and learning model transformation by examples [15]. Applicability to large models, however, is again a critical limitation in these cases as the papers openly discuss.

6 Conclusion and Future Work

We presented an approach to inter-model consistency checking by combining TGGs with linear optimization techniques. We evaluated our respective tool

support and explored its scalability with realistic and synthetically generated models. Our results show that the idea of combining a model transformation engine with optimization techniques is promising and we believe that it can be transferred to other approaches (e.g., QVT-R) to facilitate decision making. Tasks for future work include (i) experimenting with further industrial case studies as well as with academic non-trivial examples as collected in the *bx* example repository [3], (ii) comparing our approach to hand-written solutions of the same problem (developed with general purpose or bidirectional programming languages such as [16]), (iii) utilizing novel pattern matching techniques (e.g., [27,28]) to attain applicability to larger models, (iv) incremental consistency checking by reusing results from former runs, and (v) exploring new types of optimization problems beyond identifying maximal consistent portions and represent case-specific policies. Finally, our contribution paves the way to *bidirectional model integration*. Starting with two inconsistent models, consistent parts can be detected with the current contribution. Remaining parts can be synchronized again by TGGs (possibly after a conflict resolution).

Acknowledgement. This work has been funded by the German Federal Ministry of Education and Research within the Software Campus project GraTraM at TU Darmstadt, funding code 01IS12054.

References

1. Bradfield, J., Stevens, P.: Recursive checkonly QVT-R transformations with general *when* and *where* clauses via the modal Mu calculus. In: Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 194–208. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28872-2_14
2. Callow, G., Kalawsky, R.: A satisficing bi-directional model transformation engine using mixed integer linear programming. J. Object Technol. **12**(1), 1–43 (2013)
3. Cheney, J., McKinna, J., Stevens, P., Gibbons, J.: Towards a repository of BX examples. In: Candan, K.S., Amer-Yahia, S., Schweikardt, N., Christophides, V., Leroy, V. (eds.) BX 2014. CEUR Workshop Proceedings, vol. 1133, pp. 87–91 (2014). CEUR-WS.org
4. Ehrig, H., Ehrig, K., Hermann, F.: From model transformation to model integration based on the algebraic approach to triple graph grammars. ECEASST **10**, 1–15 (2008)
5. Ehrig, H., Ehrig, K., Lara, J., Taentzer, G., Varró, D., Varró-Gyapay, S.: Termination criteria for model transformation. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 49–63. Springer, Heidelberg (2005). doi:10.1007/978-3-540-31984-9_5
6. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2006)
7. Ehrig, H., Ermel, C., Golas, U., Hermann, F.: Graph and Model Transformation - General Framework and Applications. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2015)
8. Ermel, C., Hermann, F., Gall, J., Binanzer, D.: Visual modeling and analysis of EMF model transformations based on triple graph grammars. ECEASST **54**, 1–12 (2012)

9. Fadhel, A.B., Kessentini, M., Langer, P., Wimmer, M.: Search-based detection of high-level model changes. ICSM **2012**, 212–221 (2012)
10. Fleck, M., Troya, J., Wimmer, M.: Marrying search-based optimization and model transformation technology. In: Proceedings of NasBASE (2015)
11. Fleck, M., Troya, J., Wimmer, M.: Search-based model transformations with MOMoT. In: Van Gorp, P., Engels, G. (eds.) ICMT 2016. LNCS, vol. 9765, pp. 79–87. Springer, Cham (2016). doi:10.1007/978-3-319-42064-6_6
12. Guerra, E., de Lara, J.: An algebraic semantics for QVT-relations check-only transformations. Fundam. Inform. **114**(1), 73–101 (2012)
13. Gurobi: (2016). http://www.gurobi.com/
14. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y., Gottmann, S., Engel, T.: Model synchronization based on triple graph grammars: correctness, completeness and invertibility. Softw. Syst. Model. **14**(1), 241–269 (2015)
15. Kessentini, M., Sahraoui, H., Boukadoum, M.: Model transformation as an optimization problem. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 159–173. Springer, Heidelberg (2008). doi:10.1007/978-3-540-87875-9_12
16. Ko, H., Zan, T., Hu, Z.: BiGUL: a formally verified core language for putback-based bidirectional programming. In: Erwig, M., Rompf, T. (eds.) PEPM 2016, pp. 61–72 (2016)
17. Leblebici, E.: Towards a graph grammar-based approach to inter-model consistency checks with traceability support. In: Anjorin, A., Gibbons, J. (eds.) BX 2016. CEUR Workshop Proceedings, vol. 1571, pp. 35–39 (2016). CEUR-WS.org
18. Macedo, N., Cunha, A.: Implementing QVT-R bidirectional model transformations using alloy. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 297–311. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37057-1_22
19. Macedo, N., Cunha, A.: Least-change bidirectional model transformation with QVT-R and ATL. Softw. Syst. Model. **15**(3), 783–810 (2016)
20. Medini-QVT: (2016). http://projects.ikv.de/qvt
21. MoDisco: (2016). http://www.eclipse.org/MoDisco/
22. OMG: QVT Specification, V1.2 (2015). http://www.omg.org/spec/QVT/
23. Plump, D.: Termination of graph rewriting is undecidable. Fundam. Inform. **33**(2), 201–209 (1998)
24. Rekers, J., Schürr, A.: Defining and parsing visual languages with layered graph grammars. J. Vis. Lang. Comput. **8**(1), 27–55 (1997)
25. Schürr, A.: Specification of graph translators with triple graph grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer, Heidelberg (1995). doi:10.1007/3-540-59071-4_45
26. Stevens, P.: A simple game-theoretic approach to checkonly QVT relations. In: Paige, R.F. (ed.) ICMT 2009. LNCS, vol. 5563, pp. 165–180. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02408-5_12
27. Ujhelyi, Z., Bergmann, G., Hegedüs, Á., Horváth, Á., Izsó, B., Ráth, I., Szatmári, Z., Varró, D.: EMF-IncQuery: an integrated development environment for live model queries. Sci. Comput. Program. **98**, 80–99 (2015)
28. Varró, G., Deckwerth, F.: A Rete network construction algorithm for incremental pattern matching. In: Duddy, K., Kappel, G. (eds.) ICMT 2013. LNCS, vol. 7909, pp. 125–140. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38883-5_13