Towards a Graph Grammar-Based Approach to Inter-Model Consistency Checks with Traceability Support

Erhan Leblebici
Technische Universität Darmstadt, Germany
erhan.leblebici@es.tu-darmstadt.de

Abstract

Development of a complex system relies on different yet related *models* each representing the system from a particular perspective. In this respect, an important task is to *check consistency* between related models to guide subsequent decisions concerning consistency restoration. *Triple Graph Grammars* (TGGs), a particular dialect of graph grammars, are well-suited for describing consistency of two models together with correspondences. The grammar-based description leads to a precise consistency notion which is prerequisite for reliable consistency checks, and correspondences serve as explicit traceability information. Consistency checks with TGGs, however, turn out to be more difficult than consistency restoration in most cases and have not been addressed sufficiently so far. We first discuss why consistency checks with TGGs are worthwhile and identify backtracking issues making correct and efficient consistency checks challenging. Finally, we present two strategies to overcome these challenges, reflecting our work in progress towards a formally-founded consistency check approach with viable tool support.

1 Introduction and Motivation

Models are used in engineering disciplines to represent a system from a particular perspective and to abstract from irrelevant details. Depending on the variety of involved tools and domains in an engineering process, several models can exist that contain related information of the same system and thus must be kept *consistent* to each other. *Bidirectional transformations* (bx) address the challenge of consistency maintenance and play therefore an important role in model-driven landscapes. An important bx task is to perform a *consistency check* between related models to determine if (and to what extent) consistency restoration is necessary. We discuss consistency checks with *Triple Graph Grammars* (TGGs) [Sch94], a prominent and graph grammar-based bx language.

A TGG is a *grammar* whose rules construct *triples* of *graphs*. Besides two graphs representing two related models M_S and M_T (referred to as *source* and *target* model, respectively), TGGs produce a third model connecting these two, namely the *correspondence* model M_C. The grammatical characteristic of TGGs leads to a constructive, precise, and direction-agnostic notion of consistency: M_S and M_T are consistent to each other iff a triple $M_S \leftarrow M_C \rightarrow M_T$ can be constructed by the grammar. Hence, given M_S and M_T, the goal of a consistency check

In: A. Anjorin, J. Gibbons (eds.): Proceedings of the Fifth International Workshop on Bidirectional Transformations (Bx 2016), Eindhoven, The Netherlands, April 8, 2016, published at http://ceur-ws.org

is to find a valid M_C if there exists one. In case of inconsistency, a *partial* triple $M_S \supseteq M'_S \leftarrow M'_C \rightarrow M'_T \subseteq M_T$ must be explored to indicate which parts of M_S and M_T correspond to each other and which parts do not.

Related work. QVT-R [OMG15] is currently the only available standard for consistency checks. The main idea here is to specify a set of *relations* that must hold between M_S and M_T. The standard [OMG15] defines the semantics of QVT-R by translating it to QVT Core. The major problem, however, is that no sufficient formalization is provided for QVT-R and its translation to QVT Core. Consequently, scarce tool support can be observed and seminal contributions to QVT-R address defining clear semantics in the first place: In [Ste09], consistency checks with QVT-R are formalized as a game between a verifier and a refuter whose interest is to satisfy or to contradict relations, respectively. In [GdL12], QVT-R is translated to graph constraints which resort to similar formal techniques as TGGs (but without correspondences). In [MC13], QVT-R is translated to predicates over models and a solver is employed for consistency checks. Considering these state-of-the-art QVT-R approaches, the following three arguments motivate us to establish consistency checks with TGGs:

- Grammar-based consistency has a clear semantics (a model pair can either be constructed by the grammar or it cannot). Thus, given M_S, M_T, and a TGG, the consistency check is precisely and fully defined without having to translate the TGG to another formalism as done for QVT-R. Finding the correct answer efficiently is the only obstacle between theory and tool support where all TGG tools can share a common formal foundation.
- In QVT-R, consistency must be understood and checked separately in two directions. A direction-agnostic consistency notion as in TGGs, however, is arguably easier to manage for both bx tool developers and users.
- Correspondences in TGGs can be used for *traceability* purposes (e.g., change impact analysis). A partial correspondence model between inconsistent models, moreover, indicates the extent of consistency violation. QVT-R formalizations do not support traceability (with the exception of a trace-based but restrictive game variant in [Ste09]) whereas [OMG15] uses implicit traces in the underlying QVT Core translation. As argued again in [Ste09], however, these traces do not capture both directions as they are created in two directions separately.

The pioneer work concerning consistency checks with TGGs is [EEH08] where *operational* grammar rules that create correspondences for two given models are derived from a TGG. How to apply these rules in a correct order, however, remains an open issue and, if done naively, requires backtracking in many cases as we shall point out in Section 2. In [HEO+15, EEGH15], furthermore, these operational grammar rules are extended such that *existing correspondences* between two models are examined and only missing ones are created (backtracking is still required for missing correspondences). In this regard, backtracking issues can actually be mitigated when consistency checks (and correspondence creations) are performed frequently after every small modification on models and exiting correspondences are used in each run to reduce backtracking points. However, consistency checks between two large models that do not have any correspondences in between (as they are not necessarily maintained in a TGG-based environment from the beginning) are not sufficiently addressed. We focus on this case as consistency checks are typically not considered and performed in every intermediate step of a development process but rather after models have reached an advanced state.

Arguably due to backtracking issues, TGG tools generally do not support consistency checks. The only exception is HenshinTGG[1] whose consistency checks, however, only work if no decision point to be backtracked exists and fail otherwise as is already the case for simple TGGs we have experimented with. To improve the situation, we present in Section 3 two viable strategies towards providing fully-fledged tool support.

2 Background, Example, and Wrong Choices of Correspondences

Our example[2] deals with the consistency between UML operations with parameters and their Java counterparts. The left part of Figure 1 shows a consistent model pair (ignore the correspondences in the middle for now). The UML model as well as the Java model represent two operations, both named `substring`. Both operations have a parameter `beginIndex` whereas one of the operations has an additional parameter `length`. In general, consistent models in our example are simply isomorphic (this already suffices to reveal the complexity of consistency checks).

The right part of Figure 1 shows two *TGG rules* to create UML and Java models with correspondences in between, constituting thus a grammar for consistent triples of models. TGG rules are monotonic, i.e., they only create elements and never delete. Created elements in a rule are depicted green with a ++-markup where black elements represent the context required to apply the rule. The first rule (`OperationRule`) creates a pair of UML and Java operations with equal names and a correspondence in between. Similarly, the second rule (`ParameterRule`) creates a pair of UML and Java parameters for an existing pair of operations.

[1] http://github.com/de-tu-berlin-tfs/Henshin-Editor/wiki/Manual-for-HenshinTGG-Editor

[2] Available in the bx example repository at http://bx-community.wikidot.com/examples:umloperationstojavaoperations

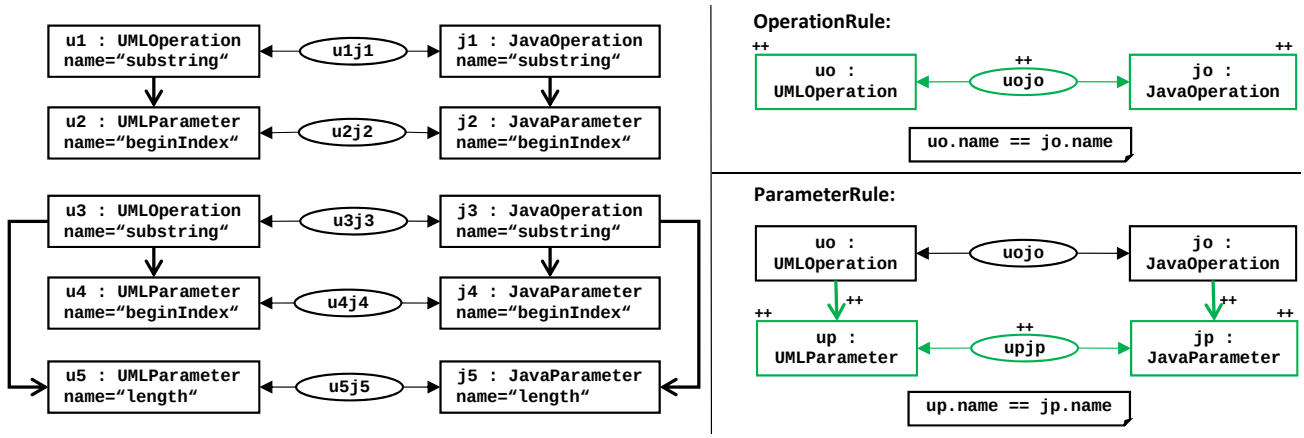

Figure 1: A consistent model pair (left) and two TGG rules that construct consistent models (right)

While TGG rules create models simultaneously, *operational* rules are automatically derived which do not create all models but *mark* (i.e., process) existing ones and create missing ones depending on the operational scenario. In a forward transformation, for example, an existing source model is marked while a correspondence and a target model are created (backward analogously). In a consistency check, an existing source and target model are marked pairwise and a correspondence model is created. Consistency check succeeds if both models are marked completely. Figure 2 depicts the consistency check rules derived for our TGG according to the formalization in [EEH08]. Markings in consistency check rules practically simulate creations in the original TGG rules (depicted as □ → ☑ which simply replaces the ++-markup). Applying these rules, the correspondences shown to the left of Figure 1 can be created and the model pair is then identified to be consistent. As shown in [EEH08], moreover, an application sequence of consistency check rules always exists to mark consistent models completely. How to find this sequence, however, remains an open issue and turns out to be challenging.

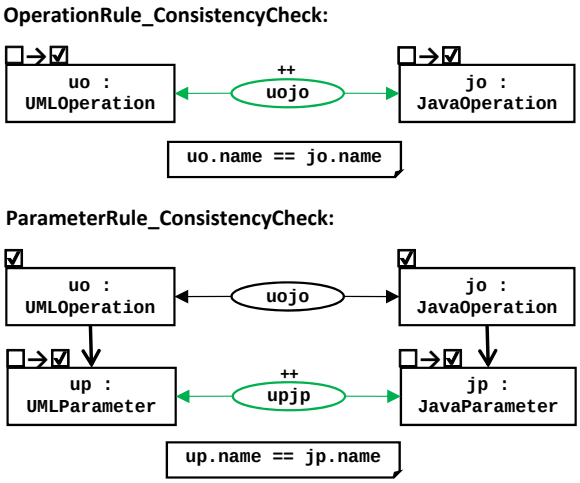

Figure 2: Consistency check rules

In Figure 3, an undesired outcome of applying the consistency check rules in Figure 2 to our model pair is depicted. Consistency check fails in this case although the models are actually consistent. The problem is that individual consistency check rules can connect wrong pairs of model elements. For example, a wrong pair of `substring` operations is connected in Figure 3. As a consequence, the `length` parameters on both sides remain unmarked without correspondence (as their parents are mistakenly not connected). Obviously, consistency checks with TGGs require backtracking in general if no appropriate control mechanism is used to govern correspondence choices. Interestingly, most of the wrong decisions are not relevant for forward (or backward) transformations but only for consistency checks. That is, pairwise marking of model elements introduces new decision points other than those in one-sided marking and leads a consistency check easily to an incorrect result.

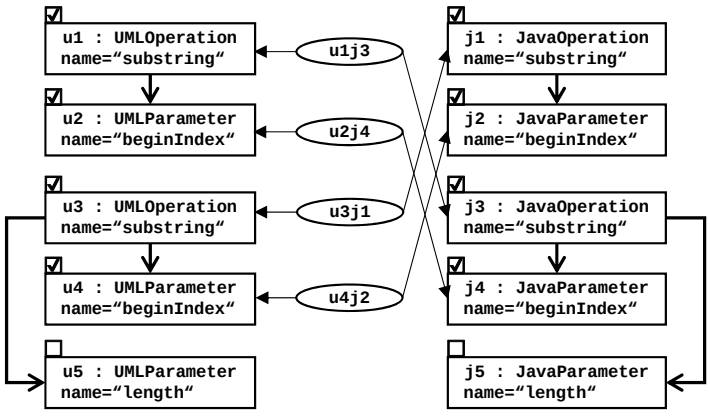

Figure 3: Undesired consistency check result

3 Strategies for Reliable and Scalable Consistency Checks

We do not consider backtracking to be a satisfactory solution. Firstly, backtracking may repeatedly discard and produce the same correspondences until a wrong decision in an arbitrary depth is corrected. Secondly, and more critically, inconsistent models directly lead to exhaustive backtracking without success whereas one of the discarded attempts may actually be of interest for a partial correspondence model. We instead discuss two strategies for reliable and scalable consistency checks without backtracking.

Create all then filter. A viable strategy against backtracking is to create all possible correspondences (including undesired ones) between two models in a brute-force manner and to determine a correct subset automatically in retrospect. In this case, some source and target elements are considered to be marked multiple times as demonstrated in Figure 4 for our example. A subset of correspondences must be then determined such that each element is marked exactly once for consistent models or at most once for inconsistent models. Fortunately, *dependencies* between correspondences can be used for an automated decision: Some correspondences are *essential* as they are the only ones marking their connected elements, e.g., `u5j5` in Figure 4. Some correspondences *imply* others, e.g., `u5j5` implies `u3j3` as it can only be created under `u3j3`. Finally, some correspondences are mutually exclusive *alternatives*, e.g., `u3j3` and `u3j1` as they both mark `u3`. A Boolean formula consisting of three parts can describe these dependencies as follows ($\Rightarrow$ denotes *implication* and $\oplus$ denotes *xor*):

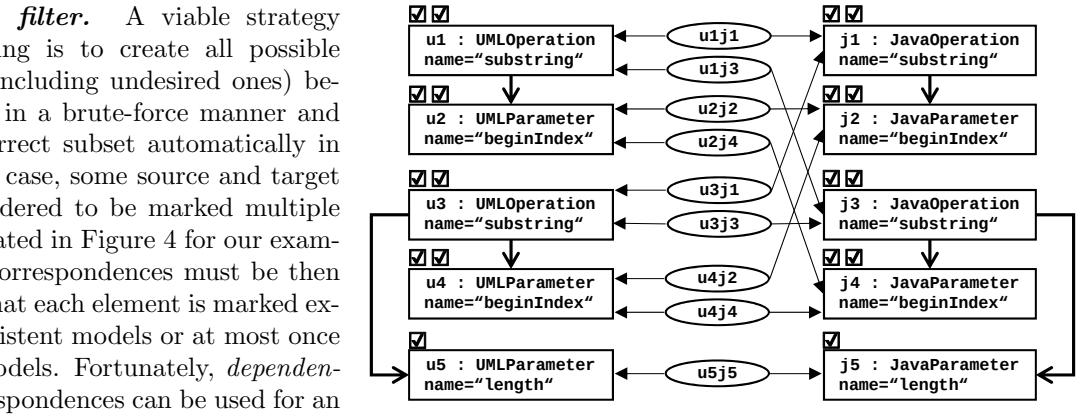

Figure 4: All possible correspondences

Essentials: `u5j5`

Implications: $(\texttt{u2j2} \Rightarrow \texttt{u1j1}) \wedge (\texttt{u4j4} \Rightarrow \texttt{u3j3}) \wedge (\texttt{u5j5} \Rightarrow \texttt{u3j3}) \wedge (\texttt{u2j4} \Rightarrow \texttt{u1j3}) \wedge (\texttt{u4j2} \Rightarrow \texttt{u3j1})$

Alternatives: $(\texttt{u1j1} \oplus \texttt{u1j3}) \wedge (\texttt{u1j1} \oplus \texttt{u3j1}) \wedge (\texttt{u2j2} \oplus \texttt{u2j4}) \wedge (\texttt{u2j2} \oplus \texttt{u4j2}) \wedge (\texttt{u3j3} \oplus \texttt{u3j1}) \wedge (\texttt{u3j3} \oplus \texttt{u1j3}) \wedge (\texttt{u4j4} \oplus \texttt{u4j2}) \wedge (\texttt{u4j4} \oplus \texttt{u2j4})$

Choosing the essential correspondence `u5j5`, from implications follows that `u3j3` must also be in the set of chosen correspondences. Consequently, `u3j1` and `u1j3` (alternatives of `u3j3`) as well as their implying correspondences `u2j4` and `u4j2` are excluded. From further implications and alternatives follows that `u1j1`, `u2j2`, and `u4j4` must be chosen as well. In general, the satisfaction of such a formula can either be outsourced to a solver or it can be implemented with a hand-crafted procedure optimized for the specific purpose of correspondence analysis. Two main advantages against backtracking are: (i) all (wrong and correct) correspondences are created only once in a progressive search and not repeatedly, and (ii) a partial correspondence model in case of inconsistency is not discarded and can be determined from all possible correspondences in the same manner. Nevertheless, the scalability of this strategy (in terms of runtime and memory consumption) strictly depends on the number of alternative correspondences. Scalable consistency checks can be expected when alternative correspondences only occur locally between two models and not in a combinatorial manner between all elements.

Heuristic-based or case-specific look-ahead. When the number of alternative correspondences grows rapidly, it is more advantageous to make decisions among correspondences already at rule application time (and not in retrospect after all possible correspondences are created). Collecting a set `C` of possible correspondence candidates that would mark a particular model element `e`, a consistency check procedure must provide and utilize a function `choose(C,e) = c` that takes `C` and `e` as input, looks ahead for upcoming elements, and returns the *best* correspondence $\texttt{c} \in \texttt{C}$. Beginning with the UML operation `u1` in our exemplary model pair, `choose({u1j1, u1j3},u1) = u1j1` already leads the consistency check procedure to a correct sequence of later correspondence creations, i.e., no other undesired correspondence occurs in the process after connecting the first UML operation to its correct counterpart in the Java model. In general, when choosing a correspondence connecting two elements, the underlying look-ahead mechanism can be supported by heuristics such as the conformance of (transitive) child elements on both sides, ideally grouped according to any relevant type and attribute information. Case-specific behaviour of `choose`, nevertheless, must be allowed by a TGG tool that supports consistency checks and implemented by the TGG designer who should be aware of the discussed

backtracking issues. An interactive implementation of `choose` where the ultimate user (who executes a consistency check) is confronted with these choices should only be considered if there is no feasible automatic decision.

4 Conclusion and Future Work

Two particular strategies are presented to avoid backtracking in consistency checks: one brute-force strategy with automated decisions and one look-ahead strategy relying on heuristics or case-specific expertise. It is also worthwhile to consider a combination of both, i.e., eliminating some alternative correspondences via a look-ahead and creating all others for an automated decision. Our next goal is to experiment with these strategies and to come up with tool support which will then be evaluated in industrial consistency tasks. Finally, it is also worthwhile to integrate these strategies into cases where consistency checks start with existing correspondences by utilizing further operational rules of [HEO+15, EEGH15].

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

[EEGH15] Hartmut Ehrig, Claudia Ermel, Ulrike Golas, and Frank Hermann. *Graph and Model Transformation - General Framework and Applications*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2015.

[EEH08] Hartmut Ehrig, Karsten Ehrig, and Frank Hermann. From Model Transformation to Model Integration based on the Algebraic Approach to Triple Graph Grammars. *ECEASST*, 10, 2008.

[GdL12] Esther Guerra and Juan de Lara. An Algebraic Semantics for QVT-Relations Check-only Transformations. *Fundam. Inform.*, 114(1):73–101, 2012.

[HEO+15] Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin, Yingfei Xiong, Susann Gottmann, and Thomas Engel. Model synchronization based on triple graph grammars: correctness, completeness and invertibility. *Software and System Modeling*, 14(1):241–269, 2015.

[MC13] Nuno Macedo and Alcino Cunha. Implementing QVT-R Bidirectional Model Transformations using Alloy. In Vittorio Cortelessa and Daniel Varro, editors, *FASE 2013*, volume 7793 of *LNCS*, pages 297–311. Springer, 2013.

[OMG15] OMG. QVT Specification, V1.2, 2015.

[Sch94] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In Ernst W. Mayr, Gunther Schmidt, and Gottfried Tinhofer, editors, *WG 1994*, volume 903 of *LNCS*, pages 151–163. Springer, 1994.

[Ste09] Perdita Stevens. A Simple Game-Theoretic Approach to Checkonly QVT Relations. In Richard F Paige, editor, *ICMT 2009*, volume 5563 of *LNCS*, pages 165–180. Springer, 2009.