Incremental Bidirectional Model Transformation with eMoflon::IBeX

Nils Weidmann, Anthony Anjorin
Paderborn University
{nils.weidmann, anthony.anjorin}@upb.de

Gergely Varró
Independent
gervarro@gmail.com

Lars Fritsche, Andy Schürr, Erhan Leblebici
Real-Time Systems Lab,
Technische Universität Darmstadt
{lars.fritsche, andy.schuerr, erhan.leblebici}@es.tu-darmstadt.de

Abstract

Consistency management is a crucial task in the context of model-driven engineering, as different teams of domain experts must be able to work concurrently with their different models. Triple Graph Grammars (TGGs) are a well-known approach to consistency management with the unique advantage of being declarative enough to address multiple consistency management operations with the same specification, while still achieving an acceptable level of scalability for realistic application scenarios. Although there have been numerous TGG-based tools in the past, to the best of our knowledge, there exists no TGG-based tool that addresses multiple consistency management operations in a conceptually and technically uniform manner. We argue that this is the reason why TGG-based tools typically do not maintain the same level of support for more than one or two such operations. In this paper, therefore, we present *eMoflon::IBeX*, a novel TGG-based tool that is able to handle model generation, model synchronisation, and consistency checking by applying essentially the same approach.

1 Introduction and Motivation

Triple Graph Grammars (TGGs) [Sch94] are a grammar-based approach to specifying and maintaining a consistency relation over two modelling languages. As a *bidirectional transformation* (bx) language, TGGs can be viewed as a practical implementation of the delta-lens framework [DXC+11, Anj16], based on the mature theory of algebraic graph transformation [EEPT06]. A TGG consists of *triple rules* specified by a developer describing how triples of connected models (source, target, and correspondence) consistently evolve together. Is a triple consistent with respect to a given TGG? Yes, if and only if it can be generated using the triple rules of the TGG. The main strength of the TGG approach is that a wide range of different consistency management operations (cf. Sect. 2) can be automatically derived from the exact same grammar. While constraint-based approaches to bx [MGC13, CREP10] might support a similar if not wider range of consistency management operations based on the same, concise specification, TGG-based tools typically scale substantially better than tools that rely fully on generic constraint solvers [MGC13]. Furthermore, while approaches to bx that focus on only a few consistency

 In: J. Cheney, H-S. Ko (eds.): Proceedings of the Eighth International Workshop on Bidirectional Transformations (Bx 2019), Philadelphia, PA, USA, June 4, 2019, published at http://ceur-ws.org

management operations (typically model synchronisation) might outperform TGGs in term of scalability and control over finer details of consistency restoration [KZH16], TGGs are more declarative in the sense that the same specification (grammar) can be used for a wider range of consistency management operations.

Although an impressive number of different TGG-based tools have been developed in the past (we discuss them in Sect. 5), no TGG-based tool we are aware of provides the same level of support for more than one or two consistency management operations. While model generation and synchronisation are often supported, consistency checking (with or without correspondence links) typically only works for either very restricted TGGs or only in the simplest of cases. We argue that this is because existing TGG-based tools do not support all these operations in a conceptually and technically *uniform* manner, within a common framework. This makes it difficult to maintain and optimise numerous operations, especially when adding new language features.

In this tool paper, we present *eMoflon::IBeX*, a new generation of TGG-based tooling that is able to handle model generation, synchronisation, and consistency checking with essentially the same uniform approach. In Sect. 2 we provide a running example and use it to present the frontend of IBeX. In Sect. 3, we give an overview of the backend of IBeX, via a static (structural) view of its architecture, and a dynamic view of the underlying algorithm. In Sect. 4, we compare eMoflon::IBeX with its predecessor *eMoflon::TiE* [LAS14a] to assess the price of delegating most of the complexity of consistency management to an incremental graph pattern matcher [LAF+17] and an ILP (Integer Linear Programming) solver [LAS17]. In Sect. 5, we compare eight TGG-based tools (including IBeX) to highlight our latest achievements, and conclude the paper in Sect. 6.

2 An Overview of the Frontend with a Running Example

As TGGs are typically used in a Model-Driven Engineering (MDE) context, the triple graphs generated by a TGG are typed via a triple of metamodels referred to as a (TGG) *schema*. Figure 1 depicts the schema for our running example, adapted from Leblebici [Leb16], in which consistency is to be maintained between UML and Java models. For presentation purposes, we restrict the example to cover only operations (methods) and their parameters (arguments). To implement a schema, source (`uml`) and target (`java`) metamodels must be provided as standard Ecore[1] metamodels to be specified with any editor of choice. The IBeX frontend provides a visualisation for such metamodels, as depicted to the left of Fig. 1. For the correspondence metamodel, IBeX provides a dedicated textual concrete syntax with which the roles (source or target) of the metamodels are assigned, and 1-1 correspondence types (`OpToMethod` and `ParamToArg`) connecting a source type with a target type can be concisely specified. This is depicted to the right of Fig. 1 (`#schema UMLToJava`). A triple of models that is well-typed with respect to the schema (correspondence links are denoted by dashed lines) is depicted in Fig. 2[2]. The operation `substring` demonstrates that a simple 1-1 mapping can be non-trivial; It is *overloaded*, once with a single parameter `beginIndex`, and once with an additional parameter `length`.

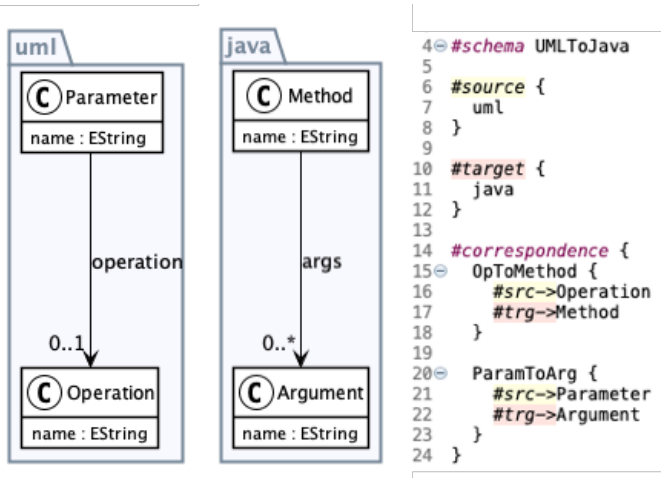

Figure 1: TGG Schema for Running Example

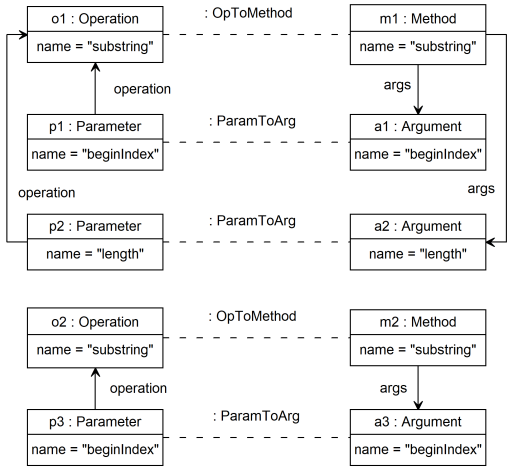

Figure 2: Consistent Triple of Models

Intuitively, the example triple is not only well-typed but is also *consistent* in the sense that the "correct" operations and parameters are connected to the "correct" methods and arguments. Figure 3 depicts two TGG

[1] The meta-metamodel of the Eclipse Modeling Framework (EMF). [2] Layout optimised manually for readability.

rules that formalise this consistency relation. The first rule `OperationToMethodRule` is depicted in the top-left corner of Fig. 3. IBeX provides an editable textual concrete syntax for TGG rules, complemented with an automatically generated read-only visualisation. A TGG rule can be applied to a given triple of models if its context (all black elements) can be matched in the triple. A rule is applied at a given match by creating all green (`++`) elements in the rule. Additional *attribute conditions* such as `eq_string`, from an extensible library of conditions implemented in Java, can be specified in a simple textual syntax. `OperationToMethodRule` requires no context and can thus be applied to any (including the empty) triple to create a connected operation and method with the same name. The second rule `ParameterToArgumentRule`, depicted to the right of Fig. 3 also in a textual and visual concrete syntax, requires a connected operation and method as context, and creates a parameter and argument with the same name for the operation and method, respectively. By applying `OperationToMethodRule` twice and `ParameterToArgumentRule` thrice, these two rules can be used to generate our example triple.

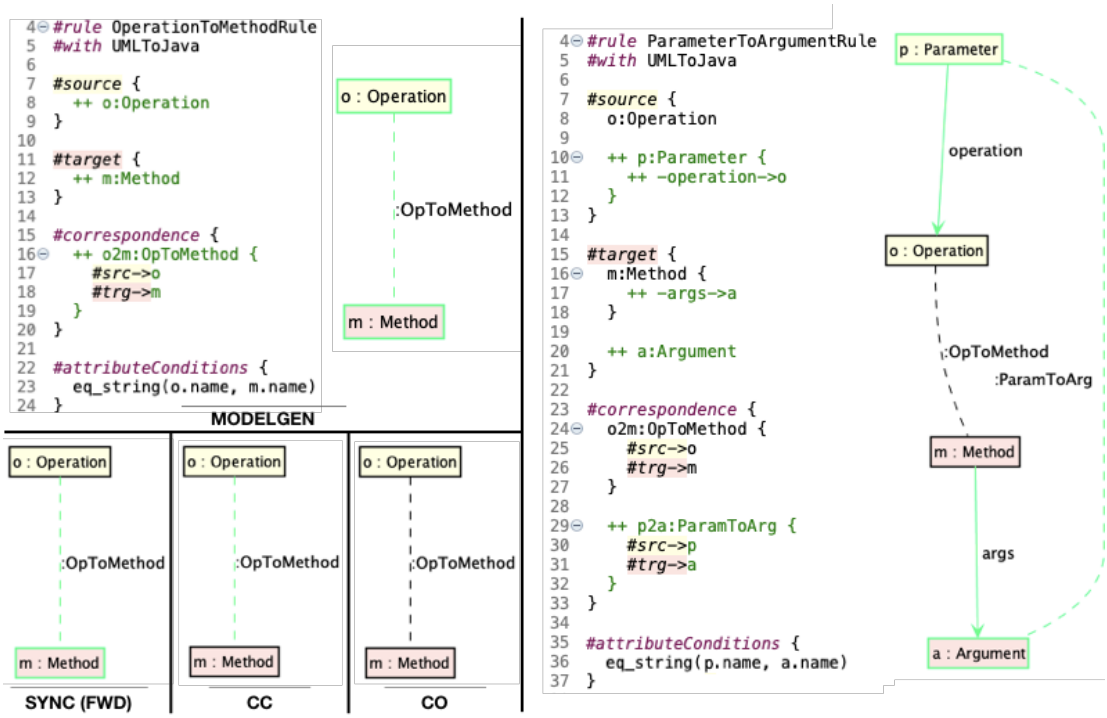

Figure 3: Specified Rules and Derived Operationalisations

The main potential of the TGG approach is that the rules specified by a developer can be automatically used to derive a host of different consistency management operations. The different variants of `OperationToMethodRule` are depicted to the left of Fig. 3. The specified rule itself can be directly used for model generation (`MODELGEN`), derived rules (in the visual concrete syntax) for model synchronisation (`SYNC-FWD`)[3], consistency checking without correspondence links (`CC`), and consistency checking with correspondence links (`CO` for *check only*) are depicted in the bottom-left corner of Fig. 3. The operationalisation process is straightforward: `SYNC` increases the context of `MODELGEN` by adding either all source (`FWD`) or all target elements (`BWD`) to its context. `CC` increases this context further by adding all source *and* target elements, while `CO` takes all elements as context. `SYNC` can be used to propagate a source (target) *delta* (a change to a model) to a corresponding target (source) delta. `CC` can be used to check if a pair of source and target models is consistent or not by attempting to create a correspondence model connecting them in a way that the triple becomes consistent. `CO` is similar but only checks for consistency and requires a correspondence model. Implementing consistency management with these derived rules in a *scalable* manner poses various challenges, some of which are discussed by Leblebici et al. [LAF+17, LAS17]. For example, as an indication of the complexity of `CC` and `CO` for our running example, note that it is impossible to decide locally if source and target elements are to be connected by a correspondence link or not. Naïve backtracking-based solutions thus explode exponentially even for small models.

[3] Backward synchronisation analogously.

3 A Structural and Dynamic Overview of the Back end

Figure 4 provides a structural overview of the architecture of eMoflon::IBeX. The front end consists of an Xtext-based[4] textual editor, coupled with a live, read-only PlantUML-based[5] visualisation, as presented in Sect. 2. The front end is completely independent of the back end and provides an extension point[6] via which it offers a TGG schema and set of rules (as EMF (meta)models). The back end takes the schema (a triple of metamodels) and set of rules as input, as well as an input triple that is expected to be typed over the schema. To simplify configuration and usage of the system, the back end generates stubs for every supported consistency management operations as Java files. These stubs can be executed directly with default settings, and of course configured and adapted as required. The back end relies primarily on two external components for performing consistency management: (i) an *incremental* graph pattern matcher to efficiently determine solution candidates, and (ii) an ILP solver to choose the optimal solution from these candidates. Arguments why this combination yields a flexible and still scalable consistency management framework are provided by Leblebici et al. [LAS17, LAF+17].

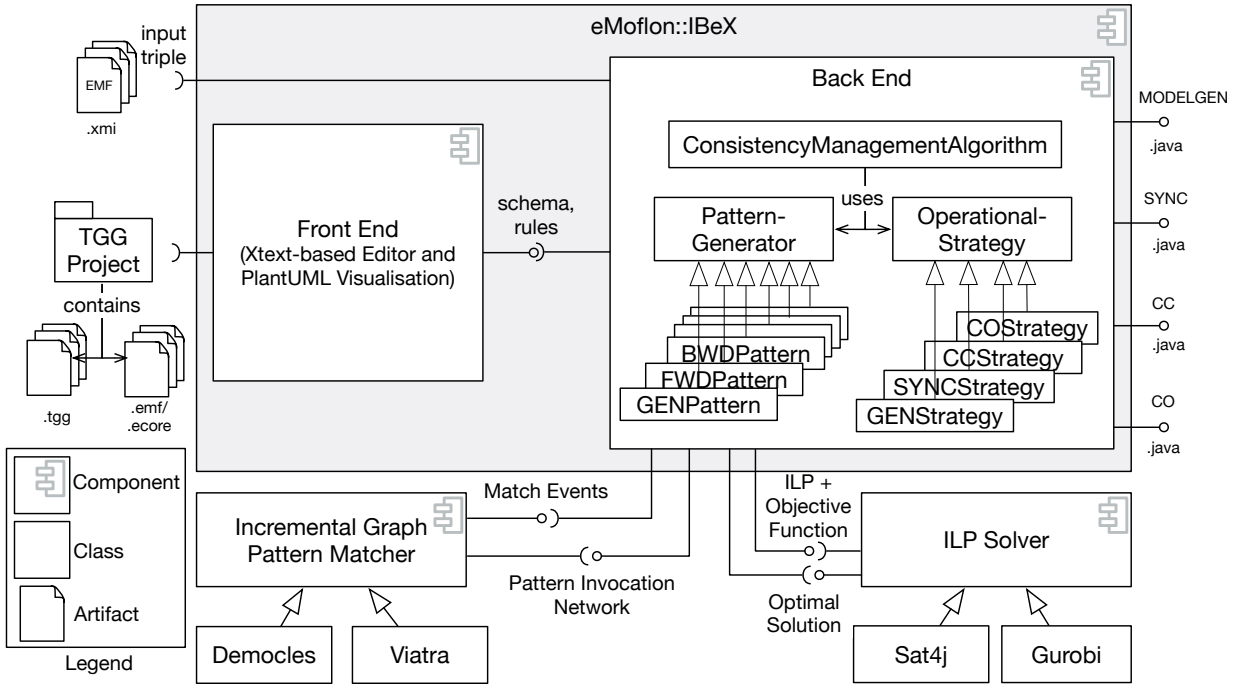

Figure 4: Architecture of eMoflon::IBeX as a Component Diagram

The interface to the incremental graph pattern matcher and ILP solver are relatively generic and can be implemented for various solvers. For the incremental pattern matcher, IBeX provides a *pattern invocation network* as input, representing the patterns to be matched structured in a network (a graph with nodes as patterns and edges as pattern invocations) to maximise reuse of partial matches. The incremental pattern matcher produces *match events* as output, signalling when new matches appear (create match events), and when old matches are violated (delete match events), as the models are manipulated. For the ILP solver, IBeX provides an ILP as input, computed from the set of solution candidates, and an objective function that depends on the specific consistency management operation. The ILP solver optimises the objective function and chooses an optimal solution from the provided candidates. Currently, IBeX primarily supports Democles [VD13] as an incremental graph pattern matcher, and is distributed with SAT4J[7] as default ILP solver. To test our interfaces, however, we have implemented prototypical adapters for VIATRA,[8] Drools,[9] and Nools[10] as alternative incremental pattern matchers, as well as well-tested adapters for alternative ILP solvers including Gurobi,[11] CBC,[12] GLPK,[13] and MIPCL.[14] Our achievements indicate that the interfaces are generic enough to enable an integration of other incremental graph pattern matchers and ILP solvers with acceptable effort.

[4] `https://www.eclipse.org/Xtext/` [5] `http://plantuml.com/` [6] In the Eclipse framework, components are *plug-ins* that can provide *extension points* and require *extensions*. [7] `sat4j.org` [8] `www.eclipse.org/viatra/` [9] `www.drools.org/` [10] `noolsjs.com/` [11] `www.gurobi.com/products/gurobi-optimizer` [12] `projects.coin-or.org/Cbc` [13] `www.gnu.org/software/glpk/` [14] `mipcl-cpp.appspot.com/`

The uniform consistency management algorithm in the back end comprises two main tasks that can be configured as required to implement various operations: the first task is to generate a suitable pattern invocation network from a set of TGG rules (via a *pattern generator*), while the second task is to restore consistency by applying a specific *strategy*. Both interfaces are implemented for the various consistency management operations. Note that there is no one-to-one mapping from patterns to strategies. The `SYNCStrategy`, for example, combines various patterns including `FWD-` and `BWDPatterns`.

Figure 5 provides a dynamic view on the uniform algorithm used for all operations as an activity diagram. The first action is *pattern generation* requiring a TGG schema and rules as input. This is followed by *pattern matching* on the input triple, based on the generated pattern invocation network. This action generates a stream of match events (create or delete), which are passed on to the *match event handling* action. Depending on the concrete operation, match events can be handled by creating, deleting, or manipulating elements in the input triple. In general, this action is non-deterministic and can be controlled by an *update policy* that, for example, can choose which match event to handle from the set of all pending events. After one or numerous match events have been handled (the handler decides when it is finished), a stop criterion (depends on the concrete operation) decides if the current state represents all solution candidates, or if the pattern matching and match event handling process is to be repeated. This stop criterion can range from a simple time-out for model generation to a check for an empty set of new match events for synchronisation (indicating that every new element has been translated). The advantage of using an *incremental* graph pattern matcher is that the latest match events can be produced relatively efficiently without any extra effort or explicit "search" for new or removed pattern matches. Note that although, e.g., model generation appears straightforward, it still requires both create and delete match events: the former due to matches involving the new structure created by the match event handler, the latter due to negative application conditions that can be violated by this newly created structure.

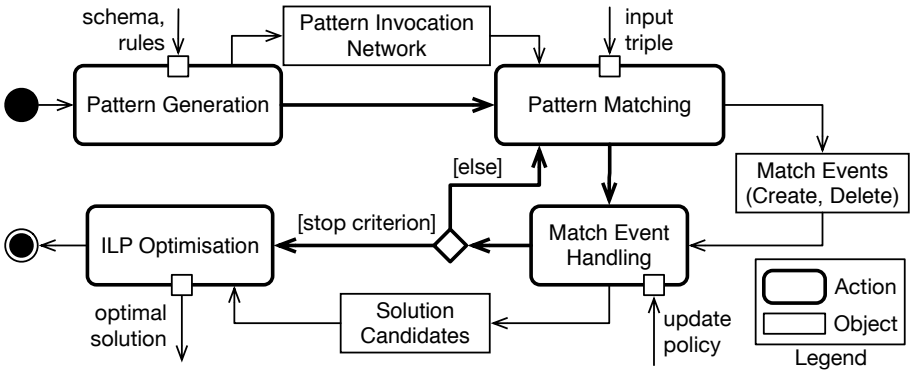

Figure 5: Uniform Consistency Management Algorithm as an Activity Diagram

When the stop criterion is fulfilled, the current set of solution candidates is passed to the final action in the process, *ILP optimisation*. This last action is optional as some of our operations (`MODELGEN` and `SYNC`) are *greedy*; a single candidate is produced and returned at this stage without any optimisation. Other operations (`CC` and `CO`), however, compute a non-trivial set of candidates from which an ILP problem is derived. This ILP ranges over matches as variables and constraints that ensure that a feasible (the specific operation defines what this means) solution is selected. The algorithm terminates with an optimal *solution* that consists in general of the output triple, and additional information if the optimal solution could not restore consistency perfectly. As an example, this additional information comprises a source and target delta for `CC`, indicating parts of the source and target models that can not be generated using the TGG (and are thus inconsistent).

4 Evaluation: A Comparison of eMoflon::IBeX with eMoflon::TiE

eMoflon::IBeX can be viewed as a complete re-engineering of its predecessor eMoflon::TiE (*Tool Integration Environment*) [LAS14a]. Both tools are similar in goal and share almost the same textual concrete syntax and visualisation for TGG rules. The similarity ends there, however, as eMoflon::TiE is a *code generator* that relies on a static search plan-based graph pattern matcher: everything is mapped to Java code at compile time including metamodels, rules, and search plans for performing pattern matching. eMoflon::IBeX is, in contrast, essentially an *interpreter* that relies on an incremental graph pattern matcher. An overview and description of

the eMoflon tool landscape together with links to the open-source projects and documentation can be found on the eMoflon project website.[15] As an interpreter, eMoflon::IBeX is more flexible and poses fewer restrictions than eMoflon::TiE, i.e., is more expressive with respect to the class of TGGs that can be specified with the tool. We have also been able to implement more operations and better strategies for synchronisation than eMoflon::TiE ever supported. These engineering advantages come at the price of *scalability* with respect to runtime and memory consumption. Interpretative pattern matching solutions tend to be factors slower than their generative counterparts [VAS12], and incremental graph pattern matching can have a substantial memory footprint as all matches and partial matches are stored in memory until the end of the entire process. To evaluate the "price" for a more flexible and general solution, we investigate the following research questions in this section:

(RQ1) How large is the runtime overhead of an interpretative approach such as eMoflon::IBeX? How does this overhead scale with respect to (meta)model size, pattern size, number of rules, and operations?

(RQ2) Is there a speed-up for performing incremental updates with the `SYNC` operation? How does this speed-up (if there is one) scale with respect to model size (constant delta size)?

We argue that a comparison of eMoflon::IBeX with eMoflon::TiE can be used to investigate these research questions, as eMoflon::TiE has been shown to scale reasonably well when compared to other TGG tools [HLG$^+$13, LAS$^+$14b], and also when compared to other bx tools [ADJ$^+$17]. This means that results from investigating the research questions with respect to eMoflon::TiE can be transferred to consistency management tools in general.

Setup: All measurements were conducted using an Intel i7 processor with 8 GB available main memory. A time-out of 20 minutes was used for `CC` and `SYNC`; all other operations were limited to 5 minutes. Runtime and memory consumption were measured for five chosen TGGs, all operations supported by both tools (`MODELGEN`, `SYNC`, and `CC`), and for model sizes from 1 thousand to 1 million elements (objects and links). To restrict the effect of outliers, the median value of three test runs was taken. The input models were generated randomly using `MODELGEN`. For measurements that require deltas, specific edits were implemented for each metamodel. The five TGGs are chosen to provide high variety with respect to their metamodels and rules:

`CompanyToIT` [AL] and `ClassInhHier2DB` [Anja] are both "small" TGGs consisting of four rules with about 10 elements (green and black nodes and edges) each, and metamodels with only about 12 elements (classes, references, and attributes) each. Although minimal, the forward direction of `CompanyToIT` is challenging, as one of the rules is chosen to yield *many* matches, of which only one is actually required and chosen. `ClassInhHier2DB` uses a Negative Application Condition (a filter NAC [HEGO10]) in the forward direction to resolve a rule conflict.

The remaining three TGGs are somewhat larger, consisting of 10–13 rules. `MoDiscoJava2UML` [Leb] (average rule size of 24 elements) uses realistic metamodels (UML and Java) with 339 and 837 elements, respectively. `VHDLTGGCodeAdapter`[16] (average rule size of 15 elements) has a source metamodel (27 elements) that is weakly typed (generic, XML-like), and a target metamodel (29 elements) that is strongly typed. This means that the forward transformation involves a lot of attribute conditions for checking labels of generic nodes (as opposed to relying on types). Finally, `FamiliesToPersons` [Anjb] (average rule size of 6 elements) has minimal metamodels (about 10 elements) but is highly non-deterministic in the backward direction.

Results: The ratio between the times needed to perform a consistency management task in eMoflon::IBeX and eMoflon::TiE is plotted for each TGG. A ratio of 5 means that eMoflon::TiE is 5 times faster than eMoflon::IBeX for the measurement point. If one of the tools timed out in all three test runs, no value is taken for the ratio[17]. We provide all measurement data[18] including memory consumption, but focus in the following on runtime.

Figure 6 depicts a plot of runtime values for `MODELGEN`. As can be expected for a code generator vs. interpreter comparison [VAS12], eMoflon::TiE clearly scales better than eMoflon::IBeX for all TGGs. Starting with factor 2 for 1000 elements, the ratio increases up to 20 for one million elements for most TGGs. As the differences are particularly pronounced for larger TGGs and larger rules, the overhead and increasing factor is probably related to the fact that IBeX stores *all* matches until the end of the entire model generation process. As a consequence, the number of potential rule applications for the model generator, and the overhead for monitoring all matches, increases steadily. All subsequent plots use the same legend and axis labels as Fig. 6 so they are not repeated.

Figures 7 and 8 depict plots of runtime for unidirectional batch transformations, i.e., source (target) models of increasing size are transformed to target (source) models from scratch. eMoflon::TiE is only faster by a factor less than 8 for `MoDiscoJava2UML` and `ClassInhHier2DB`, actually reducing steadily up to 100k before

[15] `www.emoflon.org` [16] `http://bit.ly/2tkBgYi` [17] In our experiments, either IBeX or both tools timed out.
[18] `http://bit.ly/2GnlBjB`

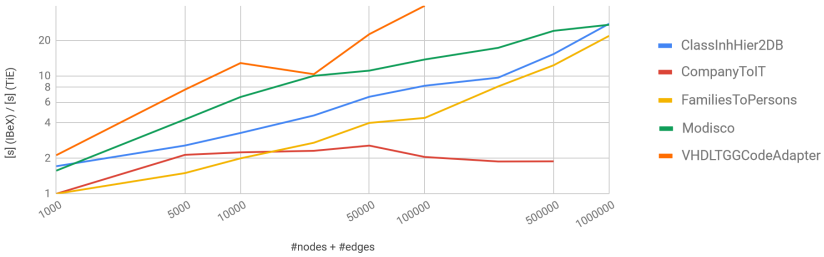

Figure 6: Runtime for Model Generation (Log/Log)

increasing again (probably due to an increase in memory consumption). For the TGGs that are challenging in the forward direction, however, eMoflon::IBeX is much less robust and does not scale in comparison to TiE. The challenge with `VHDLTGGCodeAdapter` was handling the weakly typed source models, while `CompanyToIT` yields a lot of matches that are unnecessary. Both these points have to be taken into account and avoided as much as possible if IBeX is expected to scale. Finally, while IBeX scales well for `FamiliesToPersons` up to about 10k, the factor then starts rising to more than 40 for larger models. This might be due to the fact that `FamiliesToPersons` uses very simple metamodels that are actually just lists of lists and a flat list. Such simple models are not well-suited for the heuristics used for graph pattern matchers and probably yield lots of useless partial matches. The situation for the backward direction is similar: IBeX scales reasonably well for `MoDiscoJava2UML` and `ClassInhHier2DB` again, even eventually outperforming TiE for `ClassInhHier2DB` and larger models. `CompanyToIT` and `VHDLTGGCodeAdapter` are both easier in the backward direction, and IBeX is able to produce results for larger models, with a factor of about 30–40 for the former, and about 20 for the latter. Interestingly, TiE appears to explode for `VHDLTGGCodeAdapter` for larger models, with IBeX performing better and better in comparison. Recall that `FamiliesToPersons` is highly non-deterministic in the backward direction: this explains why IBeX does not scale at all – a lot of matches are collected and only one is chosen.

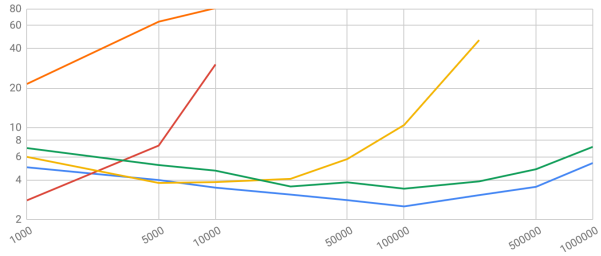

Figure 7: Runtime for Fwd. Batch (Log/Log)

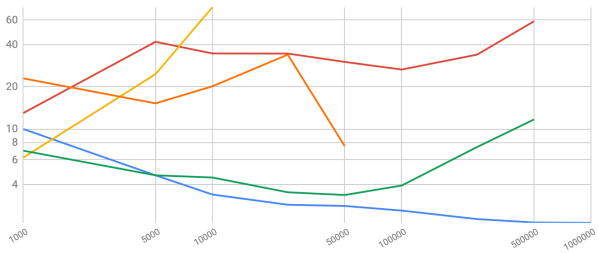

Figure 8: Runtime for Bwd. Batch (Log/Log)

The runtime behaviour for `CC` depends substantially on the particular TGG (cf. Fig. 9). While a factor of 100 is already reached for only 3k elements in the case of `CompanyToIT` and `FamiliesToPersons`, eMoflon::IBeX actually outperforms TiE for `ClassInhHier2DB` and `VHDLTGGCodeAdapter` for 8k elements. Both of the tools perform equally well for `MoDiscoJava2UML`. In general, `CC` is a much more difficult task than all other operations. Our results indicate that while IBeX can actually be advantageous for "larger" models, it is also much more susceptible to non-determinism and cases where numerous partial matches can be unnecessarily collected (only to be eventually discarded). Finally, Fig. 10 for forward synchronisation (backward was comparable) indicates the factor does not depend significantly on model size. For most TGGs, IBeX is slower by a steady factor of 5–10. While IBeX has again difficulties with `FamiliesToPersons`, its overhead steadily reduces for `MoDiscoJava2UML`.

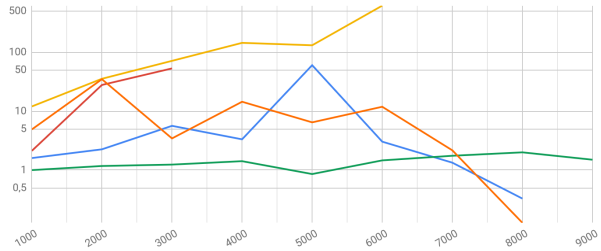

Figure 9: Runtime for Consistency Checking (Lin/Log)

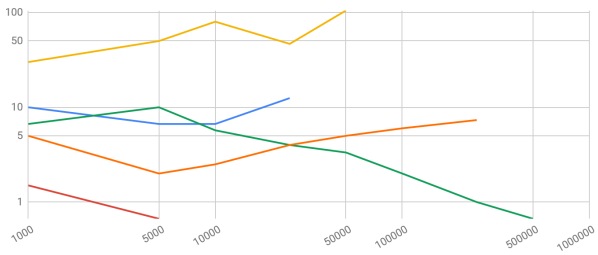

Figure 10: Runtime for Fwd. Sync. (Log/Log)

Let us conclude by reviewing our research questions. For graph-like, strongly typed and highly connected models such as UML and Java models, our results indicate that one can expect an overhead of about factor 10

with IBeX. This seems to be more for `MODELGEN`, and less for `CC`. For tree-like models, weakly typed models, and patterns that yield many matches that are eventually discarded, however, IBeX appears to be significantly slower (factor increasing with model size) especially in batch scenarios. These results are consistent with both predications of code generators vs. interpreters [VAS12], and expectations for incremental graph pattern matchers [VD13]. On a final note, recall that only operations that are currently implemented in both tools could be compared, i.e., the `CO` operation had to be excluded. We also had to restrict the comparison to TGGs that can be handled by TiE, avoiding examples that only work for IBeX. Finally, to simplify the interpretation of our results, we also omitted recent improvements, e.g., making our `SYNC` strategy more intelligent, which could only be implemented by exploiting an incremental pattern matcher with IBeX.

5 Related Work

In the last 20 years, several implementations of the TGG approach have been proposed. Table 1 summarizes the supported operations of seven TGG-based tools, including IBeX and TiE. TiE, the predecessor of IBeX, supports both unidirectional model transformation and model synchronization, as well as consistency checking. Each operation is, however, essentially a completely separate implementation making it increasingly costly to maintain and extend the tool. MoTE is a TGG-based tool with a strong focus on scalability. While it supports numerous operations, it also poses strong restrictions on the class of supported TGGs. These restrictions simplify especially `SYNC` and `CC` but also severely limit expressiveness. HenshinTGG supposedly supports all operations in our table, but as far as we can tell from simple experiments, the choice of rules for each operation must be deterministic, severely restricting the class of supported TGGs. The TGG Interpreter directly interprets TGG rules and only supports both model transformation and synchronisation. Existing surveys of TGG tools indicate, however, that the TGG interpreter does not scale compared to MoTE and eMoflon::TiE [HLG$^+$13, LAS$^+$14b]. Fujaba provided one of the first implementations of the TGG approach and – as far as we can assess – already supported both model transformation and synchronisation. EMorF interprets TGG rules and claims to support model transformation, synchronisation, and consistency checking. As both tools are no longer available, however, it is difficult to assess under which assumptions `CC` actually works. Finally, the UML tool USE was extended to support consistency management based on TGGs by translating TGG rules into OCL constraints. Specified TGG rules can be operationalised for model transformation, synchronisation and consistency checking. As far as we can tell, however, the *application* of these rules remains a manual task.

TGG Tool	Model Generation	Model Transformation	Model Synchronization	Consistency Checking	Check Only
eMoflon::TiE [LAS14a]	✓	✓	✓	✓	✗
MoTE [HLG$^+$11, GHL14]	✓	(✓)	(✓)	(✓)	✗
HenshinTGG [EHGB12]	✓	(✓)	(✓)	(✓)	(✓)
TGG Interpreter [GK10]	✗	✓	✓	✗	✗
Fujaba [GW06, BGN$^+$04]	✗	✓	✓	✗	✗
EMorF [KW12]	✗	✓	✓	✓	✗
use4tgg [DG08]	✗	(✓)	(✓)	(✓)	✗
eMoflon::IBeX	✓	✓	✓	✓	✓

Table 1: Comparison of TGG tools

6 Conclusion and Future Work

In this paper we presented eMoflon::IBeX, a novel TGG-based consistency management tool that supports numerous consistency management operations including unidirectional forward and backward transformation, model synchronization, and consistency checking with and without correspondence links. To achieve this we apply a conceptually and technically uniform interpretative algorithm, which leverages and suitably combines an incremental graph pattern matcher and an ILP solver. While our evaluation shows that IBeX is multiple factors slower than its predecessor TiE in many cases, we believe that it is worth the price to simplify the implementation of further operations and the general maintenance of the tool and its support for all operations. As future work we plan to further exploit IBeX to add support for *tolerance* [Ste14] to all our operations, and implement *model integration*, which requires reflective updates [DKL18], conflict detection and resolution, and user interaction.

References

[ADJ+17] Anthony Anjorin, Zinovy Diskin, Frédéric Jouault, Hsiang-Shang Ko, Erhan Leblebici, and Bernhard Westfechtel. Benchmarx reloaded: A Practical Benchmark Framework for Bidirectional Transformations. In Romina Eramo and Michael Johnson, editors, *Proceedings of the 6th International Workshop on Bidirectional Transformations co-located with The European Joint Conferences on Theory and Practice of Software, BX@ETAPS 2017, Uppsala, Sweden, April 29, 2017.*, volume 1827 of *CEUR Workshop Proceedings*, pages 15–30. CEUR-WS.org, 2017.

[AL] Anthony Anjorin and Marius Lauder. CompanyToIT v0.1 in Bx Examples Repository. `http://bx-community.wikidot.com/examples:companytoit`. Date retrieved: 12 Feb 2019.

[Anja] Anthony Anjorin. ClassDiagramsToDataBaseSchemas v0.1 in Bx Examples Repository. `http://bx-community.wikidot.com/examples:classdiagramstodatabaseschemas`. Date retrieved: 12 Feb 2019.

[Anjb] Anthony Anjorin. FamilyToPersons (B) v0.1 in Bx Examples Repository. `http://bx-community.wikidot.com/examples:familytopersons`. Date retrieved: 12 Feb 2019.

[Anj16] Anthony Anjorin. An Introduction to Triple Graph Grammars as an Implementation of the Delta-Lens Framework. In Jeremy Gibbons and Perdita Stevens, editors, *Bidirectional Transformations - International Summer School, Oxford, UK, July 25-29, 2016, Tutorial Lectures*, volume 9715 of *Lecture Notes in Computer Science*, pages 29–72. Springer, 2016.

[BGN+04] Sven Burmester, Holger Giese, Jörg Niere, Matthias Tichy, Jörg P. Wadsack, Robert Wagner, Lothar Wendehals, and Albert Zündorf. Tool Integration at the Meta-model Level: The Fujaba Approach. *STTT*, 6(3):203–218, 2004.

[CREP10] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. JTL: A Bidirectional and Change Propagating Transformation Language. In Brian A. Malloy, Steffen Staab, and Mark van den Brand, editors, *Software Language Engineering - Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers*, volume 6563 of *Lecture Notes in Computer Science*, pages 183–202. Springer, 2010.

[DG08] Duc-Hanh Dang and Martin Gogolla. On Integrating OCL and Triple Graph Grammars. In Michel R. V. Chaudron, editor, *Models in Software Engineering, Workshops and Symposia at MODELS 2008, Toulouse, France, September 28 - October 3, 2008. Reports and Revised Selected Papers*, volume 5421 of *Lecture Notes in Computer Science*, pages 124–137. Springer, 2008.

[DKL18] Zinovy Diskin, Harald König, and Mark Lawford. Multiple Model Synchronization with Multiary Delta Lenses. In Alessandra Russo and Andy Schürr, editors, *Fundamental Approaches to Software Engineering, 21st International Conference, FASE 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings.*, volume 10802 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2018.

[DXC+11] Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki, Hartmut Ehrig, Frank Hermann, and Fernando Orejas. From State- to Delta-Based Bidirectional Model Transformations: The Symmetric Case. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings*, volume 6981 of *Lecture Notes in Computer Science*, pages 304–318. Springer, 2011.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[EHGB12] Claudia Ermel, Frank Hermann, Jürgen Gall, and Daniel Binanzer. Visual Modeling and Analysis of EMF Model Transformations Based on Triple Graph Grammars. *ECEASST*, 54, 2012.

[GHL14] Holger Giese, Stephan Hildebrandt, and Leen Lambers. Bridging the Gap between Formal Semantics and Implementation of Triple Graph Grammars - Ensuring Conformance of Relational Model Transformation Specifications and Implementations. *Software and System Modeling*, 13(1):273–299, 2014.

[GK10] Joel Greenyer and Ekkart Kindler. Comparing Relational Model Transformation Technologies: Implementing Query/View/Transformation with Triple Graph Grammars. *Software and System Modeling*, 9(1):21–46, 2010.

[GW06] Holger Giese and Robert Wagner. Incremental Model Synchronization with Triple Graph Grammars. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006, Proceedings*, volume 4199 of *Lecture Notes in Computer Science*, pages 543–557. Springer, 2006.

[HEGO10] Frank Hermann, Hartmut Ehrig, Ulrike Golas, and Fernando Orejas. Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars. In Jean Bézivin, Richard Mark Soley, and Antonio Vallecillo, editors, *Proceedings of the First International Workshop on Model-Driven Interoperability, MDI@MoDELS 2010, Oslo, Norway, October 3-5, 2010*, pages 22–31. ACM, 2010.

[HLG+11] Stephan Hildebrandt, Leen Lambers, Holger Giese, Dominic Petrick, and Ingo Richter. Automatic Conformance Testing of Optimized Triple Graph Grammar Implementations. In Andy Schürr, Dániel Varró, and Gergely Varró, editors, *Applications of Graph Transformations with Industrial Relevance - 4th International Symposium, AGTIVE 2011, Budapest, Hungary, October 4-7, 2011, Revised Selected and Invited Papers*, volume 7233 of *Lecture Notes in Computer Science*, pages 238–253. Springer, 2011.

[HLG+13] Stephan Hildebrandt, Leen Lambers, Holger Giese, Jan Rieke, Joel Greenyer, Wilhelm Schäfer, Marius Lauder, Anthony Anjorin, and Andy Schürr. A Survey of Triple Graph Grammar Tools. *ECEASST*, 57, 2013.

[KW12] Lilija Klassen and Robert Wagner. EMorF - A Tool for Model Transformations. *ECEASST*, 54, 2012.

[KZH16] Hsiang-Shang Ko, Tao Zan, and Zhenjiang Hu. BiGUL: A Formally Verified Core Language for Putback-based Bidirectional Programming. In Martin Erwig and Tiark Rompf, editors, *Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 61–72. ACM, 2016.

[LAF+17] Erhan Leblebici, Anthony Anjorin, Lars Fritsche, Gergely Varró, and Andy Schürr. Leveraging Incremental Pattern Matching Techniques for Model Synchronisation. In Juan de Lara and Detlef Plump, editors, *Graph Transformation - 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18-19, 2017, Proceedings*, volume 10373 of *Lecture Notes in Computer Science*, pages 179–195. Springer, 2017.

[LAS14a] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. Developing emoflon with emoflon. In Davide Di Ruscio and Dániel Varró, editors, *Theory and Practice of Model Transformations - 7th International Conference, ICMT 2014, Held as Part of STAF 2014, York, UK, July 21-22, 2014. Proceedings*, volume 8568 of *Lecture Notes in Computer Science*, pages 138–145. Springer, 2014.

[LAS+14b] Erhan Leblebici, Anthony Anjorin, Andy Schürr, Stephan Hildebrandt, Jan Rieke, and Joel Greenyer. A Comparison of Incremental Triple Graph Grammar Tools. *ECEASST*, 67, 2014.

[LAS17] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. Inter-model Consistency Checking Using Triple Graph Grammars and Linear Optimization Techniques. In Marieke Huisman and Julia Rubin, editors, *Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10202 of *Lecture Notes in Computer Science*, pages 191–207. Springer, 2017.

[Leb] Erhan Leblebici. UML Operations to Java Operations v0.1 in Bx Examples Repository. `http://bx-community.wikidot.com/examples:umloperationstojavaoperations`. Date retrieved: 12 Feb 2019.

[Leb16] Erhan Leblebici. Towards a Graph Grammar-Based Approach to Inter-Model Consistency Checks with Traceability Support. In Anthony Anjorin and Jeremy Gibbons, editors, *Proceedings of the 5th International Workshop on Bidirectional Transformations, Bx 2016, co-located with The European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 8, 2016.*, volume 1571 of *CEUR Workshop Proceedings*, pages 35–39. CEUR-WS.org, 2016.

[MGC13] Nuno Macedo, Tiago Guimarães, and Alcino Cunha. Model Repair and Transformation with Echo. In Ewen Denney, Tevfik Bultan, and Andreas Zeller, editors, *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, pages 694–697. IEEE, 2013.

[Sch94] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In Ernst W. Mayr, Gunther Schmidt, and Gottfried Tinhofer, editors, *Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG '94, Herrsching, Germany, June 16-18, 1994, Proceedings*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163. Springer, 1994.

[Ste14] Perdita Stevens. Bidirectionally tolerating inconsistency: Partial transformations. In Stefania Gnesi and Arend Rensink, editors, *Fundamental Approaches to Software Engineering - 17th International Conference, FASE 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8411 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2014.

[VAS12] Gergely Varró, Anthony Anjorin, and Andy Schürr. Unification of Compiled and Interpreter-Based Pattern Matching Techniques. In Antonio Vallecillo, Juha-Pekka Tolvanen, Ekkart Kindler, Harald Störrle, and Dimitrios S. Kolovos, editors, *Modelling Foundations and Applications - 8th European Conference, ECMFA 2012, Kgs. Lyngby, Denmark, July 2-5, 2012. Proceedings*, volume 7349 of *Lecture Notes in Computer Science*, pages 368–383. Springer, 2012.

[VD13] Gergely Varró and Frederik Deckwerth. A Rete Network Construction Algorithm for Incremental Pattern Matching. In Keith Duddy and Gerti Kappel, editors, *Theory and Practice of Model Transformations - 6th International Conference, ICMT 2013, Budapest, Hungary, June 18-19, 2013. Proceedings*, volume 7909 of *Lecture Notes in Computer Science*, pages 125–140. Springer, 2013.