eMoflon::Neo - Consistency and Model Management with Graph Databases

Nils Weidmann[1], Anthony Anjorin[2]

[1]*Paderborn University, Zukunftsmeile 2, 33102 Paderborn, Germany*
[2]*IAV GmbH Ingenieurgesellschaft Auto und Verkehr, Nordhoffstraße 5, 38518 Gifhorn, Germany*

Abstract

Maintaining the consistency of interrelated models is an important task in the context of Model-Driven Engineering (MDE). Appropriate tool support is crucial to achieve an adequate level of automation required for successful model management in general, and consistency maintenance in particular. Numerous MDE tools, including the tools in the eMoflon toolsuite, build upon the Eclipse Modeling Framework (EMF), a de-facto MDE standard. While EMF is a great framework, it has some drawbacks regarding scalability and flexibility with respect to metamodel conformance.

As our focus in recent years has turned to concurrent synchronisation and conflict resolution as an optimisation problem, we have decided to explore graph databases as an alternative infrastructure for handling our runtime model operations. In this paper we present the consistency management tool eMoflon::Neo as the latest addition to the eMoflon toolsuite, based on EMF for *specification models*, and the graph database Neo4j for all *runtime models*.

Keywords

Consistency Management, Model Management, Graph Databases

1. Introduction and Motivation

Consistency management plays a central role in the context of Model-Driven Engineering (MDE) and involves numerous operations including model transformation, (concurrent) synchronisation, and consistency checking. For consistency management tools to be of practical use, they must provide an adequate level of automation of these operations.

Triple Graph Grammars (TGGs) [1] are a rule-based bidirectional transformation (bx) language, well-suited for MDE as models are adequately handled as typed, attributed graphs. From a specified set of declarative rules that formally defines a language of consistent model triples, *operational* rules for different consistency management tasks are automatically derived. Although the Eclipse Modelling Framework (EMF) provides a solid basis for developing modelling tools, our experience from developing the eMoflon toolsuite [2] over the years is that EMF has some drawbacks when it comes to implementing model management tools. Especially with our recent focus on *concurrent* model synchronisation [3] and our hybrid approach of combining graph pattern matching and constraint solving to address conflict resolution as an optimisation problem, we have identified the following limitations directly related to representing our runtime models as EMF data structures.

Bx 2021: 9th International Workshop on Bidirectional Transformations, part of STAF, June 21, 2021
nils.weidmann@upb.de (N. Weidmann); anthony.anjorin@iav.de (A. Anjorin)

Scalability: EMF models must fit completely into the main memory for operating on them, which limits the handling of very large models. While we are not necessarily interested in extremely large models per se, we (i) represent traceability links and other bookkeeping information such as various markers explicitly in models, (ii) generate multiple candidate structures before using a constraint solver to pick the best result. Both points mean that we have to handle effective model sizes factors larger than the actual input model sizes.

Flexibility: Although the relatively strict conformance relation between EMF models and their metamodels certainly has its advantages, it is more often a hindrance that our algorithms have to work around. There are mainly two reasons for this: (i) being able to enrich model elements with markers and other extra information often simplifies analyses and bookkeeping operations, and (ii) when collecting candidate structures for a final optimisation step, we need to construct a "super model" that violates metamodel constraints such as multiplicities and single containment relations. EMF, however, does not support attributes for edges, and temporarily extending or relaxing metamodels of loaded models is non-trivial [4].

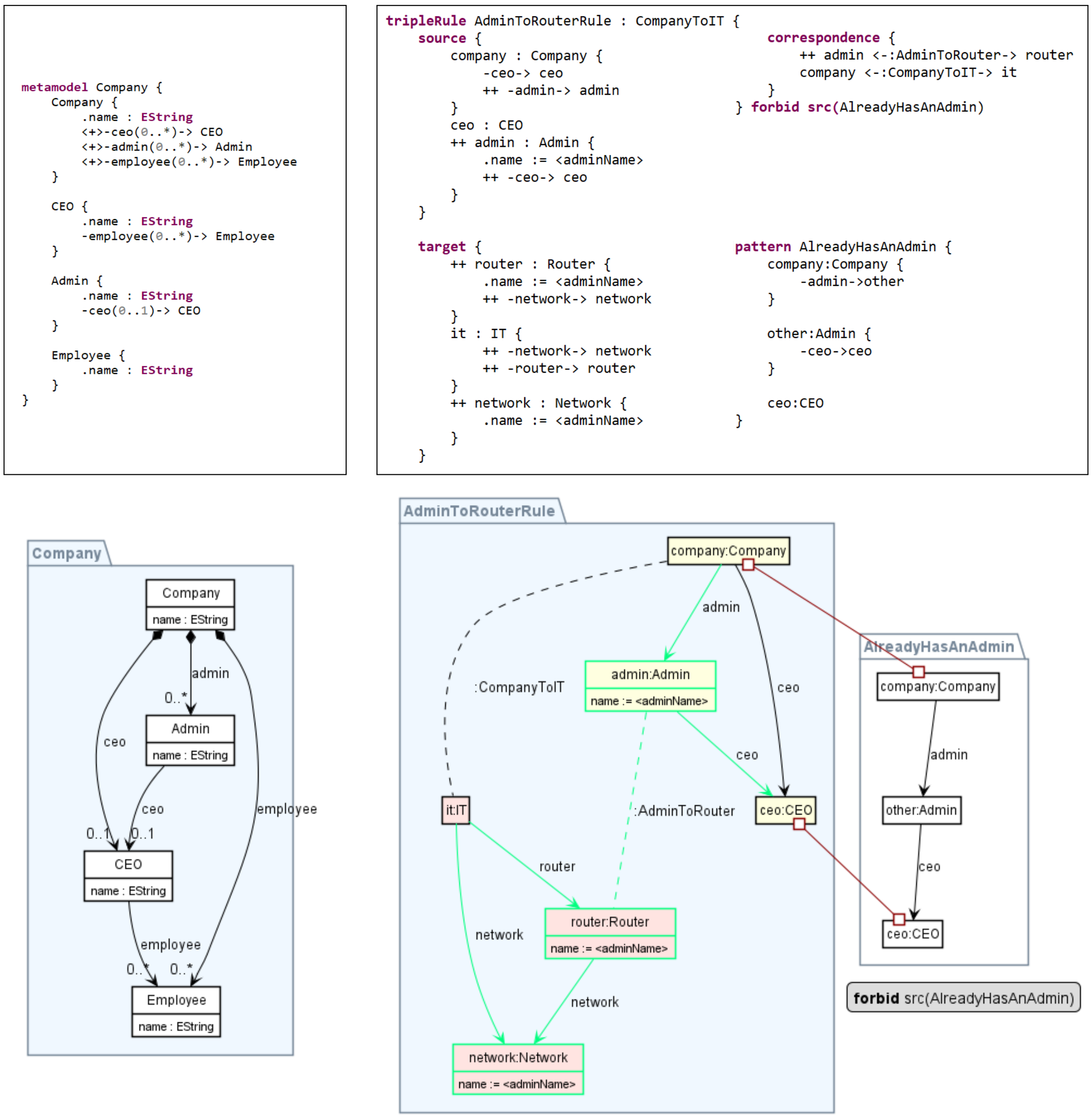
NoSQL databases in general, and graph databases in particular, have gained popularity in recent years and have been successfully leveraged for developing MDE tools [5, 6]. Using a graph database to represent runtime models can address both aforementioned issues: Firstly, graph databases promise improved scalability via on-demand caching, indexing, and a native respresentation of nodes and edges. Secondly, models and metamodels are (typically) both represented as plain graphs with type edges and constraints representing the conformance relation. This means that the relation can be (temporarily) violated and later reestablished as required with the standard infrastructure.

In this paper, we present *eMoflon::Neo* [7], the latest addition to the eMoflon toolsuite supporting bx based on the TGG formalism and using the graph database Neo4j [8] for handling all runtime models and model operations. eMoflon::Neo provides a modelling language eMoflon Specification Language (eMSL) that supports modelling, metamodelling, constraints as graph patterns, graph transformation rules, and TGGs, all with a uniform textual concrete syntax and a read-only visual concrete syntax based on PlantUML [9]. To investigate the strengths and weaknesses of using graph databases for model management regarding runtime performance, we compare our tool to eMoflon::IBeX [10], a comparable EMF-based TGG tool also from the eMoflon toolsuite. We measure different consistency management operations for different examples from the bx example repository [11], and for growing model sizes using both tools.

The remainder of the paper is organised as follows: Section 2 introduces the tool's front-end and its modelling languages with a uniform textual and visual concrete syntax. An overview of the software architecture of eMoflon::Neo is provided in Sect. 3; its runtime performance is analysed with an experimental evaluation in Sect. 4. Related approaches are discussed in Sect. 5, while Sect. 6 concludes the paper.

2. Overview of the Frontend

An overview of the front-end of eMoflon::Neo is provided in Fig. 1 using the bx example *CompanyToIT* [12]. In this example, a simplified organisational structure of a Company is to be kept consistent with a corresponding IT infrastructure. The top left of Fig. 1 depicts the textual



```
metamodel Company {
    Company {
        .name : EString
        <+>-ceo(0..*)-> CEO
        <+>-admin(0..*)-> Admin
        <+>-employee(0..*)-> Employee
    }

    CEO {
        .name : EString
        -employee(0..*)-> Employee
    }

    Admin {
        .name : EString
        -ceo(0..1)-> CEO
    }

    Employee {
        .name : EString
    }
}
```

```
tripleRule AdminToRouterRule : CompanyToIT {
    source {
        company : Company {
            -ceo-> ceo
            ++ -admin-> admin
        }
        ceo : CEO
        ++ admin : Admin {
            .name := <adminName>
            ++ -ceo-> ceo
        }
    }

    target {
        ++ router : Router {
            .name := <adminName>
            ++ -network-> network
        }
        it : IT {
            ++ -network-> network
            ++ -router-> router
        }
        ++ network : Network {
            .name := <adminName>
        }
    }

    correspondence {
        ++ admin <-:AdminToRouter-> router
        company <-:CompanyToIT-> it
    }
} forbid src(AlreadyHasAnAdmin)

pattern AlreadyHasAnAdmin {
    company:Company {
        -admin->other
    }

    other:Admin {
        -ceo->ceo
    }

    ceo:CEO
}
```

Figure 1: Overview of selected frontend components of eMoflon::Neo

specification of the source metamodel, i.e., the Company metamodel using eMSL, the eMoflon Specification Language, a family of modelling languages with a uniform textual concrete syntax supported by an Xtext-based editor. A `Company` consists of `CEO`s, `Admin`s, and `Employee`s, all identified via their respective names. For each class of the metamodel, there is a (nested) block containing attribute definitions and outgoing edges to other classes. The most important UML language features for specifying associations, such as multiplicities, aggregation and composition are supported. The corresponding read-only visualisation of the metamodel is depicted in the bottom left of Fig. 1 in a visual concrete syntax based on PlantUML[1], which

[1]https://plantuml.com

has already been used for rule visualisation in eMoflon::IBeX. To complement the textual eMSL editor, PlantUML diagrams are automatically generated for all eMSL entities including (meta-)models, graph patterns, constraints and rules. The visualisation is dynamic in the sense that it constantly adapts to the current selection in the textual editor. It is not only useful to obtain a helpful overview, but can also be used for navigation as elements in the view can be hyperlinked to locations in the corresponding textual eMSL files.

Consistency management with eMoflon::Neo is based on the TGG formalism [1]. The TGG for defining consistent model triples for the CompanyToIT example consists of multiple rules, one of which is depicted to the top right of Fig. 1 in a textual concrete syntax, and to the bottom right in the corresponding visual concrete syntax. The rule `AdminToRouterRule` creates an `Admin` in the company model and links it to a `Router` in the IT model. An existing `company`, a corresponding `it`, and the `ceo` of the company are required as context in order to match the rule in the source model. Context elements are coloured black in the visual syntax, whereas created elements are green and have a ++ mark-up in the textual syntax.

The graph *pattern* `AlreadyHasAnAdmin` is attached to the rule as a *negative application condition*, guaranteeing that no admin has been added to the company before. The pattern consists of a `Company`, a `CEO` and another `Admin`. When attached to the rule, the company and the ceo are matched to the same objects in both the pattern and the rule (indicated textually by using the same names, and visually by connecting these variables). In a similar manner, patterns can also be attached to metamodels as negative, positive, or implication *constraints*. All basic constraint types can be further combined via logical connectors.[2] Derived consistency management operations must guarantee that all constraints hold for generated output models.

3. Architectural Overview

The software architecture of eMoflon::Neo is depicted in Fig. 2 as a component diagram. As mentioned in Sect. 2, the eMSL language (file extension `.msl`) is used to uniformly specify all involved model management artefacts. For Graph Transformation (GT) and TGG projects, both metamodels and (triple) graph grammars are specified and stored in this format.

The back end of eMoflon::Neo can be subdivided into several components as depicted in Fig. 2. At compile time, the `Rule Compiler` uses the TGG specification to generate GT operational rules (also in eMSL) for all supported operations. The `Generator`, composed itself of several modules, is used to perform all consistency management operations, which can be configured via the generated operational rules and Java API code.

Finally, a `Cypher Query Translator` connects the back end to the Neo4j database, which contains all runtime models. Cypher[3] queries are generated to collect matches for operational rules and apply them on the models. The results are returned as an array of IDs for nodes and edges, which are either part of the match, or have been created by rule applications. Most operations (FWD_OPT (forward transformation), BWD_OPT (backward transformation), CO (check only), CC (correspondence creation) and CS (concurrent synchronisation)) are supported by an Integer Linear Programming (ILP) solver to choose a subset of actual rule applications

[2]The interested reader is referred to Ehrig et al. [13] for further details on graph constraints.
[3]Cypher is the declarative pattern matching and transformation language for Neo4j.

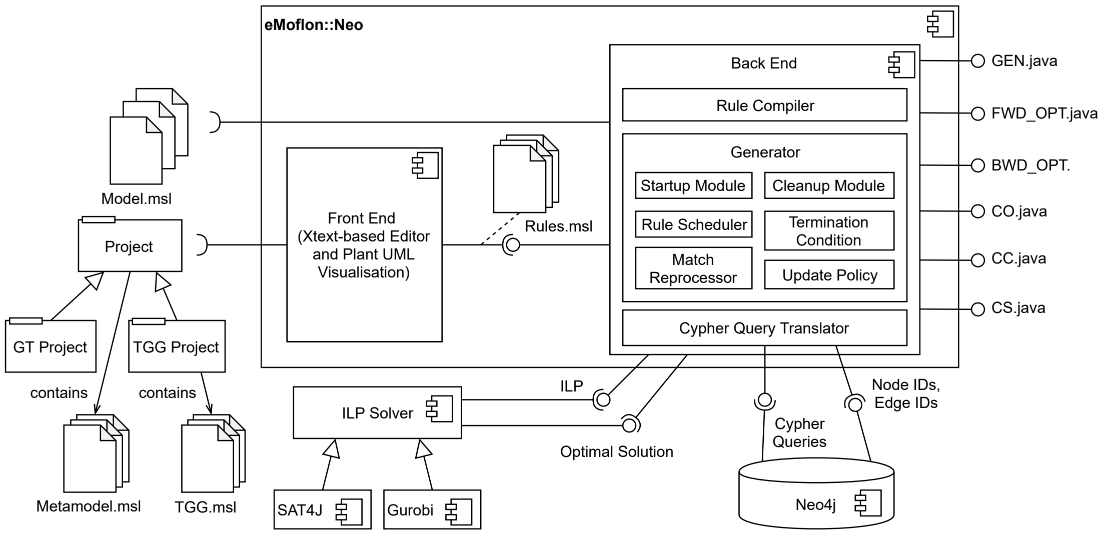


Figure 2: Architecture of eMoflon::Neo

from a predetermined superset. This approach simplifies guaranteeing formal properties such as correctness and completeness for all operations, as opposed to "greedy" strategies that apply rules immediately [3]. eMoflon::Neo currently supports SAT4J [14] and Gurobi [15] as solvers; adapters for other solvers can be added as required.

All consistency management operations follow a common work-flow, which we denote as the "core cycle", depicted in Fig. 3 as an activity diagram. Each activity is implemented as a module, which can be reused and combined with other modules to configure an operation. The startup module performs initialisation steps, such as setting temporary translation markers to their default value. In a loop, matches for (potential) rule applications and other patterns (e.g. for constraints) are collected: The rule scheduler selects rules for the subsequent pattern matching step, for which a maximum number of matches can be set. This is especially helpful for model generation (GEN), and can also be useful for other operations on very large models. For pattern matching, the first costly step in the database, the scheduling request is translated into a cypher query for the database. Based on the query results, matches are added to a match container. If this container is non-empty, matches are selected from the match container to be applied according to an update policy. While it is possible and greatly improves performance to select multiple matches to be applied in parallel, the update policy must guarantee that these matches are not in conflict with each other, i.e., make sure that only one of the potentially conflicting matches are chosen to be applied. The selected matches are then applied in a subsequent step in the database. Depending on the update policy, it is possible that there are still unused matches in the match container at this stage. Match reprocessing denotes the strategy applied to determine which matches can be safely used for the next iteration, and which have become invalid and must thus be removed. In a final step of the main loop, a pre-defined termination condition is checked. Such a condition could be, e.g., that no new matches were found in the last iteration. If this condition does not yet hold, a new iteration of the main loop begins. Otherwise, the clean-up module prepares the operation's termination. Depending on the concrete operation, a clean-up can entail removing all temporary markers, performing ILP solving to determine the optimal result from a set of candidates, deleting all other sub-optimal candidates, etc. In this step, it is also guaranteed that the produced result fulfils all posed constraints.

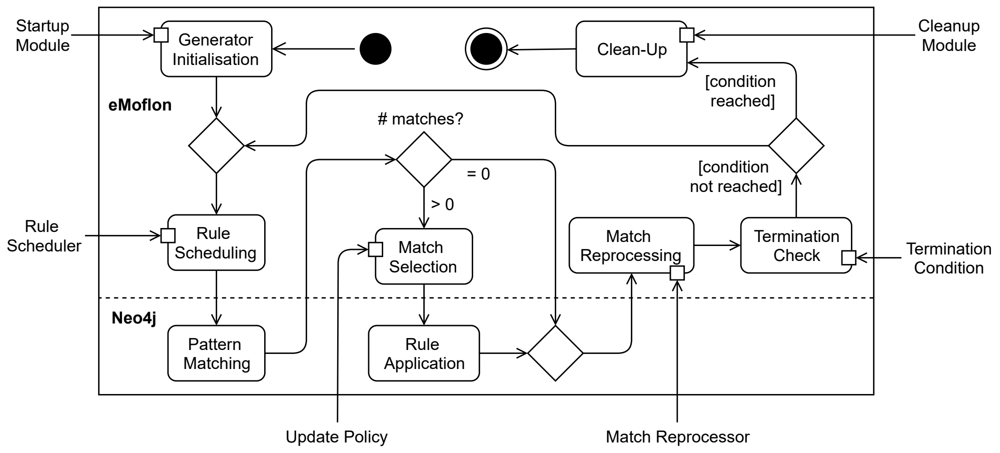


Figure 3: Core Cycle

4. Evaluation

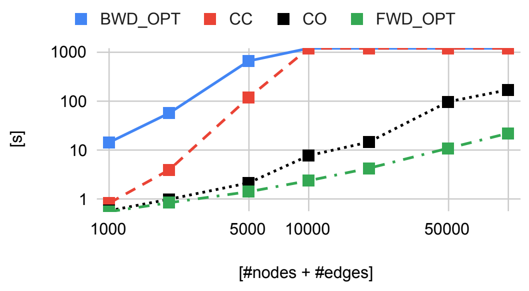
To assess the scalability of eMoflon::Neo, we measured its runtime performance using examples *FamiliesToPersons* [16], *ClassDiagramToDatabaseSchema* [17] and *CompanyToIT* [12] from the bx example repository. We compared the runtime measurements to those for our previous EMF-based tool eMoflon::IBeX [4] in a comparable setting, taking only operations with a similar implementation in both tools into account. We investigate the following research questions:

(RQ1) How does the use of graph databases relate to runtime performance? Are differences for growing (meta-)model sizes or an increasing number of rules observable?

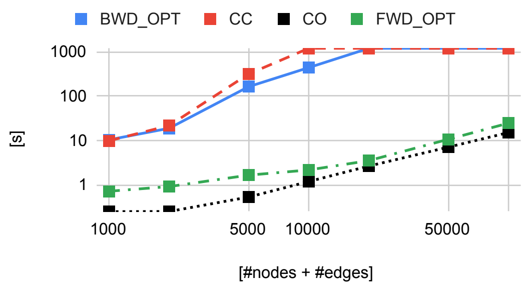
(RQ2) Are there differences between the supported operations regarding runtime performance? For which operations is the use of graph databases especially beneficial?

Setup: The three examples were tested for model sizes from 1,000 to 100,000 elements (nodes and edges). We repeated each test run five times with a time-out of 20 minutes and took the median to reduce the effect of outliers. The execution environment for the test runs was a standard notebook with an Intel Core i7 (1.80 GHz), 16GB RAM, and Windows 10 64-bit. eMoflon::Neo was installed based on an Eclipse IDE for Java and DSL Developers, version 2021-03 (4.19.0) with JDK version 13. 4GB RAM were allocated to the JVM running the tests, while 8GB were allocated to Neo4j (version 3.5.8). Gurobi 8.1.1 was used as an ILP solver.

Results: The runtime measurements for the three examples are depicted in Fig. 4a - 6b for IBeX and Neo. Note that a logarithmic scale is used on both axes to show results for small and large models in the same diagram. For FamiliesToPersons (Fig. 4), IBeX and Neo perform equally well for FWD_OPT, whereas Neo shows better scalability for all other operations. The ratio converges to 1 as IBeX reached the time-out for BWD_OPT and CC for 10,000 elements and Neo



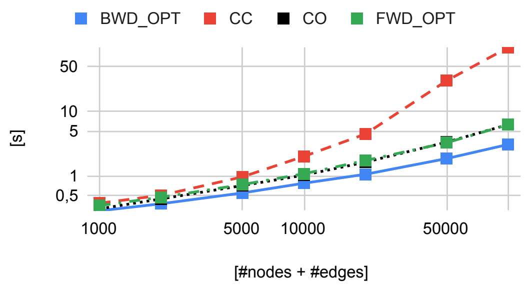
(a) eMoflon::IBeX



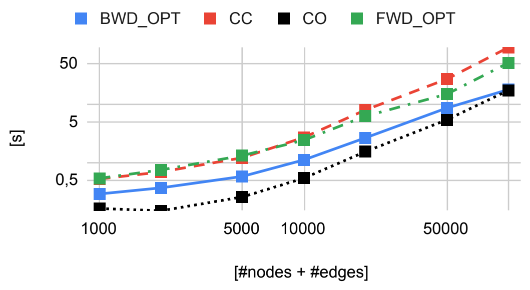
(b) eMoflon::Neo

Figure 4: Runtime Measurements: FamiliesToPersons

for 20,000 elements. IBeX appears to scale better for ClassDiagramToDatabaseSchema (Fig. 5) with the exception of CC on large models. We assume that the linear, hierarchical metamodel structures of this example are advantageous for the pattern matcher of IBeX. For CompanyToIT



(a) eMoflon::IBeX



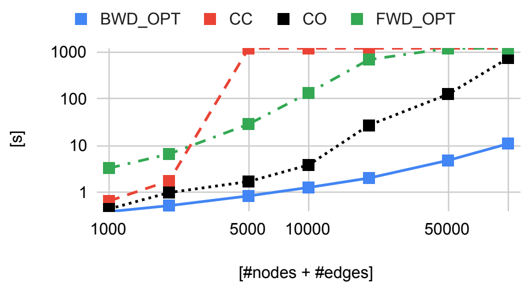
(b) eMoflon::Neo

Figure 5: Runtime Measurements: ClassDiagramToDatabaseSchema

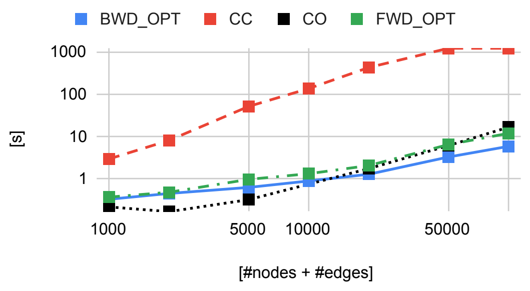
(Fig. 6), the runtime differences are substantial for all operations except BWD_OPT. The ratio decrease for FWD_OPT and CC is again caused by IBeX reaching the timeout earlier than Neo (50,000 and 5,000 elements). Compared to the other examples, CompanyToIT has slightly larger rules and metamodels, and generally a more complex pattern structure.

Summary: The results indicate that eMoflon::Neo scales better than eMoflon::IBeX with increasing rule and metamodel complexity, whereas IBeX might show a better performance for simpler TGGs (RQ1). The gain in performance, however, depends more on the nature of the concrete example than on the particular operation (RQ2).

Threats to validity: As a baseline for our comparison, we used measurements for the mid 2019 version of IBeX, which might not completely reflect the current state of the tool. We restricted the comparison to similarly implemented operations, not involving strategies such as (concurrent) model synchronisation, which is implemented very differently in both tools. As we have shown that the results strongly depend on the concrete examples, it would be important to test with further realistic (industrial) examples to gain more insights on scalability.



(a) eMoflon::IBeX



(b) eMoflon::Neo

Figure 6: Runtime Measurements: CompanyToIT

5. Related Work

There are several EMF-based TGG tools, including MoTE [18], EMorF [19], Henshin-TGG [20], and the TGG-Interpreter [21], each supporting different consistency management operations. As a successor of eMoflon::Tie [22], the EMF-based components of eMoflon::IBeX [23, 10] are similar to the corresponding components in eMoflon::Neo. For this reason, we chose eMoflon::IBeX for the runtime performance assessment in Sect. 4 in order to investigate the effect of switching from EMF to Neo4j for TGG-based consistency management. Our evaluation indicates that the combination of EMF and incremental pattern matching can be a bottleneck for larger model sizes in combination with more complex rules and metamodels. Concerning general tooling improvements, eMoflon::Neo covers *all* modelling tasks with eMSL, providing a uniform textual and visual concrete syntax. We have found this to be beneficial especially for teaching as students only have to learn how to use one tool and one consistent family of languages.

Neo4j has been used as an underlying graph database for the GT tool GRAPE [5]. An embedded Domain-Specific Language (DSL) in Closure is used to define rules, from which Cypher statements are generated to query the database. GRAPE uses a textual concrete syntax together with a visualisation, and supports graph transformations on untyped graphs.

Alqahtani and Heckel use Neo4j for TGG-based model transformations [24]. TGG rules are translated into Gremlin code, which is an alternative query language for Neo4j. In an experimental performance comparison with eMoflon::IBeX, their approach shows better scalability results. Their implementation, however, only supports forward and backward transformations without completeness guarantees, whereas eMoflon::Neo covers numerous other TGG operations. Daniel et al. [6] use Gremlin for ATL-based graph transformations in a similar fashion. Besides Neo4j, adapters for other NoSQL databases exist, such as OrientDB and MongoDB.

NeoEMF [25, 26] was recently proposed as a seamless EMF-compatible layer over Neo4j and other NoSQL databases. The advantages of graph databases with respect to scalability and the well-known EMF resource handling are synergetically combined, which makes it possible to attach NeoEMF to EMF-based bx tools. Model transformations are supported, but take place in main memory and have to be replicated in the database. Furthermore, using this intermediate layer restricts the control over how, e.g., types are mapped to Neo4j, and prevents leveraging all advantages of the particular database, such as attributed edges for bookkeeping operations.

6. Conclusion and Future Work

In this paper we presented eMoflon::Neo as a novel addition to the eMoflon toolsuite. eMoflon::Neo leverages Neo4j as a graph database for all runtime models and TGG-based consistency operations. Our comparison with a similar EMF-based TGG tool indicates that this allows for improved scalability, especially for more complex examples and for growing model size. Concerning usability of the tool, especially for teaching, eMoflon::Neo provides a novel specification language eMSL that uniformly supports modelling, metamodelling, patterns, constraints, rules and TGGs. As future work we plan to improve the interoperability with EMF-based tools regarding model and metamodel exchange. As our evaluation was restricted to a comparison with respect to runtime performance, we plan to investigate on the benefits regarding flexibility and potential drawbacks of storing (meta-)models in an external graph database. We also plan to further optimise the current set of TGG-based operations, using industrial examples of realistic size and complexity.

Acknowledgements

This work was partially supported by the North Rhine Westphalian Ministry of Economic Affairs, Innovation, Digitalisation and Energy (MWIDE) through the Pro-LowCode project (005-2011-0022). We would like to thank Deekshitha Veeraraje Urs, Jannik Hinz, Maximilian Schmidt, and Mario Treiber (Paderborn University) for their valuable contributions to developing the tool.

References

[1] A. Schürr, Specification of Graph Translators with Triple Graph Grammars, in: E. W. Mayr, G. Schmidt, G. Tinhofer (Eds.), 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*, Springer, 1994, pp. 151–163. doi:10.1007/3-540-59071-4_45.

[2] eMoflon Developer Team, eMoflon, https://emoflon.org, 2021.

[3] N. Weidmann, L. Fritsche, A. Anjorin, A search-based and fault-tolerant approach to concurrent model synchronisation, in: R. Lämmel, L. Tratt, J. de Lara (Eds.), Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2020, Virtual Event, USA, November 16-17, 2020, ACM, 2020, pp. 56–71. doi:10.1145/3426425.3426932.

[4] N. Weidmann, A. Anjorin, E. Leblebici, A. Schürr, Consistency management via a combination of triple graph grammars and linear programming, in: O. Nierstrasz, J. Gray, B. C. d. S. Oliveira (Eds.), Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2019, Athens, Greece, October 20-22, 2019, ACM, 2019, pp. 29–41. doi:10.1145/3357766.3359544.

[5] J. H. Weber, GRAPE - A graph rewriting and persistence engine, in: J. de Lara, D. Plump (Eds.), Graph Transformation - 10th International Conference, ICGT 2017, Marburg, Ger-

many, July 18-19, 2017, Proceedings, volume 10373 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 209–220. doi:10.1007/978-3-319-61470-0_13.

[6] G. Daniel, F. Jouault, G. Sunyé, J. Cabot, Gremlin-ATL: a scalable model transformation framework, in: G. Rosu, M. D. Penta, T. N. Nguyen (Eds.), Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017, IEEE Computer Society, 2017, pp. 462–472. doi:10.1109/ASE.2017.8115658.

[7] eMoflon Developer Team, eMoflon-Neo. A Neo4j-based implementation of eMoflon, https://github.com/eMoflon/emoflon-neo, 2021.

[8] Neo4j, Inc., Neo4j, https://neo4j.com/, 2021.

[9] PlantUML, PlantUML in a nutshell, https://plantuml.com, 2021.

[10] N. Weidmann, A. Anjorin, L. Fritsche, G. Varró, A. Schürr, E. Leblebici, Incremental bidirectional model transformation with emoflon: : Ibex, in: J. Cheney, H. Ko (Eds.), Proceedings of the 8th International Workshop on Bidirectional Transformations co-located with the Philadelphia Logic Week, Bx@PLW 2019, Philadelphia, PA, USA, June 4, 2019, volume 2355 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019, pp. 45–55. URL: http://ceur-ws.org/Vol-2355/paper4.pdf.

[11] The BX Community, The Bx Examples Repository, http://bx-community.wikidot.com/examples:home, 2021.

[12] The BX Community, CompanyToIT, http://bx-community.wikidot.com/examples:companytoit, 2021.

[13] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer, Fundamentals of Algebraic Graph Transformation, Monographs in Theoretical Computer Science. An EATCS Series, Springer, 2006. doi:10.1007/3-540-31188-2.

[14] Artois University, CNRS, Sat4j the boolean satisfaction and optimization library in Java, https://www.sat4j.org/, 2021.

[15] Gurobi Optimization, LLC., Gurobi Optimization. The Fastest Solver, https://www.gurobi.com/, 2021.

[16] The BX Community, FamiliesToPersons, http://bx-community.wikidot.com/examples:familytopersons, 2021.

[17] The BX Community, ClassDiagramToDatabaseSchema, http://bx-community.wikidot.com/examples:classdiagramstodatabaseschemas, 2021.

[18] H. Giese, L. Lambers, B. Becker, S. Hildebrandt, S. Neumann, T. Vogel, S. Wätzoldt, Graph transformations for MDE, adaptation, and models at runtime, in: M. Bernardo, V. Cortellessa, A. Pierantonio (Eds.), Formal Methods for Model-Driven Engineering - 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18-23, 2012. Advanced Lectures, volume 7320 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 137–191. doi:10.1007/978-3-642-30982-3_5.

[19] L. Klassen, R. Wagner, EMorF - A Tool for Model Transformations, ECEASST 54 (2012). doi:10.14279/tuj.eceasst.54.768.

[20] C. Ermel, F. Hermann, J. Gall, D. Binanzer, Visual Modeling and Analysis of EMF Model Transformations Based on Triple Graph Grammars, ECEASST 54 (2012). doi:10.14279/tuj.eceasst.54.771.

[21] J. Greenyer, E. Kindler, Comparing Relational Model Transformation Technologies: Implementing Query/View/Transformation with Triple Graph Grammars, Software and System Modeling 9 (2010) 21–46. doi:10.1007/s10270-009-0121-8.

[22] E. Leblebici, A. Anjorin, A. Schürr, Developing eMoflon with eMoflon, in: D. D. Ruscio, D. Varró (Eds.), 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*, Springer, 2014, pp. 138–145. doi:10.1007/978-3-319-08789-4_10.

[23] N. Weidmann, A. Anjorin, P. Robrecht, G. Varró, Incremental (unidirectional) model transformation with emoflon: : Ibex, in: E. Guerra, F. Orejas (Eds.), Graph Transformation - 12th International Conference, ICGT 2019, Held as Part of STAF 2019, Eindhoven, The Netherlands, July 15-16, 2019, Proceedings, volume 11629 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 131–140. doi:10.1007/978-3-030-23611-3_8.

[24] A. Alqahtani, R. Heckel, Model based development of data integration in graph databases using triple graph grammars, in: M. Mazzara, I. Ober, G. Salaün (Eds.), Software Technologies: Applications and Foundations - STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers, volume 11176 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 399–414. doi:10.1007/978-3-030-04771-9_29.

[25] G. Daniel, G. Sunyé, A. Benelallam, M. Tisi, Y. Vernageau, A. Gómez, J. Cabot, NeoEMF: A multi-database model persistence framework for very large models, in: J. de Lara, P. J. Clarke, M. Sabetzadeh (Eds.), Proceedings of the MoDELS 2016 Demo and Poster Sessions co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2016), Saint-Malo, France, October 2-7, 2016, volume 1725 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 1–7.

[26] G. Daniel, G. Sunyé, A. Benelallam, M. Tisi, Y. Vernageau, A. Gómez, J. Cabot, NeoEMF: A multi-database model persistence framework for very large models, Sci. Comput. Program. 149 (2017) 9–14. doi:10.1016/j.scico.2017.08.002.