Heterogeneous Model Query Optimisation

Qurat ul ain Ali
Department of Computer Science
University of York
York, UK
quratulain.ali@york.ac.uk

***Abstract*—With the growing size and complexity of software systems, the underlying models also grow in size proportionally. These large-scale models pose scalability issues for model-driven engineering technologies. These models can be persisted in various backend technologies (such as file systems, document and relational databases) and can be represented in different formats such as XMI and Flexmi. Several tailored high-level model management languages such as OCL and EOL enable developers to work on different backend technologies in a uniform way by shielding them from the complexities of different backends. On the contrary, performance with respect to execution time in tailored model management languages programs becomes one of the major scalability bottlenecks. In this work, we propose an architecture built on top of existing model query languages to facilitate query optimisation. The proposed approach will benefit from compile-time static analysis and automatic program rewriting to optimise queries operating over heterogeneous backend technologies. Optimisation strategies and performance will vary depending on the type of queries and the backend modelling technology. We expect to significantly improve performance (decrease in one order of magnitude of execution time) for model management programs, particularly over large-scale models.**

***Index Terms*—model querying, static analysis, model-driven engineering**

I. Problem and Motivation

Model-driven engineering (MDE) has proven to have several benefits over traditional software development methodologies such as quality, maintainability and productivity. To continue the broader use of MDE in industrial projects, it is crucial that MDE technologies scale well with larger and complex applications. A typical MDE workflow includes several tasks, including model validation, model-to-model transformations, and model-to-text transformations. All these tasks mentioned above have a common set of queries/expressions operating over model elements. As these queries grow complex, they will significantly impact performance both in terms of execution time and memory footprint. Moreover, usually in industrial projects, most MDE tools have poor performance handling very large models (VLMs) [1]. One example of large scale models in the automotive industry are the models of the Automotive Open System Architecture (AUTOSAR) [2], containing millions of model elements. While executing complex and computationally expensive queries over such large models, a significant performance cost in terms of execution time is incurred. Scalability is one of the main challenging factor in the industry adoption of MDE [3].

A. Motivating Example

The problem is demonstrated with the help of a motivating example. Consider a small excerpt of Java metamodel as illustrated in Fig 1. *SingleVariableAccess* has a non-containment reference to *VariableDeclaration* named as *variable*. A vali-

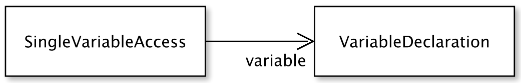

Fig. 1. Excerpt of Java metamodel

dation program can be written in Epsilon Validation Language - a language of Epsilon[1] to validate a model conforming to a metamodel. An example EVL validation to validate models conforming to Java metamodel is shown in Listing 1.

```
model Java driver EMF {
  nsuri = "http://www.eclipse.org/MoDisco/
  Java/0.2.incubation/java"
};
pre{
}
context Java!VariableDeclaration {
  constraint variableIsUsed {
    check: Java!SingleVariableAccess.all
    .select(sva|sva.variable=self)
    .notEmpty()
  }
}
```

Listing 1. Example EVL constraint before optimisation

In the constraint *variableIsUsed*, we are checking that every *VariableDeclaration* is accessed at least once. If we evaluate this constraint over a model containing M number of *VariableDeclaration*s and N number of *SingleVariableAccess*. The cost of evaluating the constraint would be $O(M*N)$.

One possible optimisation is to detect such expressions where all instances of a type are filtered based on a specific field. We can pre-compute an index based on this and then just retrieve instances by searching through this pre-computed index. Such an optimisation would reduce the cost of evaluating the same contraint to$O(M)+O(N)$, considering the complexity

[1]https://www.eclipse.org/epsilon/

of computing and populating an index as *O(M)* and searching this index as *O(1)*.

```
1 model Java driver EMF {
2   nsuri = "http://www.eclipse.org/MoDisco/
3   Java/0.2.incubation/java"
4 };
5 pre{
6   Java.createIndex("SingleVariableAccess",
7   "variable");
8 }
9 context Java!VariableDeclaration {
10   constraint variableIsUsed {
11     check: Java.findByIndex(
12       "SingleVariableAccess","variable",
13       self).notEmpty()
14     }
15   }
```

Listing 2. Example EVL constraint after optimisation

Such an optimisation can be applied behind the scenes to the program shown in Listing 1, and the optimised program can be automatically rewritten as shown in Listing 2. In this case, an index is pre-computed in Line 6 and then searched through in Line 11. This is just one optimisation pattern, there can be other efficient ways to execute a program, such as translating the high-level program to the native language of the model persistence technology. The aim of this PhD work is to produce such a solution that can provide optimisation for model management programs(such as query, validation, transformation) operating over heterogeneous model persistence technologies. This would enable the developers to write their code in a technology-agnostic form (such as in EVL, ETL), while still benefiting from technology specific optimisations. The optimisation algorithm/strategies would be different depending on the model persistence technology. Such optimisations enable reusing performance benefits already provided by the model persistence technologies. Programs operating over multiple persistance technologies (such as Simulink, EMF) concurrently can also benefit from the proposed optimisations.

B. *Research Objectives*

- **RO-1:** Identify the performance challenges involved in executing complex queries over large models represented in heterogeneous formats (EMF, Simulink etc.) and stored in different back-ends (XMI files, relational databases etc.)
- **RO-2:** Identify reusable optimisation primitives and patterns across different formats and back-ends using static analysis of high-level language code
- **RO-3:** Propose algorithms for optimisation of queries operating on low-code system models captured using different modelling languages and model representation formats.
- **RO-4:** Ensure to preserve the semantics of the original program while rewriting the optimised program
- **RO-5:** Evaluate the results of the proposed algorithms in terms of execution time and memory footprint over various back-end technologies.

II. Related work

In this section, we will present various state-of-the-art approaches related to scalable model querying over heterogeneous modelling back-ends.

There are two principal categories for querying models from a language perspective: i) Native querying ii) Back-end independent querying. Being the most straightforward approach, the former is very efficient as it is tailored for the back-end persistence technology. Examples include Cypher or Gremlin for NoSQL databases and SQL for relational databases. These query languages are specifically created for the back-ends they target, often providing efficient querying (e.g. index backed) methods. While being efficient for their particular back-ends, they have drawbacks when it comes to heterogeneous technologies [4] as they often cannot be used or will not be performant for other technologies. Moreover, if model persistence is changed, this technology-specific querying approach requires considerable effort to change model management programs relying on it to function.

On the contrary, back-end independent querying queries models using high-level languages that abstract over heterogeneous back-ends and model representation formats. Examples of such type of high-level languages include OCL and EOL. This abstraction is commonly implemented using an intermediate layer such as the OCL pivot metamodel [5] or the Epsilon Model Connectivity (EMC) Layer[2].

Queries written in high-level languages can be translated to their appropriate native query language. Several researches have proposed ways of such mappings. For example a tool called Mogwai [6] translates OCL and ATL expressions to Gremlin scripts - a query language for NoSQL databases. This shifts the majority of the computation of queries to the database layer, and it makes use of possible optimisation strategies for this back-end technology. Similarly, in [7] authors have presented a tool that generates SQL queries from OCL expressions. In [8], a runtime translation of SQL is presented that enables querying relational databases using EOL. This translation doesn't take into account any static analysis and program rewriting to optimise the translated program.

Another approach, for pre-computing the expensive sub-expressions to speed up complex queries, is presented in [9]. Users define these as derived attributes which are then cached to speed up queries utilising such attributes. This approach does not automatically detect such expensive sub-expressions and requires specification by the developer both for their identification and implementation.

Program-aware strategies for optimising queries make use of compile-time static analysis. A program-aware approach is presented in [10], which pre-computes and caches *allInstances*

[2]https://www.eclipse.org/epsilon/doc/emc/

of a type, if a program makes multiple calls to *allInstances*. Another optimisation strategy as proposed in [11] suggests how combining parallelisation, lazy evaluation and short-circuiting can significantly increase the performance of queries over large models.

In [12], an allocation optimisation approach is presented by a combination of heuristics, to reduce the usage of resources by minimising remote network traffic and computation cost, minimising resources needed for evaluation of queries.

A distributed index-based execution of model validation programs is introduced in [13]. A validation program is decomposed and distributed among several machines and cores, to perform parallel execution. This is implemented for EVL programs and is performant on multiple machines, especially ones with multiple logical cores.

A compiler is presented in [14] which uses static information extracted by ANATLyzer [15]. The compiler generates efficient code by optimising at the transformation level. This approach is limited to EMF-based models, and such optimisation is done at the transformation level for the model to model transformations only.

III. Proposed Solution

Considering the research challenges and objectives, we propose an approach for optimising queries operating over heterogeneous low-code system models. The architecture of the envisioned approach is depicted as a block diagram in Fig 2. The primary purpose of this framework is to be able to automatically rewrite expensive queries in an input model management program written in technology agnostic language operating over heterogeneous models to a more efficient form. The rewritten program should be efficient in terms of execution time. The rewriting process will be taking memory footprint into consideration to make a trade-off. Program rewriting is based on information extracted through static analysis. The rewriting and optimisation will vary based on the specific backend technology or technologies the query is operating over. To our knowledge, we have not found the solution to this problem in the literature. In a low-code platform, the underlying models can be of different modelling technologies and stored in different backend formats. After a model management program is parsed, an Abstract Syntax Tree (AST) is generated. This AST may not include any type information attached to its nodes. Before execution, the static analyser component will analyse the program and populate type-related information into the AST, also referred to as an Abstract Syntax Graph. Abstract Syntax Graph would then be used by the rewriters involved in a program (depending on the type of models it needs to access), to rewrite this program behind the scenes into an optimised form.

A. Static Analysis

The static analyser component analyses the program and the metamodels to which the source/target models conform. The static analyser consists of a visitor that traverses the input program sequentially and will populate type information for every element of this program, to yield a type-resolved abstract syntax graph. Static analysis is a fundamental block and can be used for various purposes, such as to facilitate program editing and error checking. For this purpose, we use this type information to check type compatibility to produce the necessary compile-time errors as a helpful by-product.

B. Query Optimisation

A query optimisation block can contain several optimisers, but which ones will actually be invoked depends on the input program. If a program involves cross-validation of a Simulink and an EMF model, their respective rewriters/optimisers would be invoked. This entire process is done before the execution of the program. The reason for using several orthogonal optimisers is twofold:

- Technologies may offer different optimisations specific to their backend storage, such as index-backed methods provided by Simulink.
- If there was a single optimiser, then it would have to know about all the other models accessed by the program in question to function.

SQL Query Optimiser: In SQL query optimiser, we propose to translate the queries to the native language of the underlying persistence technology, i.e. SQL. This is to take benefit of the optimisations already provided by the underlying technology. Tailored SQL queries are much faster on relational databases to execute compared to high-level languages using only basic query functions (like a blanket select * expression to return all instances of a type, to be filtered afterwards by the execution engine). We take in the AST and visit each node to translate to the constructs of SQL as in [16]. For instance, *Database!Requirements.all.select(r|r.id = 45)* would be translated to *Database.runSql("select * from Requirements where id =45")*

EMF Query Optimiser: EMF is a widely used modelling technology in the modelling community. To efficiently query large-scale EMF models, we use a static analyser to analyse the program and detect which custom indices can be created. One possible optimisation is that a given model management program is visited sequentially to detect expressions which include filtering all instances based on one or more attributes. This class of expressions can make use of such in-memory indices at run-time to execute faster. The program is then automatically rewritten to search through the created in-memory custom indices. Creating custom indices also has an overhead, which is expected to regularly pay off in the case of large-scale models, when the index is used multiple times in the program. Call graph analysis is used to decide whether an expression can potentially be executed multiple times within a single program. The analysis checks if an expression is inside the body of a loop or it's a part of an operation which is called from a loop. If an optimisable expression is detected to be potentially executed multiple times, it is considered to be a candidate for indexing.

Another optimisation for programs operating over EMF-based models is translating certain expensive Epsilons expres-

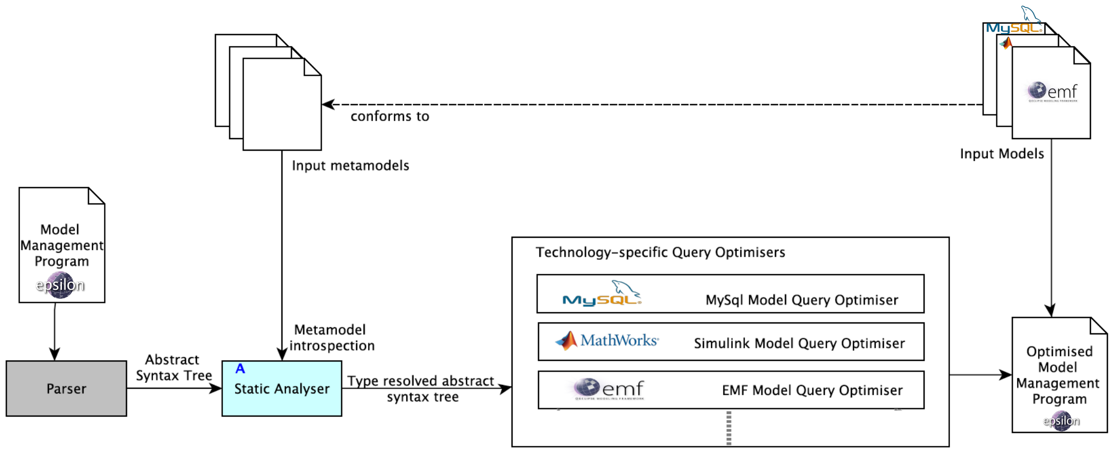
conforms to
Input metamodels
Input Models
Model Management Program
Technology-specific Query Optimisers
MySql Model Query Optimiser
Simulink Model Query Optimiser
EMF Model Query Optimiser
Metamodel introspection
Parser
Abstract Syntax Tree
Static Analyser
Type resolved abstract syntax tree
Optimised Model Management Program

Fig. 2. Architecture of the proposed solution

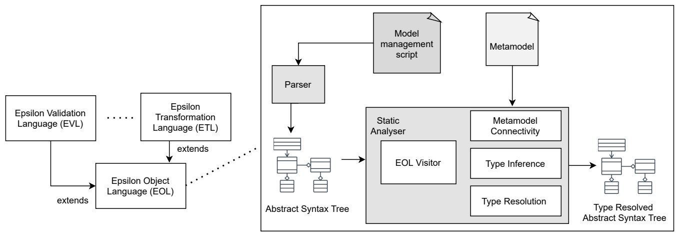

Fig. 3. Architecture of static analysis

sions into Viatra patterns. This is to leverage the benefit of the incremental engine of Viatra. The plan is to translate first-order logical operations to their corresponding Viatra graph patterns and then execute those translated patterns by invoking the Viatra Engine. This is expected to potentially reduce the execution time in the case of large-scale models.

For model to model transformations over EMF models, a dependency graph would be created between rules so that when an equivalent rule is called, it should not be searched in the transformation trace. For transformation chains, the chain would be statically analysed to see what rules are mandatory to be executed depending on if they actually take part in generating the end target model.

Simulink Query Optimiser: In the context of model management programs executing over Simulink models, the execution time can be reduced by leveraging the various built-in MATLAB commands, such as index-backed built-in methods. We propose to translate expressions which can utilise such built-in commands to compute results more efficiently. This translation to MATLAB commands would be performed at compile-time with the help of information extracted from static analysis. For Instance, if we have a given EOL expression: *Simulink!Subsystem .allInstances.exists(s|s.name = self.subsystem)* it can be translated to a index backed MATLAB method *findBlocks()* as : *Simulink.findBlocks('Subsystem' ,'name', self.subsystem).notEmpty()*

IV. Plan for Evaluation and Validation

In this section, I will present how the correctness of the rewritten program will be validated. Also, I will describe the process to evaluate the proposed solution in comparison with other state-of-the-art approaches.

A. Validation

As the proposed solution for optimising queries will rewrite the program behind the scenes, it is important to ensure that semantics of the original program are preserved. I will use automated JUnit equivalence tests for validating the correctness of our rewritten program, comparing the results obtained from the execution of the original program with those of the rewritten program.

B. Evaluation

The proposed solution will be evaluated on validation constraints and transformations operating over large EMF models available in LinTra [3] conforming to the Java metamodel by MoDisco [17] - a model-driven reverse engineering project. LinTra has large models reverse-engineered from the source code of Eclipse projects ranging from 100K to 4.35M model elements. I plan to evaluate the proposed approach preferably on model management programs that access and query, validate or transform different models such as MySQL, Simulink and EMF concurrently to see if the approach performs efficiently in case of heterogeneous models. I will use publicly available large Simulink models and model management programs from GitHub.

V. Expected Contributions

The concrete contributions of this doctoral research are listed as follows:

Contribution 1: A framework on-the-top of existing model management languages for reducing the execution time of queries operating over heterogeneous modelling technologies.

[3]http://atenea.lcc.uma.es/projects/LinTra.html

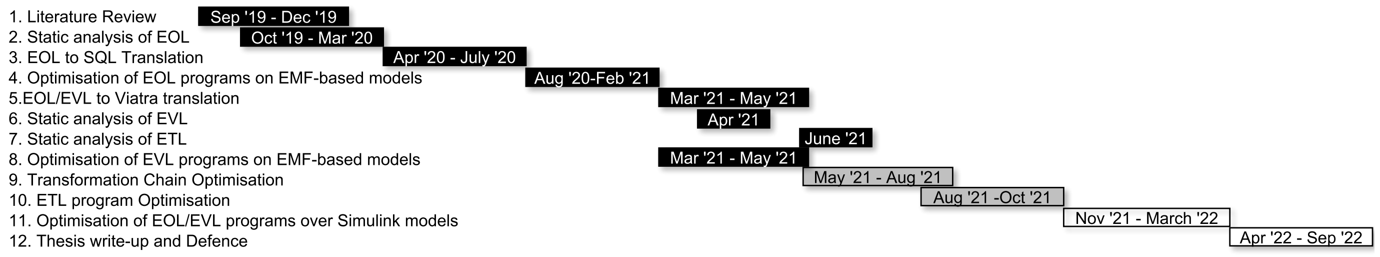

Fig. 4. Gantt Chart of Research Tasks

Contribution 2: A static analyser component to populate static information such as types (based on input/output meta-models) for model management programs.

Contribution 3: A query translation faciltiy for automatically translating an appropriate subset of EOL expressions to their corresponding optimised SQL queries.

Contribution 4: A query rewriting faciltiy for optimising EOL, EVL and ETL programs over EMF models, by performing a number of optimisations such as use of custom indices or translating to Viatra patterns.

Contribution 5: A query translation faciltiy for programs operating over Simulink models. This facility will automatically translate EOL expressions to their corresponding optimised MATLAB commands (such as leveraging index-backed operations).

Contribution 6: An equivalence test suite for ensuring the validity of rewritten programs (such that the semantics of the original program are preserved).

VI. Current status and Further Work

In this section, I will list the results achieved so far, current status of the research work and further work along with timeline. The overall progress is depicted in the Fig 4.

A. Milestones Achieved

I started my doctoral program by reviewing a large number of research papers (See Fig 4 highlighted in black) related to scalability challenges in MDE then model querying in particular. I also spent some of the time reviewing literature related to query optimisation in databases, as this is a well-studied problem with similar objectives. From a practical perspective, I started familiarising myself with MDE frameworks such as EMF, Epsilon, Viatra. After analysing literature, an approach that uses compile-time static analysis and several query optimisers/rewriters is proposed, which is also presented as a workshop article in [16].

Meanwhile, I started working on adding static analysis capabilities to Epsilon. As all other Epsilon languages are built on top of EOL, I started by implementing the static analysis of the core language of Epsilon i.e. EOL and then extended it for EVL and ETL to add support for language specific constructs. Static analysis is open-source and is publicly available at [18] to be used on top of Epsilon.

After implementing static analysis, I implemented a compile-time mapping strategy from a subset of EOL expressions to SQL queries. This is to efficiently query relational databases while still using tailored model management languages.

For EMF-based models, I implemented optimisation of type-level queries using static analysis by creating in-memory indices. We plan to submit this work as a conference paper. Recently I worked on translating EOL expression to graph patterns (VQL) [19] for efficiently querying large-scale EMF models. VQL's incremental engine considerably reduce the execution time. We submitted an article at the 2nd Lowcode workshop at MODELS '21.

B. Current Status

Currently, I am working on (See Fig 4 highlighted in grey) optimising ETL transformations for EMF-based models at the rule level (exploiting dependencies between rules). In the same direction, I am working on utlitising static type information in the optimisation of transformation chains. I am also working on optimising various expressions in EOL and EVL programs by exploiting static information already extracted.

C. Next Steps

After the above-mentioned steps, I plan (See Fig 4 highlighted in white) to devise a strategy for compile-time translation of EOL expressions to native MATLAB commands for efficiently querying Simulink models. Finally, I will conclude the results obtained along with testing and benchmarking and write the dissertation by Sep 2022.

Acknowledgments

The work in this paper has been partially supported by the Lowcomote Training Network [20], which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie grant agreement n° 813884. I would like to thank my supervisors Prof. Dimitris Kolovos and Dr Konstantinos Barmpis for their continuous guidance throughout this program.

References

[1] M. Tisi, S. Martínez, and H. Choura, "Parallel execution of atl transformation rules," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2013, pp. 656–672.

[2] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange, "Autosar–a worldwide standard is on the road," in *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, vol. 62, 2009, p. 5.

[3] J. Hutchinson, J. Whittle, and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," *Science of Computer Programming*, vol. 89, pp. 144–161, 2014, special issue on Success Stories in Model Driven Engineering. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642313000786

[4] K. Barmpis and D. Kolovos, "Evaluation of contemporary graph databases for efficient persistence of large-scale models." *The Journal of Object Technology*, vol. 13, p. 3:1, 07 2014.

[5] E. Willink, "Aligning ocl with uml," *ECEASST*, vol. 44, 01 2011.

[6] G. Daniel, G. Sunyé, and J. Cabot, "Scalable queries and model transformations with the mogwai tool," in *International Conference on Theory and Practice of Model Transformations*. Springer, 2018, pp. 175–183.

[7] M. Egea, C. Dania, and M. Clavel, "Mysql4ocl: A stored procedure-based mysql code generator for ocl," *ECEASST*, vol. 36, 01 2010.

[8] D. Kolovos, R. Wei, and K. Barmpis, "An approach for efficient querying of large relational datasets with ocl-based languages," vol. 1089, pp. 46–54, 01 2013.

[9] K. Barmpis and D. S. Kolovos, "Towards scalable querying of large-scale models," in *European Conference on Modelling Foundations and Applications*. Springer, 2014, pp. 35–50.

[10] R. Wei and D. S. Kolovos, "An efficient computation strategy for allinstances ()." in *BigMDE@ STAF*, 2015, pp. 32–41.

[11] S. Madani, D. Kolovos, and R. F. Paige, "Towards optimisation of model queries: a parallel execution approach," *Journal of Object Technology*, vol. 18, no. 2, 2019.

[12] J. Makai, G. Szárnyas, I. Z. Ráth, D. Varró, and Á. Horváth, "Optimization of incremental queries in the cloud," 2015.

[13] S. Madani, D. Kolovos, and R. Paige, "Distributed model validation with epsilon," *Software and Systems Modeling*, 03 2021.

[14] J. Sánchez Cuadrado, L. Burgueño, M. Wimmer, and A. Vallecillo, "Efficient execution of atl model transformations using static analysis and parallelism," *IEEE Transactions on Software Engineering*, vol. PP, pp. 1–1, 07 2020.

[15] J. S. Cuadrado, E. Guerra, and J. de Lara, "Static analysis of model transformations," *IEEE Transactions on Software Engineering*, vol. 43, no. 9, pp. 868–897, 2017.

[16] Q. Ali, D. Kolovos, and K. Barmpis, "Efficiently querying large-scale heterogeneous models," in *International Conference on Model Driven Engineering Languages and Systems*. ACM, 2020, pp. 1–5.

[17] H. Brunelière, J. Cabot, G. Dupé, and F. Madiot, "Modisco: A model driven reverse engineering framework," *Information and Software Technology*, vol. 56, no. 8, pp. 1012–1032, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584914000883

[18] "Static Analysis built on-the-top of Epsilon," https://github.com/epsilonlabs/static-analysis.git.

[19] D. Varró, G. Bergmann, Á. Hegedüs, Á. Horváth, I. Ráth, and Z. Ujhelyi, "Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework," *Software and Systems Modeling*, vol. 15, no. 3, pp. 609–629, 2016.

[20] M. Tisi, J.-M. Mottu, D. S. Kolovos, J. De Lara, E. M. Guerra, D. Di Ruscio, A. Pierantonio, and M. Wimmer, "Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms," in *STAF 2019 Co-Located Events Joint Proceedings: 1st Junior Researcher Community Event, 2nd International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems, and 1st Research Project Showcase Workshop co-located with Software Technologies: Applications and Foundations (STAF 2019)*, Jul. 2019.