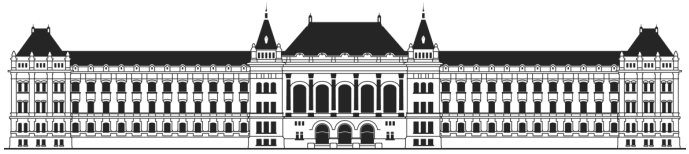

MŰEGYETEM 1782
Budapest University of Technology and Economics
Department of Measurement and Information Systems

Program Analysis Techniques for Model Queries and Transformations

PhD thesis booklet

Zoltán Ujhelyi
MSc in technical informatics

Advisor:
Dr. Dániel Varró, DSc
professor

Budapest, 2016

1 Preliminaries

Model-driven Engineering The discipline of *model-driven engineering (MDE)* [BG05; WHR14] aims to address the increasing complexity of software development by focusing on the definition and analysis of high level models that describe various aspects of the developed system on different levels of abstraction. This approach provides better understandability, faster development lifecycle and increased software quality, e.g. fewer bugs [HWR14], which makes MDE especially useful in critical system development where system failure may lead to human injury or significant damage in property [Bro+14].

Simultaneous increase in quality and productivity is primarily achieved in MDE by early validation of system models and automated generation of various design and implementation artifacts. Early validation allows to highlight conceptual flaws well before such flaws are detected by traditional testing techniques. Automated source code generation from high-level behavioural models reduces both development and verification time. Furthermore, high-level models also enable the automated synthesis of test cases, configuration files, documentation, etc.

As models become ubiquitous as the main artifacts in systems development in many application domains [Béz05], a wide range of modeling languages have been introduced. They may represent the software or hardware architecture (e.g., UML [Obj11c], SysML [Obj10] or AADL [SAE09]), the external or internal behavior (e.g., UML statecharts [Obj11c], BPMN [Obj11a], BPEL [OAS07]), the reliability or performance of components (e.g., MARTE [Obj11b]) or support formal analysis, such as Petri nets [Mur89] to model distributed system or Kripke structures used in model checking [Cla08]. Furthermore, particular industries often make use of platform description languages, such as AUTOSAR [AUT12] in the automotive domain, or ARINC 653 for Integrated Modular Avionics [Pri08].

Model Queries and Transformations Several important engineering tasks supported by model-based tools rely on *model queries*. Such queries, by definition, retrieve those elements from a model that satisfy a query condition, which can be a complex composition of subconditions. Their applications include early model analysis [Aut+16; Bar+15], like well-formedness constraint validation or the calculation of various model metrics. Moreover, model queries can to ease the derivation of higher-level view models.

Automated *model transformations (MT)* are used to create or update models in different modeling formalisms from one or more existing models. Model transformations may generate various artifacts, like test cases, documentation or other models [Bon+99; Rus08]. Furthermore, by the generation of formal models, it is possible to execute formal analysis on the developed models while effectively hiding the theoretical mathematical details and enabling the designers to use well-known, high-level models to specify their systems [Bon+99].

Code generators that map models to textual artifacts, like source code or configuration descriptors can be considered as a special kind of model transformation where the output is either directly text, or an abstract syntax graph of the textual output. Such generators, like Acceleo [Acc], the Epsilon Generation Language (EGL) [Epsilon; Ros+08] or Xtend [Xtend], are widely used in modeling tools, and are often considered an important aspect of productivity gain by MDE [HWR14].

The correctness of such model queries and transformations is critical as undetected errors can invalidate the analysis results or they might propagate into the target application. Due to the increasing complexity of queries and transformations, ensuring their correctness becomes increasingly difficult.

1.1 Analysis of Model Queries and Transformations

Model transformations are captured in the form of a model transformation program, which is a regular piece of software. Elementary model transformation steps are often formulated by highly data-driven, declarative transformation rules, while imperative control structures help assemble complex transformations from these elementary steps. Furthermore, the input and output models of transformations are complex data structures. As a consequence of this, existing validation results of software engineering has to be extended when adapting them for transformation programs [Bau+06; Küs06], especially for designing complicated, industrial transformations, where debugging and validation plays a crucial role.

There is a wide range of techniques available for the verification of model transformations, surveyed in [AW13]. Following its categorization, the approaches range from generating test cases and oracles, formal analysis using theorem proving, model checking to graph theory-based proofs.

- The main challenge of *testing model transformations* [Bau+10] is their complex, but structured input and output data (models), the use of model management environments (e.g. to create or validate models), and a large set of transformation languages and techniques. To generate instance models that cover the transformation programs sufficiently, often equivalence partitioning [Stu+07; WKC08] is used. In [FSB04] an alternative, mutation-based approach is proposed. To provide test oracles, multiple approaches are available [MBL08]: provide a known-to-be-correct reference transformation, and compare the results; provide an expected output model manually; or describe the expected results using constraints, e.g., by the use of OCL [Bau+06] or graph patterns [OW09].

- *Theorem proving* techniques are also very diverse. *Direct* approaches consider transformation rules directly, either by generating an equivalent of the transformation as the input of a theorem prover tool [Cal+11], or by extracting some invariants from the specifications [Cab+10; ALL09]. Other, *indirect* approaches verify properties of the output model, such as executing well-formedness analysis [ER10] or checking the conformance to semantics metamodel [Bar+08]. Further techniques, including the Amphion tool [Van+98] used by NASA, or the AutoBayes code generator [Sch+03], provide evidence to check its results. In general, indirect approaches are less complex as they only consider the outputs generated by the transformation, however, the transformation itself is not verified, and requires additional analysis after each execution.

- *Model checking* methods focus generally on verifying dynamic properties of the model transformations, such as termination using techniques like state space exploration [LBA10; Var04; Ren04] or bisimulation [KN07]. A mapping to Colored Petri nets was proposed in [Wim+09] to verify static and dynamic properties, including termination or liveness. The main benefit of model checker tools is that in case of verification failure, a counterexample is retrieved showing an execution trace to the location of the fault; on the other hand, the well-known state-space explosion problem frequently limits its practical usability. An additional problem in case of models is that the graphs themselves may grow (creation or deletion of new objects) resulting in infite state-spaces [BCK08; RD06] requiring the introduction of some abstractions over the graph models.

- As models are usually expressed as graphs, the verification of transformation methods, especially graph transformations, can rely on mathematical proofs based on graph theory. Several transformation properties are already evaluated this way, such as termination [BKZ14; Var+06] or confluence [Küs06; TG15]. Additionally, as graph transformation

approaches can be used to specify bidirectional transformations, verification of invertibility [Her+11] is also useful.

When comparing the various techniques, a commonality is that their usage requires medium-to-high level effort [AW13], since the respective papers focus more on the core algorithms themselves rather than their automation and adaptation. Additionally, the sheer complexity of the analysis tasks make most verification approaches computationally intensive.

A complementary approach would be the use of lightweight analysis techniques, such as static type checking or (pattern-based) static analysis tools (like the FindBugs [Fin] or PMD [PMD14] tools for Java programs) to identify common developer problems during development. However, less research has been dedicated to investigate such techniques for model transformations. In fact, such techniques are omitted from [AW13] despite their practical usefulness is already proven in other software engineering applications.

1.2 Debugging Model Queries and Transformations

In traditional software engineering if a (permanent) software *fault is detected* in a program, developers begin a process called debugging [STJ83], where the main goal is to *localize the fault*. Finally, the localized fault is *repaired*.

Fault detection can either be a result of automated or manual testing, or it can be initiated by a report from the end-user. Generally, fault localization is considered the most complex part of debugging; localized faults are usually easy to fix. Debugger tools usually allow stopping the program execution at any given time using *breakpoints*, and the *current program state*, such as the set variables, is presented while allowing modifications as well.

Debugging transformation programs [MV11] may conceptually work similarly. Usually, faults are detected by finding incorrect output models, resulting in simple questions [HLR07] like "Why are there no objects of a selected type in the target model?" or "What might have caused this violation of a metamodel constraint?".

However, fault localization is problematic for similar reasons as testing model transformations: inputs and outputs are complex, structured models defined in model management environments. When halting a transformation program at a *breakpoint*, the current state must include the program variables and *the models* as well. Furthermore, in case of large models, context-sensitive filtering is required to visualize the models meaningfully.

Furthermore, in case of high level transformation languages, especially that of declarative nature and/or graphical syntax, there might be a large conceptual gap between transformation definition and execution. E.g., most transformation languages offer primitives to evaluate complex conditions, like OCL expressions or graph patterns. This is especially the case for translation-based approaches where a regular piece of software is derived from the transformation specification [Wag+11].

There are different approaches that address some issues of debugging model transformation programs, such as input model tainting [Dho+10] that mutates existing models to identify corner cases where the transformation is incorrect; or the use of omniscient or forensic debuggers [HLR07; Cor14] that take logged execution traces, and allow both forward and backward stepping during the executed steps.

Still, current transformation development environments provide only limited debugger support, requiring additional work for the remaining challenges such as integrated model visualization or filtering support.

2 Research Questions and Challenges

My research has been motivated by practical challenges to assist the lightweight analysis of model queries and transformations for developers by locating and highlighting errors. As opposed to complex verification approaches, my focus was to propose lightweight techniques that provide direct, integrated feedback to developers with a fast response time.

Such lightweight analysis techniques are often useful as they can be fast enough to report errors as soon as the developer commits them, making it easy to provide a fix. Similarly, during debugging, analysis techniques may provide a hint where the problem is potentially located. Lightweight techniques are less precise than formal verification approaches, thus they cannot guarantee that a query or transformation is free of flaws. However, their fast, integrated nature complements verification techniques by providing early error detection.

I identified three challenges for the lightweight analysis of model queries and transformations.

Challenge 3: How to observe the structure of models during execution?

During the debugging of model transformations it is common that certain parts of model elements are important with regards to the error being investigated, but no model visualization is available that directly *displays* a developer-specified subset of the models, and *updates* it as the transformation executes. Such a visualization would be useful to monitor the active state of a(n internal) state machine or follow the creation of traceability links during the process of model transformation.

There is a large number of methods and tools for displaying graphical views of models [God+07; Bul08; WEK04]. However, they are not suitable for the definition of ad-hoc visualizations required by the transformation debugging scenario, as they either require manual customization to filter the model to display the right information, or require coding to create the visualization, making the approach more expensive to use.

Furthermore, they either require to export the model before they can be displayed, or require additional programming effort to display the model. Model export can usually be written easily - in some cases, such as in the GrGen.net tool [Gei+06] it is available with the framework, - however, to monitor the changes in the transformation this model needs to be updated. Manually integrating visualization tools to the transforation is a possibility, but for transformation debugging scenarios the implementation effort often exceeds the cost of finding the related error without the help of such a visualization.

Other environments like Sirius [Sirius] or Papyrus [Papyrus] allow defining view models, but they still require significant effort to develop (mainly because the complex feature set of the tools), and updating the visualization after changes requires careful planning.

In this thesis I propose a lightweight approach to define graphical views based on annotated graph patterns. The approach relies on incremental graph pattern matching and event-driven model transformations to provide an always current notation model that can be displayed using a wide range of model visualization tools.

Challenge 1: How to efficiently detect frequent flaws in model query and transformation specifications?

During model query and transformation development, it is easy to make mistakes that result in syntactically correct queries that always return an undesired result set. These problems are hard to find in a debugging session because of the high-level, declarative specification. On the other

hand, some common causes can be efficiently identified, such as incorrect ordering of parameters during a call; referencing an incorrect variable or even some copy-paste errors, e.g. missing or extra constraints in a query.

As these issues are usually trivial to fix once identified, automatic detection is most effective if it can be executed when editing the queries or transformations. However, this requires an efficient approach that does not require a high level of resources. Furthermore, it is required to back-annotate the internal analysis results to the original source code, and present the results visually.

Type checking and the verification of the graph structure of pattern definitions are two ways to detect common errors statically: they can be executed fast, while still identifing common problems mentioned before. In this thesis, I will propose to perform such analysis over model queries and transformations and evaluate its capabilities.

Challenge 2: How to identify the root cause of complex transformation errors?

During debugging a developer often starts from the point in the program where an error first manifests, and attempts to reason about the executed operations that led to that state. This kind of reasoning is useful during model query or transformation debugging as well, however it must include both the model query or transformation program *and* the model the it was executed on.

Such a process can be supported by automatically highlighting model elements and program statements that might contribute to the transformation error, and the developer does not have to consider the excluded (independent) elements and program points, making it easier to identify the root cause of the issue.

Program slicing [Wei82] is a widely used technique in traditional programming languages [BGG06; GL91; RH07; Tót+10; SGM02] for such root cause detection. In this dissertation, I define the slicing problem for model queries and transformations, and propose two algorithms for calculating such slices.

3 Novel Scientific Contributions

3.1 Static Type Checking of Model Queries and Transformations

There is a class of common programmer errors, such as incorrectly ordered parameters or invalid variable assignments, that can be efficiently revealed statically on model queries and transformations by executing type checking, usage counting or reachability analysis. As such issues occur commonly, revealing them before executing helps query and transformation developers providing error-free programs. I formulated the following contributions related to the detection of erroneous specification (in fulfillment of Challenge 1):

Contribution 1 *I proposed a set of lightweight static analysis techniques to identify common mistakes in model queries and transformations: type checking, variable usage counting and reachability analysis. I integrated these approaches into the development environment of the* VIATRA *framework.*

1. ***Type checking and type inference with constraint satisfaction problems.*** *I defined type checking and type inference by mapping type safety of model queries and transformations to a series of constraint satisfaction problems. The proposed mapping provides detailed localization in case of type errors [4, 15, 25, 26].*

2. ***Type constraints of model queries.*** *I defined type rules for graph patterns supporting all constraint types of the languages of the VIATRA and EMF-IncQuery frameworks [4, 14, 26].*

3. ***Type constraints of transformation programs.*** *I defined type rules for graph transformation rules and abstract state machine rules, and evaluated its performance using various traversal strategies [4, 15, 26].*

4. ***Static analysis techniques for model queries.*** *I proposed variable usage counting and reachability analysis to detect common errors in model queries [3], such as unused variables and disjunct constraints.*

My results build upon the query and transformation languages of the VIATRA and EMF-IncQuery frameworks. The VIATRA transformation language [VB07] was developed by András Balogh and Dániel Varró, while the query language of EMF-IncQuery [14] is part of Gábor Bergmann's thesis [Ber13].

3.2 Incremental Graphical Views

Queries defined for a transformation specify some aspects of the underlying model that is relevant during the debugging of transformation programs. Reusing these queries as lightweight declarations of visualizations allows fast creation of graphical views, making it possible to visualize interesting parts of the models fast. Furthermore, incremental query evaluation techniques would enable the implementation of efficient update mechanisms, visualizing the changes of the model, e.g. during a transformation.

I formulated the following thesis contributions for the observation of the structure of models during execution (fulfilling Challenge 2):

Contribution 2 *I defined an approach to derive and incrementally maintain graphical views defined declaratively by model queries. I demonstrated its feasibility using an implementation over the VIATRA framework.*

1. ***Graphical view definition using annotated queries.*** *I extended the language of EMF-IncQuery with annotations that allow the declarative definition of graphical view models. The annotations uniformly define how to display data in various (graphical, tree- or list-based) viewers [14, 16].*

2. ***Incremental graphical view maintenance based on incremental queries.*** *I proposed an approach for deriving and maintaining graphical views using incremental query evaluation techniques by mapping query results to displayed items and translating change notifications to the addition or removal of view elements. The mapping rules are automatically derived from the view definition.*

The implementation of the graphical views builds on the incremental pattern matcher techniques [Ber+10] of EMF-IncQuery, which is part of the PhD thesis of Gábor Bergmann [Ber13],

and an event-driven virtual machine where Ábel Hegedüs was the main contributor. Csaba Debreceni has generalized my approach to support the construction of incrementally maintained arbitrary logical view models using a more generic set of annotations in our joint paper [16] together with a performance assessment; my own contributions to the work are the annotation-based view model definitions, the notational metamodel and the transformation workflow. In his MSc thesis, Péter Lunk (under my supervision) implemented a prototype debugger for VIATRA exploiting the annotation-based graphical views.

3.3 Slicing Model Transformations

Program slicing [Wei82] approaches are widely used for dependency analysis in traditional programming environments [Xu+05; Tip95]; similar approaches have been proposed in the context of MDD for model slicing [LK10; SP08; Sha+10]. However, the direct application of these techniques is problematic for model transformations, as it requires *simultaneous* slicing of the models and the transformation program. I formulated the following contributions to locate root causes of complex transformation errors (to address Challenge 3):

Contribution 3 *I defined the concepts of model transformation slicing to identify the dependencies between model elements and transformation program constructs. I adapted dynamic backward slicing algorithms for model transformation slicing and demonstrated the feasibility of the approach in the context of modern model query and transformation frameworks.*

1. ***Slicing of Model Transformations.*** *I defined the problem of slicing model transformation programs to identify model elements and transformation constructs dependent on a slicing criterion. I identified the main challenges of model transformation slicing by adapting traditional concepts of program slicing [12].*
2. ***Dynamic Backward Slicing of Model Queries.*** *I defined a technique that calculates dynamic backward slices during the evaluation of model queries based on Rete networks [6].*
3. ***Dynamic Backward Slicing of Model Transformation Programs.*** *I developed an algorithm that calculates dynamic backward slices of model transformation programs based on recorded execution traces of transformations [11].*
4. ***Evaluation of the Dynamic Backward Slicing Approach.*** *I evaluated the approach using different kind of transformation programs to assess its scalability [11].*

4 Application of Results

The practical relevance of the methods and techniques outlined in the current thesis are demonstrated in this section by listing some recent applications of the scientific results.

Tools I: EMF-IncQuery Development Environment

The EMF-IncQuery open source model query framework adapts the Rete algorithm for incremental query evaluation techniques over EMF models, one of the most widely used modeling

environments today. Query evaluation serves as a basis for additional features such as automated validation, data binding, incremental graphical views and model transformation. The EMF-IncQuery framework is a major research contribution of Gábor Bergmann's PhD thesis.

The experience gained when developing the type checker for Viatra (Contribution 1) influenced both the design and the implementation of the pattern language of EMF-IncQuery: on one hand, inferrable type constraints can be omitted, resulting in shorter pattern definitions; on the other hand optional type definitions for pattern parameters can reduce the runtime cost of evaluating the type safety of pattern calls.

The development environment supports several lightweight analysis techniques, such as type checking (Contribution 1) and model visualization support (Contribution 2). Other validation strategies, such as variable counting and connectedness analysis, were implemented (in collaboration with Gábor Bergmann and Ádám Dudás, a BSc student under our joint supervision) to find other typical issues with query definitions.

Tools II: Viatra Model Transformation Framework

The Viatra model transformation framework is an open source Eclipse project featuring a complete transformation development environment based on a custom metamodeling core, including a dedicated transformation language and support for both local search-based and incremental query evaluation stategies. I have been serving as co-leader of the project for three years by the the of writing this thesis.

Contribution 1 enabled to extend the Viatra2 model transformation framework with a type checker implementation, while the results of Contribution 3 were used to extend the debugging capabilities of the framework.

The new version of Viatra3 provides a high-level model for developing both batch and incremental transformations, and relies on the EMF-IncQuery project for query evaluation, making the project capable of working with EMF-based models natively.

The modular, plug-in based architecture of EMF-IncQuery and Viatra enables easy integration with existing EMF-based modeling tools and applications, such as Papyrus UML [Papyrus], Capella [Capella], Sirius [Sirius] or Artop [Artop]. Furthermore, while we illustrate EMF-IncQuery and Viatra in the context of EMF models, core queries and transformations are regular Java programs which have been successfully adapted to other technological spaces (outside Eclipse), such as for Matlab [Hor+14; HRS15] or MPS [TV16].

The projects have already been successfully used in various industrial projects, including, but not limited to the development of an incremental code generator for executable UML models [Hor+15] and well-formedness validation of AutoSAR models in the automotive domain [Ber+10]. These projects both motivated the research presented in this dissertation, and benefited from the integrated static analysis or graphical view generation capabilities presented here.

Applications I: Incremental Graphical Views for Integrated Modular Avionics

The results of Contribution 2 are applied in Trans-IMA, a cooperative research project between Embraer (the large Brazilian airframer) and our research group. The aim of the Trans-IMA project [Hor+14] was to develop a modeling tool that allows engineers to automatically generate integrated Matlab Simulink models for analyzing the complete hardware-software architecture of the system using a high-level allocation process mapping functions to their computing units of the hardware platforms.

The proposed approach uses a set of models describing the required avionics functions (Functional Architecture Model, FAM) and the underlying hardware architecture (Platform Description

Model, PDM) as inputs that are combined in an Allocation Specification Editor requiring inputs from the developer, then an allocation process generates an Integrated Architecture Model (IAM) that stores all Simulink-specific configurations, such as library links and model references. All these models are available in a set of graphical editors with multiple viewpoints that highlight different segments of the underlying model.

These viewpoint visualizations were built using the Graphical Viewers functionality (Contribution 2), relying on the yFiles for Java [WEK04] graph visualization library, supporting (1) incremental layout algorithms scaling to thousands of nodes and edges, (2) hierarchical graphs and closable group nodes and (3) detailed fine-tuning options of layout and visualization parameters.

The declarative definition of the visualization supported quick prototyping (using the default display options provided by the graph viewers). After the set of promising graphical views were collected, the visualizations could be customized using the Java API of yFiles, such as selecting colors, layout algorithms, etc.

Applications II: Using Model Queries for Software Maintenance

In a cooperative project with Refactoring 2011 Kft. the applicability of model query and transformation tools was evaluated for software code refactoring. The project aimed to reduce the effects of software erosion by detecting coding anti-patterns and propose automatic refactorings that fix these issues, but leaving it up to the developer to decide how to handle the revealed issues. The process relied on the capabilities of the Columbus reverse engineering tool [Fer+02].

The proposed refactoring workflow consisted of four steps: (1) creating an abstract syntax graph (ASG) from the source code using Columbus, (2) searching for predefined anti-patterns in this syntax graph, (3) executing the wanted refactorings on the ASG, and finally (4) writing back the changed model into the original source code. As steps (2) and (3) could benefit from high-level specification languages used for model query and transformations they were evaluated by comparing a traditional, Java-based implementation with another relying on EMF-IncQuery and Viatra.

The benchmarking has shown that the different approaches complement each other well: a manually optimized model representation requires less memory, allowing to load larger models into the memory, while indexing-based techniques, such as the Rete algorithm in EMF-IncQuery perform well if the anti-patterns have to be re-evaulated after incremental model updates.

For the identification of the smells a selection of graph pattern were implemented in EMF-IncQuery, where the enhancements to the development environment and the graphical viewers support for debugging eased development. Furthermore, the created performance benchmark allows to measure the scalability of different pattern matching strategies on real-world data sets.

The conference paper describing the benchmarking [9] received the best paper award at the IEEE CSMR-WCRE 2014 Software Evolution Week, Antwerp, Belgium, February 3-6, 2014, and an extended version was published in the Journal of Information and Software Technology [2].

5 Publication list of Zoltán Ujhelyi

Number of publications: 26
Number of peer-reviewed publications: 20
Number of independent citations: 43

International, peer-reviewed journal papers

[1] 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". In: *Software & Systems Modeling* 15.3 (2016), pp. 609–629. DOI: 10.1007/s10270-016-0530-4

[2] **Z. Ujhelyi**, G. Szőke, Á. Horváth, N. I. Csiszár, L. Vidács, D. Varró, and R. Ferenc. "Performance comparison of query-based techniques for anti-pattern detection". In: *Information and Software Technology* 65 (2015), pp. 147–165. DOI: 10.1016/j.infsof.2015.01.003

[3] **Z. Ujhelyi**, Á. Hegedüs, G. Bergmann, Á. Horváth, I. Ráth, and D. Varró. "EMF-IncQuery: An Integrated Development Environment for Live Model Queries". In: *Science of Computer Programming* 98, Part 1 (2015). Fifth issue of Experimental Software and Toolkits (EST): A special issue on Academics Modelling with Eclipse (ACME2012), pp. 80–99. DOI: 10.1016/j.scico.2014.01.004

[4] **Z. Ujhelyi**, Á. Horváth, and D. Varró. "Static Type Checking of Model Transformation Programs". In: *ECEASST* 38 (2010). Ed. by A. Corradini, T. Margaria, J. Padberg, and G. Taentzer. URL: http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/567

[5] Á. Hegedüs, **Z. Ujhelyi**, I. Ráth, and Á. Horváth. "Visualization of Traceability Models with Domain-specific Layouting". In: *ECEASST* 32 (2010). Ed. by T. Margaria, J. Padberg, and G. Taentzer. URL: http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/510

International, peer-reviewed conferences

[6] **Z. Ujhelyi**, G. Bergmann, and D. Varró. "Rete network slicing for model queries". English. In: *Graph Transformation*. Ed. by R. Echahed and M. Minas. Vol. 9761. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 137–152. DOI: 10.1007/978-3-319-40530-8_9

[7] G. Bergmann, I. Dávid, Á. Hegedüs, Á. Horváth, I. Ráth, **Z. Ujhelyi**, and D. Varró. "Viatra 3: A Reactive Model Transformation Platform". English. In: *Theory and Practice of Model Transformations*. Ed. by D. Kolovos and M. Wimmer. Vol. 9152. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 101–110. DOI: 10.1007/978-3-319-21155-8_8

[8] M. Búr, **Z. Ujhelyi**, Á. Horváth, and D. Varró. "Local Search-Based Pattern Matching Features in EMF-IncQuery". English. In: *Graph Transformation*. Ed. by F. Parisi-Presicce and B. Westfechtel. Vol. 9151. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 275–282. DOI: 10.1007/978-3-319-21145-9_18

[9] **Z. Ujhelyi**, Á. Horváth, D. Varró, N. I. Csiszár, G. Szőke, L. Vidács, and R. Ferenc. "Anti-pattern detection with model queries: A comparison of approaches". In: *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*. Best paper award. Acceptance rate: 31%. Feb. 2014, pp. 293–302. DOI: 10.1109/CSMR-WCRE.2014.6747181

[10] G. Bergmann, Á. Hegedüs, Á. Horváth, I. Ráth, **Z. Ujhelyi**, and D. Varró. "Integrating Efficient Model Queries in State-of-the-Art EMF Tools". In: *Objects, Models, Components, Patterns*. Ed. by C. Furia and S. Nanz. Vol. 7304. Lecture Notes in Computer Science. Acceptance rate: 31%. Springer Berlin / Heidelberg, 2012, pp. 1–8. DOI: 10.1007/978-3-642-30561-0_1

[11] **Z. Ujhelyi**, Á. Horváth, and D. Varró. "Dynamic Backward Slicing of Model Transformations". In: *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. ICST '12. Acceptance rate: 27%. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1–10. DOI: 10.1109/ICST.2012.80

[12] **Z. Ujhelyi**, Á. Horváth, and D. Varró. "Towards Dynamic Backwards Slicing of Model Transformations". In: *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. 2011, pp. 404–407. DOI: 10.1109/ASE.2011.6100084

[13] G. Bergmann, Á. Hegedüs, Á. Horváth, I. Ráth, **Z. Ujhelyi**, and D. Varró. "Implementing efficient model validation in EMF tools". In: *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Nov. 2011, pp. 580–583. DOI: 10.1109/ASE.2011.6100130

[14] G. Bergmann, **Z. Ujhelyi**, I. Ráth, and D. Varró. "A Graph Query Language for EMF Models". In: *Theory and Practice of Model Transformations*. Ed. by J. Cabot and E. Visser. Vol. 6707. Lecture Notes in Computer Science. Acceptance rate: 27%. Springer Berlin / Heidelberg, 2011, pp. 167–182. DOI: 10.1007/978-3-642-21732-6_12

[15] **Z. Ujhelyi**. "Static Type Checking of Model Transformation Programs". In: *Graph Transformations*. Ed. by H. Ehrig, A. Rensink, G. Rozenberg, and A. Schürr. Vol. 6372. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, pp. 413–415. DOI: 10.1007/978-3-642-15928-2_36

International, peer-reviewed workshops

[16] C. Debreceni, Á. Horváth, Á. Hegedüs, **Z. Ujhelyi**, I. Ráth, and D. Varró. "Query-driven Incremental Synchronization of View Models". In: *Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*. VAO '14. York, United Kingdom: ACM, 2014, 31:31–31:38. DOI: 10.1145/2631675.2631677

[17] **Z. Ujhelyi**, T. Szabó, I. Ráth, and D. Varró. "Developing and Visualizing Live Model Queries". In: *Proceedings of the 1st Workshop on the Analysis of Model Transformations (AMT) @ MoDELS'12*. AMT '12. Innsbruck, Austria: ACM, 2012. DOI: 10.1145/2432497.2432505

[18] Á. Hegedüs, **Z. Ujhelyi**, and G. Bergmann. "Solving the TTC 2011 Reengineering Case with VIATRA2". In: *TTC 2011: Fifth Transformation Tool Contest, Post-Proceedings*. Ed. by P. V. Gorp, S. Mazanek, and L. Rose. Vol. 74. EPTCS. Zürich, Switzerland: Open Publishing Association, 2011, pp. 136–148. DOI: 10.4204/EPTCS.74.13

[19] Á. Hegedüs, **Z. Ujhelyi**, and G. Bergmann. "Saying Hello World with VIATRA2 - A Solution to the TTC 2011 Instructive Case". In: *TTC 2011: Fifth Transformation Tool Contest, Post-Proceedings*. Ed. by P. V. Gorp, S. Mazanek, and L. Rose. Vol. 74. EPTCS. Zürich, Switzerland: Open Publishing Association, 2011, pp. 302–324. DOI: 10.4204/EPTCS.74.25

[20] Á. Hegedüs, **Z. Ujhelyi**, G. Bergmann, and Á. Horváth. "Ecore to Genmodel case study solution using the VIATRA2 framework". In: *Transformation Tool Contest (TTC '10)*. Ed. by P. V. Gorp, S. Mazanek, and A. Rensink. Malaga, Spain, July 2010

National conferences

[21] **Z. Ujhelyi**. "Def-Use Analysis of Model Transformation Programs with Program Slicing". In: *Proceedings of the 18th PhD Minisymposium*. Budapest University of Technology, Economics, Department of Measurement, and Information Systems. Budapest, 2011, pp. 46–49

[22] **Z. Ujhelyi**. "Static Type Checking of Model Transformation Programs". In: *Conference of PhD Students in Computer Science*. 2010, pp. 413–415

[23] **Z. Ujhelyi**. "Static Analysis of Model Transformations". In: *Proceedings of the 17th PhD Minisymposium*. Budapest University of Technology, Economics, Department of Measurement, and Information Systems. 2010, pp. 26–27

[24] **Z. Ujhelyi**. "Modelltranszformációk statikus analízise". In: *Tavaszi Szél Konferenciakiadvány*. In hungarian. 2009

Technical reports and online content

[25] **Z. Ujhelyi**, Á. Horváth, and D. Varró. *A Generic Static Analysis Framework for Model Transformation Programs*. Technical Report TUB-TR-09-EE19. Budapest University of Technology and Economics, June 2009

[26] **Z. Ujhelyi**, Á. Horváth, and D. Varró. *Static Type Checking of Model Transformations by Constraint Satisfaction Programming*. Technical Report TUB-TR-09-EE20. Budapest University of Technology and Economics, June 2009

Acknowledgements

My research work was partially supported by the EU projects CONCERTO (ART-2012-333053), DIANA (AERO1-030985), MONDO (ICT-611125) and SECURECHANGE (ICT-FET-231101), the Hungarian CERTIMOT (ERC_HU-09-1-2010-0003) project, the TÁMOP (4.2.2.B-10/1–2010-0009) grant, the GOP (1.2.1-11-2011-0002) grant and a collaborative project with Embraer.

References

[Acc] The Eclipse Project. *Acceleo*. http://www.eclipse.org/acceleo.

[ALL09] M. Asztalos, L. Lengyel, and T. Levendovszky. "A Formalism for Describing Modeling Transformations for Verification". In: *Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation*. MoDeVVa '09. 2009, 2:1–2:10. DOI: 10.1145/1656485.1656487.

[Artop] *Artop: The AUTOSAR Tool Platform*. https://www.artop.org/.

[Aut+16] M. Autili, A. Bertolino, G. D. Angelis, D. D. Ruscio, and A. D. Sandro. "A Tool-Supported Methodology for Validation and Refinement of Early-Stage Domain Models". In: *IEEE Transactions on Software Engineering* 42.1 (2016), pp. 2–25. DOI: 10.1109/TSE.2015.2449319.

[AUT12] AUTOSAR Consortium. *The AUTOSAR Standard*. http://www.autosar.org/. 2012.

[AW13] L. Ab. Rahim and J. Whittle. "A survey of approaches for verifying model transformations". In: *Software & Systems Modeling* (2013). DOI: 10.1007/s10270-013-0358-0.

[Bar+08] P. Barbosa, F. Ramalho, J. de Figueiredo, and A. dos S.Junior. "An Extended MDA Architecture for Ensuring Semantics-Preserving Transformations". In: *32nd Annual IEEE Software Engineering Workshop, 2008. SEW '08*. 32nd Annual IEEE Software Engineering Workshop, 2008. SEW '08. 2008, pp. 33–42. DOI: 10.1109/SEW.2008.8.

[Bar+15] L. Baresi, G. Blohm, D. S. Kolovos, N. Matragkas, A. Motta, R. F. Paige, A. Radjenovic, and M. Rossi. "Formal verification and validation of embedded systems: the UML-based MADES approach". In: *Software & Systems Modeling* 14.1 (2015), pp. 343–363. DOI: 10.1007/s10270-013-0330-z.

[Bau+06] B. Baudry, T. Dinh-Trong, J.-M. Mottu, D. Simmonds, R. France, S. Ghosh, F. Fleurey, and Y. Le Traon. "Model Transformation Testing Challenges". English. In: *ECMDA workshop on Integration of Model Driven Development and Model Driven Testing*. 2006.

[Bau+10] B. Baudry, S. Ghosh, F. Fleurey, R. France, Y. Le Traon, and J.-M. Mottu. "Barriers to Systematic Model Transformation Testing". In: *Commununications of the ACM* 53.6 (2010), pp. 139–143. DOI: 10.1145/1743546.1743583.

[BCK08] P. Baldan, A. Corradini, and B. König. "A framework for the verification of infinite-state graph transformation systems". In: *Information and Computation* 206.7 (2008), pp. 869–907. DOI: 10.1016/j.ic.2008.04.002.

[Ber+10] G. Bergmann, Á. Horváth, I. Ráth, D. Varró, A. Balogh, Z. Balogh, and A. Ökrös. "Incremental Evaluation of Model Queries over EMF Models". In: *Model Driven Engineering Languages and Systems*. Vol. 6394. Lecture Notes in Computer Science. 2010, pp. 76–90. DOI: 10.1007/978-3-642-16145-2_6.

[Ber13] G. Bergmann. "Incremental Model Queries in Model-Driven Design". PhD dissertation. Budapest University of Technology and Economics, 2013.

[Béz05] J. Bézivin. "On the unification power of models". In: *Software & Systems Modeling* 4.2 (2005), pp. 171–188. DOI: 10.1007/s10270-005-0079-0.

[BG05] S. Beydeda and V. Gruhn. *Model-Driven Software Development*. Springer-Verlag New York, Inc., 2005.

[BGG06] A. Beszedes, T. Gergely, and T. Gyimothy. "Graph-Less Dynamic Dependence-Based Dynamic Slicing Algorithms". In: *2006 Sixth IEEE International Workshop on Source Code Analysis and Manipulation*. 2006, pp. 21–30. DOI: 10.1109/SCAM.2006.17.

[BKZ14] H. J. S. Bruggink, B. König, and H. Zantema. "Termination Analysis for Graph Transformation Systems". In: *Theoretical Computer Science: 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*. Ed. by J. Diaz, I. Lanese, and D. Sangiorgi. Springer Berlin Heidelberg, 2014, pp. 179–194. DOI: 10.1007/978-3-662-44602-7_15.

[Bon+99] A. Bondavalli, M. Dal Cin, D. Latella, and A. Pataricza. "High-level Integrated Design Environment for Dependability". WORDS'99, 1999 Workshop on Real-Time Dependable System. 1999.

[Bro+14] M. Broy, S. Kirstan, H. Krcmar, and B. Schätz. "What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry?" In: *Software Design and Development: Concepts, Methodologies, Tools, and Applications*. 2014, pp. 310–334. DOI: 10.4018/978-1-4666-4301-7.ch017.

[Bul08] I. Bull. "Model Driven Visualization: Towards A Model Driven Engineering Approach For Information Visualization". Ph.D. Thesis. University of Victoria, BC, Canada, 2008.

[Cab+10] J. Cabot, R. Clarisó, E. Guerra, and J. de Lara. "Verification and validation of declarative model-to-model transformations through invariants". In: *Journal of Systems and Software*. Computer Software and Applications 83.2 (2010), pp. 283–302. DOI: 10.1016/j.jss.2009.08.012.

[Cal+11] D. Calegari, C. Luna, N. Szasz, and Á. Tasistro. "A Type-Theoretic Framework for Certified Model Transformations". In: *Formal Methods: Foundations and Applications*. Ed. by J. Davies, L. Silva, and A. Simao. Lecture Notes in Computer Science 6527. Springer Berlin Heidelberg, 2011, pp. 112–127.

[Capella] PolarSys. *Capella*. http://www.polarsys.org/capella.

[Cla08] E. M. Clarke. "The Birth of Model Checking". In: *25 Years of Model Checking*. Ed. by O. Grumberg and H. Veith. Lecture Notes in Computer Science 5000. Springer Berlin Heidelberg, 2008, pp. 1–26.

[Cor14] J. Corley. "Exploring Omniscient Debugging for Model Transformations". In: *Joint Proceedings of MODELS 2014 Poster Session and the ACM Student Research Competition (SRC) co-located with the 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 28 - October 3, 2014*. Vol. 1258. CEUR Workshop Proceedings. 2014, pp. 63–68. URL: http://ceur-ws.org/Vol-1258/src3.pdf.

[Dho+10] P. Dhoolia, S. Mani, V. S. Sinha, and S. Sinha. "Debugging model-transformation failures using dynamic tainting". In: *Proceedings of the 24th European conference on Object-oriented programming*. ECOOP'10. ACM ID: 1883983. 2010, pp. 26–51.

[Epsilon] The Eclipse Project. *Epsilon*. https://www.eclipse.org/epsilon/.

[ER10] M. Egea and V. Rusu. "Formal executable semantics for conformance in the MDE framework". In: *Innovations in Systems and Software Engineering* 6.1-2 (2010), pp. 73–81. DOI: 10.1007/s11334-009-0108-1.

[Fer+02] R. Ferenc, Á. Beszédes, M. Tarkiainen, and T. Gyimóthy. "Columbus – Reverse Engineering Tool and Schema for C++". In: *Software Maintenance, 2002. Proceedings. International Conference on*. 2002, pp. 172–181. DOI: 10.1109/ICSM.2002.1167764.

[Fin] FindBugs. *FindBugs – Find Bugs in Java Programs*. http://findbugs.sourceforge.net/. (Visited on 05/16/2013).

[FSB04] F. Fleurey, J. Steel, and B. Baudry. "Validation in model-driven engineering: testing model transformations". In: *2004 First International Workshop on Model, Design and Validation, 2004. Proceedings*. 2004 First International Workshop on Model, Design and Validation, 2004. Proceedings. 2004, pp. 29–40. DOI: 10.1109/MODEVA.2004.1425846.

[Gei+06] R. Geiß, G. Batz, D. Grund, S. Hack, and A. Szalkowski. "GrGen: A Fast SPO-Based Graph Rewriting Tool". English. In: *Graph Transformations*. Vol. 4178. Lecture Notes in Computer Science. 2006, pp. 383–397. DOI: 10.1007/11841883_27.

[GL91] K. Gallagher and J. Lyle. "Using Program Slicing in Software Maintenance". In: *IEEE Transactions on Software Engineering* 17.8 (1991), pp. 751–761. DOI: 10.1109/32.83912.

[God+07] P. Godinho, B. Meiguins, A. Goncalves Meiguins, R. Casseb do Carmo, M. de Brito Garcia, L. Almeida, and R. Lourenco. "PRISMA - A Multidimensional Information Visualization Tool Using Multiple Coordinated Views". In: *Information Visualization, 2007. IV '07. 11th International Conference*. 2007, pp. 23–32. DOI: 10.1109/IV.2007.90.

[Her+11] F. Hermann, H. Ehrig, F. Orejas, K. Czarnecki, Z. Diskin, and Y. Xiong. "Correctness of Model Synchronization Based on Triple Graph Grammars". In: *Model Driven Engineering Languages and Systems*. Ed. by J. Whittle, T. Clark, and T. Kühne. Lecture Notes in Computer Science 6981. Springer Berlin Heidelberg, 2011, pp. 668–682.

[HLR07] M. Hibberd, M. Lawley, and K. Raymond. "Forensic Debugging of Model Transformations". In: *Model Driven Engineering Languages and Systems*. Ed. by G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil. Lecture Notes in Computer Science 4735. Springer Berlin Heidelberg, 2007, pp. 589–604.

[Hor+14] Á. Horváth, Á. Hegedüs, M. Búr, V. Varró, R. R. Starr, and S. Mirachi. "Hardware-software allocation specification of IMA systems for early simulation". In: *2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)*. 2014, pp. 4D3-1–4D3-15. DOI: 10.1109/DASC.2014.6979474.

[Hor+15] Á. Horváth, I. Ráth, Á. Hegedüs, and Á. Balogh. "Decreasing your Coffee Consumption with Incremental Code Regeneration". In: *EclipseCon France*. https://www.eclipsecon.org/france2015/session/decreasing-your-coffee-consumption-incremental-code-regeneration. 2015.

[HRS15] Á. Horváth, I. Ráth, and R. R. Starr. "Massif – the love child of Matlab Simulink and Eclipse". In: *EclipseCon North America*. https://www.eclipsecon.org/na2015/session/massif-love-child-matlab-simulink-and-eclipse. 2015.

[HWR14] J. Hutchinson, J. Whittle, and M. Rouncefield. "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure". In: *Science of Computer Programming* 89, Part B (2014). Special issue on Success Stories in Model Driven Engineering, pp. 144–161. DOI: 10.1016/j.scico.2013.03.017.

[KN07] G. Karsai and A. Narayanan. “On the Correctness of Model Transformations in the Development of Embedded Systems”. In: *Composition of Embedded Systems. Scientific and Industrial Issues*. Ed. by F. Kordon and O. Sokolsky. Lecture Notes in Computer Science 4888. Springer Berlin Heidelberg, 2007, pp. 1–18.

[Küs06] J. M. Küster. “Definition and validation of model transformations”. In: *Software & Systems Modeling* 5.3 (2006), pp. 233–259. DOI: 10.1007/s10270-006-0018-8.

[LBA10] L. Lúcio, B. Barroca, and V. Amaral. “A Technique for Automatic Validation of Model Transformations”. In: *Model Driven Engineering Languages and Systems*. Ed. by D. C. Petriu, N. Rouquette, and Ø. Haugen. Lecture Notes in Computer Science 6394. Springer Berlin Heidelberg, 2010, pp. 136–150. DOI: 10.1007/978-3-642-16145-2_10.

[LK10] K. Lano and S. Kolahdouz-Rahimi. “Slicing of UML Models Using Model Transformations”. In: *Model Driven Engineering Languages and Systems*. Vol. 6395. Lecture Notes in Computer Science. 2010, pp. 228–242. DOI: 10.1007/978-3-642-16129-2_17.

[MBL08] J.-M. Mottu, B. Baudry, and Y. Le Traon. “Model transformation testing: oracle issue”. In: *IEEE International Conference on Software Testing Verification and Validation Workshop, 2008. ICSTW ’08*. 2008, pp. 105–112. DOI: 10.1109/ICSTW.2008.27.

[Mur89] T. Murata. “Petri nets: Properties, analysis and applications”. In: *Proceedings of the IEEE* 77.4 (1989), pp. 541–580. DOI: 10.1109/5.24143.

[MV11] R. Mannadiar and H. Vangheluwe. “Debugging in Domain-Specific Modelling”. In: *Software Language Engineering*. Ed. by B. Malloy, S. Staab, and M. v. d. Brand. Lecture Notes in Computer Science 6563. Springer Berlin Heidelberg, 2011, pp. 276–285.

[OAS07] OASIS. *Web Services Business Process Execution Language Version 2.0 (OASIS Standard)*. "http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html". 2007.

[Obj10] Object Management Group. *OMG System Modeling Language (SysML)*. http://www.omg.org/spec/SysML/index.htm. 2010.

[Obj11a] Object Management Group. *Business Process Model And Notation (BPMN)*. http://www.omg.org/spec/BPMN/index.htm. 2011.

[Obj11b] Object Management Group. *UML Profile For MARTE: Modeling And Analysis Of Real-Time Embedded Systems*. http://www.omg.org/spec/MARTE/index.htm. 2011.

[Obj11c] Object Management Group. *Unified Modeling Language (UML)*. http://www.omg.org/spec/UML/index.htm. 2011.

[OW09] F. Orejas and M. Wirsing. “On the Specification and Verification of Model Transformations”. In: *Semantics and Algebraic Specification*. Ed. by J. Palsberg. Lecture Notes in Computer Science 5700. Springer Berlin Heidelberg, 2009, pp. 140–161.

[Papyrus] The Eclipse Project. *Papyrus*. http://www.eclipse.org/papyrus.

[PMD14] PMD. *PMD checker*. http://pmd.sourceforge.net/. 2014. (Visited on 2014).

[Pri08] P. Prisaznuk. “ARINC 653 role in Integrated Modular Avionics (IMA)”. In: *Digital Avionics Systems Conference, 2008. IEEE/AIAA 27th*. 2008. DOI: 10.1109/DASC.2008.4702770.

[RD06] A. Rensink and D. Distefano. "Abstract Graph Transformation". In: *Electronic Notes in Theoretical Computer Science* 157.1 (2006), pp. 39–59. DOI: 10.1016/j.entcs.2006.01.022.

[Ren04] A. Rensink. "The GROOVE Simulator: A Tool for State Space Generation". English. In: *Applications of Graph Transformations with Industrial Relevance*. Vol. 3062. Lecture Notes in Computer Science. 2004, pp. 479–485. DOI: 10.1007/978-3-540-25959-6_40.

[RH07] V. P. Ranganath and J. Hatcliff. "Slicing concurrent Java programs using Indus and Kaveri". In: *International Journal on Software Tools for Technology Transfer* 9.5 (2007), pp. 489–504. DOI: 10.1007/s10009-007-0043-0.

[Ros+08] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. C. Polack. "The Epsilon Generation Language". In: *Model Driven Architecture – Foundations and Applications: 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings*. Ed. by I. Schieferdecker and A. Hartman. Springer Berlin Heidelberg, 2008, pp. 1–16. DOI: 10.1007/978-3-540-69100-6_1.

[Rus08] J. Rushby. "Automated Test Generation and Verified Software". In: *Verified Software: Theories, Tools, Experiments*. 2008, pp. 161–172. DOI: 10.1007/978-3-540-69149-5_18.

[SAE09] SAE International. *Architecture Analysis & Design Language (AADL)*. http://standards.sae.org/as5506a/. 2009.

[Sch+03] J. Schumann, B. Fischer, M. Whalen, and J. Whittle. "Certification support for automatically generated programs". In: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003*. 2003. DOI: 10.1109/HICSS.2003.1174914.

[SGM02] G. Szilágyi, T. Gyimóthy, and J. Małuszyński. "Static and Dynamic Slicing of Constraint Logic Programs". In: *Automated Software Engineering* 9 (1 2002), pp. 41–65.

[Sha+10] A. Shaikh, R. Clarisó, U. K. Wiil, and N. Memon. "Verification-driven slicing of UML/OCL models". In: *25th IEEE/ACM Int. Conf. on Automated Software Engineering*. 2010, pp. 185–194.

[Sirius] The Eclipse Project. *Sirius*. http://www.eclipse.org/sirius.

[SP08] I. Schaefer and A. Poetzsch-Heffter. "Slicing for model reduction in adaptive embedded systems development". In: *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems*. SEAMS '08. 2008, 25–32. DOI: 10.1145/1370018.1370024.

[STJ83] R. L. Sedlmeyer, W. B. Thompson, and P. E. Johnson. "Knowledge-based fault localization in debugging". In: *Journal of Systems and Software* 3.4 (1983), pp. 301–307. DOI: 10.1016/0164-1212(83)90016-X.

[Stu+07] I. Stuermer, M. Conrad, H. Doerr, and P. Pepper. "Systematic Testing of Model-Based Code Generators". In: *IEEE Transactions on Software Engineering* 33.9 (2007), pp. 622–634.

[TG15] G. Taentzer and U. Golas. "Towards Local Confluence Analysis for Amalgamated Graph Transformation". In: *Graph Transformation: 8th International Conference, ICGT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 21-23, 2015. Proceedings*. Ed. by F. Parisi-Presicce and B. Westfechtel. Springer International Publishing, 2015, pp. 69–86. DOI: 10.1007/978-3-319-21145-9_5.

[Tip95] F. Tip. "A survey of program slicing techniques". In: *Journal of Programming Languages 3(3)* (1995), pp. 121–189.

[Tót+10] G. Tóth, P. Hegedűs, Á. Beszédes, T. Gyimóthy, and J. Jász. "Comparison of Different Impact Analysis Methods and Programmer's Opinion: An Empirical Study". In: *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java*. PPPJ '10. 2010, pp. 109–118. DOI: 10.1145/1852761.1852777.

[TV16] S. E. Tamás Szabó and M. Voelter. "IncA: A DSL for the Definition of Incremental Program Analyses". In: *Automated Software Engineering, 31st IEEE/ACM International Conference on*. Accepted. 2016.

[Van+98] J. Van Baalen, P. Robinson, M. Lowry, and T. Pressburger. "Explaining synthesized software". In: *13th IEEE International Conference on Automated Software Engineering, 1998. Proceedings*. 13th IEEE International Conference on Automated Software Engineering, 1998. Proceedings. 1998, pp. 240–248. DOI: 10.1109/ASE.1998.732661.

[Var+06] D. Varró, S. Varró–Gyapay, H. Ehrig, U. Prange, and G. Taentzer. "Termination Analysis of Model Transformations by Petri Nets". In: *Graph Transformations*. Ed. by A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg. Lecture Notes in Computer Science 4178. Springer Berlin Heidelberg, 2006, pp. 260–274. DOI: 10.1007/11841883_19.

[Var04] D. Varró. "Automated formal verification of visual modeling languages by model checking". English. In: *Software and Systems Modeling* 3.2 (2004), pp. 85–113. DOI: 10.1007/s10270-003-0050-x.

[VB07] D. Varró and A. Balogh. "The Model Transformation Language of the VIATRA2 Framework". In: *Science of Computer Programming* 68.3 (2007), pp. 214–234.

[Wag+11] D. Wagelaar, M. Tisi, J. Cabot, and F. Jouault. "Model Driven Engineering Languages and Systems: 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings". In: ed. by J. Whittle, T. Clark, and T. Kühne. Springer Berlin Heidelberg, 2011. Chap. Towards a General Composition Semantics for Rule-Based Model Transformation, pp. 623–637. DOI: 10.1007/978-3-642-24485-8_46.

[Wei82] M. Weiser. "Programmers use slices when debugging". In: *Communications of the ACM* 25.7 (1982), pp. 446–452. DOI: 10.1145/358557.358577.

[WEK04] R. Wiese, M. Eiglsperger, and M. Kaufmann. "yFiles — Visualization and Automatic Layout of Graphs". In: *Graph Drawing Software*. Ed. by M. Jünger and P. Mutzel. Mathematics and Visualization. Springer Berlin Heidelberg, 2004, pp. 173–191. DOI: 10.1007/978-3-642-18638-7_8.

[WHR14] J. Whittle, J. Hutchinson, and M. Rouncefield. "The State of Practice in Model-Driven Engineering". In: *IEEE Software* 31.3 (2014), pp. 79–85. DOI: 10.1109/MS.2013.65.

[Wim+09] M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger. "Right or Wrong? Verification of Model Transformations using Colored Petri Nets". In: *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09)*. 2009.

[WKC08] J. Wang, S.-K. Kim, and D. Carrington. "Automatic Generation of Test Models for Model Transformations". In: *19th Australian Conference on Software Engineering, 2008. ASWEC 2008*. 19th Australian Conference on Software Engineering, 2008. ASWEC 2008. 2008, pp. 432–440. DOI: 10.1109/ASWEC.2008.4483232.

[Xtend] The Eclipse Project. *Xtend - Modernized Java*. `http://www.eclipse.org/xtend`.

[Xu+05] B. Xu, J. Qian, X. Zhang, Z. Wu, and L. Chen. "A brief survey of program slicing". In: *ACM SIGSOFT Software Engineering Notes* 30.2 (2005), pp. 1–36.