ADVANCED TECHNIQUES FOR THE IMPLEMENTATION OF MODEL TRANSFORMATION SYSTEMS

PhD thesis booklet

GERGELY VARRÓ
MSc in technical informatics

SUPERVISORS:
DR. KATALIN FRIEDL, PhD
ASSOCIATE PROFESSOR

DR. DÁNIEL VARRÓ, PhD
ASSISTANT PROFESSOR

PROF. DR. RER. NAT. ANDY SCHÜRR
PROFESSOR

BUDAPEST, APRIL 2008
1 Preliminaries and Objectives of the Research

1.1 Model-Driven Engineering

Model-Driven Development (MDD) has recently become a leading trend in software engineering. The aim of MDD is to carry out a thorough system modeling before implementation. Key ideas of MDD are to create models of the software on various abstraction levels and from various viewpoints and to support automatic code generation from these models. The main advantages of the MDD concept are the reuse of high abstraction level models and an increase in productivity by high degree of automation.

A well-known realization of the MDD principle is the Model-Driven Architecture (MDA) [KWB03, Obj01c] initiative of the Object Management Group (OMG). The aim of MDA is to separate software or business functionality from platform details, which is achieved as follows.

The conceptual design of functionalities of the software or business system is captured in the form of a platform independent model (PIM), which constitutes a reusable model represented on a high abstraction level. PIMs can survive changes in realization technologies and software architectures. A platform-specific model (PSM) is also a model of the system under design, but in addition, it is linked to an underlying technological platform such as a specific execution platform, software architecture, operating system or database. Frequently, a PSM is produced from the PIM by an automated model transformation [Béz05], which is followed by a code generation phase that automatically prepares the implementation for the system from the PSM.

MDA tools [Leo07, TA05] should provide support for (i) creating and editing models, (ii) checking completeness and consistency, (iii) calculating metrics, (iv) transforming models to other models or program code, (v) composing several source models, (vi) model-based testing, (vii) simulating the execution of the systems represented by models, and (viii) re-engineering by transforming legacy systems to well-formed models.

In order to carry out all these tasks, a large number of modeling languages and tools are used in a typical model-driven development process. Powerful domain-specific modeling environments (such as the GMF in Eclipse, or the DSM framework of Microsoft Visual Studio) frequently provide rich support for developing editors, and code generators. However, the design and execution of model transformations are usually not supported properly in these industrial tools. This thesis primarily focuses on to provide advanced support for executing complex model transformations within and between these modeling languages.

1.2 Model transformation

Model transformation plays a key role in the overall process of MDA. The aim of model transformation is to carry out automated translations within and between modeling languages. The MDD approach requires these transformations to be (i) captured by a high-level, specification language, (ii) automatically executed by efficient algorithms and techniques, and (iii) extensively supported by industrial quality tools.

The first requirement is fulfilled by Query/Views/Transformations (QVT) [QVT03], which is an OMG standard for specifying multi-directional model transformations. Since QVT is a recent standard with a short history, only initial prototypes (e.g., mediniQVT [KE07], MTF [Alp04]) have been developed, and efficient implementations scaling up to complex model transformation are still missing.

While the model transformation community lacks QVT-based tools, a large variety of tools using different concepts and techniques (see the excellent surveys of [CH03, MG05, MGVK05]) have already been implemented. As demonstrated by [CH03, EGdL05, 1, 4, 5], the declarative
and rule-based approach of graph transformation is one popular technique for capturing model transformations.

1.3 Graph transformation as a model transformation approach

Graph transformation (GT) [EEKR99, Roz97] provides a declarative language for specifying the manipulation of graph models by means of GT rules, which consist of a left-hand side (LHS), a right-hand side (RHS) graph, and negative application conditions (NAC) [HHT96] attached to the LHS. Model manipulation is carried out by replacing a matching of the LHS in the model by an image of the RHS. This is performed in two phases. In the performance critical pattern matching phase, matchings of the LHS are sought in the model and the absence of the graph structures of NACs is checked and ensured. In the updating phase, the selected matching parts are modified based on the difference of LHS and RHS.

From a theoretical viewpoint, the time complexity of graph pattern matching is bound by a polynomial whose exponent linearly depends on the size of the LHS. As a consequence, no fast algorithms can be guaranteed to exist for the general version of graph pattern matching, as it leads to the subgraph isomorphism problem, for which NP-completeness has been proved [Ata99]. However, in practice, the rules are typically fixed in advance resulting in a bound (but not necessarily small) exponent, and GT tools use different kinds of heuristic based strategies for accelerating the pattern matching process. The most popular algorithms are based (i) either on constraint satisfaction, which interpret LHS as a set of constraints to be satisfied by the matchings in the model, (ii) or on local searches, which start from matching a single node and extend the matching step-by-step by mapping neighboring nodes in a recursive or iterative process. Heuristics in this latter technique are frequently driven by search plans [Zün96], which define the processing order of nodes in the LHS.

After examining several GT tools (AGG [Tae00, ERT99], AToM3 [dLV02], BOTL [MB03], DiaGen [Min02], FUJABA [KNNZ00], GREAT [AKS03], Groove [Ren04a], PROGRES [SWZ99], Tiger [BEK+06], V1ATRA2 [BV06], VMTS [LLMC06]) with essentially different pattern matching strategies and heterogeneous execution environments, I have discovered the following problems, which have become the challenges for my research.

- **Lack of objective measurements for memory and time.** Only estimates existed about the memory usage and the run-time performance of model transformation tools, and their exact characteristics have never been objectively assessed due to lack of measurements.

- **Insensitivity of algorithms to models under transformation.** Optimization of pattern matching algorithms only exploited restrictions imposed by the problem domain (i.e., metamodels), but additional information in the model under transformation was not used. Furthermore, pattern matching algorithms were inflexible and insensitive to the transformation flow, although significant changes might be experienced in the structure and size of instance models, while the transformation progresses from the beginning to the end.

- **Missing support for incremental transformations.** All the examined tools performed batch transformations by exploiting only the matchings calculated for the rule execution step in turn, but the possibility to preserve matchings between consecutive rule applications have not been examined and analyzed.

- **Integration of transformations to existing tools.** Although a large variety of highly sophisticated standalone graph transformation solutions were available, their integration into existing MDA tools was inhibited by the missing support for transformation APIs.
2 Research Method

These observations basically determined the main line of my research and the individual challenges to be solved.

**Designing a benchmarking framework for graph transformation.** In the first phase of my research, I found that only estimates existed about the run-time performance of graph transformation tools.

For this reason, I surveyed the literature on the existing methods for the comparison and categorization of graph transformation tools. Additionally, I examined how the quantitative performance assessment technique of benchmarking has been used in other fields of software engineering by following the guidelines of [SEH03].

During this analysis, I found that only qualitative specification examples existed for graph transformation approaches, but their goal was to demonstrate the expressiveness of the given approach. I also recognized that the performance aspects of graph transformation tools have only been assessed by using estimates based on best-practice engineering considerations, but these estimates have never been confirmed by repeatable measurements performed under well-defined circumstances. At the same time, an intensifying demand could be perceived in the graph transformation community for establishing benchmarks for assessing strengths and weaknesses of different heuristics and tool implementations, which could later help in laying down the directions of further developments.

Therefore, in a preparatory step, I collected typical transformation scenarios from the literature. For each such scenario, I searched for a corresponding implementation, which can later be used as a benchmark example. In parallel with the collection of scenarios, I examined some existing tools including their pattern matching algorithms. During this analysis, I focused on such acceleration techniques and heuristics that are used by several tools.

In Thesis 1, I defined the benchmarking framework by selecting such graph transformation tools that support a wide range of features and by defining the configurations of measurement settings. Then, I measured the execution times of transformations in several rounds.

**Building a graph transformation engine on top of a relational database.** If the paradigm of model driven software engineering is aimed to be used in industrial projects, then model transformations should operate on huge system models as well, and they should be easily integrated into existing tools.

For this reason, in the beginning of my research, I surveyed the algorithms and techniques used in graph transformation tools of that time. Additionally, based on the related literature [Béz01], I collected and analyzed the requirements for model transformation prescribed by tools supporting model-driven development.

Based on this review, I found that all the tools performed in-memory translations, and no analysis investigated the transformation of huge system models, which were unable to fit the main memory. Moreover, I noticed that several large distributed applications already existed in the software industry, in which models are persisted in a relational database.

To overcome the problem of exceeding main memory limits, I proposed to perform the transformation on models stored on disks, which enables the handling of larger models for the price of slower runtime operations. This idea can be realized by building graph transformation on top of a relational database management system (RDBMS) as presented in Thesis 2. This solution also supports the integration of model transformation functionality into such legacy systems, which store models in RDBMSs.
Integration of graph transformation modules into existing enterprise applications. I also investigated some MDA and model transformation tools of that time by focusing on their integration capabilities. Thus, I assessed whether the concept of modularity appeared among their design and development principles, and I checked whether they support standard APIs in any sense.

As a result of this investigation, I found that MDA tools can typically access models via standard interfaces like the Eclipse Modeling Framework [EMF] or the Java Metadata Interface [JMI], while enterprise applications use Enterprise Java Beans [EJB06] for the same purpose. On the other hand, the corresponding transformation modules rarely have a modular structure, and they cannot be accessed in a standardized way.

Therefore, in parts of Theses 2 and 3, I demonstrated how new model transformation approaches can be used in enterprise applications by providing a modular implementation in the form of Enterprise Java Beans on the Java 2 Enterprise Edition [Sun] platform.

Designing model and transformation flow sensitive graph pattern matching algorithms. While designing the RDBMS based graph transformation approach, I noticed that the execution plans of SQL queries, which drive the graph pattern matching process in the underlying database, frequently change as the transformation progresses depending on the actual database content.

By reviewing the source codes of available graph transformation tools and the related publications, I examined what kind of support these tools provide for the run-time modification of heuristics that drive the graph pattern matching algorithm.

As a result of this review, I found that the heuristics used by the implemented algorithms are fixed during the complete execution of transformations, and they neglect the size and structure of the model under transformation.

Therefore, in Thesis 3, I introduced the concept of model-sensitivity for search plan driven heuristics, which estimate their expected performance by using statistical data collected from instance models. Additionally, to overcome the problem of hard-wired heuristics, I proposed a new adaptive approach for graph pattern matching, where the optimal strategy can be selected from previously generated heuristics by using the model based estimations.

Elaborating incremental graph transformation. In computer engineering, caches are widely used for high-speed data retrieval in such cases, when data access is more frequent than modification. Since this might also characterize graph transformation in specific cases, the growing tendency in the capacity of RAMs opens up the possibility for applying such techniques, which can accelerate graph pattern matching for the price of increased memory consumption and slower updates.

For this reason, I investigated the pattern matching algorithms of existing tools to check whether they support cache based solutions to speed-up the transformation process. Additionally, I explored the incremental techniques that had been applied in other fields of computer engineering.

Based on this analysis, I found that the implementation of the investigated graph transformation engines followed the same principle. First a matching occurrence of the left-hand side of the graph transformation rule was searched by graph pattern matching. Then the engine performed some local modification to add or remove graph elements to the model, and the entire process started all over again. As the information on a previous matching was lost when a new transformation step was initiated, the complex and expensive graph pattern matching phase was restarted from scratch each time. I recognized that this behaviour could be a performance bottleneck (as demonstrated later by [KS06]).

Therefore, in Thesis 4, we proposed foundational data structures and algorithms for incremental graph pattern matching where all complete and non-extensible partial matchings are stored...
explicitly. The set of these matchings is then updated incrementally when triggered by the modifi-
cations of the instance model. To support this process, we additionally introduced a notification
mechanism for quickly identifying those partial matchings, which are candidates for extension or
removal when the model is modified.

3 New Scientific Results

3.1 Benchmarking Framework for Graph Transformation

By surveying the literature [CH03, SEH03, MG05] on the existing methods for the comparison
and categorization of graph transformation tools, and on benchmarking in software engineering re-
lated fields, I realized that only qualitative specification examples existed for graph transformation
approaches, but their performance aspects had never been assessed quantitatively.

Thesis 1 I proposed a benchmarking framework, which enables quantitative performance analy-

sis and comparison of heuristics in graph transformation tools.

- **Definition and categorization of the features of graph transformation problems.** By ana-
lyzing typical scenarios described by graph transformation, I defined and categorized such
characteristics of graph transformation problems that have high influence on tool perfor-
manceed, such as pattern size, the maximum degree of nodes, the number of matchings and the
length of transformation sequences (Sec. 5.2.2).

- **Identification and categorization of the optimization strategies in graph transformation
tools.** By analyzing the most popular graph transformation tools, I identified and cate-
gorized their typical optimization strategies such as parallel rule execution, 'as long as
possible' rule application, multiplicity based optimization, and parameter passing, which
have significant impact on execution time (Sec. 5.2.3).

- **Specification of benchmark examples.** I adapted standard benchmarking terminology for
graph transformation, and I prepared benchmark example specifications for typical model
interpretation and model transformation scenarios (Sections 5.3 and 5.4).

- **Quantitative comparison of the speed-up effects of optimization strategies in graph trans-
formation tools.** Based on the benchmarking framework, I carried out measurements on
several tools using different parameter settings and combinations of optimization strate-
gies. I used the measurement results for assessing the acceleration effects of optimization
strategies (Sec. 5.5).

This thesis is based upon Chapter 5 of the PhD dissertation, and it was published in [11, 18,
19, 20].

3.2 Graph Transformation in Relational Databases

A natural requirement for an MDA tool is to support transformations on huge system models as
well. By surveying the features of several advanced tools [AKS03a, KNNZ00, ERT99, SWZ99,
LLMC06] available at the beginning of my research, I experienced that all of them performed
in-memory translations, which might fail due to exceeding main memory limits in case of huge
models. Therefore, it seemed to be practical to perform the transformation on models stored on
disks, which enables the handling of larger models for the price of slower runtime operations.
Thesis 2 I elaborated a provenly correct method for executing graph transformation built on top of a relational database, and assessed the performance of the approach in several measurement settings involving different combinations of databases, parameters and optimization strategies.

- **Graph pattern matching in relational databases.** I elaborated a provenly correct method, which automatically transforms graph patterns to SQL queries whose behaviour corresponds to the pattern matching phase of graph transformation (Sections 6.4.2 and 6.4.3).

- **Modification phase of graph transformation on top of relational databases.** I elaborated a provenly correct method, which generates such SQL commands, whose behaviour corresponds to the updating phase of graph transformation (Sec. 6.4.4).

- **Quantitative performance analysis of the method.** By using the object-relational mapping as a benchmark example, I examined the efficiency of this technique in several measurement settings involving different combinations of databases, parameters and optimization strategies (Sec. 6.5).

- **Portable queries for graph pattern matching.** I proposed a database independent and portable pattern matching approach that uses declarative EJB QL queries to implement graph pattern matching (Sec. 6.6).

This thesis is based upon Chapter 6, and it was published in [2, 7, 9, 13].

3.3 Adaptive Graph Transformation

By reviewing the pattern matching algorithms of graph transformation tools [AKS03a, KNNZ00, ERT99, SWZ99, LLMC06] available at the beginning of my research, I found that the heuristics used by these algorithms are fixed during the complete execution, and they neglect the size and structure of the model under transformation. Therefore, it seemed to be practical to introduce the concept of model-specific search plans which use statistical data collected from typical instance models specified by transformation designers.

Thesis 3 I elaborated an adaptive method for executing model-specific search plans in order to improve the performance of graph transformation in its pattern matching phase.

- **Costs and optimization mechanisms for model-specific search plans.** I defined a cost function for model-specific search plans which estimates the size of the search space that would be traversed during search plan execution. In order to find a low cost search plan according to the special cost function defined for model-specific search plans, I elaborated an optimization technique by customizing traditional greedy algorithms (Sec. 7.3).

- **Adaptive graph transformation engine.** I elaborated an adaptive graph transformation engine which is able to select the optimal pattern matching strategy at execution time from the set of precompiled strategies by exploiting run-time model statistics (Sections 7.4.1 and 7.5.1).

- **EJB3-based prototype engine.** I prepared an EJB3-based prototype of the adaptive graph transformation engine for the Java 2 Enterprise Edition (J2EE) platform by generating code for pattern matching and cost calculation functionalities of concrete strategies, and by implementing a stateless session bean that selects the optimal strategy at run-time (Sections 7.4.2 and 7.5.2).
• Quantitative evaluation of Java, EJB3 and EJB QL based pattern matching. After fixing a common search plan, I examined and compared the efficiency of Java, EJB3 and EJB QL based pattern matching implementations on a benchmark example (Sec. 7.6).

This thesis is based upon Chapter 7, and it was published in [3, 10, 12, 15, 16, 17]. The results of this thesis have been integrated into the VIAKRA2 model transformation framework [1, 4, 5].

3.4 Incremental Graph Transformation

By investigating how pattern matching algorithms of existing GT tools use cache based techniques to speed-up the transformation process, I found that caches are typically used during a single rule application for retrieving frequently accessed graph nodes or edges, but not for reusing common matchings of subsequent transformation steps. On the other hand, by surveying other fields of computer engineering, I found that incremental techniques have been used successfully [BGT91, Hud87, GMS93], thus, it seemed to be practical to elaborate an incremental method for graph pattern matching, which stores partial matchings in memory and updates them incrementally, when notifications about model changes arrive.

Thesis 4 I elaborated a notification framework based incremental technique for graph pattern matching. Additionally, I assessed the performance of the approach by comparing it to a traditional graph transformation tool.

• Data structures for in-memory incremental graph transformation. In order to support incremental graph transformation, I proposed data structures for maintaining, efficiently storing, invalidating, and notifying partial matchings, and for accelerating the retrieval of complete matchings (Sec. 8.3).

• Algorithms for in-memory incremental graph transformation. By using these data structures, I elaborated algorithms for incremental graph pattern matching, in which complete and partial matchings of LHS and NAC patterns of a rule are stored explicitly in a snapshot tree in the main memory, and they are updated incrementally when the instance model is modified by also taking into account invalidations due to matchings of negative condition patterns (Sec. 8.4).

• Quantitative performance analysis of incremental graph transformation. By using a benchmark example, I examined and compared the run-time performance of the incremental and the traditional graph transformation approaches (Sec. 8.5).

• Incremental graph transformation in relational databases. I elaborated a method for incremental graph transformation, which maintains and stores partial matchings of graph transformation rules in relational database tables, which are updated incrementally, when the instance model is changed (Sec. 8.6).

This thesis is based upon Chapter 8, and it was published in [6, 7, 8, 14].

4 Utilization of New Scientific Results

The practical utilization of new theoretical results presented in the current thesis is now summarized.
Pattern matcher of the V\textsc{IATRA2} model transformation framework. Results of Thesis 3 are directly utilized in the development of the V\textsc{IATRA2} model transformation framework. Adaptive and model-sensitive techniques have already been built into the pattern matching module of Release 3.

Based on the experience of Thesis 4, a Rete-based approach has been developed by a graduate student. This prototype engine is now an alternative of the non-incremental pattern matching module of V\textsc{IATRA2} to be used for domain-specific editors.

Graph transformation tools with advanced search plans. Performance considerations have recently become emphatic design criteria in the development process of many graph and model transformation tools, which are now shortly surveyed.

The Transformation Execution Environment of the MOLA tool [KBC04, KCS05] performs RDBMS based graph transformation by generating and executing SQL queries and commands. Though this approach uses a different database schema, the basic idea is similar to the one presented in Thesis 2 as stated in [KCS06].

Graph Rewrite Generator (GrGen) [GBG+_06], which is a generative programming system for graph transformation, has been recently reimplemented for the .NET framework. The optimization technique [Bat05, Bat06] of this new version is highly similar to the adaptive and model-specific pattern matching approach of Thesis 3 with minor differences in operation cost assignment and search plan cost calculation.

Research for implementing efficient pattern matching in graph transformation. The previous list also indicates that a recent research boost can be experienced in the field of developing efficient graph pattern matching heuristics and techniques. Many of these directions build upon the results of my theses.

As reported in [GBG+_06], the benchmarking framework of Thesis 1 has been directly used by the developers of the GrGen tool for measuring the run-time performance of their tool. In addition, in [GK07] they suggested several improvements for the framework itself, for the benchmark implementations, and for chronometry issues. The benchmarking framework is mentioned in several papers [FMRS07, ZKZ07] and Master’s theses [Fal05, Kar07, Lil06, Sza05] as well.

The feasibility of the adaptive, model-sensitive graph pattern matching technique of Thesis 3 has recently been confirmed by the developers of the GrGen tool after performing a quantitative analysis reported in [BKG07], which also used the benchmarking framework proposed by Thesis 1. According to [BKG07], the adaptive approach can be an order of magnitude faster than any other known graph transformation systems.

Publications in the Subject of the Dissertation

Number of publications: 20
Number of peer reviewed publications: 16
Number of independent citations: 71

Journal papers (appeared abroad)


**Journal papers (appeared in Hungary)**


**International conference and workshop papers**


Conference and workshop papers in Hungarian


Electronic publications (Technical reports)


References


[KCS05] Audris Kalnins, Edgars Celms, and Agris Sostaks. Model transformation approach based on MOLA. In Jean Bézivin, Bernhard Rumpe, Andy Schürr, and Laurence


