ONLINE VALIDATION OF VISUAL MODEL TRANSFORMATIONS

Ph.D. Thesis Booklet

László Lengyel

Advisors:
Dr. Hassan Charaf, Ph.D.
Associate Professor

Dr. Tihamér Levendovszky, Ph.D.

BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
DEPARTMENT OF AUTOMATION AND APPLIED INFORMATICS

Budapest, 2006
László Lengyel

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

1111 Budapest, Goldmann György tér 3.

e-mail: lengyel@aut.bme.hu
tel: +36(1)4631007
fax: +36(1)4633478

Advisors:
Dr. Hassan Charaf, Ph.D.
Associate Professor

Dr. Tihamér Levendovszky, Ph.D.
1 Preliminaries and Objectives

Model-based development (MBD) is an increasingly applied method in producing software artifacts. MBD is driven by model transformation that is attempted to bridge the large semantic gaps between high-level models and low-level languages. The goal of the model transformation-based researches is to achieve more flexible, efficient, configurable and validated transformations.

Graph rewriting [Ehrig 1979] [Blostein et al. 1995] [Ehrig & Taenzer 1996] [Rozenberg 1997] [Ehrig et al. 1999] [Baresi & Heckel 2002] is a widely used technique for model transformation. Especially visual model transformations can be expressed by graph transformations, since graphs are well-suited to describe the underlying structures of models. The atoms of graph transformations are rewriting rules, each rule consists of a left-hand side graph (LHS) and right-hand side graph (RHS). Applying a graph rewriting rule means finding an isomorphic occurrence (match) of LHS in the graph the rule being applied to (host graph), and replacing this subgraph with RHS. Graph rewriting has many branches, in case of software models the algebraic approaches [Ehrig 1979] [Rozenberg 1997], namely, the double pushout (DPO) and the single pushout (SPO) approaches turned out to be the most useful. These methods use abstract algebraic and category theoretical [Pierce 1991] [Barr & Wells 1999] constructs to formalize the process of rewriting.

Among the modeling languages the most known and used is Unified Modeling Language (UML) [OMG UML]. UML is a graphic language to specify, visualize, build, and document the software system artifacts. Beyond UML there are fields where special languages are needed, since UML would be too general for them. For this reason, Domain-Specific Languages (DSLs) [Czarnecki & Eisenecker 2000] continue to play an important role in modeling software and hardware systems. Domain-Specific Modeling (DSM) languages provide a viable solution for improving development productivity by raising the level of abstraction beyond coding. With DSM, the models are made up of elements representing concepts that are part of the problem domain world, not the code world. DSM languages follow domain abstractions and semantics, allowing developers to perceive themselves as working directly with domain concepts. In many cases, full final product artifacts can be automatically generated from these high-level specifications with domain-specific code generators.

Most models consist of a number of “nodes and edges”, pictures and some accompanying text. The information conveyed by such a model has a tendency to be incomplete, informal, imprecise, and sometimes even inconsistent. To define the visual models precisely, additional constraints must be specified beyond the diagrams. The Object Constraint Language (OCL) [OMG OCL] is a formal language for analysis and design of software systems. It is a subset of the UML standard that allows software developers to write constraints and queries over object models. OCL expressions are unambiguous and make the model more precise and more detailed. These expressions can be checked by automated tools to ensure that they are correct and consistent with other elements of the model. Similarly to models, it is worth to define model transformation rules precisely with constraints. Structural and attribute transformation methods cannot perform and express the issues, which can be addressed by constraint validation.

Model-driven development approaches (e.g., Model-Integrated Computing (MIC) [Sprinkle 2004] [Sztipanovits & Karsai 1997] and OMG’s Model-Driven Architecture (MDA) [OMG MDA] [OMG MDAGuide]) emphasize the use of models at all stages of system development. They have placed model-based approaches to software development
MIC advocates the use of domain-specific concepts to represent system design. Domain-specific models are then used to synthesize executable systems, perform analysis or drive simulations. Using domain concepts to represent system design helps increase productivity, makes systems easier to maintain and evolves and shortens the development cycle.

MDA offers a standardized framework to separate the essential, platform-independent information from the platform-dependent constructs and assumptions. A complete MDA application consists of a definitive platform-independent model (PIM), one or more platform-specific models (PSM) including complete implementations, one on each platform that the application developer decides to support. The platform-independent artifacts are mainly UML and other software models containing enough specification to generate the platform-dependent artifacts automatically by model compilers.

Transformations appear in many, different situations in a model-based development process. A few representative examples are as follows. (i) Refining the design to implementation; this is a basic case of PIM/PSM mapping [Agrawal et al. 2003] [Clark et al. 2001]. (ii) Aspect weaving; the integration of aspect models/code into functional artifacts is a transformation on the design [Assmann & Andreas 1999] [Gray et al. 2003]. (iii) Application of design patterns [Karsai 2001]. (iv) Analysis and verification; analysis algorithms can be expressed as transformations on the design [Assmann 1996]. Furthermore, UML-based development also driven by metamodeling and model-based development [Akehurst 2000] [Clark et al. 2002].

One can conclude that transformations in general play an essential role in model-based development, thus, there is a need for highly reusable model transformation tools. These tools must make the model transformation flexible and expressive, therefore, they should preferably defined visually. Furthermore, in order to the output of the transformation be valid both syntactically and semantically the model transformation should be validated.

Examine a model transformation that generates relational database management system (RDBMS) model from class diagram (also referred to as object-relational mapping) results that in the generated model there are tables with columns, and several columns are signed as primary or foreign keys. We would like to require certain things from the transformation. In case of successful execution we want the transformation to guarantee the followings:

- Classes that are marked as non-abstract in the source model should be transformed into a single table of the same name in the target model. The resultant table should contain one added primary key column, one or more columns for each attribute in the class, and one or more columns for associations based on the next rule.

- In general, an association may, or may not, map to a table. It depends on the type and multiplicity of the association.
  - Many-to-many (N:N) associations should mapped to distinct tables. The primary keys for both related classes should become attributes of the association table (foreign keys). Foreign keys should not allow NULL value, because a link between two objects requires that both of them be known.
  - One-to-many (1:N) associations using one or more foreign key columns should be merged into the table for the class on the “many” side.
- For one-to-one (1:1) associations also the foreign key should be buried optionally in one of the affected tables.

- Parent class attributes should be mapped into tables created from inherited classes.

- An association class should be transformed based on the multiplicity of the association. For N:N association the attributes of the association class become columns of the distinct table. For a 1:N or 1:1 the attributes of the association class become columns of the table in which the foreign key is buried.

The required rules jointly guarantee that the generated database is in third normal form [Blaha 1997].

In case of traversing approaches, at the implementation level, system validation can be achieved by testing. Various tools and methodologies have been developed to assist in testing the implementation of a system (for example, unit testing, mutation testing, and white/black box testing). However, in case of model transformation environments, it is not enough that the input model is a valid instance of the input metamodel and the transformation engine is validated. The transformation specification should also be validated. There are only few and not complete facilities provided for testing offline transformation specifications in an executable style. Also, there are several model transformation environment provides well-defined interfaces that allow to implement transformations using an optional object-oriented programming language. Related to the expected output there is nothing that can be guaranteed by these transformations. The transformation should be tested: not only the syntactical but the semantical correctness is also required. In fact, the testing requires huge efforts, and even after the testing it is not guaranteed that the transformation produces the expected output for all valid input. The reason is that there is no real possibility that the testing covers all the possible cases. But, in the case study the model transformation should guarantee that in the generated output model each table has primary key, each class attribute is part of a table, each many-to-many association has a distinct table, and foreign keys do not allow NULL value.

There is a need for a solution that can online validate model transformation specifications: what cannot be validated generally before the execution time (offline) that should be validated online.

Furthermore, model transformations often need to follow an algorithm that requires a strict control over the execution sequence of the transformation rules. The control flow should support the sequencing of transformation rules, conditional branching, and parameter passing.

The objective of my research was to design and implement a model transformation framework that supports the high-level control over the model transformations, the online validation transformation, and as well as the efficient constraint checking required by the validation. Taking these aspects into account, my goals were the following:

1. **Online validation of visual model transformations.**

2. **Create a new and efficient constraint evaluation method.**

3. **Create a new visual control flow language that supports sequencing transformation rules, branching with OCL constraints, hierarchical rules, parallel executions of the rules, iteration, and recursion.**
4 Work out a method that facilitates to predict the behavior of model transformations, including the decision of termination property of model transformations.

5 Elaborate a consistent constraint management method, which supports the reuse of the transformation rules and constraints.

6 Work out constraint normalization algorithms, which accelerate constraint evaluation and the whole transformation process.

2 Methodological Summary

The requirements above decided the orientation of my research and the individual tasks I should have solved one after another. The starting points of my work were the already existing model transformation tools, their operation, the constraint management mechanisms, furthermore, the methods used to define transformation rules and control flow. At Department of Automation and Applied Informatics of BUTE we developed the Visual Modeling and Transformation System (VMTS) [VMTS] that is a metamodeling and model transformation framework. During the development we have utilized the mathematical background of formal languages, the research results of graph rewriting and metamodel-based model transformation, and we have taken into consideration the experiences related to the already existing model transformation systems.

It holds for the whole research method that all of the expected results took aim at practically applicable solutions. The theoretical results of the research have been realized in different VMTS modules. Therefore, some of the VMTS components are the practical results of the research.

The methods of the theoretical research are determined by the actual tasks, the already existing subsolutions, and the defects of the existing solutions. Several times I had to come to know fields that are not related closely to model transformation, but I hoped they improve the quality, usability, and efficiency of the model transformation. From these fields the application of the aspect-oriented methods [Filman et al. 2005] was really efficient that is detailed in Thesis II.

3 Novel Scientific Results

In general, software system requirements are specified by humans. These specifications should be understood by the designer of the model transformation and translated into a given formalism. In model-based software development such a formalism is the OCL constraints propagated to model transformation rules. In general, the offline validation of model transformations is algorithmically hard, therefore, validation is performed online in the proposed approach: during the model transformation process. To achieve an intuitive and convenient treatment, the expressions of the formalism can be specified with high-level constructions. The designers of the transformation should comprehend and handle both the problem and its specification. In Thesis I, I introduce the online validation of the model transformations. Furthermore, I provide optimized algorithms to support efficient constraint evaluation during transformation.

The validation introduces a new aspect, the constraints assigned to model transformation rules, which often crosscut the model transformation rules. To support the separation
of the constraints and model transformation rules and to improve the reusability aspect-oriented methods are applied. I have worked out an efficient aspect-oriented constraint management method that is introduced in Thesis II. Furthermore, weaver algorithms are provided for constraint propagation and constraint normalization that result more efficient transformation execution.

Constraints also take place in the control flow of the transformations. I have worked out a domain-specific visual control flow language to support high-level definition of transformations. In this case the control flow constraints and the termination property of the transformations should be validated. In general, the termination is undecidable, but there are certain cases when it can be proved. Besides the termination analysis, the method also supports the prediction of further transformation properties.

The results that facilitate the online validated model transformation, the aspect-oriented constraint management, the high-level specification of transformations, and the analysis of the behavior of the transformations are divided into three theses.

**Thesis I**

**Validated Model Transformation with Efficient Constraint Checking**

I have proven the general validation, general preservation, and general guarantee, which are the basis of the online validated model transformation. I have worked out the following algorithms: (i) Rule Constraint Validator (RCV) - a naive algorithm for constraint validation, (ii) Invariant Analysis - an offline algorithm, that is independent from the execution of the transformation rules. This algorithm supports to explore the contradictions between the metamodel and transformation rule constraints immediately after the transformation rule specification. (iii) Persistent Analysis - an algorithm that performs the constraint evaluation already during the matching process, and (iv) the Optimalized Rule Constraint Validator (ORCV) - the combination of the RCV algorithm with the Persistent Analysis algorithm.

The ORCV algorithm performs the constraint validation during the model transformation that takes into consideration both metamodel and transformation rule constraints. Thesis I is contained by the forth chapter of the dissertation. Related publications: [7][8][15][16][21][22][24][25][30][34][37][43].

**Definition 3.1.** A **precondition** assigned to a transformation rule is a boolean expression that must be true at the moment when the transformation rule is fired. A **postcondition** assigned to a transformation rule is a boolean expression that must be true after the completion of a transformation rule. If a precondition of a transformation rule is not true, then the transformation rule fails without being fired. If a postcondition of a transformation rule is not true after the execution of the transformation rule, the transformation rule fails.

A direct corollary of it is that an OCL expression in LHS is a precondition to the transformation rule, and an OCL expression in RHS is a postcondition to the transformation rule. A transformation rule can be fired if and only if all conditions enlisted in LHS are true. Also, if a transformation rule finished successfully, then all conditions enlisted in RHS must be true.
Definition 3.2. • A transformation rule $S$ validates a property $P$ specified by a boolean expression, when the following condition always holds: if a property $P$ was true before the rule $S$ it remains true after the execution of the rule $S$, and if $P$ is false before the rule $S$, the rule $S$ fails.

• A transformation rule $S$ preserves a property $P$ specified by a boolean expression, when the following condition always holds: if a property $P$ was false before the rule $S$ it remains false after the execution of the rule $S$. Furthermore, if a property $P$ was true before the rule $S$ it remains true after the execution of the rule $S$.

• A transformation rule $S$ guarantees a property $P$ specified by a boolean expression, when the following condition always holds: if a property $P$ was true before the rule $S$ it remains true after the execution of the rule $S$, and if $P$ is false, the rule $S$ changes property $P$ to true.

Table 1 illustrates the truth table of the validation, preservation and guarantee properties.

<table>
<thead>
<tr>
<th></th>
<th>property $P$ before the rule $S$</th>
<th>property $P$ after the rule $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td>rule $S$ fails</td>
</tr>
<tr>
<td>Preservation</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>Guarantee</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Table 1: Truth table of the validation, preservation and guarantee properties

Definition 3.3. If an OCL constraint is specified in RHS of a transformation rule, $x@pre$ means the value of $x$ immediately before the rule was fired even if the value of $x$ has not changed. Similarly, the $x@preS$ means the value of $x$ immediately before the rule $S$ was fired.

The $x@pre$ is used when we refer to the value of the $x$ before the execution of the same rule, but we use the $x@preS$ if there is a finite sequence of rules and we refer to the value of the $x$ predicting the execution of the rule $S$.

Pre- and postconditions defined as OCL constraints and propagated to the transformation rules represents low-level constructions. On the other hand, the validation, preservation and guarantee properties are high-level constructions.

Definition 3.4. A model transformation is validated if satisfies a set of high-level constructions.

Subthesis I.1

I have given high level constructions for constraint definitions that are close to the human reasoning. Utilizing Definition 3.2 I have proved the general validation, preservation, and guarantee:
A rule $S$ validates a property $P$ for an input model $M$ if the property $P$ is enlisted in both pre- and postconditions of the rule $S$, and the rule $S$ has been executed successfully for the model $M$.

If a rule $S$ validates a property $P$ for an input model $M$ without conditions in the rule $S$ for the property $P$, then the property $P$ can be enlisted in $S^{LHS}$ and $S^{RHS}$ without changing the result of the rule $S$ for the model $M$.

A finite sequence of rules $S_0$, $S_1$ ... $S_{n-1}$ validates a property $P$ for an input model $M$ if the property $P$ is enlisted both in preconditions of the rule $S_0$ and in the postconditions of the rule $S_{n-1}$, and the finite sequence of rules $S_0$, $S_1$ ... $S_{n-1}$ has been executed successfully for the model $M$.

If a finite sequence of rules $S_0$, $S_1$ ... $S_{n-1}$ validates a property $P$ for an input model $M$ without conditions in $S_0^{LHS}$ and $S_{n-1}^{RHS}$ for the property $P$, then the property $P$ can be enlisted in $S_0^{LHS}$ and $S_{n-1}^{RHS}$ without changing the result of the finite sequence of rules $S_0$, $S_1$ ... $S_{n-1}$ for the model $M$.

(Where $S^{LHS}$ and $S^{RHS}$ denote the left-hand side and the right-hand side of the transformation rule $S$, furthermore, the $S_0$ and the $S_{n-1}$ sign the first and the last rule of the finite sequence of rules $S_0$, $S_1$ ... $S_{n-1}$.)

A rule $S$ preserves a property $P$ for an input model $M$ if the expression (NOT $P$ or $P@pre$) and ($P$ or NOT $P@pre$) is enlisted as an OCL expression in the postconditions of the rule $S$, and the rule $S$ has been executed successfully for the model $M$.

If a rule $S$ preserves a property $P$ for an input model $M$ without a postcondition in the rule $S$ for the property $P$, then the expression (NOT $P$ or $P@pre$) and ($P$ or NOT $P@pre$) as an OCL expression can be enlisted in $S^{RHS}$, and the rule $S$ also preserves the property $P$ for the model $M$.

A finite sequence of rules $S_0$, $S_1$ ... $S_{n-1}$ preserves a property $P$ for an input model $M$ if the expression (NOT $P$ or $P@preS_0$) and ($P$ or NOT $P@preS_0$) is enlisted as an OCL expression in the postconditions of the rule $S_{n-1}$, and the finite sequence of rules $S_0$, $S_1$ ... $S_{n-1}$ has been executed successfully for the model $M$.

If a finite sequence of rules $S_0$, $S_1$ ... $S_{n-1}$ preserves a property $P$ for an input model $M$ without conditions in $S_{n-1}^{RHS}$ for the property $P$, then the expression (NOT $P$ or $P@preS_0$) and ($P$ or NOT $P@preS_0$) as an OCL expression can be enlisted in $S_{n-1}^{RHS}$, and the finite sequence of rules $S_0$, $S_1$ ... $S_{n-1}$ also preserves the property $P$ for the model $M$.

A rule $S$ guarantees a property $P$ for an input model $M$ if the property $P$ is enlisted in the postconditions of the rule $S$, and the rule $S$ has been executed successfully for the model $M$.

If a rule $S$ guarantees a property $P$ for an input model $M$ without a postcondition in the rule $S$ for the property $P$, then the property $P$ can be enlisted in $S^{RHS}$, and the rule $S$ also guarantees the property $P$ for the model $M$.

A finite sequence of rules $S_0$, $S_1$ ... $S_{n-1}$ guarantees a property $P$ for an input model $M$ if the property $P$ is enlisted in the preconditions of rule $S_i$ (where $1 \leq i \leq n$), and $S_i$, $S_{i+1}$ ... $S_{n-1}$ preserve the property $P$, or the property $P$ is enlisted in the postconditions of rule $S_i$ (where $1 \leq i \leq n$), moreover $S_{i+1}$, $S_{i+2}$ ... $S_{n-1}$ preserve
the property \( P \), and the finite sequence of rules \( S_0, S_1 \ldots S_{n-1} \) has been executed successfully for the model \( M \).

If a finite sequence of rules \( S_0, S_1 \ldots S_{n-1} \) guarantees a property \( P \) for an input model \( M \) without conditions in \( S_{n-1}^{RHS} \) for the property \( P \), then the property \( P \) can be enlisted in \( S_{n-1}^{RHS} \), and the finite sequence of rules \( S_0, S_1 \ldots S_{n-1} \) also guarantees the property \( P \) for the model \( M \).

Furthermore, I have shown that the general validation, preservation and guarantee are the basis of the validated model transformation: Let transformation \( T \) contain rules \( S_0, S_1 \ldots S_{n-1} \) specified by pre- and postconditions. If the transformation \( T \) has been executed successfully for an input model \( M \), then the generated output model \( M' \) satisfies the high-level constructions defined by the pre- and postconditions of the transformation rules.

Therefore, the modeler's task is to create the adequate transformation rules and specify them with high-level constructions. If the execution of the transformation finishes successfully, then it produces a valid result that satisfies the high-level constructions defined by the pre- and postconditions.

After I have examined and proved the validated model transformation I have worked out a constraint validator algorithm that can be applied during the model transformation. The algorithm evaluates the constraints in two separated phases: (i) after the matching process it evaluates the preconditions of the transformation rule on the matched submodels (Definition 3.1), and (ii) immediately after the transformation it checks the output based on the postconditions of the transformation rule (Definition 3.1).

The Rule Constraint Validator Algorithm. Fig. 1 presents a block diagram to illustrate how the Rule Constraint Validator (RCV) algorithm validates the constraints contained by the transformation rules during the model transformation. The algorithm does not interpret the constraints during the transformation, but they are converted to source code and a compiled binary by an OCL compiler. The compiled binary supports the constraint validation during the transformation [20] [28].

I have proven that the computational complexity of the RCV algorithm for constraint validating is the following:

- \( O(s) + O(lg v_m) \) - once for a transformation rule, and
- \( O(lg v) \) - for each rule firing,

where \( s \) is the number of the symbols in the OCL constraints, \( v_m \) is the number of the metamodel vertices and \( v \) is the number of the input model vertices.

The RCV algorithm evaluates constraints after the matches are found, but metamodel-based model transformation rules facilitate to evaluate certain constraints before the rewriting process, during the matching process. The Invariant Analysis (IA) algorithm can be applied independently form transformation rule firing, immediately after the creation or modification of the rules. The algorithm is executed only in the case of rule modification, therefore it does not increase the complexity of the transformation. For optimizing the RCV algorithm I have worked out the Persistent Analysis (PA) algorithm that can be applied during the matching process. The algorithms significantly accelerate the transformation process, because the contradictions between the metamodel and transformation rule constraints are explored in the earliest phase.
Figure 1: Constraint validation during model transformation

The **Invariant Analysis (IA) Algorithm**. The IA algorithm compares the constraints contained by the metamodel and the transformation rules. The goal of the Invariant Analysis is to reveal immediately after the transformation rule specification if a constraint in the transformation rule contradicts a constraint in the metamodel. The IA can be applied to examine both preconditions and postconditions.

After the specification of a transformation rule, the rule is modified only few times, but it is fired on optional number of times. Invariant Analysis is independent from the rule firing; it runs only few times for a transformation rule, immediately after the rule creation and modification, therefore the complexity of the Invariant Analysis does not increase the complexity of the transformation process.

**Subthesis I.2**

I have proven that the computational complexity of IA algorithm is $O\left(\sum_{i=1}^{v_r} \lg c_m + (c_{ri} \cdot c_{mi})\right)$. Where $v_r$ denotes the number of the pattern rule nodes in the transformation rules, $c_{ri}$ is the number of the constraints contained by the pattern rule node $i$, and $c_{mi}$ is the number of the constraints contained by the metamodel node of the same type.

Furthermore, I have shown that if the Invariant Analysis is used during the transformation rule creation, and the algorithm detects that at least one of the constraints in the transformation rule contradicts a constraint in the metamodel, then the algorithm marks the contradicting constraint. Since the rule would be unsuccessful, and therefore the saved computational time is $SC_{IA} \geq n^k - C_{IA}$. $SC_{IA}$ denotes the saved computational time using Invariant Analysis, $n^k$ is the complexity of the matching process (where $n$ is the number of the input model nodes, and $k$ is the number of nodes in the matched submodel) and $C_{IA}$ is the computational complexity of the Invariant Analysis. The saved computational time presented here is valid only for the first rule firing, because we can assume that after the warning the designer of the transformation will correct the wrong constraint. If the constraint is not corrected, then the execution of the rule is interrupted and the saved computational time is $n^k$ at each attempt to fire the rule.

The **Persistent Analysis (PA) Algorithm**. The PA algorithm evaluates the con-
straints continuously during the matching process. PA algorithm is executed in parallel with the matching process, therefore it evaluates only the preconditions of the transformation rules.

Constraints checked by the PA algorithm are not evaluated after the matching process, therefore the computational complexity of the PA algorithm does not increase the complexity of the rule firing. Furthermore, during the matching process the nodes, the constraints are evaluated on, are available, therefore there is no need further query, thus the algorithm reduces the total time of the transformation.

Subthesis I.3

I have shown that the computational complexity of the PA algorithm is at most $O(k)$, where $k$ is the number of nodes in the matched submodel. In addition, I have proven that the Persistent Analysis algorithm does not increase the complexity of firing a rule. Using the Persistent Analysis algorithm, the saved computational time is $n^{k-r}$ if the algorithm finds an unsatisfied constraint, while it evaluates the constraints on the matched input model node $r$.

The Optimized RCV Algorithm. Based on the results above I have worked out the Optimized RCV (ORCV) algorithm. The ORCV algorithm completes the RCV algorithm with the Persistent Analysis algorithm that results a more powerful constraint validation algorithm. The ORCV algorithm explores already during the matching process if there are not satisfied constraints propagated to the transformation rules, therefore it saves the further part of the matching and the rewriting process.

Subthesis I.4

I have proven that the computational complexity of the Optimized RCV algorithm never exceeds the computational complexity of the RCV algorithm:

- $C_{ORCV} = C_{RCV}$ - if the Persistent Analysis algorithm cannot evaluate any constraint during the matching process.

- The computational complexity of the ORCV algorithm is at least with $p \times \lg v$ less than the computational complexity of the RCV algorithm if the Persistent Analysis algorithm can evaluate $p$ constraints during the matching process, and all of the constraints are satisfied. In this case the saved computational time is $SC_{ORCV} \geq p \times \lg v$.

- The computational complexity of the ORCV algorithm is at least with $n^{k-r}$ less than the computational complexity of the RCV algorithm if the Persistent Analysis algorithm finds an unsatisfied constraint, while evaluates the constraints on the matched input model node $r$.

Based on the results above I have worked out the online validated model transformation, and I have made possible that the constraint evaluation can be executed already during the matching process of the model transformation.
THESIS II
Aspect-Oriented Constraint Management

Validation of the model transformations are based on the constraints propagated to the model transformation rules. Often, constraints appear several times in transformations and crosscut the transformation rules. I have worked out an aspect-oriented constraint management method that facilitates consistent constraint management and makes the transformation rules and the constraints reusable. I have introduced a new type of aspect, the constraint aspect that fits into visual model transformation frameworks and provides more efficient constraint management and processing methods. I have proven that the removal and the latter weaving of the constraints do not have effect on the results of the model transformations. In addition, I have worked out normalization algorithms, which eliminates navigation steps from the constraints and accelerates the constraint evaluation and the whole transformation.

Thesis II is contained by the fifth chapter of the dissertation. Related publications: [9][10][15][22][28][29][30][31][32][35][37][38][41][45][51][52].

Aspect-Oriented Constraints. I have worked out an aspect-oriented constraint management that supports the metamodel-based model transformation. With this method we can avoid the crosscutting constraints appearing in model transformation rules. The method facilitates to define constraints independently from transformation rules and later propagate them to the appropriate pattern rule nodes. The method results a simple and consistent constraint management.

I have worked out the type-based weaving, which is driven by the metatype information of the pattern rule nodes of the transformation rules, and the property-based weaving that is supported by the weaving constraint. With these methods constraints can be woven to several pattern rule nodes that satisfies the required weaving conditions.

Constraint Aspect. I have introduced a new type of aspect: the constraint aspect. A constraint aspect contains not only textual conditions described by the OCL constraints but it has structure, type and multiplicity conditions and weaving constraints as well.

Subthesis II.1

I have proven that OCL constraints can be converted into equivalent constraint aspects. I have worked out an algorithm (CREATECONSTRAINTASPECT) to support the conversion, and I have shown that the computational complexity of the algorithm is $O(n)$, where $n$ denotes the number of the navigation steps in the processed constraint.

The NORMALIZECONSTRAINT and DECOMPOSECONSTRAINT Algorithms. Supporting constraint normalization I have worked out the NORMALIZECONSTRAINT and DECOMPOSECONSTRAINT algorithms. The algorithms apply constraint relocation and constraint decomposition to eliminate navigation steps from the constraints.

I have introduced the AND/OR clauses that facilitate to decompose constraint expressions connected with or/xor operations.

Subthesis II.2

I have shown that the complexity required by the AND/OR clause management does not reach the computational complexity that is saved by the application of the clauses.
I have given conditions for the applicability of the normalization algorithms and I have proven that they improve the efficiency.

Furthermore, I have shown that normalization methods are applicable not only for the constraint aspects and transformation rules but also for optional UML class diagram.

Subthesis II.3

I have shown that with constraint relocation the model generated by the NormalizeConstraint algorithm contains as few navigation steps as possible. In addition, I have shown why the constraint replacement cannot be allowed through edges that allow zero multiplicity.

The computational complexity of the NormalizeConstraint algorithm is $O(\sum_{i=1}^{c} n_i + v^3)$, where $c$ denotes the number of the propagated constraints contained by the transformation rule, $n_i$ is the number of the navigation steps contained by the constraint $i$, and $v$ denotes the number of pattern rule nodes in the transformation rule.

Subthesis II.4

I have proven that applying the DecomposeConstraint algorithm, the number of the navigation steps in the constraints contained by the output model is minimal.

I have shown that the proposed constraint relocation and constraint decomposition do not modify the result of the constraint evaluation.

The GlobalConstraintWeaver and the ConstraintAspectWeaver Algorithms. I have worked out the GlobalConstraintWeaver and the ConstraintAspectWeaver (GCW and CAW) algorithms that propagates OCL constraints and constraint aspects to pattern rule nodes. Weaver algorithms are executed before the transformation firing. The outputs of the weavers are not stored as new transformation rules or transformations but as weaving configurations. The results are handled as links between the constraints and transformation rules. The weaving should be accomplished once for a set of constraints and a transformation, but the weaving result can be executed optional time.

Subthesis II.5

I have shown that the computational complexity of the GlobalConstraintWeaver algorithm is at most $O(\sum_{i=1}^{c} \sum_{j=1}^{s} (\lg v_r + n_{ij} \cdot v_r^h + n_{ij}))$, where $v_r$ denotes the number of the nodes in the actual transformation, $\lg v_r$ is the complexity of querying a node from the database, $c$ denotes the number of the constraints to be propagated to the transformation rules, and $s$ is the number of the transformation rules. Furthermore, $n_{ij}$ is the number of the nodes in the transformation rule $j$ with the same type as the context of the constraint $i$. $v_r^h$ is the worst case for finding an isomorphic submodel, where $k$ is the size of the submodel.

In addition, I have proven that the computational complexity of the ConstraintAspectWeaver algorithm is at most $O(\sum_{i=1}^{ca} \sum_{j=1}^{s} (n_{ij}^k + \sum_{p=1}^{c_i} m_{jp}))$, where $ca$ denotes the number of the constraint aspects, $s$ is the number of the transformation rules, $c_i$ is the
number of the constraints contained by the constraint aspect $i$, $n_i^k$ is the complexity of the metatype-based matching (worst case), $n_j$ is the number of pattern rule nodes contained by the transformation rule $j$, and $k_i$ is the number of pattern rule nodes contained by the constraint aspect $i$. The $m_{jp}$ is the number of pattern rule nodes with the metatype that corresponds to the context information of constraint $p$.

Furthermore, I have shown that with the presented constraint management, not only individual transformation rules but whole transformations can be validated as well.

Subthesis II.6

I have proven that using constraint aspects in model transformation provides the same result as the OCL constraints.

In addition, I have shown that working with constraint aspects is more efficient than OCL constraints. The structure of the constraint aspects facilitates to apply the metatype-based matching to accelerate the weaving process. Furthermore, I have shown that the evaluation complexity of the propagated normalized constraint aspect reaches only in the worst case the evaluation complexity of the propagated OCL constraint.

Constraint aspects facilitate a more efficient constraint management and transformation firing in metamodel-based model transformation approaches.

An overview of the different aspect-oriented constraint notions and the presented algorithms is depicted in Fig. 2.

The results of the thesis facilitate the aspect-oriented constraint management in metamodel-based model transformation frameworks. The approach provides a consistent and efficient constraint management, furthermore, it supports the reuse of the transformation rules and the constraints.
The new visual control flow language (Visual Control Flow Language, VCFL) that supports sequencing transformation rules, branching with OCL constraints, hierarchical transformation rules, parallel execution of the rules and iteration as well as recursion. I have proposed a method for composing metamodel-based model transformation rules. I have proven that taking into account certain conditions the application of the composed transformation rule is equivalent with the execution of the original transformation rules. I have shown that the proposed algorithms facilitate the analysis of the behavior of the transformations. Such a predictable property is the termination of the transformations. I have given and proven conditions for the termination of the VCFL transformations.

Thesis III is contained by the sixth chapter of the dissertation. Related publications: [11][12][14][16][22][30][37][39][41][43][44][46][47][49][50][53].

I have worked out a new UML activity diagram-based control flow language that supports the following constructs:

- **Sequencing Transformation Rules.** Sequencing transformation rules results a transformation that contains the rules in an ordered sequence ($S_0, S_1 \ldots S_{n-1} \in RULES$). Assume the case that the input model of rule $i$ ($S_i$) is the model $M_i$ and the result of the $S_i$ is the $M_{i+1}$ (where $0 \leq i \leq n-1$). In this case the input model of the rule $i + 1$ ($S_{i+1}$) is the model $G_{i+1}$. This means that during the execution of the rule sequence, each rule works on the result of the previous rule. The interface of the transformation rules allows the output of one rule to be the input of another rule, in a dataflow-like manner. This is used to sequence expression execution. In VCFL, this construction is referred to as external causality. An external causality creates a mapping between a node contained by RHS of the rule $i$ and a node contained by LHS of the rule $i + 1$. This feature accelerates the matching and reduces the complexity, because the rule $i$ can provide partial match to the rule $i + 1$.

- **Branching with OCL Constraints.** In control flow models decision elements supports the constraint-based branching. OCL constraints assigned to the decision elements can choose between the paths of optional numbers. The control flow path is chosen based on the properties of the actual input model and the success of the last transformation rule. Each branch has an exact OCL guard condition. Constraints belonging to different branches cannot have common part. This restriction ensures that the control flow execution is deterministic.

- **Hierarchical Transformation Rules.** VCFL supports hierarchical specification of the transformation rules. High-level rules can be created by composing a sequence of primitive rules and can be viewed as separate transformation modules. Often, the OCL constraints assigned to a decision object do not cover all possible cases. It could result that none of the branch paths is selected in certain cases: the parent rule of the actual transformation handles the control flow.
• **Iteration, Tail Recursion and Parallel Executions of the Rules.** The iteration is achieved with the help of the decision objects and the OCL constraints contained by them. Recursion is resulted by the composition of the iteration and external causalities. The parallel execution of the independent transformation rules is supported by the *Fork* and *Join* elements.

I have worked out a comfortable error handling mechanism: if a transformation rule fails, and the next element in the control flow is a decision element then it could provide the next branch based on the OCL statements evaluated on the actual model and the result of the previous rule execution. If no decisions can be found, the control is transferred to the parent state, if there is no parent state, the transformation terminates with error.

I have provided algorithms to check whether a transformation contains isolated or illegal transformation rules and to validate that the OCL constraints assigned to a decision object are disjoint.

I have assigned two specific attributes to each transformation rule: Exhaustive and MultipleMatch. An exhaustive transformation rule is executed continuously as long as LHS of the rule could be matched to the input model. The MultipleMatch attribute of a rule allows the matching process to find all not overlapping occurrences of LHS in the input model, and replacing is executed on all the found places.

The **VTSCOMPOSING Algorithm.** I have worked out algorithms for composing and self-composing metamodel-based model transformation rules. Besides the structure of the transformation rules, the algorithms take into account the metatypes of the pattern rule nodes and the external causalities between the transformation rules as well. These algorithms supports the analysis of the transformations, exhaustive transformation rules, and loops.

I have differentiated and examined three transformation rule structures: *tree*, *inserting* and *deleting* structures.

**Subthesis III.1**

*Let S be a transformation rule with tree, inserting or deleting structure that defines external causalities between the model elements appearing both on RHS and LHS. In addition, let $S^n$ be the transformation rule resulted by composing n times the rule S. I have proven that applying the transformation rule $S$ n times on a finite input model is equivalent to a single execution of the rule $S^n$.*

Furthermore, I have shown that if the transformation rules $S_j, S_{j+1} \ldots S_k$ are applicable successfully for a finite input model $M$, and the only composed transformation rule $S_C$ that can be created from transformation rules $S_j, S_{j+1} \ldots S_k$, then it has the same effect on the input model $M$ as the transformation rules $S_j, S_{j+1} \ldots S_k$.

During the transformation rule composition we have to take into consideration the multiplicity values of the pattern edges. I have worked out rules that define the multiplicity values in the composed transformation rules.

An exhaustive transformation rule must contain either an attribute modification or an element deletion to prevent that the same match is found again and again by the matching process. Furthermore, the termination is ensured with the following type transformation rule as well.
Definition 3.5. A terminating creation rule $S_{TC}$ contains an OCL constraint $C$ in $S_{LHS}^{TC}$, which must stand for the input models matched to the $S_{LHS}^{TC}$, and as a result of the rule execution, the condition required by the constraint $C$ becomes false. Furthermore, if the rule $S_{TC}$ can be composed with itself, then for all possible composition $S_{i}^{RHS}$ is not included by the $S_{i}^{LHS}$.

Subthesis III.2

I have proven that exhaustively applied TC transformation rules terminate for arbitrary finite input model.

I have shown that a control flow without exhaustive transformation rules and loops always terminates. In addition, I have proven that a control flow terminates if all exhaustive transformation rule and loop terminate.

Furthermore, I have given conditions, and I have proven when a VCFL transformation does not terminate because of an exhaustive transformation rule or a loop:

- If $S \in T$ is an exhaustive transformation rule and $S_{LHS}^{TC} \subseteq S_{RHS}^{TC}$, the transformation $T$ does not terminate.
- If $S \in T$ is an exhaustive transformation rule, $S$ does not contain delete and modify type internal causalities, and $S$ is not a TC rule, then the transformation $T$ does not terminate.
- If $L \in T$ is a loop and $S_{C}$ is the composition of the transformation rules $S_{h}, S_{h+1}, \ldots S_{h+n} \in L$, $S_{C}^{LHS} \subseteq S_{C}^{RHS}$, and the only exit condition of $L$ is the SystemLastRuleSucceed, then the transformation $T$ does not terminate.

I have worked out an algorithm that validates the presented properties on VCFL transformations. I have shown that the VMTS termination algorithm cannot make decision in all cases. In certain cases the decision cannot be made based on the VCFL model only, or cannot hold for all valid input model.

The results of the Thesis III provides a high-level visual control flow language. Furthermore, taking into account certain conditions the results facilitate the analysis of the transformations and the prediction of their behavior.

The general validation, preservation, guarantee, the aspect-oriented constraint management, and the high-level visual control flow together form an efficient, online, and validated model transformation.

4 Application of the Novel Scientific Results

The novel scientific results appear in VMTS metamodeling and model transformation framework, and in its applications. Based on the case studies mentioned below we can present the applicability to the industry, but the number of the applications is not finite. It is expected to increase with the growing interest.

Related publications: [7][15][16][19][20][21][22][23][25][30][34][37][40][41][42][43][46][52][53].

I have shown the followings via the Visual Modeling and Transformation System (VMTS) software package:
The model transformation validated online can be realized with constraint management and validation methods.

The online constraint validation accelerates the whole model transformation process.

Aspect-oriented constraint management is an efficient part of the model transformation systems. It makes the transformation rules and the constraints reusable.

Constraint relocation and decomposition are applicable for UML class diagram-based models.

A visual control flow language can be realized as a stereotyped UML activity diagram. Such a visual transformation language can support sequenced transformation rules, constraint-driven branching, hierarchical rules, recursion, iteration and parallel rule execution.

Metamodel-based model transformation rules can be composed using the presented composing algorithm.

Certain termination properties of the transformations can be validated offline, without input models.

I have solved the validated generation of the database model (Section 1) and further engineering problems guaranteeing a set of high-level constructions expecting from the transformation.

Framework

To prove the practical applicability of the results I have developed the following components of the VMTS: (i) AGSI Core (Attributed Graph Architecture Supporting Inheritance), which using a relation database management system stores the models and supports their query and modifications, (ii) VMTS Rule Editor, which facilitates the specification of the metamodel-based model transformation rules, (iii) VMTS Control Flow Designer that supports the visual definition of control flow models, and (iv) VMTS Aspect-Oriented Constraint Manager, which supports the creation, modification and weaving of aspect-oriented constraints and constraint aspects. The further components of the software package are developed by Gergely Mezei, Tihomér Levendovszky, and numerous students.

Visual Modeling and Transformation System (VMTS) is an n-layer metamodeling environment which supports editing models according to their metamodels, and allows
László Lengyel

Online Validation of Visual Model Transformations

Figure 4: VMTS block diagram

specifying Object Constraint Language (OCL) constraints. Models are formalized as directed, labeled graphs. VMTS uses a simplified class diagram for its root metamodel ("visual vocabulary"). Also, VMTS is a model transformation system, which transforms models using graph rewriting techniques. Moreover, the tool facilitates the validation of the constraints specified in the transformation rule during the model transformation process.

Applications

The metamodel-based, visual, and validated model transformation has been applied several times to solve engineering problems. (i) The method facilitates the model-based unification of mobile platforms. From the same input models (resource model and statechart diagram) with the help of model transformations we have generated applications for different mobile platforms [30] [37] [40] [43] [46]. (ii) The model transformation-driven software maintenance is also supported. The method with the help of the constraints differentiates the new, modified, and the already processed model elements. Based on it the approach generates source code only from the new and modified part of the model [34]. (iii) The approach facilitates the validated model transformation. It provides a solution for the tasks similar to the validated database model generation (Section 1) [7] [37] [41]. (iv) The VMTS Presentation Framework (VPF) uses a flexible plug-in-based architecture to offer individual, metamodel-dependent visualization and editing features. In order to increase the efficiency of the development we apply model transformation that automatically generates the classes of the domain-specific language elements from the model-based
graphical language definition. (v) We have developed model processors for UML and other domain-specific languages [30] [37] [40] [43]. We have generated C# source code from UML class diagram [33], and C++ source code for the Quantum Framework [Samek 2002] platform from UML statechart diagram [25].

In order to achieve the validated model transformation we have to utilize the results of the Thesis I, which facilitates the validation, preservation, and guarantee of the attribute values, and providing a more efficient constraint evaluation mechanism accelerate the whole transformation process. With the help of the results of the Thesis II we can manage the constraints with aspect-oriented methods. The model transformation converting class diagram to database model consists of nine transformation rules and contains 6 different constraints. With conventional methods two of the constraints crosscut the transformation. The first constraint appeared in 9 transformation rules 30 times, and the second constraint was assigned to 6 transformation rules 16 times. Without the presented aspect-oriented methods the constraint management would be difficult and inconsistent. With the results of the Thesis III we can define the transformations on high-level and the branching conditions with OCL constraints.

The sample transformations illustrate that VMTS is an efficient model transformation tool, which using constraint validation supports the online validated model transformation.

5 Related Publications

Book

Journals

3 T. Levendovszky, L. Lengyel, H. Charaf, “Implementing a Metamodel-Based Model Transformation System”, Buletinul Stiintific al Universitatii “Politehnica” din Timisoara, ROMANIA Seria AUTOMATICA si CALCULATOARE PERIODICA POLITEHNICA, Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE Vol.49 (63), 2004, ISSN 1224-600X.


László Lengyel  

Online Validation of Visual Model Transformations


Publications in International Conference Proceedings


6 Citations

[18] is cited by:


7 References


