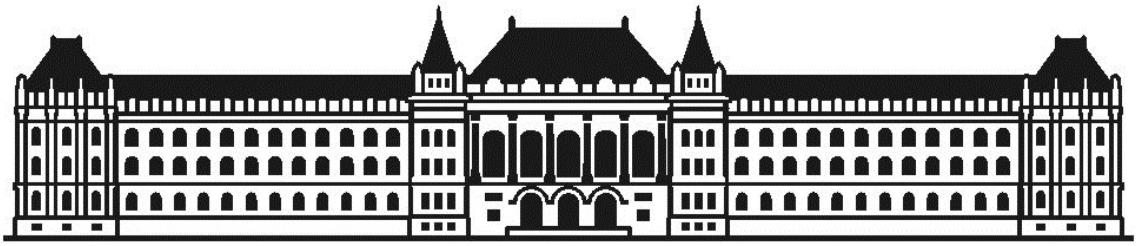

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Ph.D. Thesis Booklet

Automated Offline Verification of Graph Rewriting-Based Model Transformations

by

Márk Asztalos

Advisors:
Tihamér Levendovszky Ph.D.
László Lengyel Ph.D.

Budapest, 2012.

Ph.D. Thesis Booklet

Márk Asztalos

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

1117 Budapest, Magyar tudósok krt. 2. (Building Q) .

e-mail: asztalos@aut.bme.hu
tel: +36(1)463-42-25

Advisors:

Dr. Tihamér Levendovszky, Ph.D.
Dr. László Lengyel, Ph.D.

1 Background and Goals

Nowadays the application of visual models has become a commonly used technique in software engineering. Model-based software development can significantly improve the understandability of the software, decrease the development time, and make the use of automated development methods possible, which in turns leads to increased productivity [SVC06]. Moreover, models usually raise the level of abstraction as well, which is advantageous when designing large and complex systems. Models can also be understood by domain experts, which is an important step in the evolution of languages designed for software development. Typically, models consist of nodes representing the entities of a domain and edges representing the relations between the entities. The nodes and edges usually have attributes that specifies the properties of the elements. Therefore, models can be formalized as graphs that provide a strong mathematical background along with a user friendly visual representation [EEPT06]. Graph-based models are widely used in different areas of engineering and science. An advantage of the use of such models is that they may be visualized by domain-specific concrete syntax. Although graphical models are often used, this is not mandatory, textual models are also an option.

Unified Modeling Language (UML) [FS99] standardized by the Object Management Group (OMG) [Obj07] is the de-facto standard language to describe the static structure and behavior of object-oriented systems. Since UML is a general purpose language, it is not always easy to express aspects related to a concrete application domain. Therefore, domain-specific modeling languages (DSMLs) [KT08] have gained focus recently to specify modeling languages for specific application domains. This approach to model-based software development therefore aims to find domain-specific abstractions and make them accessible through formal modeling. One of the most popular forms to define visual DSMLs is metamodeling [Nor99, Küh06, HR00]. *Metamodeling* means using a specific DSML to model the specification of a language. This model is called the *metamodel* of the language and the models created using this language are referred to as the *instance models* of the metamodel.

In many model-based approaches [KWB03, SK97], model processing programs [SK03, BBG+06, MCG05] (often called as model transformations) are regularly used to update existing models, or generate new ones. Typical scenarios for the application of such programs are: (i) code generation, e.g. generating executable program code from UML diagrams or from domain-specific modeling languages (note that textual code is often treated as a model as well), (ii) generating lower-level models from higher-level ones, or reverse engineering higher-level models from lower-level ones, (iii) model refactoring, (iv) providing dynamic semantics (behavior) to models, (v) synchronizing models. The input and output of such programs are the models themselves. A model processing program can be written in any programming language. However, I have mentioned that models are usually defined on higher abstraction levels, therefore, it is reasonable to implement model processing programs using languages that provide the same abstraction level as the models themselves. For models that are represented by different types of graphs, algebraic graph rewriting [EEPT06, Hec06] is a frequently used technique to formalize model transformations. The visual representation and high abstraction level of such programs makes it possible to use them in larger, more complex systems. In the theory of algebraic graph rewriting systems, *rewriting rules* specify the operation primitives to modify graphs (this modification is formalized by the double-pushout (DPO) or Single Pushout (SPO) approach using category theory [BW90]). Informally, a rule is defined by a left-hand-side (LHS) and a right-hand-side (RHS) graph. Given an input graph, the application of a rule means finding an isomorphic occurrence of LHS (match) in the input graph and replacing the occurrence by RHS. In the context of graph rewriting, the term transformation sometimes covers the execution of a concrete sequence of rewriting rules on a concrete input model. In my terminology, the term *(graph rewriting-based) model transformation* is the specification (not the execution) of a model processing program whose semantics is based on graph rewriting systems and is specified by a set of rewriting rules as well as an additional control mechanism that defines the execution order (scheduling) of the rules. This control structure can be formalized in many ways (different approaches have been categorized in [BFG96, SV09]) e.g.

as a directed graph whose nodes are the individual rules.

Model-driven software engineering techniques are of great importance [GH06, WML10], because in several industries – such as aerospace, automotive, and medical –, the certification of safety, reliability, or security critical systems is essential [GHN10, WL06], and the formal background of these techniques largely facilitates the formal analysis. Since models are extensively used, the verification of the model processing programs has become an emerging research field, because, such programs are usually applied repeatedly in an automated way [SK03, KWB03]. During the verification, we need to guarantee that the program works correctly, hence, we do not need to validate the output models of each execution separately. Verification means determining the correctness of the transformation in the sense that it satisfies certain functional and non-functional properties. Functional properties concern the output model only or the relation between the input and the output models (i.e. how the input model has been modified) and are usually domain-specific. Besides the functional properties, non-functional properties of the transformations should be analyzed, such as *termination* and *confluence*. Termination is an important property of any program, however, in the case of model transformations, the non-deterministic parts (e.g. finding a match) of the execution makes it harder to be proved and demands specialized analysis methods. In the context of model transformations, confluence covers determinism. When a software developer implements a program, most often, this program needs to produce the same output for the same input when executed repeatedly. Therefore, although non-deterministic execution comes from the nature of graph rewriting rules, it is useful to analyze if the result of the transformation will be the same for different executions.

Verification of graph rewriting-based model transformations has become possible [ALS+12], because the solid mathematical basis of algebraic graph transformations facilitates their automated analysis. Verification methods for model transformations can be divided into three categories based on the generality of the results: *online*, *static* and *offline* methods.

- The verification is called *online* if the output model is validated during the execution [Len06] by the execution engine. This means that when the execution of the transformation successfully finishes, we know that certain properties are satisfied by its output, because otherwise the transformation would have failed. However, this method does not guarantee that the transformation successfully terminates. We usually refer to this method as *online validation*.
- The verification is called *static* if the correctness of the model transformation is proved with respect to a concrete input model, however, the analysis is performed without executing the transformation. An example for such a method can be found in [KN08]. To emphasize the importance of this method and the difference between online and static analysis, it worth mentioning that model transformations are often used to define graph grammars that are a set of rewriting rules along with a concrete input model [Roz97a]. In this case, not just the application of a concrete rewriting rule may imply non-determinism, but the rules to be applied are also chosen non-deterministically just as in the case of string rewriting. Non-determinism causes that even if one execution of a model transformation results in a correct output, it is not guaranteed that the next execution with the same input model also results in a correct output. Therefore, it can be seen that the results of the static analysis is more general than that of online validation.
- A verification technique is called *offline* only if the definition of the transformation and the language specification that describes the models to be transformed are used during the analysis process. The results of the offline analysis are general in the sense that they are independent from the concrete input models. The main advantage of this method is that the offline analysis needs to be performed only once, because, it guarantees the correctness of the result of any possible execution. However, the obvious disadvantage of this approach is the increased complexity of the analysis itself. It worth mentioning that, for example, the termination of a graph rewriting transformation is undecidable in general [Plu98].

Since model processing programs are used repeatedly in an automated way, analysis methods for

their formal verification gained focus recently and have become a research field with high importance. Graph rewriting-based model transformation is a promising candidate for the specification of model processing programs, because this method has many advantages:

- Graphs are the most often used formalism for the specification of visual modeling languages and instance models. By definition, a graph rewriting-based model transformation specifies the processing of graph-based models.
- Graph rewriting-based model transformations are based on the background of algebraic graph rewriting and graph grammars, which provide a strong formal basis for formal analysis methods.
- A graph rewriting-based model transformation has a special structure, because it is built from operational primitives called rewriting rules. This structure makes it possible to define transformations at the same abstraction level as the models and even provide domain-specific concrete syntax for their visualization. In this case, domain experts may be able to understand certain aspects of the processing.

Open Problems

My research focuses on the offline verification methods. Most often, formal verification of model transformations is performed manually or the methods can be applied only to a certain transformation class in specific application domains or for the analysis of only certain types of properties [BH07], therefore, there is an increasing need for automated verification methods and tools. It is a usual approach that the verification of a model transformation is performed by converting the transformation itself into a general purpose formal domain where analysis methods are already available. A typical example is the translation of a transformation into the input of theorem provers, or a special analyzer tools [ABK07]. Therefore, the disadvantage of such methods is that during the analysis of a model transformation, we need to define a mapping, i.e. we translate the requirements to be verified from the current application domain of the transformation to the formal domain where automated analysis methods are available. The requirements can be complex and may be domain-specific properties that are hard to be interpreted in another, general domain. In the opposite direction, when an error is recognized in the general formal domain, it should be also interpreted in the application domain. It can be seen that, because the mapping is not always symmetric between the two domains, it is hard to provide the mapping of the domain specific properties and the found problems of the analysis domain. Based on this discussion, the main challenges related to the offline verification of graph rewriting-based model transformations can be concluded as follows.

- However, there exist several methods to translate the problem domain into a general formal analysis domain, it would be beneficial to provide formalism where the properties to be verified can be directly expressed, hence, symmetry between the application domain and the analysis domain does not need to be maintained by translating the properties.
- Moreover, it would be advantageous to provide a coherent formalism for the description of the rewriting rules such that it could be easily checked if a rule guarantees the satisfaction of many types of functional properties.
- Similarly, we would need a formalism to describe the control mechanism of the rewriting rules such that the properties proved to be true by individual rules could be easily propagated through the control structure. In this way, we could derive properties that are satisfied when the complete transformation terminates.
- Since the formal verification of graph rewriting-based model transformations is an undecidable problem, it is challenging to find decidable subsets of the problem. Obviously, this can be reached by restricting the possible transformations or the properties to be analyzed. However, the real challenge is to find a subset that can solve engineering problems.
- It would largely increase the efficiency of the analysis methods to find recurring problems that are usually need to be solved during the verification of transformations. These patterns could

be used and their results could be applied automatically.
- It is also an important open question if certain problems that are undecidable by the automated methods could be specified as patterns that could be analyzed in advance.

Goals of My Research

This work focuses on the offline analysis of algebraic graph rewriting-based model transformations. The goal of my research is to support the automated verification of model transformations by formal methods and to integrate these methods into an existing modeling and model transformation framework. In more details, the goals of my research are as follows:

- To provide a general formal background for the description of model transformation. This makes it possible to provide platform independent analysis methods that are not restricted to specific domains, transformation classes or properties.
- To provide a formal language that is able to express functional properties to be verified. I want this language to be extendable later by new types of properties, hence, the analysis methods could be improved in the future.
- To provide algorithms that support the automation of recurring tasks during the analysis of model transformation definitions. The goal of these algorithms is to leverage the efforts of the manual analysis.
- To provide algorithms for the formal, offline analysis of rewriting rules and control mechanisms.
- Moreover, these methods should be able to perform the analysis using the presented formal background. This would make it possible to express the properties to be verified as well as the found errors in a single formal domain that is specific to the current application domain.
- Since the automated verification of all properties is not possible in general, my methods must be designed to be able to efficiently make use of the knowledge of domain experts by using it during the application of automated methods. To exploit this knowledge, I wanted to provide a set of well-specified techniques and design guidelines that makes it possible for the developer to design a model transformation that can be more efficiently analyzed.
- To provide an implementation of the theoretical analysis framework in an existing modeling and model transformation tool.

2 Methodological Summary

The open issues outlined in the previous section have defined the direction of my research. To develop a suitable formalism, I have studied several approaches of model-based development and model transformations [KT08, SK03, MCG05]. To provide a coherent formalism for the description of my results, I have investigated the mathematical background of graphs (attributed graphs, typed graphs, labeled graphs) that are used to formalize domain-specific visual languages, metamodels and models [EEPT06].

To formalize the processing of graph-based models, I have investigated the formal background of graph rewriting (graph transformations) and graph grammars [Roz97b]. Category theory [Pie91, BW90] is a well-known mathematical modeling language with a wide area of applications in computer science. Therefore, as it is common in the model transformation community, I used category theory to formulate propositions and carry out proofs related to graph rewriting systems concisely. I have studied different approaches for the formal definition of model transformations. As it has been already outlined, a model transformation consists of several rewriting rules and an additional control mechanism that specifies the scheduling of the rules. I have investigated different methods for the specification and the application of the rules, the most widely used approaches are the Single

Pushout (SPO) and Double Pushout (DPO) approaches. Moreover, I have investigated several control mechanisms (priority-based ordering, layering, explicit control flow graphs) [SV09].

To perform verification of graph rewriting-based model transformations in practice, I have investigated several existing model transformation frameworks, where analysis of the transformations is supported on the tool level [dLV02, Tae04, Agr03, CHM+02, KNNZ99]. I have also investigated formal analysis methods and case studies for the verification of specific model transformations [ALS+12, LBA10, Pen09]. Although, my work concentrates on the verification of functional properties, I have also investigated work related to the formal analysis of non-functional properties such as confluence and termination [LPE06, EEPT06, EEdL+05]. From the experience distilled from them, I have developed my own verification framework, and developed a logic-based language that is able to express properties to be verified. and a calculus to reason about these properties. To develop this calculus, provide inference rules, and analyze the soundness and completeness, I have investigated mathematical logic [HR04, BA93], especially propositional logic, and non-canonical logic systems based on propositional logic.

At the Department of Automation and Applied Informatics of BUTE, we have developed Visual Modeling and Transformation System (VMTS) [VMT10] that is a multi-level modeling and model transformation framework. During my research, I have realized my theoretical framework in VMTS. The implemented framework supports semi-automated verification of model transformations.

During the development of my methods, I have used an iterative and incremental approach. I have tested my concepts during the verification of several model transformation case studies and used the experience distilled from them to improve my framework.

3 Novel Scientific Results

The scientific results of my research are summarized in four theses that are further divided into subtheses. I have proved the theoretical results by mathematical methods and illustrated them on model transformation case studies. The theses are outlined in the following. Their structure and the correspondences between them are illustrated in Figure 1.

I. The first contribution provides a uniform formal background that makes formal analysis of model transformations possible. It formalizes primary artifacts used to specify modeling languages and models based on typed graphs. I have introduced three important categories of category theory, which makes it possible to use them for the description of graph rewriting systems. A method for the specification and analysis of attribute constraints of models is also detailed.

II. The second contribution focuses on the definition of a propositional logic-based language that can express properties of the transformations to be verified. I have specified the syntax and the semantics of the language. To perform deduction on the expressions, I have provided sound inference rules. I have also analyzed the completeness of the inference logic. The second contribution also presents an algorithm for the implementation of the inference logic, and analyzes the capabilities of this algorithm.

III. The third contribution builds on the first two theses. It provides a method to describe the declarative interface of rewriting rules and their applications on models. It also provides a formal method to analyze individual rewriting rules of model transformations. Moreover, I have introduced the definition and formalized several instances of Model Transformation Analysis (MTA) methods. MTA techniques are the formal description of recurring tasks performed during the analysis phases as well. MTA design patterns serve as pre-analyzed building blocks that can be used in the implementation and analysis of model transformations.

IV. The fourth contribution focuses on the realization of the theoretical framework. I have developed a textual programming language (MTV) to write programs that represent model transformations according to the formalism presented in the first three theses. I have implemented a verification

framework that is able to process such programs and analyze the specified transformations. I have developed algorithms that can generate such program code automatically from the model transformations implemented in VMTS. Moreover, the framework is ready to work together with other model transformation tools as well. I have also demonstrated the practical relevance of my methods on the verification of a case study.

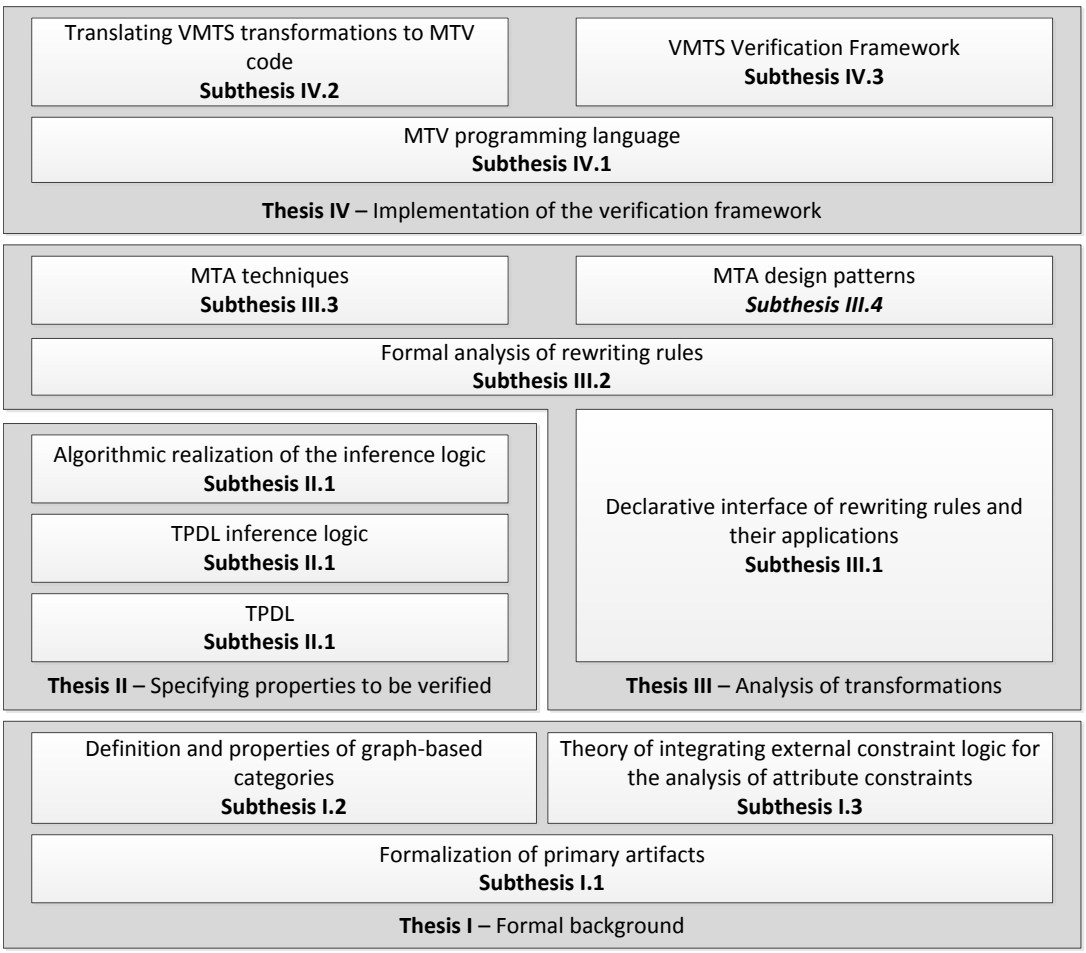

Figure 1: Outline of the theses

Thesis I – Formal Background for the Analysis of Model Transformations

This thesis provides a formal framework based on typed graphs according to [EEPT06]. In this thesis, I present a method to describe modeling languages, models, patterns of models and relations between these entities. Informally, the main definitions are as follows:

- A *metamodel interface* is an abstraction of a metamodel that is used to specify the language of instance models. A metamodel interface is a type graph with inheritance such that for each node and edge, a set of attribute names are assigned.
- An *instance graph* of a metamodel interface is a graph typed over the type graph of the metamodel interface. In an instance graph, values can be specified for each attribute. Such an *attribute value assignment* is formalized as a function that assigns a value for each pair of element and attribute.
- An *instance model* of a metamodel interface is simply an instance graph with an attached attribute value assignment.

- *Abstract attribute constraints* are logical functions defined over the attributes of the elements of a concrete graph. A *pattern* of a metamodel interface is simply an instance graph with an attached set of attribute constraints defined over this graph.
- A *weakly typed morphism* is a traditional graph homomorphism defined between two graphs of a metamodel interface, but during the definition of the mapping types of the elements and inheritance between the node types are taken into account. If a weakly typed morphism is injective and total, it is called a *pattern morphism*.
- An in-place model transformation processes a single model, i.e. given an input model, during the transformation, elements are removed, new elements are added and attributes of the elements may be modified. The output model is this modified input model. Therefore, to specify possible pairs of input-output (source-target) models, we use two graphs and a partial morphism between them. This morphism selects the elements that have been preserved during the transformation. To formalize these source-target relations, I introduce the definition of *relation graphs*, *relation models* and *relation patterns*.
- *Relation pattern morphisms* are morphisms defined between relation graphs.

To be able to use these formal artifacts to describe graph rewriting systems, three new categories of category theory are introduced, and I prove several desirable properties of them.

- Category $\mathbf{GraphsWM}_T$ consists of instance graphs of a typed graph T as objects and weakly typed morphisms as morphisms.
- Category $\mathbf{GraphsPM}_{\mathfrak{M}}$ consists of instance graphs of a metamodel interface $\mathfrak{M}$ as objects and pattern morphisms as morphisms.
- Category $\mathbf{RelGraphsPM}_{\mathfrak{M}}$ consists of relation graphs of a metamodel interface $\mathfrak{M}$ as objects and relation pattern morphisms as morphisms.

Constraints defined on the attributes are handled in an abstract way. I have defined two abstract relations of attribute constraints.

- Two constraints are *conflicting* if it is not possible to satisfy both constraints at once.
- A constraint is *derivable* from another if the satisfaction of the second constraint implies that of the first.

These relations can be analyzed and may be determined by external constraint logic systems that are outside the boundaries of our core framework. To do so, I have defined the interface of functions that can analyze these relations. I have defined restrictions on the constraints, which makes it possible to propagate their analysis to general constraint logic systems. This is advantageous, because constraint logic programming and constraint satisfaction analysis has its own research field and, hence, we are able to integrate its methods.

Thesis I is presented in Chapter 4 of the dissertation.
Publications related to this thesis are: [5, 23, 4, 3, 18, 36].

Subthesis I.1 *I have provided a formalism that is able to describe artifacts (including metamodels, models, patterns of models and relations between these entities) used during the formal analysis of model transformations, I have proved that models can be equivalently described by patterns. I have shown that an attribute value assignment of a source graph can be mapped to a target graph along a pattern morphism, and I have proved that the mapped attribute value assignment is compatible with the target graph, i.e. it is valid attribute value assignment of the target graph. I have also shown that an abstract attribute constraint of a source graph can be mapped to a target graph along a pattern morphism, and I have proved that the mapped constraint is compatible with the target graph. I have defined the mapping of attribute value assignments along pattern morphisms in the reverse direction, and I have proved that the mapping of a complete attribute value assignment of a target graph along a weakly typed morphism in the reverse direction results in a valid complete attribute value assignment of a source graph.*

Subthesis I.2 *I have proved that* $\mathbf{GraphsWM_T}$ *is a valid category in category theory. Similarly, I have proved that the set of pattern morphisms is closed under composition, and, hence,* $\mathbf{GraphsPM_{\mathfrak{M}}}$ *is also a valid category in category theory. I have proved that the category* $\mathbf{GraphsPM_{\mathfrak{M}}}$ *satisfies the following three properties: (i) the pushout can be always constructed along pattern morphisms, (ii) given two pattern morphisms with a common source, if one of them is strongly typed, then the opposite pattern morphism will be also strongly typed in the resulting pushout construction, (iii) given a pair of pattern morphisms with a common target graph, a minimal pair-factorization can be always constructed. I have proved that* $\mathbf{RelGraphsPM_{\mathfrak{M}}}$ *is a valid category in category theory.*

Subthesis I.3 *I have defined relations of sets of abstract attribute constraints. I have provided sufficient conditions to derive the properties of patterns and pattern morphisms from these relations. I have defined the interface of functions that are used to analyze the relations of constraints, and, hence, I have shown that external constraint logic that is used to determine the validity of the abstract attribute constraints can be integrated into my framework. I have proved that different implementations of the presented function interfaces cannot contradict to each other, i.e. the requirements of the functions guarantees consistency.*

Thesis II – Formalizing Functional Properties of Model Transformations

In this thesis, a propositional logic-based language called Transformation Property Description Language (TPDL) is defined that can express a set of functional properties of transformations to be verified. The atomic expressions of TPDL have been defined by their interface: an atomic expression is specified by a logical function in the domain of relation models. This makes it possible to extend this language later with the capability to describe more types of properties. Based on this interface, I have introduced the definition of a concrete type of atomic TPDL expression called *relation pattern condition.*

To perform reasoning on the expressions of TPDL, inference rules are introduced and their soundness is analyzed. These rules make it possible to analyze implications, i.e. prove that a TPDL expression is a semantic entailment of another expression.

I have provided the analysis of the decidability of the problem of determining if an implication is always satisfied. I have showed that the problem is undecidable in general, however, I have presented a decidable subset of the problem domain.

An important contribution of Thesis II is a method to realize the reasoning in the form of an algorithm. Since the problem is undecidable in general, it is important to specify a limit for the execution of the algorithm. However, it is reasonable to require that this limit make it possible for the algorithm to solve the problems in the subset of decidable implications. The reasoning algorithm is presented, its complexity is analyzed, and an appropriate limit function is provided.

Thesis II is presented in Chapter 5 of the dissertation.
Publications related to this thesis are: [5, 23, 22, 4, 2, 19, 16, 17, 10].

Subthesis II.1 *I have provided the interface of a propositional logic-based formal language (TPDL) that is able to express several functional properties of model transformations including correspondences between the source and target models. I have provided the syntax and the semantics of a concrete type of TPDL atom called relation pattern condition that are built from relation patterns. I have proved that equivalent relation pattern conditions can be specified by means of isomorphic relation patterns used in the expressions. I have proved that according to the semantics of TPDL, the inference rules defined for propositional logic are valid inference rules in TPDL inference logic.*

Subthesis II.2 *I have provided a set of inference rules for TPDL expressions and I have proved that these inference rules are sound. I have provided sufficient conditions for the applicability of the inference rules and provided algorithms for the decidability of these conditions. I have proved that it is undecidable in general if an implication is always satisfied. I have proved that there is a subset of the implications for which the previous property is algorithmically decidable.*

Subthesis II.3 *I have provided the definition of limit functions and introduced a valid limit function. I have provided an algorithm for analyzing the satisfiability of implications built from TPDL formulae. I have proved that the algorithm always terminates if a valid limit function is provided for any input. I have proved that the algorithm always retrieves a solution for the algorithmically decidable subset of the problem domain presented in Subthesis II.2 when the previously introduced limit function is used.*

Thesis III – Formal Analysis of Transformation Segments

Thesis III focuses on (i) the verification of the properties of individual rewriting rules, (ii) the verification of complex transformation segments, and (iii) the propagation of results of their analysis through the whole control structure of the model transformations, hence, deriving the properties that can be verified not only for these segments, but for the complete transformation as well.

I provide a method to specify the declarative interface of rewriting rules and the interface of the application of a rewriting rule. Based on these interfaces, it is possible to analyze individual rewriting rules of model transformations. By the analysis of a rewriting rule, my goal is to prove properties that are true after the application of a rule independently from the concrete input.

I have formalized a directed graph-based control mechanism to specify the execution order of the rules. This specification is used to propagate the properties that are proved to be true after the application of the single rewriting rules through the control flow of the transformations. By this propagation, the properties that will be true when the complete transformation finishes can be derived. These properties are described by a TPDL formula called *final formula*. If a formula to be verified can be inferred from the final formula, then the property can be proved to be satisfied by all possible outputs. If the opposite of the formula to be verified can be inferred, then it is proved that no output can satisfy the property. Otherwise, the verification framework cannot state anything about the satisfaction of the property.

Another contribution of this thesis is the definition of the concept of two types of *MTA* (Model Transformation Analysis) *methods*.

- There are recurring techniques that are usually applied intuitively in the formal analysis of model transformations. An *MTA technique* is a precise, formal documentation of such a method which cannot be expressed as a partial transformation. We have multiple intentions with defining an *MTA technique*: (i) The current intuitive method should be known by others. (ii) The formal definition of a technique makes the automation of the technique possible in several cases, which supports the analysis process. I introduce the *Intact Element* MTA technique that is used to identify the rules of a transformation that leaves certain elements intact. The automation of this technique may help the developer decrease the number of rules that need to be analyzed. I also introduce the *Composite Rule* MTA technique that formalizes how two sequential rewriting rules can be composed into a single rule. There are many benefits of the automation of this method. For example, it is easier to analyze a single rule, then multiple sequential rules.
- *MTA design patterns* are similar to traditional design patterns [GHJV95] used in object-oriented systems. An MTA design pattern expresses a more or less complex construction that describes how to implement a concrete functionality in a transformation. In other words, we define a partial model transformation that solves a general recurring transformation task. By customizing a design pattern, it can be applied in a concrete transformation, where a variant of the documented

situation occurs. An important feature of MTA design patterns is that the defined constructions (partial transformations) can be analyzed separately, and the analysis could be attached to the documentation. This independent analysis can be reused in the analysis of the concrete model transformation, where the design pattern is used. In other words, the application of a design pattern results that certain properties automatically hold for the related part of the concrete transformation. This method facilitates the analysis of complex transformations and also supports automation. In this thesis, I introduce the *Traverser* MTA design pattern that specifies to iteratively process all instances of a certain pattern.

Thesis III is presented in Chapter 6 and 7 of the dissertation.
Publications related to this thesis are: [5, 23, 21, 22, 4, 27, 1, 9, 12, 2, 3, 7, 8, 34, 33, 26, 14].

Subthesis III.1 *I have provided a formalism to declaratively specify the interface of rewriting rules based on the category* $\mathbf{GraphsPM}_{\mathfrak{M}}$. *I have specified the declarative interface of a direct graph transformation that is based on the interface of rewriting rule interfaces. I have provided a sufficient condition for the applicability of a rewriting rule based on its interface. I have provided a sufficient condition for the non-applicability of a rewriting rule based on its interface.*

Subthesis III.2 *I have provided a formalism to specify control flow graphs of model transformations. I have provided sufficient conditions to derive formulae that are proved to be true after the successful application of rules based on their interface. I have provided sufficient conditions to derive formulae that are proved to be true after the unsuccessful application of rules based on their interface. I have shown how the previous sufficient conditions can be algorithmically analyzed.*

Subthesis III.3 *I have introduced MTA (Model Transformation Analysis) techniques that are used to automate recurring tasks during the formal analysis of graph rewriting-based model transformations formalized by the formal framework presented in Thesis I. I have introduced and formalized the Intact Element MTA technique. I have given a sufficient condition for the detection of weakly and strongly intact elements of a rewriting rule interface. I have provided the interface of the composition of sequential rules and the interface of the application of the composed rule. Based on this definition, I have presented the Composite Rule MTA technique. According to this technique, I have adapted concurrency theorem to the formalism presented in Thesis I. and proved the validity of the adapted theorem.*

Subthesis III.4 *I have introduced and formalized the Traverser MTA design pattern that is a generally described transformation segment with an attached formal analysis for the iterative processing of a set of elements. I have formally described the structure of the design pattern. By the formal analysis of this design pattern, I have given sufficient conditions for the termination of the transformation segment. I have also showed how the execution time can be computed in the case when the sufficient conditions hold. I have proved a TPDL formula that can be proved to be true after the application of the transformation segment. I have proved that for any model transformation that implemented this design pattern, the results of the analysis automatically hold.*

Thesis IV. Application of the Novel Scientific Results

The theoretical framework presented in my thesis has been realized in Visual Modeling and Transformation System (VMTS), a multi-level modeling and model transformation framework. During its implementation, my goal was to illustrate the applicability of the theoretical results. The tool can be publicly accessed at `http://vmts.aut.bme.hu`.

MTV Language

I have developed a textual programming language (MTV – Model Transformation Verification) whose constructs conform to the formal artifacts presented in Subthesis I.1. More precisely, with MTV, we are able to specify: metamodel interfaces, (relation) graphs, attribute constraints, (relation) patterns, (relation) pattern morphisms, rewriting rule interfaces, control flows, and TPDL formulae. In other words, given a metamodel, a model transformation and verifiable properties, we can write a program in MTV that conforms to the formalism presented in the previous chapters. My verification framework reads such program code and performs the analysis of the model transformations specified in it.

The MTV programming language is the basis of my framework, because the input for the verification is provided written as MTV code. This means that my verification framework is not restricted to the analysis of model transformations implemented in VMTS, any model transformation tool can translate its transformations into MTV, which can be processed by my system.

VMTS Verification Framework

In the following, the implementation of the framework is presented in VMTS. On the top of Figure 2, the workflow of the analysis of a model transformation is depicted. Given a model transformation definition, it first needs to be converted to MTV code. One component of the framework is able to automatically generate the MTV code for a model transformations defined in VMTS. It parses the constraint and imperative code as well and recognizes certain constructions in them. Of course, the developer can modify or extend the generated code any time.

The implemented verification framework is able to parse MTV code, recognize the model transformation definitions and it generates a custom user interface where the current assignment can be seen. The user can choose the rewriting rule to be analyzed, or is able to manually attach formulae to any of the edges, hence, during the next rule analysis, this information can be taken into account. After the manual refinement of the state of assignments, the final formula is computed and checked if the verifiable formula (provided by the user) can be inferred. At the bottom of Figure 2, the components of the realized framework are presented.

The model transformation framework and the analysis framework are two different components. They only communicate to each other using MTV program code. Therefore, it is not obligatory that the MTV code that is processed must come from an existing VMTS model transformation. Other model transformation frameworks can also translate their transformation definitions into MTV programs and provide it for the framework.

Tool Support for MTA Techniques and Design Patterns

VMTS can be extended by addons. These separately developed components can use and extend the services of the core VMTS framework. This makes it possible to provide new functionality easily and make these functions available from the user interface. The API of VMTS provides access to both the VMTS models and the in-memory representation of the already parsed MTV code.

In VMTS, I have developed a separate addon for each MTA technique presented above. I have also implemented an addon for the *Traverser* MTA design pattern. By this addon, the user can select a transformation segment and check if it meets certain requirements of the design pattern. Moreover, the verified properties of the design pattern can be used in the analysis of the current model transformation.

Applications of the scientific results are presented in Chapter 8 of the dissertation.
Publications related to this thesis are: [15, 24, 25, 27, 32, 20, 28, 30, 31, 12, 35, 13, 11, 6, 29].

Subthesis IV.1 *I have designed a tool-independent textual programming language called MTV that*

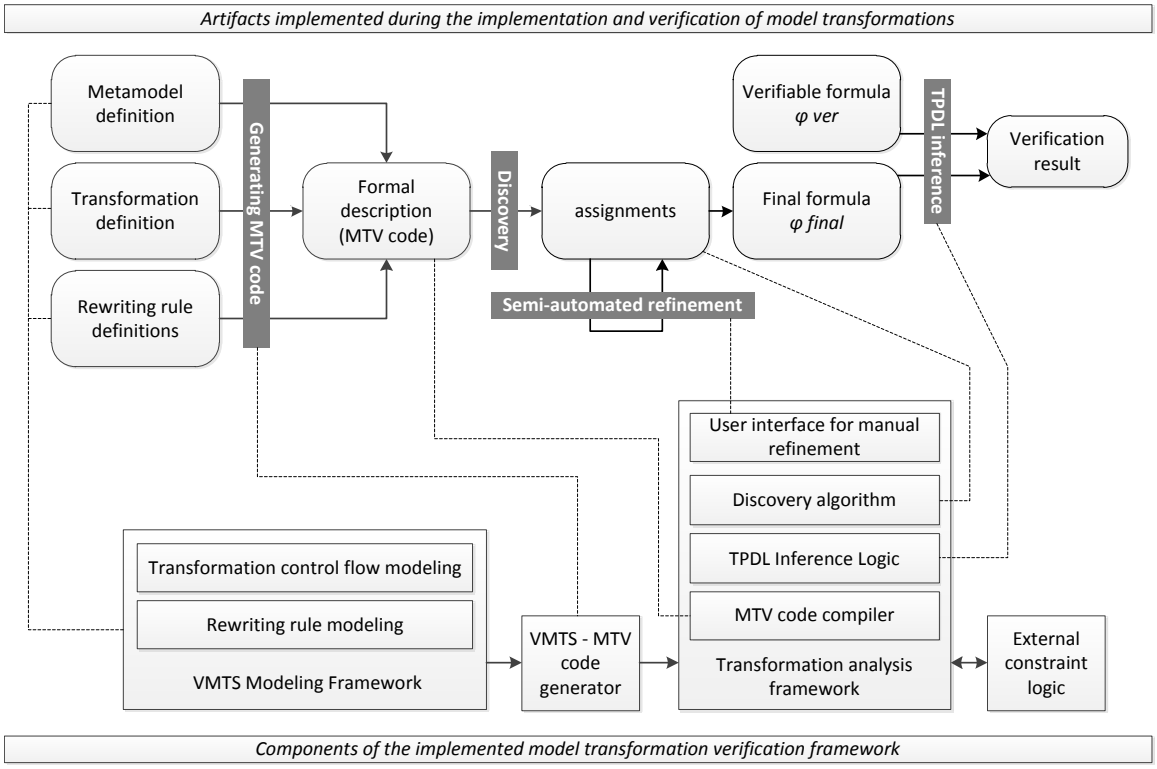

Figure 2: Overview of the model transformation verification framework

is able to specify model transformations with the formalism presented in Thesis I. I have shown that typical engineering verification problems can be formulated as MTV programs. Also, I have demonstrated that by using MTV, the transformations to be analyzed by the implemented framework are not restricted to transformations implemented in VMTS.

Subthesis IV.2 *I have provided algorithms that are able to process model transformations defined in VMTS and automatically generate MTV code from them ready for analysis. I have shown that the verification framework can be implemented such that it is able to analyze control flow definitions.*

Subthesis IV.3 *I have demonstrated that my theoretical framework can be realized as a component-based reusable, and extensible software package that can reason about an arbitrary MTV input, and is able to compose predefined MTA patterns, and integrate them into the reasoning process.*

4 Related Publications

Journals

[1] **Márk Asztalos**, István Madari, and László Lengyel. Towards formal analysis of multi-paradigm model transformations. *Simulation*, 86(7):429–452, 2010. **I.F.: 0.611**

[2] **Márk Asztalos**, Péter Ekler, László Lengyel, Tihamér Levendovszky, István Madari, and Tamás Vajk. Automated formal verification of graph rewriting-based model transformations. *Buletinul Stiintific Al Universitatii Politehnica Din Timisoara-Seria Automatica Si Calculatoare*, 55(69):175–184, 2010.

[3] **Márk Asztalos**, Péter Ekler, László Lengyel, Tihamér Levendovszky, Tamás Mészáros, and Gergely Mezei. Automated verification by declarative description of graph rewriting-based model transformations. *Electronic Communications Of The EASST*, 42:12, 2010.

[4] **Márk Asztalos**, László Lengyel, and Tihamér Levendovszky. Toward automated verification of model transformations: A case study of analysis of refactoring business process models. *Electronic Communications of the EASST*, 21, 2009.

[5] **Márk Asztalos**, László Lengyel, and Tihamér Levendovszky. A formal framework for automated verification of model transformations. *Software Testing Verification & Reliability*, 2012. (submitted).

[6] **Márk Asztalos**, István Madari, Tamás Mészaros, Tamás Vajk, and Gergely Mezei. Szakterület-specifikus modellezés. *Híradástechnika*, LXV(5-6):25–30, 2010.

[7] **Márk Asztalos**, Eugene Syriani, Manuel Wimmer, and Marouane Kessentini. Simplifying model transformation chains by rule composition. *Lecture Notes in Computer Science*, 6627:293–307, 2011. Proceedings of the 2010 international conference on Models in software engineering.

[8] **Márk Asztalos**, Eugene Syriani, Manuel Wimmer, and Marouane Kessentini. Towards transformation rule composition. *Electronic Communications of the EASST*, 42:13, October 2011. 4th International Workshop on Multi-Paradigm Modeling - MPM'10.

[9] **Márk Asztalos**, Péter Ekler, László Lengyel, and Tihamér Levendovszky. Verification of model transformations to refactoring mobile social networks. *Electronic Communications of the EASST*, 32:12, 2010.

[10] Péter Ekler and **Márk Asztalos**. Performance analysis of maintaining mobile-based social network models. *WSEAS Transactions on Computers*, 9(12):1391–1400, dec 2010.

[11] Tamás Mészáros, **Márk Asztalos**, and Gergely Mezei. Overlapped execution of rewriting rules in graph transformation systems. *Buletinul Stiintific Al Universitatii Politehnica Din Timisoara-Seria Automatica Si Calculatoare*, 55(3):143–150, 2010.

[12] Tamás Mészáros, Gergely Mezei, Tihamér Levendovszky, and **Márk Asztalos**. Manual and automated performance optimization of model transformation systems. *International Journal of Software Tools for Technology Transfer*, 12(3-4):231–243, July 2010.

[13] Tamás Vajk, **Márk Asztalos**, and Gergely Mezei. Formal analysis of incremental code generation in ocl compilers. *Buletinul Stiintific Al Universitatii Politehnica Din Timisoara-Seria Automatica Si Calculatoare*, 55(69):117–122, 2010.

[14] Dániel Varró, **Márk Asztalos**, Dénes Bisztray, Artur Boronat, Duc-Hanh Dang, Rubino Geiss, Joel Greenyer, Pieter Van Gorp, Ole Kniemeyer, Anantha Narayanan, Edgars Rencis, and Erhard Weinell. Transformation of UML Models to CSP: A Case Study for Graph Transformation Tools. In Andy Schürr, Manfred Nagl, and Albert Zündorf, editors, *Applications of Graph Transformations with Industrial Relevance*, volume 5088 of *Lecture Notes in Computer Science*, pages 540–565. Springer Berlin / Heidelberg, 2008.

Publications in International Conference Proceedings

[15] László Angyal, **Márk Asztalos**, László Lengyel, Tihamér Levendovszky, István Madari, Gergely Mezei, Tamás Mészaros, László Siroki, and Tamás Vajk. Towards a fast, efficient and customizable

domain-specific modeling framework. In *Proceedings of the IASTED International Conference*, pages 11–16, Innsbruck, Austria, February 2009. ACTA Press.

[16] **Márk Asztalos**. Comparison of termination criteria for graph transformation systems. In Vajk István, editor, *Proceedings of the Automation and Applied Computer Science Workshop: AACS'07*, Budapest, Hungary, 2007.

[17] **Márk Asztalos**. Offline analysis of the properties of transformation UML to CSP. In Vajk István, editor, *Proceedings of the Automation and Applied Computer Science Workshop: AACS'08*, pages 259–271, Budapest, Hungary, 2008. ISBN: 978-963-420-955-3.

[18] **Márk Asztalos**, Péter Ekler, László Lengyel, Tihamér Levendovszky, and Tamás Mészáros. Formalizing models with abstract attribute constraints. In *Third International Workshop on Graph Computation Models*, pages 65–80, Enschede, Hollandia, 2010.

[19] **Márk Asztalos**, Péter Ekler, László Lengyel, Tihamér Levendovszky, and Tamás Vajk. MCDL: A language for specifying graph conditions with attribute constraints. In *Proceedings of Workshop on Model-Driven Engineering, Verification, and Validation*, pages 43–48, Oslo, Norway, 2010.

[20] **Márk Asztalos** and László Lengyel. A metamodel-based matching algorithm for model transformations. In *6th IEEE International Conference on Computational Cybernetics (ICCC 2008)*, pages 151–155, Stara Lesna, Slovakia, November 2008.

[21] **Márk Asztalos**, László Lengyel, and Tihamér Levendovszky. A formalism for automated verification of model transformations. In *International Symposium of Hungarian Researchers on Computational Intelligence and Informatics (CINTI)*, pages 377–390, Budapest, Hungary, November 2009.

[22] **Márk Asztalos**, László Lengyel, and Tihamér Levendovszky. A formalism for describing modeling transformations for verification. In *MoDeVVa'09 : Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation*, pages 1–10, New York, NY, USA, 2009. ACM.

[23] **Márk Asztalos**, László Lengyel, and Tihamér Levendovszky. Towards automated, formal verification of model transformations. In *Third International Conference on Software Testing, Verification and Validation*, pages 15 – 24, Paris, France, April 2010. IEEE.

[24] **Márk Asztalos**, László Lengyel, Tihamér Levendovszky, and Hassan Charaf. Graph transformation contest - UML to CSP transformation. In *Applications of Graph Transformation 2007 (AGTIVE) - Graph Transformation Tool Contest*, page 5, Kassel, Germany, 2007.

[25] **Márk Asztalos**, László Lengyel, Tihamér Levendovszky, and Hassan Charaf. Termination analysis of the transformation UML to CSP. In *Inproceedings of Computational Intelligence and Informatics 8th International Symposium of Hungarian Researchers*, pages 611–622, Budapest, Hungary, November 2007.

[26] **Márk Asztalos** and István Madari. VTL: an improved model transformation language. In Vajk István and Iváncsy Renáta, editors, *Proceedings Automation and Applied Computer Science Workshop 2009: AACS'09*, pages 185–195, Budapest, Hungary, 2009. ISBN: 978-963-420-978-2.

[27] **Márk Asztalos**, István Madari, Tamás Vajk, László Lengyel, and Tihamér Levendovszky. Formal verification of model transformations : An automated framework. In *2010 IEEE International Joint Conference on Computational Cybernetics and Technical Informatics (ICCC-CONTI)*, pages 493–498, Timisoara, Romania, may 2010.

[28] **Márk Asztalos**, Tamás Mészáros, and László Lengyel. Generating executable BPEL code from BPMN models. In *Grabats 2009 5th International Workshop on Graph-Based Tools – graph transformation tool contest*, page 5, Zürich, Switzerland, 2009.

[29] **Márk Asztalos** and Tamás Vajk. Automated offline verification of graph rewriting-based model transformations. In Vajk István and Iváncsy Renáta, editors, *Proceedings of the Automation and Applied Computer Science Workshop 2010 (AACS'10)*, Budapest, Hungary, 2010. Department of Automation and Applied Informatics, ISBN: 978-963-313-004-9.

[30] László Lengyel, István Madari, **Márk Asztalos**, and Tihamér Levendovszky. Validating query/view/transformation relations. In *Proceedings of Workshop on Model-Driven Engineering, Verification, and Validation*, pages 7–12, Oslo, Norway, 2010.

[31] Tihamér Levendovszky, Tamás Mészáros, Péter Ekler, and **Márk Asztalos**. DSML-aided develepement for mobile P2P systems. In Matti Rossi, Jonathan Sprinkle, Jeff Gray, and Juha-Pekka Tolvanen, editors, *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09)*, pages 51–56, Orlando, USA, 2009. HSE Print.

[32] István Madari, **Márk Asztalos**, Tamás Mészáros, László Lengyel, and Hassan Charaf. Implementing QVT in a domain-specific modeling framework. In *5th International Conference on Software and Data Technologies (ICSOFT)*, pages 304–307, Athens, Greece, July 2010.

[33] Tamás Mészáros, **Márk Asztalos**, and Gergely Mezei. An overlapped execution technique for graph rewriting rules. In *IEEE International Joint Conferences on Computational Cybernetics and Technical Informatics (ICCC-CONTI 2010)*, pages 281–286, 2010.

[34] Tamás Mészáros, **Márk Asztalos**, Gergely Mezei, and Hassan Charaf. Performance optimization of exhaustive rules in graph rewriting systems. In *ICSOFT 2010 - Proceedings of the 5th International Conference on Software and Data Technologies*, volume 2, pages 292–295, 2010.

[35] Tamás Vajk and **Márk Asztalos**. Incremental code generation in Object Constraint Language compilers. In Vajk István and Iváncsy Renáta, editors, *Proceedings of the Automation and Applied Computer Science Workshop 2010 (AACS'10)*, pages 71–84, Budapest, Hungary, 2010. ISBN: 978-963-313-004-9.

[36] Tamás Vajk, Zoltán Dávid, **Márk Asztalos**, Gergely Mezei, and Tihamér Levendovszky. Runtime model validation with parallel object constraint language. In *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation*, pages 1–7, Wellington, New-Zealand, 2011. ACM.

5 Citations

[1] is cited by:

Eugene Syriani, Jeff Gray. Challenges for Addressing Quality Factors in Model Transformation: In 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), pp. 929-937.

[25] is cited by:

Louis M. Rose, Markus Herrmannsdoerfer, Steffen Mazanek, Pieter Van Gorp, Sebastian Buchwald, Tassilo Horn, Elina Kalnina, Andreas Koch, Kevin Lano and Bernhard Schätz, Manuel Wimmer. Graph and model transformation tools for model migration: Graph and model transformation tools for model migration Software and Systems Modeling, Springer Berlin / Heidelberg, , 1-37, 2012

[14] is cited by:

Federico Banti, Rosario Pugliese Francesco Tiezzi1. An accessible verification environment for UML models of services, Journal of Symbolic Computation, Volume 46, Issue 2, February 2011, Pages 119-149, ISSN 0747-7171

James Sharp, Helen Treharne. CASE Technical Report: Automated Transformations from VHDL to CSP, 2011.

Kawtar Benghazi, Luis Garrido, Manuel Noguera, Maria V. Hurtado, Lawrence Chung. Extending and Formalizing UML 2.0 Activity Diagrams for the specification of time-constrained business processes. Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on , vol., no., pp.93-100, 19-21 May 2010

Dang, D.-H., Gogolla, M. Precise model-driven transformations based on graphs and metamodels (2009) SEFM 2009 - 7th IEEE International Conference on Software Engineering and Formal Methods, art. no. 5368074, pp. 307-316.

[22] is cited by:

Lasalle Jonathan, Bouquet Fabrice, Legeard Bruno, Peureux Fabien. SysML to UML model transformation for test generation purpose, ACM SIGSOFT Software Engineering Notes, Volume 36 Issue 1, January 2011 Pages 1-8, 2011

Horacio López, Fernando Varesi, Marcelo Vinolo, Daniel Calegari, Carlos Luna. Estado del arte de verificación de transformación de modelos Reporte Técnico RT 10-07 Facultad de Ingeniería, Universidad de la República Montevideo, Uruguay, 2010.

Calegari Daniel, Szasz Nora. Verificacioi de transformaciones de modelos. Una Revision del Estado del Arte (Version Extendida), Reporte Tecnico RT 12-05, Instituto de Computacion - Facultad de Ingenieria, Universidad de la Republica Montevideo, Uruguay, ISSN 0797-6410, 2012

[28] is cited by:

Espinosa Enrique D, Frausto Juan, Rivera Ernesto J. Markov Decision Processes for Optimizing Human Workflows, Service Science 2: (4) pages 245-269., 2010.

[27] is cited by:

Calegari Daniel, Szasz Nora. Verificaciói de transformaciones de modelos. Una Revisión del Estado del Arte (Versión Extendida), Reporte Tecnico RT 12-05, Instituto de Computacion - Facultad de Ingenieria, Universidad de la Republica Montevideo, Uruguay, ISSN 0797-6410, 2012

[23] is cited by:

Martin Strecker. Locality in Reasoning about Graph Transformations, International Symposium on Applications of Graph Transformation With Industrial Relevance (AGTIVE), Budapest, Hungary 2011.

Márcio Filipe Caetano Mateus. Measuring Data Transfer in Heterogeneous IoT Environments, Dissertacao para obtencao do Grau de Mestre em Engenharia Electrotécnica e de Computadores, Universidade Nova De Lisboa, 2012.

Moussa Amrani, Levi Lucio, Gehan Selim, Benoit Combemale, Jurgen Dingel, Hans Vangheluwe, Yves Le Traon, James R. Cordy. A Tridimensional Approach for Studying the Formal Verification of Model Transformations, In Proceeding of the IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), Pages 921-928 IEEE Computer Society, ISBN: 978-0-7695-4670-4, 2012.

Hanh Nhi Tran, Christian Percebois. Towards a Rule-Level Verification Framework for Property-Preserving Graph Transformations ICST (International Conference on Software Testing, Verification and Validation) Workshop on Verification and validation Of model Transformations (VOLT), pages 946-953., Montreal, Canada (2012)

Hanna Schölzel, Hartmut Ehrig, Frank Hermann, Christoph Brandt. Propagation of Constraints along Model Transformations Based on Triple Graph Grammars, Electronic Communications of the EASST, 41, (14 pages) 2011.

Léveque, T., Carlson, J., Sentilles, S., Borde, E. Flexible semantic-preserving flattening of hierarchical component models (2011) Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011, art. no. 6068319, pp. 31-38.

Fabian Büttner, Jordi Cabot, Martin Gogolla. On validation of ATL transformation rules by transformation models, In Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation, article no. 9.

Marko Jurisic. Transition between process models (BPMN) and service models (WS-BPEL and other standards): A systematic review, Journal of Information and Organizational Sciences, 35:(2), 163-171. (2011)

Lúcio L, Barroca B, Amaral V. A technique for automatic validation of model transformations, ure Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6394 LNCS:(PART 1), 136-150. (2010)

[12] is cited by:

Amir Hossein Ghamarian, Maarten de Mol, Arend Rensink, Eduardo Zambon and Maria Zimakova. Modelling and analysis using GROOVE, International Journal on Software Tools for Technology Transfer (STTT) Volume 14, Number 1 (2012), 15-40,

Krause, C., Maraikar, Z., Lazovik, A., Arbab, F. Modeling dynamic reconfigurations in Reo using high-level replacement systems (2011) Science of Computer Programming, 76 (1), pp. 23-36.

Tisi, M., Martínez, S., Jouault, F., Cabot, J. Lazy execution of model-to-model transformations (2011) Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6981 LNCS, pp. 32-46.

Louis M. Rose, Markus Herrmannsdoerfer, Steffen Mazanek, Pieter Van Gorp, Sebastian Buchwald, Tassilo Horn, Elina Kalnina, Andreas Koch, Kevin Lano, Bernhard Schätz, et al. Graph and model transformation tools for model migration, Software and Systems Modeling, 2012

Enrico Biermann, Claudia Ermel, Leen Lambers, Ulrike Prange, Olga Runge and Gabriele Taentzer. Introduction to AGG and EMF Tiger by modeling a Conference Scheduling System, International Journal on Software Tools for Technology Transfer July 2010, Volume 12, Issue 3-4, pp 245-261

[30] is cited by:

Pejman Salehi. A Model Based Framework for Service Availability Management, PhD thesis, Concordia University, Montreal, Canada, 2012

6 References

[ABK07] K. Anastasakis, B. Bordbar, and J. M. Küster. Analysis of Model Transformations via Alloy. In *Model Driven Engineering, Verification, and Validation: Integrating Verification and Validation in MDE (MODEVVA07)*, pages 47–56, October 2007.

[Agr03] Aditya Agrawal. Graph Rewriting And Transformation (GReAT): A Solution For The Model Integrated Computing (MIC) Bottleneck. *International Conference on Automated Software Engineering*, 0:364, 2003.

[ALS+12] Moussa Amrani, Levi Lucio, Gehan Selim, Benoit Combemale, Jurgen Dingel, Hans Vangheluwe, Yves Le Traon, and James R. Cordy. A tridimensional approach for studying the formal verification of model transformations. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, ICST '12, pages 921–928, Washington, DC, USA, 2012. IEEE Computer Society.

[BA93] M. Ben-Ari. *Mathematical logic for computer science.* Prentice-Hall International series in computer science. Prentice-Hall International, Hempel Hempstead England, 1993.

[BBG+06] Jean Bézivin, Fabian Büttner, Martin Gogolla, Frederic Jouault, Ivan Kurtev, and Arne Lindow. Model transformations? transformation models! In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *Model Driven Engineering Languages and Systems*, volume 4199 of *Lecture Notes in Computer Science*, pages 440–453. Springer Berlin / Heidelberg, 2006.

[BFG96] Dorothea Blostein, Hoda Fahmy, and Ann Grbavec. Issues in the practical use of graph rewriting. In *5th Workshop on Graph Grammars and Their Application To Computer Science, Lecture Notes in Computer Science*, pages 38–55. Springer, 1996.

[BH07] Dénes Bisztray and Reiko Heckel. Rule-level verification of business process transformations using CSP. *ECEASST*, 6, 2007.

[BW90] Michael Barr and Charles Wells. *Category Theory for Computing Science.* Prentice Hall, 1990.

[CHM+02] György Csertán, Gábor Huszerl, István Majzik, Zsigmond Pap, András Pataricza, and Dániel Varró. VIATRA: Visual Automated Transformations for Formal Verification and validation of UML models. In Julian Richardson, Wolfgang Emmerich, and Dave Wile, editors, *Proc. ASE 2002: 17th IEEE International Conference on Automated Software Engineering*, pages 267–270, Edinburgh, UK, September 23–27 2002. IEEE Press.

[dLV02] Juan de Lara and Hans Vangheluwe. AToM3: A Tool for Multi-formalism and Metamodelling. In *FASE '02: Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, pages 174–188, London, UK, 2002. Springer-Verlag.

[EEdL+05] H. Ehrig, K. Ehrig, J. de Lara, G. Taentzer, D. Varró, and Sz. Varró-Gyapay. Termination criteria for model transformation. *FASE*, pages 49–63, 2005.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*, volume XIV of *Monographs in Theoretical Computer Science. An EATCS Series.* Springer, 2006.

[FS99] Martin Fowler and Kendall Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (2nd Edition).* Addison-Wesley Professional, August 1999.

[GH06] Holger Giese and Stefan Henkler. A survey of approaches for the visual model-driven development of next generation software-intensive systems. *J. Vis. Lang. Comput.*, 17(6):528–550, December 2006.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, Boston, MA, 1995.

[GHN10] Holger Giese, Stephan Hildebrandt, and Stefan Neumann. Model synchronization at work: Keeping sysml and autosar models consistent. In Gregor Engels, Claus Lewerentz, Wilhelm Schäfer, Andy Schürr, and Bernhard Westfechtel, editors, *Graph Transformations and Model-Driven Engineering*, volume 5765 of *Lecture Notes in Computer Science*, pages 555–579. Springer Berlin Heidelberg, 2010.

[Hec06] Reiko Heckel. Graph transformation in a nutshell. *Electron. Notes Theor. Comput. Sci.*, 148(1):187–198, February 2006.

[HR00] D. Harel and B. Rumpe. Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff. Technical report, Jerusalem, Israel, Israel, 2000.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, New York, NY, USA, 2004.

[KN08] Gabor Karsai and Anantha Narayanan. Towards Verification of Model Transformations Via Goal-Directed Certification. pages 67–83. 2008.

[KNNZ99] Thomas Klein, Ulrich A. Nickel, Jörg Niere, and Albert Zündorf. From UML to Java And Back Again. Technical Report tr-ri-00-216, University of Paderborn, Paderborn, Germany, September 1999.

[KT08] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, March 2008.

[Küh06] Thomas Kühne. Matters of (meta-)modeling. *Software and System Modeling*, 5(4):369–385, 2006.

[KWB03] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[LBA10] Levi Lucio, Bruno Barroca, and Vasco Amaral. A technique for automatic validation of model transformations. In *MoDELS (1)*, pages 136–150, 2010.

[Len06] László Lengyel. *Online Validation of Visual Model Transformations*. PhD thesis, Budapest University of Technology and Economics, Department of Automation and Applied Informatics, 2006.

[LPE06] Tihamér Levendovszky, Ulrike Prange, and Hartmut Ehrig. A Termination Criteria for DPO Transformations with Injective Matches. In *Graph Transformation for Verification and Concurrency*, pages 102–116, Bonn, Germany, August 2006.

[MCG05] Tom Mens, Krzysztof Czarnecki, and Pieter Van Gorp. A taxonomy of model transformation. In *Proc. Dagstuhl Seminar on "Language Engineering for Model-Driven Software Development". Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl*. Electronic, 2005.

[Nor99] Gregory G. Nordstrom. *Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments*. PhD thesis, Nashville, United States, 1999.

[Obj07] Object Management Group (OMG). *OMG Unified Modeling Language (OMG UML) Superstructure Specification, V2.1.2*, November 2007.

[Pen09] Karl-Heinz Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, Department of Computing Science, University of Oldenburg, Oldenburg, 2009.

[Pie91] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists*. MIT Press, 1991.

[Plu98] Detlef Plump. Termination of graph rewriting is undecidable. *Fundam. Inf.*, 33(2):201–209, 1998.

[Roz97a] G. Rozenberg. Handbook on graph grammars and computing by graph transformation, foundations, vol.1. *World Scientific,*, 1997.

[Roz97b] Grzegorz Rozenberg. *Handbook on Graph Grammars and Computing by Graph Transformation: Foundations*, volume 1. World Scientific, Singapore, 1997.

[SK97] J. Sztipanovits and G. Karsai. Model-integrated computing. *Computer*, 30(4):110 –111, April 1997.

[SK03] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20(5):42–45, September 2003.

[SV09] Eugene Syriani and Hans Vangheluwe. Matters of model transformation. Technical Report SOCS-TR-2009.2, 2009.

[SVC06] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[Tae04] Gabrielle Taentzer. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In *Application of Graph Transformations with Industrial Relevance (AGTIVE 2004)*, volume 3062 of *LNCS 3062*, pages 446–453. Springer, 2004.

[VMT10] Visual Modeling and Transformation System. `http://vmts.aut.bme.hu`, 2010.

[WL06] Alan Wassyng and Mark Lawford. Software tools for safety-critical software development. *Int. J. Softw. Tools Technol. Transf.*, 8(4):337–354, August 2006.

[WML10] Alan Wassyng, Tom Maibaum, and Mark Lawford. On software certification: we need product-focused approaches. In *Proceedings of the 15th Monterey conference on Foundations of Computer Software: future Trends and Techniques for Development*, Monterey'08, pages 250–274, Berlin, Heidelberg, 2010. Springer-Verlag.