MŰEGYETEM 1782

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

Automated Model Transformations for the Analysis of IT Systems

PhD thesis booklet

Dániel Varró
MSc in technical informatics

Supervisor:
Dr. András Pataricza, PhD
associate professor

Budapest, December 2003

1. Preliminaries and Objectives of the Research

For the past decades, the main trend in software engineering has been almost exclusively dominated by the object-oriented modeling and programming paradigm. However, the first object-oriented programming languages and design methods have been specific to the peculiarities of software development teams, thus many of such languages with slight (or major) differences existed in parallel. Due to the increasing complexity of software intensive systems, the design process has also become distributed in addition to the system itself. As a result, different software development teams had to face communication problems more and more frequently which originated in the differences of object-oriented design methodologies and modeling languages.

The real breakthrough in the field was the creation of the **Unified Modeling Language (UML)** [RJB99], which has become the de facto standard and visual object-oriented modeling language. The success of UML partially lies in the fact that it is a *standard* notation, where designers and programmers use a common language to specify the system thus preventing communication problems caused by different backgrounds in education. The other main success factor of UML is its *visual* nature which enables a step-wise and view-oriented design process for a software system using intuitive visual diagrams.

However, both academic and industrial applications have revealed several shortcomings of the language concerning, especially, its imprecise semantics and the *lack of adaptability in domain specific applications*. Due to its monolith nature, UML is hard to be tailored to the needs of domain specific applications (such as real-time systems, scheduling, etc.) which has resulted in a large collection of UML Profiles with unintuitive graphical notation. Nevertheless, the most severe problem is still the *lack of precise semantics* originating in the fact that while *the structure of model elements* in the language is *precisely defined* (by a corresponding metamodel and well-formedness constraints), *the semantics of a model element is defined informally* in pure (and sometimes poor) English. As a consequence, the definition of some constructs in the language are either incomplete or ambiguous, thus it can be interpreted differently by the designer and the programmer.

Formal methods in the design of IT systems. The lack of precise semantics for UML is more critical when using UML for the design of reliable (such as telecommunication and banking systems) and safety-critical systems (such as safety equipments and monitoring services in railway systems or in aviation), where a failure of a service may result in severe material/financial losses or even in casualties. In such systems, the traditional testing-based development process is insufficient, since it is impossible to cover each potential fault by a corresponding test case due to complexity and unanticipated faults. *During the **verification** and **validation** of critical systems, we have to prove with mathematical preciseness that the system under design will fulfill its requirements.*

The verification of dependable systems is carried out using **formal methods**, but the wide use of formal methods is hindered by several factors. First, formal methods require a thorough mathematical background which is frequently lacked by software engineers. Moreover, the *faithfulness of mathematical models*, and the *consistency of the UML-based system model and the mathematical models* are hard to be guaranteed.

In the late 1990s, the Fault Tolerant Systems Research Group at the Department of

Measurement and Information Systems at Budapest University of Technology and Economics participated in a European ESPRIT project with the acronym HIDE (High-level Design Environment for Dependability) [BDCL$^+$01, 1]. The aim of the HIDE project was to bridge the gap between the intuitiveness of high-level visual system models and the preciseness of mathematical models by **automated model transformations**. UML-based system models are first *automatically transformed* into various mathematical models, and then, the results of the analysis are *automatically back-annotated* into the source UML description thus hiding the underlying mathematics from system architects.

Several model transformations have been designed and implemented within the HIDE framework for proving functional correctness of the system [LMM99] or carrying out quantitative analysis [BMM99]. However, the benchmark experiments and applications have revealed that the transformation scripts themselves were the quality bottleneck of the HIDE framework. The main problems were the following:

- The *design* of transformations was carried out in an ad hoc way. Furthermore, there was a huge abstraction gap between the mathematical notations used in publications and the low-level transformation language.
- Without proper *automation*, the implementation of a complex transformation required a huge cost (several person month) even if the theoretical design of the transformation was already well-founded.
- Without a formal mathematical background of transformations, it was impossible, in practice, to prove the *correctness of transformations*. However, without such correctness proofs, it is impossible to decide whether a problem detected by the mathematical analysis originates from a flaw in the system model or in the transformation program.

The objective of my research was to design and implement a model transformation framework which provides automated program generation and verification facilities for transformations specified preferably in a high-level visual notation. I set up the following requirements which have to be fulfilled by a complex automated model transformation system:

- **Requirement 1** The easy–to–understand (visual) and mathematically precise *description of source and target modeling languages and models*, which is close to existing industrial standards;
- **Requirement 2** A general *high-level specification method* for the formal description of *model transformations*;
- **Requirement 3** An efficient *back-annotation* that back-project the results of the mathematical analysis into the UML design;
- **Requirement 4** A (preferably automated) method for *formally proving correctness and completeness* of transformations;
- **Requirement 5** The *automatic synthesis of transformation programs* implementing the provenly correct specification of the model transformation;

2. Research Method

These requirements basically determined the main line of my research and the individual subtasks to be solved.

Designing the structure of modeling languages. Initially, I investigated the methodology and the language constructs of MOF metamodeling (Meta Object Facility [OMG03]), which is considered to be the de facto standard of specifying the structure of modeling languages in engineering domains. The MOF metamodeling introduces a so-called four-layer architecture for separating the concerns of the modeling paradigm (metametamodel), the modeling languages (metamodels), the modeled system (models) and the objects of the real world, where the elements of a higher metalevel are *instantiated* exactly one metalevel below.

I revealed several weaknesses of MOF metamodeling which hinder the unique treatment of modeling languages taken from either mathematical or engineering domains despite the fact that its visual and object-oriented nature makes MOF an intuitive design notation for engineers.

Therefore, while keeping the visual notation and elementary constructs of MOF, I extended the methodology of metamodeling in depth by introducing dynamically reconfigurable inheritance and instantiation relations which makes the confusing notion of metalevels obsolete (Thesis 1). A special emphasis was put on to define a minimal core of metamodeling elements which are more expressive than MOF constructs (decrease in size, increase in expressiveness).

My aim was not to define a merely theoretical metamodeling framework, but also (i) to set up consistency constraints for this framework and (ii) to define a set of elementary operations which guarantee the consistent manipulations of models.

Visual specification of model transformations. In parallel with the research of metamodeling techniques, I started to investigate what kind of precise formalisms exist in the literature to specify transformations between modeling languages. In early stages of my research (in 1999), scientific research in the field was in a rather preliminary state.

As soon as getting familiar with *graph transformation*, the adaptation of the paradigm to model transformation tasks has become an obvious and highly desirable goal since graph transformation provides a visual yet mathematically precise method to capture a large variety of transformations.

This adaptation primarily included the combination of graph transformation and VPM metamodeling, the introduction of elementary control structures and the conceptual development of reference metamodels to support the back-annotation of model transformations. My new scientific results in this field are summarized in Thesis 2 of the dissertation where I propose a formalism for the uniform operational specification of model transformations and modeling languages.

Automating model transformations. The HIDE project, which preceded my research, has already demonstrated that the automatic synthesis of a provenly correct *implementa-*

tion of a model transformation is critical in a model transformation-based verification and validation process of IT systems.

For that reason, I investigated how a graph transformation virtual machine can be implemented in various programming environments. I concluded that, although several graph transformation tools exist in academic fields, all of them are research prototypes, thus the usability, adaptability and scalability of these tools are questionable for industrial size model transformations.

When examining how to specify the process of automated program generation, I found that in the majority of code generators built into industrial UML CASE tools this process is rather ad hoc. Rare exceptions are those modern tools that use standard UML Action Semantics as their specification language.

Therefore, in Thesis 3, I propose a new method for program generation where the generation process is specified by consecutive model transformations. I also elaborated a Prolog-based and an Action Semantics-based virtual machine for graph transformation systems.

Formal verification of modeling languages. Since the operational semantics of both modeling languages and transformations between such languages are defined by graph transformation, it is essential to find automated methods for the formal analysis of such specifications.

For proving the correctness of a system model, a large number of tools are available based on model checking or theorem proving. We can typically draw more general conclusions on a model by using theorem provers; however, their use requires significant amount of mathematical expertise and user interaction. Thereafter, I focused my attention on model checkers which provide a very high-level of automation (naturally, they can only examine a restricted subset of models).

Overviewing the literature on the existent support for a model checking-based analysis of graph transformation systems, I (surprisingly) found that the practical feasibility of existing (theoretical) approaches for model checking graph transformation systems has not been demonstrated by using off-the-shelf model checker tools (or any other kind of tool support).

In Thesis 4, I first introduce a new method for model checking graph transformation systems, which is mathematically sound and practically feasible as well (where practical feasibility is assessed on well-known verification benchmarks).

Formal verification of model transformations. As the specification of automated model transformations can also be erroneous, it necessitates finding formal (and preferably automated) ways of proving the correctness of a model transformation.

First I introduced a set of criteria which has to be fulfilled by any model transformation considered to be correct. In this respect, we may distinguish between the syntactic and semantic correctness of model transformations. For a syntactic correctness analysis, one has to decide whether the result of the transformation is a well-formed model of the target language. In case of semantic correctness analysis, our goal is to decide if the model transformation preserves (transformation specific) correctness properties.

In Thesis 4, I propose a method based on planner algorithms for proving syntactic correctness, and I develop a new technique to decide whether a property is left invariant by a transformation based upon the model checking of source and target models with operational semantics defined by graph transformation systems.

3. New Scientific Results

3.1. VPM: The Mathematics of Metamodeling

When investigating the Meta Object Facility (MOF) metamodeling standard [OMG03], I discovered several inconsistencies, redundancies and missing concepts in a metamodeling language where minimality, mathematical preciseness, and expressiveness is critical for the quality of specification for a large scale of visual modeling languages. Moreover, due to the lack of formal semantics, the applicability of MOF is problematic for a precise and meta-level description of mathematical models.

Thesis 1. *I defined a visual, and formally precise metamodeling (VPM) framework based upon the structure of traditional mathematical definitions and the generalization of the traditional inheritance and instantiation relations that handles uniformly arbitrary metamodels and models taken from both engineering and mathematical domains.*

- ***Structural refinement of models.*** *Based upon a minimal core subset of MOF elements (classes/entities, associations/connections, attributes/mappings), I elaborated the refinement calculus of structural model elements (Sec. 2.4), which handles the refinement (i.e., inheritance and instantiation) of all model elements (associations, attributes and packages) in addition to the refinement of classes.*

- ***Static consistency analysis.*** *I proposed a general method (Sec. 2.5) based upon the partial order relation imposed by inheritance and instantiation relations that allows to formally detect and automatically resolve models and metamodels violating the refinement axioms of VPM models.*

- ***Algebraic description of VPM models.*** *I proposed meta-level and model-level (Sec. 3.3.1, 3.3.1 and 3.5) algebraic representations and well-formedness constraints of VPM models based on the mathematical paradigm of abstract state machines [Gur95].*

- ***Elementary VPM operations.*** *I proposed elementary operations (in Sec. 3.4) that provenly guarantee the consistent manipulation of the algebraic representation of VPM models.*

- ***Separation of static and dynamic model elements.*** *In addition to traditional static model elements, I introduced the concepts of dynamic model elements (in Sec. 2.2.1), which is indispensable for the specification of dynamic semantics for modeling languages.*

The thesis is based upon Chapter 2 and 3 of the PhD dissertation, and it was published in [3,4,12,13,18,27–29].

It is worth emphasizing that these refinement relations can be reconfigured dynamically during the evolution of VPM models. In this way, behavioral descriptions (see Thesis 2) can also be stored and manipulated as models, hence I adapted the Neumann principle in the VPM metamodeling framework.

3.2. Formal Specification of Modeling Languages and Model Transformations

I examined the different model transformation approaches in the UML literature (such as [Mil02,AK02,HKT02,Whi02,dLV02]). As a conclusion, I drew that the majority of approaches providing a close fit to best engineering practice typically used visual techniques for the specification of model transformations based upon the paradigms of metamodeling and graph transformation. On the other hand, I experienced that (i) the adaptation of existing visual formalisms to the VPM metamodeling framework was problematic, (ii) back-annotation was typically not supported, (iii) there was no means to support the reuse of dynamic behavioral specifications, and (iv) the design of model transformation was carried out in an ad hoc way.

Thesis 2. *Based on the paradigm of graph transformation, I elaborated a mathematically precise and visual formalism that simultaneously supports the (i) meta-level definition of an operational semantics to an arbitrary modeling language and (i) the high-level specification of model transformations within and between such modeling languages.*

- ***An ASM semantics of graph transformation.*** *I proposed a formal operational semantics to graph transformation rules (Sec. 4.3) based on abstract state machines which captures the semantic differences of major graph transformation approaches.*
- ***Control structures.*** *In order to restrict the non-determinism of graph transformation, I defined basic control structures in the form of a control flow graph (Sec. 4.4.1).*
- ***Support for back-annotation.*** *In order to provide means for the back-annotation of the results of a formal analysis carried out on the system model, I introduced the concepts of reference models and metamodels (Sec. 4.4.4) that interrelate the elements of the source and target modeling language.*
- ***Refinement of dynamic behavior.*** *I proposed a refinement of graph transformation rules (Sec. 4.5) in order to enable the reuse of dynamic behavior when specifying the operational semantics of further modeling languages and model transformations.*

The thesis is based upon Chapter 4 of the PhD dissertation, and it was published in [2–4,8,11,12,14,16,18–21,24–27,29].

3.3. Automated Model and Program Generation

Overviewing existing graph transformation based model transformation approaches and tools, I realized that general purpose graph transformation systems could be used conceptually for the automation of model transformations (see [HKT02], for instance), but the support they provide is frequently insufficient for transformations executed on large UML models due to the fact that it is difficult to adapt them to existing UML CASE tools. Moreover, as these transformation tools are research prototypes, the transformation programs generated by them cannot be directly used in industrial transformations.

Thesis 3. *In order to support the implementation of model transformations, I proposed automated program (and model) generation techniques that automatically synthesize a transformation program from the high-level specification of the transformation (defined by metamodeling and graph transformation techniques) which can be executed on an arbitrary model of the modeling language(s).*

1. ***Automated program generation by model transformation.*** *I proposed a reflective way (consisting of consecutive model transformation steps) to provide automated program generation for model transformations (Sec. 5.2.3).*

2. ***A Prolog implementation of model transformation virtual machine.*** *I elaborated a Prolog implementation of a virtual machine executing high-level specifications of model transformations (Sec. 5.2.1 and 5.2.2). The implementation of graph transformation rules is based upon a graph pattern matching algorithm using depth-first search with optimized clause ordering and the powerful unification mechanism of Prolog. The realization of the control flow graph exploits the meta-programming facilities of Prolog.*

3. ***Action Semantics description of model transformations.*** *I proposed how to map model transformations into standard Action Semantics expressions (in Sec. 5.4). Graph transformation rules are simulated by an algorithm using local searches in an object-oriented structure, while the control flow graph is implemented by built-in compound actions of the Action Semantics standard.*

The thesis is based upon Chapter 5 of the PhD dissertation, and it was published in [5, 11, 22, 23, 27].

3.4. Formal Verification of Visual Modeling Languages and Model Transformations

A natural requirement for a complex model transformation system is the automated formal verification of models and transformations. The question was first addressed in the literature by providing a set of correctness and completeness criteria in [2, 15, 19]. Examining the existing techniques and tools supporting the automated formal analysis for graph transformation systems capturing the dynamic behavior of a model, I found that while the correctness analysis of graph transformation systems has been investigated recently in several papers [BK02, PE02, Hec98] from a theoretical point of view, no general and modeling language independent approach existed that is directly applicable and provenly feasible in practice as well. However, since model checking tools frequently have to face the state space explosion problem, the practical feasibility of verification approaches should be indispensable.

Thesis 4. *I presented an approach for the automated verification of any specific instance model of an arbitrary modeling language (with static structure defined by metamodeling and operational semantics defined by graph transformation systems) using existing model checker tools. Moreover, I defined consistency criteria for model transformations and I proposed methods to formally verify the syntactic (proving language containment) and semantic consistency (aiming at property preservations) of such transformations.*

- ***Provenly correct transition system representation.*** *For any graph transformation system I defined an encoding (Sec. 6.3.1 and 6.3.2) which derives a behaviorally equivalent transition system.*
- ***Optimized model description.*** *I introduced several optimizations (Sec. 6.3.3) in the transition system encoding of a graph transformation system exploiting the fact that graph transformation rules only modify (by definition) dynamic model elements thus all the static parts of a model can be eliminated after compile-time preprocessing.*
- ***Performance analysis of verification.*** *I demonstrated the practical feasibility of the approach by assessing the efficiency of model checking the target transition system on benchmark verification problems (Sec. 6.5). As a conclusion, I derived the "few complex is better than many simple" principle as a verification rule of thumb which may help the transformation engineers to design graph transformation systems which are efficient from a verification aspect.*
- ***Consistency criteria for model transformations.*** *I proposed general consistency criteria for model transformations (Sec. 7.1.1) which has to be provenly satisfied by any specific model transformation.*
- ***Proving syntactic correctness and completeness of model transformations.*** *I elaborated a method based on planner algorithms (Sec. 7.2) for proving the syntactic correctness and completeness of model transformations. As a consequence, we may decide whether the target model (obtained as a result of a model transformation) is provenly a well-formed model of the target modeling language.*
- ***Semantic consistency analysis of model transformations.*** *I proposed a general and automated framework (in Sec. 7.3) which allows to investigate whether certain (transformation specific) semantic consistency properties are preserved by a model transformation.*

The thesis is based upon Chapter 6 and 7 of the PhD dissertation, and it was published in [2,4,6,7,9,10,14,15,17,19,20].

One can conclude that my approach is the first complex solution in the literature for a language independent formal verification of visual modeling languages using model checking techniques which is also assessed on practical benchmarks.

4. Utilization of New Scientific Results

In order to demonstrate the practical, industrial utilization of theoretical results presented in the current thesis, we (where the term “we” also refers to several students working on their Master’s or PhD thesis) carried out the development of a set of tools supporting our model transformation approach in addition to basic research.

The VIATRA model transformation framework and its applications. Adapting the main industrial standards (including XMI, MOF and UML), I designed and implemented in Prolog the VIATRA model transformation system, which carries out model transformations (following our theoretical foundations of Thesis 2 and 3) on models given in a standard XMI format by automatically generating and executing Prolog programs derived from high-level rule descriptions specified in a UML notation.

Using the VIATRA model transformation system in an Hungarian research project (IKTA 065/2000: A framework for designing and testing dependable and safety critical systems), I designed and implemented model transformations, which were tested afterwards on UML based system models taken from industrial partners. As a conclusion, we drew that even an academic prototype tool such as VIATRA had an acceptable run-time performance even for real-size models.

Modeling benchmarks. In order to demonstrate the expressiveness and practical feasibility of our model transformation approach, I proposed a formal operational semantics for the well-known visual modeling languages of Petri nets and Extended Hierarchical Automata (which is a standard formalization of UML statecharts) in the form of model transformation systems. I also proved that this semantics is provenly equivalent in each case with the traditional mathematical description of the language.

A design methodology for model transformations. I also elaborated a design methodology for model transformations which defines the main phases of a design cycle introducing the concepts of static, dynamic and derived model elements on the metamodel level, and dividing the process of model transformations into an initialization and an execution phase.

Model checking modeling languages. We developed a tool called CheckVML [10] (based upon the foundations of Thesis 4) which automatically derives a Promela model as the output from the metamodel, the instance model and the set of graph transformation rules supplied as the input. This Promela model serves as the input of the SPIN model checker tool [Hol97]. After that, feeding the property to be proved (given in the form of temporal logic formulae) directly to SPIN we may decide for any specific model of an arbitrary modeling language whether the given property holds.

Ongoing development and future work. An ongoing activity aims at developing a metamodeling tool based on our VPM foundations (of Thesis 1) and integrating it into the VIATRA framework afterwards. The overall objective is to develop a new generation of the so-called Meta-CASE tools which simultaneously provides mathematical preciseness, and reusability of both static and dynamic semantic descriptions for a wide range of visual modeling languages.

Further activities aim at implementing the encoding of model transformation rules into Action Semantics expressions in an off-the-shelf UML CASE tool (with support for Action Semantics). As a result, all the theoretical foundations can be integrated into an industrial environment as well.

5. Publications in the Subject of the Dissertation

Parts in a Book

[1] D. Varró. UML modeling of web applications. In C. Garcia and A. Pataricza, editors, *E-Business for SMEs*. Prentice Hall, 2004. Reviewed by the publisher and the editors.

Journal papers (appeared in abroad)

[2] D. Varró, G. Varró, and A. Pataricza. Designing the automatic transformation of visual languages. *Science of Computer Programming*, 44(2):205–227, August 2002, Elsevier.

[3] D. Varró and A. Pataricza. VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML. *Journal of Software and Systems Modelling*, (2)3:187–210, October 2003, Springer.

[4] D. Varró. Automated formal verification of visual modeling languages by model checking. *Journal of Software and Systems Modelling*, 2003, Springer. Accepted to the Special Issue on Graph Transformation and Visual Modelling Techniques.

Journal papers (appeared in Hungary)

[5] D. Varró and A. Pataricza. UML Action Semantics for model transformation systems. *Periodica Polytechnica,* 2003. In press.

International conference and workshop papers

[6] L. Baresi, R. Heckel, S. Thöne, and D. Varró. Modeling and analysis of architectural styles. In *Proc ESEC 2003: European Software Engineering Conference,* pages 68–77, Helsinki, Finland, September, 2003, ACM Press.

[7] L. Baresi, R. Heckel, S. Thöne, and D. Varró. Modeling and analysis of architectural styles based on graph transformation. In *The 6th ICSE Workshop on Component Based Software Engineering: Automated Reasoning and Prediction*, Portland, Oregon, USA, May 3-4.

[8] Gy. Csertán, G. Huszerl, I. Majzik, Zs. Pap, A. Pataricza, and D. Varró. VIATRA: Visual automated transformations for formal verification and validation of UML

models. In *Proc. ASE 2002: 17th IEEE International Conference on Automated Software Engineering*, pages 267–270, Edinburgh, UK, September 23–27 2002. IEEE Press.

[9] G. Salamon, D. Varró, and A. Pataricza. Formal verification of model transformation systems. In *EDCC 2002: Fourth European Dependable Computing Conference: Fast Abstracts*, pages 15–16, Toulouse, France, October 23–25 2002.

[10] Á. Schmidt and D. Varró. CheckVML: A tool for model checking visual modeling languages. In P. Stevens, J. Whittle and G. Booch, editors. *Proc. UML 2003: 6th International Conference on the Unified Modeling Language*, volume 2863 of LNCS, pages 92–95, San Francisco, CA, USA, October 20-24 2003, Springer.

[11] D. Varró. Automatic program generation for and by model transformation systems. In H.-J. Kreowski and P. Knirsch, editors, *Proc. AGT 2002: Workshop on Applied Graph Transformation*, pages 161–173, Grenoble, France, April 12–13 2002.

[12] D. Varró. A formal semantics of UML Statecharts by model transition systems. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. ICGT 2002: 1st International Conference on Graph Transformation*, volume 2505 of *LNCS*, pages 378–392, Barcelona, Spain, October 7–12 2002. Springer-Verlag.

[13] D. Varró. A pattern-based constraint language for metamodels. In T. Csendes, editor, *Proc. CSCS 2002: The Third Conference of PhD Students in Computer Science*, page 109, Szeged, Hungary, July 1–4 2002.

[14] D. Varró. Towards symbolic analysis of visual modelling languages. In P. Bottoni and M. Minas, editors, *Proc. GT-VMT 2002: International Workshop on Graph Transformation and Visual Modelling Techniques*, volume 72 of *ENTCS*, pages 57–70, Barcelona, Spain, October 11-12 2002. Elsevier.

[15] D. Varró. Towards formal verification of model transformations. In T. Jones, editor, *PhD Student Workshop of FMOODS 2002, Formal Methods for Open Object-Based Distributed Systems*, Enschede, The Netherlands, March 20–22 2002.

[16] D. Varró, Sz. Gyapay, and A. Pataricza. Automatic transformation of UML models for system verification. In J. Whittle et al., editors, *WTUML'01: Workshop on Transformations in UML*, pages 123–127, Genova, Italy, April 7th 2001.

[17] D. Varró and A. Pataricza. Automated formal verification of model transformations. In J. Jürjens, B. Rumpe, R. France and E.B. Fernandez *Proc. CSDUML 2003: Workshop on Critical Systems Development in UML*, pages 63–78, San Francisco, CA, USA, October 20–24 2003. Published as Technical Report, TUM-I0323, September, Technische Universität München.

[18] D. Varró and A. Pataricza. Metamodeling mathematics: A precise and visual framework for describing semantics domains of UML models. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *Proc. Fifth International Conference on the Unified Modeling Language – The Language and its Applications*, volume 2460 of *LNCS*, pages 18–33, Dresden, Germany, September 30 – October 4 2002. Springer-Verlag.

[19] D. Varró, G. Varró, and A. Pataricza. Designing the automatic transformation of visual languages. In H. Ehrig and G. Taentzer, editors, *GRATRA 2000 Joint APPLIGRAPH and GETGRATS Workshop on Graph Transformation Systems*, pages 14–21, Berlin, Germany, March 25–27 2000.

[20] D. Varró, G. Varró, and A. Pataricza. Visual graph transformation in system verification. In E. Gramatova, H. Manhaeve, and A. Pawlak, editors, *DDECS 2000 Design and Diagnostics of Electronic Circuits and Systems*, pages 137–141, Bratislava, Slovakia, April 5–7 2000.

Conference and workshop papers in Hungarian

[21] D. Varró. Visual automated model transformation. In *Mini–Symposium 2001*, pages 48–49, Budapest University of Technology and Economics, Department of Measurement and Information Systems, January 31 – February 1 2001. IEEE Hungary Section (BUTE Student Branch).

[22] D. Varró. Automated program generation in VIATRA. In *Mini–Symposium 2002*, pages 34–35, Budapest University of Technology and Economics, Department of Measurement and Information Systems, February 4–5 2002. IEEE Hungary Section (BUTE Student Branch).

[23] D. Varró. UML Action Semantics for model transformation systems. In *Mini–Symposium 2003*, pages 22–23, Budapest University of Technology and Economics, Department of Measurement and Information Systems, February 2003. IEEE Hungary Section (BUTE Student Branch).

[24] G. Huszerl, I. Majzik, Zs. Pap, D. Petri, A. Pataricza, and D. Varró. Keretrendszer nagymegbízhatóságú, biztonságkritikus rendszerek fejlesztéséhez és teszteléséhez. In *OOOK 2002: 5. Országos Objektum-Orientált Konferencia*, Dobogókõ, Hungary, October 16–17 2002. In Hungarian.

[25] D. Varró and A. Pataricza. UML modellek automatikus transzformációi. In *OOOK 2002: 5. Országos Objektum-Orientált Konferencia*, Dobogókõ, Hungary, October 16–17 2002. In Hungarian.

[26] D. Varró and P. Domokos. A VIATRA modelltranszformációs rendszer. In Bitay E. (editor) *FMTÜ 2003: Fiatal Magyarok Tudományos Ülésszaka*, pages 51–54, Kolozsvár, Románia, March 21–22, 2003, Erdélyi Múzeum Egyesület. In Hungarian.

Electronic publications (Technical reports)

[27] D. Varró and A. Pataricza. Mathematical model transformations for system verification. Technical report, Budapest University of Technology and Economics, May 2001.

[28] D. Varró and A. Pataricza. A unifying semantic framework for multilevel metamodelling. Technical report, Budapest University of Technology and Economics, October 2001.

[29] D. Varró and A. Pataricza. Formalizing model transformation systems by abstract state machines. Technical report, Budapest University of Technology and Economics, June 2003.

6. References

[AK02] D. Akehurst and S. Kent. A relational approach to defining transformations in a metamodel. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *Proc. Fifth Intern. Conf. on the Unified Modeling Language – The Language and its Applications*, volume 2460 of *LNCS*, pages 243–258, Dresden, Germany, September 30 – October 4 2002. Springer-Verlag.

[BDCL$^+$01] A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza, and G. Savoia. Dependability analysis in the early phases of UML based system design. *International Journal of Computer Systems - Science & Engineering*, 16(5):265–275, 2001.

[BK02] P. Baldan and B. König. Approximating the behaviour of graph transformation systems. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. ICGT 2002: First International Conference on Graph Transformation*, volume 2505 of *LNCS*, pages 14–29, Barcelona, Spain, October 7–12 2002. Springer.

[BMM99] A. Bondavalli, I. Majzik, and I. Mura. Automatic dependability analyses for supporting design decisions in UML. In *Proc. HASE'99: The 4th IEEE International Symposium on High Assurance Systems Engineering*, pages 64–71, 1999.

[dLV02] J. de Lara and H. Vangheluwe. AToM3: A tool for multi-formalism and meta-modelling. In R.-D. Kutsche and H. Weber, editors, *5th Intern. Conf., FASE 2002: Fundamental Approaches to Software Engineering, Grenoble, France, April 8-12, 2002, Proceedings*, volume 2306 of *LNCS*, pages 174–188. Springer, 2002.

[Gur95] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger (editor), *Specification and Validation Methods*, Oxford University Press, 1995.

[Hec98] R. Heckel. Compositional verification of reactive systems specified by graph transformation. In *Proc. FASE: Fundamental Approaches to Software Engineering*, volume 1382 of *LNCS*, pages 138–153. Springer, 1998.

[HKT02] R. Heckel, J. Küster, and G. Taentzer. Towards automatic translation of UML models into semantic domains. In *Proc. AGT 2002: Workshop on Applied Graph Transformation*, pages 11–21, Grenoble, France, April 2002.

[Hol97] G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[LMM99] D. Latella, I. Majzik, and M. Massink. Automatic verification of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing*, 11(6):637–664, 1999.

[Mil02] D. Milicev. Automatic model transformations using extended UML object diagrams in modeling environments. *IEEE Transactions on Software Engineering*, 28(4):413–431, April 2002.

[OMG03] Object Management Group. *Meta Object Facility Version 2.0*. `http://www.omg.org`.

[PE02] J. Padberg and B. J. Enders. Rule invariants in graph transformation systems for analyzing safety-critical systems. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. ICGT 2002: Firs International Conference on Graph Transformation*, volume 2505 of *LNCS*, pages 334–350, Barcelona, Spain, October 7–12 2002. Springer.

[PL99] I. Paltor and J. Lilius. vUML: A tool for verifying UML models. In R. J. Hall and E. Tyugu, editors, *Proc. of the 14th IEEE Intern. Conf. on Automated Software Engineering, ASE'99*. IEEE, 1999.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[Whi02] J. Whittle. Transformations and software modeling languages: Automating transformations in UML. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *Proc. Fifth Intern. Conf. on the Unified Modeling Language – The Language and its Applications*, volume 2460 of *LNCS*, pages 227–242, Dresden, Germany, September 30 – October 4 2002. Springer-Verlag.