AUTOMATIC GRAPH TRANSFORMATION IN SYSTEM VERIFICATION

Dániel Varró, Gergely Varró, András Pataricza
Technical University of Budapest,
Department of Measurement and Information Systems,
Budapest, 1111, Mûegyetem rkp. 3-11.
Hungary[1]
{pataric,varro}@mit.bme.hu

Abstract. *The use of formal verification methods is essential in the design process of dependable computer controlled systems. A complex environment should support the semi-formal specification as well as the formal verification of the desired system. The efficiency of applying these formal methods will be highly increased if the underlying mathematical background is hidden from the designer. In such an integrated system effective techniques are needed to transform the system model to different sort of mathematical models supporting the assessment of system characteristics. The current paper introduces our research results towards a general-purpose model transformation engine. This approach results in yielding a provenly correct and complete transformation code by combining the powerful techniques of graph transformation, planner algorithms and deductive databases.*
Keywords*: formal verification, graph transformation, visual languages, planner algorithms, deductive databases.*

1 Introduction

1.1 Formal Verification in Software Design

During the design process of dependable computer controlled systems verifying whether the system fulfils its requirements or needs some re-design is indispensable at each level of abstraction and after each step of system model refinement in order to spare time and resources.

When such a refinement step is performed, system characteristics like dependability, timeliness and correctness are assessed. The characteristics are subsequently confronted with requirements, therefore only provenly correct refinement steps are allowed.

Several sophisticated mathematical tools exist supporting the assessment of these system parameters (e.g. SPIN model checker). However, they require special mathematical skills and a thorough knowledge of the underlying mathematics for their use. Moreover, additional obstacles (such as problems in the consistency and faithfulness of different sort of models) arise due to the lack of an automated transformation and back-annotation between the mathematical and designer models.

These problems of consistency and faithfulness are planned to be eliminated by the European ESPRIT project HIDE (High-level Integrated Design Environment for Dependability) [2] with the participation of the Department of Measurement and Information Systems at Technical University of Budapest, Pisa Dependable Computing Center, University of Erlangen and two UML provider enterprises.

1.2 HIDE

The HIDE project (regarded as a novel approach concerned with the process of dependable system design) aims at the integration of the textual system specification and system model into a semi-formal system description based on the *Unified Modeling Language (UML),* hence bridging the gap between a practice-oriented CASE methodology and sophisticated mathematical tools.

[1] This work was supported by the Hungarian National Scientific Foundation under contract OTKA T030804.

Formal verification, quantitative and timeliness analysis tools will be made available to the designer within the design environment, and these mathematical models are planned to be derived automatically from the user-end UML specification.

The system model is stored in a central HIDE repository (a conventional relational database) for gaining an open, tool-independent architecture due to this architectural redundancy.

An overview of the designated architecture of HIDE can be observed later in Fig. 1.

1.3 Mathematical Model Transformation

The step when the input language of a mathematical tool is generated from the UML model repository is called **mathematical model transformation**.

Several semi-formal transformation algorithms have already been designed and implemented for e.g. formal verification of functional properties [5] and quantitative analysis of dependability attributes [3,4].

The inverse direction of model transformation (referred as **back-annotation**) is of immense importance if some design faults (e.g. a deadlock) are detected during the mathematical analysis. After an automated back-annotation these problems appear in the UML system model allowing the designer to fix these conceptual bugs.

Unfortunately, the conventional (i.e. heuristic) way of model transformation as used in the pilot phase of HIDE raises several problems due to the large number of target mathematical analysis tools. Moreover, HIDE is an open environment in order to support user-defined checks. This way an integration possibility of transformations has to be provided to the end-user.

- No unique and formal description of transformation algorithms exists therefore their implementation was hand-written in the form of PL/SQL scripts and rather ad hoc (inconvenient for implementing complex transformations).
- The formal verification of these scripts (aiming to prove correctness and completeness) was almost impossible, hence their quality is a bottleneck of the entire architectural approach.
- Each model has to be verified individually disregarding from the same underlying algorithmical skeletons.

1.4 Visual Automated Graph Transformation

In the following an alternate solution will be sketched based on the theoretical and practical results of **graph transformation**, **deductive databases** and **planner algorithms** in order to obtain a system supporting the graphical design and verification of similar sort of model transformation with the capability of automatic generation of transformation code.

Such a solution undoubtedly increases the level of confidence that can be put on a system since the derivation of the mathematical models (used as inputs for different formal verification tools) is provenly **correct** and **complete**, moreover, tiresome implementation steps are also obsolete.

Fault-tolerant elements added to the system from a pre-defined set can be evaluated from several aspects by well-known formal analysis techniques, thus highly improving dependability. Moreover, the back-annotation of mathematical analysis results is also supported offering a possible candidate for HIDE architecture.

Figure 1. gives an overview of the designated architecture of HIDE. In the rest of the report the system components are discussed shortly.

2 Transformation System Components

2.1 Graph Transformation and Visual Languages

Graphs are well-known and frequently used means to represent system states, complex objects, diagrams and networks like e.g. flowcharts, entity-relationship diagrams or Petri nets [1]. Rules have proved to be extremely useful for describing computations by local transformation. Areas like language definition,

logic and functional programming, algebraic specification etc. are prominent witnesses of the role of rules.

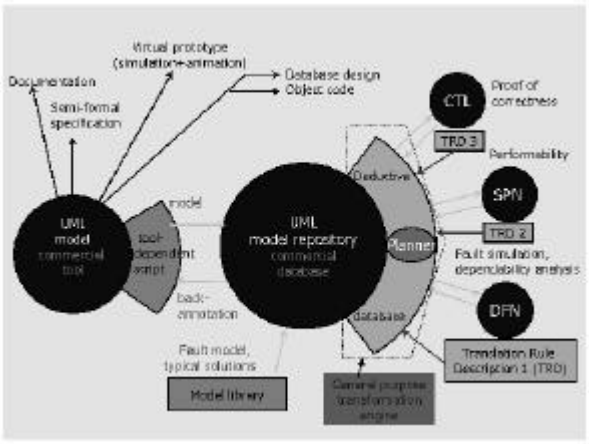

Fig.1. Proposed architectural framework of the HIDE environment

Graph transformation (also known as graph rewriting) combines the advantages of both, graphs and rules, into an individual computational paradigm. A **graph transformation rule** is a special pair of pattern graphs where the instance defined by the left-hand side is substituted with the instance defined by the right-hand side when an applying such a rule (similarly to the well-known grammar rules of Chomsky in computational linguistics).

The theory and application of **visual languages** is also based on the strong paradigm of graph transformation. Visual languages (defined by special graph transformation rules, usually called graph grammars) use coupled graph layers (logical and graphical) for supporting the visual information contained in graphical diagrams (such as UML statecharts or Petri nets).

Visual languages are used in our case for defining both the source and target models of a transformation, moreover, the transformation rules are also a special sort of graph grammar rules, referred as *Tranlation Rule Description (TRD)* in Fig. 1.

The visual transformation rules are local in a sense that they handle only a small amount of model elements at a time therefore the designer does not need to concentrate on the entire transformation problem.

2.2 Planner algorithms

Planner algorithms (e.g. in [6]) are a sort of hierarchical search algorithms widely used in artificial intelligence. According to the "divide and conquer" principle they divide the original problem into smaller parts before trying to solve them. Finally, these partial solutions are merged together resulting in a solution of the original problem.

The input of these algorithms are expressions describing the **initial and goal state** (usually first-order logic formulae) while the output is a correct **plan**, which is a sequence of permitted operations providing a trajectory from the initial state to the goal state.

The operations are structured as a **precondition** and an **action** part, where preconditions describe the (positive or negative) conditions that must held before performing the specific operation. The action part describes the necessary changes to the next state of the state-space.

Planner algorithms also play a major role in a transformation system as the two most essential questions that arise in connection with a given set of transformation rules can be answered by means of planners.

- The **correctness problem** refers to whether the application of transformation rules always yields a syntactically correct target model or not. (Naturally, transformation designers are responsible for semantical correctness)
- The **completeness problem** refers to whether the given set of rules is complete, i.e., each situation allowed in the source (input) model is covered by a corresponding rule.

As planner algorithms operate constructively, any problems arisen in connection with correctness and completeness are indicated immediately. Therefore the construction of a novel set of rules by adding or altering one or several rules in the previous set is simple and convenient.

As a consequence, the entire transformation is based on theoretical background thus highly increasing the faithfulness and consistency of the target mathematical model.

2.3 Deductive Databases

Since the HIDE architectural framework is based on storing system and mathematical models in a central relational repository our designated transformation system has to operate on these databases. As a consequence, a model transformation proceeds between the source and target database tables, narrowing our possibilities of system and mathematical model description.

However, as transformation rules of large complexity manipulate the source and target models therefore intelligent and effective database search algorithms are necessitated.

In order to fulfil both requirements (complex searches on relational databases) a deductive database was implemented by using Prolog hence obtaining a purely declarative data manipulation language with backtracking yielding a legible and verifiable transformation code.

A suitable set of database tables can automatically be derived from the visual grammar and transformation rules (in an SQL DDL format).

3 Concluding remarks

In the current paper our research towards a general-purpose transformation engine performing mathematical model transformations has been summarized. For this specific purpose we integrated the powerful techniques of visual languages, planner algorithms and deductive databases in order to obtain a provenly correct and complete and automatically generated transformation code.

A sample transformation has already been implemented and investigated as a structural benchmark of HIDE environment (considering its behaviour in time for source models of medium size), with promising results. A source model of 2000 database objects was transformed within 15 minutes. Since the analysis of a user model by mathematical tools using our transformed model as input must handle extremely large problem spaces, therefore, according to our experiments, this model transformation does not take more than a few percentage of the total time.

However, our research is still in an early phase, such a transformation system supporting formal verification of the model transformation process seems to be a promising candidate for the HIDE environment aiming at an integrated, easy-to-understand handling of both system and mathematical models in order to increase dependability.

References

[1] ANDRIES, M. et al.: Graph transformation for specification and programming, Science of Computer Programming 34 (1999) 1-54.

[2] BONDAVALLI, A. - DAL CIN, M. - LATELLA, D. - PATARICZA A.: High-level Integrated Design Environment for Dependability. Invited paper to {WORDS'99}, 1999 Workshop on Real-Time Dependable Systems (1999).

[3] BONDAVALLI, A. - MAJZIK, I. - MURA, I.: Automatic dependability analyses for supporting design decisions in UML, (1998).

[4] DAL CIN, M. - HUSZERL, G. - KOSMIDIS K.: Evaluation of safety-critical system based on guarded statecharts. In Proc. HASE'99, 4th IEEE International Symposium on High Assurance Systems Engineering, (1999).

[5] LATELLA, D. - MAJZIK, I. - MASSINK M.: Towards a formal operational semantics of UML Statechart Diagrams. In Proc, IFIP TC6/WG6.1, 3rd International Conference on Formal Methods for Open Object-Oriented Distributed Systems, February, (1999).

[6] WELD, D.: An introduction to least commitment planning. AI Magazine, (1994).