Designing the Automatic Transformation of Visual Languages

Dániel Varró, Gergely Varró, and András Pataricza

Technical University of Budapest [*],
Department of Measurement and Infomation Systems
Budapest, 1111, Müegyetem rakpart 3-11.
Hungary
e-mail: pataric@mit.bme.hu

Abstract. The process of developing depandable, safety–critical systems controlled by computers requires a formal verification of conceptual and architectural choices by using different mathematical tools. According to a novel approach of IT system design, these input models to formal mathematical analysis are transformed automatically from the system model. Up to now, the design and implementation of such transformations was rather ad hoc missing any formal descriptions and methods. In this paper we present our efforts towards a model transformation system based on a powerful integration of graph transformation, planner algorithms and deductive databases in order to obtain an automatically generated, provenly correct and complete transformation code.

1 Introduction

For most computer controlled systems, especially dependable, real-time systems for critical applications, an effective design process requires an early validation of the concepts and architectural choices, without wasting time and resources before realizing whether the system fulfills its objectives or needs some re-design.

In order to increase the level of confidence that can be put on a system mathematical tools (based on formal verification methods) are used to assess the most important system parameters of timeliness, performability or dependability.

Unfortunately, sophisticated verification tools require a thorough knowledge of underlying mathematics therefore special skills are needed for dependable IT system designers. Moreover, faithfulness and consistancy of a mathematical model can hardly be guaranteed.

In order to avoid these modeling problems, novel approaches concerned with the process of IT system design have turned up recently ([BCLP99]) aiming at the integration of textual system specification and system model into a semi–formal system description based on the *Unified Modeling Language (UML)* (see [BJR99] for a complete reference).

[*] This work was supported by the Hungarian National Scientific Foundation Grant OTKA T030804

The UML models of specific CASE tools (such as of Rational Rose and Innovator) are transformed into a central repository (which is a standard commercial database) for obtaining an open tool–independent architecture.

The mathematical models are planned to be automatically derived from the UML–based system model, moreover, the results of the mathematical analysis are also automatically back–annotated, therefore the problems (e.g. a deadlock) turned up in the mathematical model can straightly be observed in the system model.

The validation of designs described using UML is the main objective of the European ESPRIT project HIDE (High-level Integrated Design Environment for Dependability)[BCLP99] with the participation of the DMIS at TUB, Pisa Dependable Computing Center, University of Erlangen and two UML tool provider enterprises.

The purpose of HIDE is to allow the designer to use UML as a front-end for the specification of both the system and the user requirements, and to bridge the gap between a practice-oriented CASE methodology and sophisticated mathematical tools.

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 [LMM99] and quantitative analysis of dependability attributes [BMM98] [CHK99].

Unfortunately, the conventional way (i.e. experiences in the experimental implementation process) of model transformation raised several problems due to the openness of the HIDE architecture. (Openness means that the mathematical models of different verification tools can be generated from the central system model.)

- No unique and formal description of transformation algorithms existed therefore their implementation was hand–written and rather ad hoc (inconvenient for implementing complex transformations).
- As these scripts were usually written in PL/SQL their formal verification (aiming to prove correctness and completeness) is almost impossible, hence their quality is a bottleneck of the entire architectural approach.
- However, the transformation algorithms have similar underlying algorithmical skeletons, each model has to be verified individually.

2 Research Objectives

The aim of the current research is to provide a framework for the process of designing general mathematical model transformations and to introduce a novel approach of a translation system, which supports the automated generation of the transformation code of a proven quality.

Such an automated transformation has to fulfil at least the following requirements

- The description of the problem has to be simultaneously natural and mathematically precise;
- On the other hand, this description has to be highly independent from the target mathematical tools.

Overviewing the previous results in the literature the **visual languages and graph transformation** assured the best fit to the purpose of **translation rule description (TRD)** (Sec. 3.1), as on one hand they preserve the visual information stored in CASE models and on the other they fit well to typical mathematical paradigms.

After a user has specified his set of rules in the TRD typically two main questions arise.

- **correctness problem**: Whether this set of rules is correct in the sense that it does not contain contradictions or ambiguous operations.
- **completeness problem** Whether this set of rules is complete, i.e., each situation allowed in the source (input) language is covered by a corresponding rule.

As a novel idea, **planner algorithms** (introduced in Sec. 3.2) are used as a core to prove correctness and completeness of a transformation therefore there is an extensive increase in its reliability as only the design of elementary rules needs intuition and the deep relationship between them is automatically generated. Planner algorithms are well-known to artificial intelligence experts and used for designing sequences of complex operations,

Mathematical transformations necessitate (as the entire transformation is rule-driven) complex search operations. In order to avoid a reduction in efficiency in contrast with hand-written (therefore nearly optimal SQL) codes, intelligent and effective search algorithms are needed. As a practical solution, we carried out a **deductive database** (Sec. 3.3) placed above traditional relational databases.

Figure 1 shows the proposed architectural framework of HIDE.

In the following section, the main features of our general purpose transformation system will be introduced.

3 A Model Transformation System

Our desired model transformation system has to support at least the following features:

- Description of input and output (source and target) models;
- Translation rule description (TRD);
- An engine for proving correctness and completeness of TRDs;
- A database for storing both the models and the rules;
- A transformation engine built upon this database providing automatically generated transformation code;

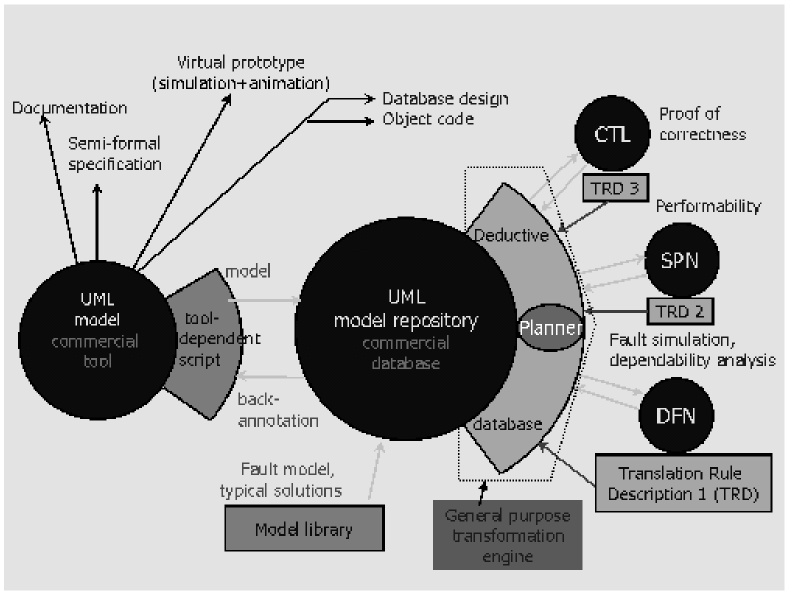

Fig. 1. Proposed architectural framework of the HIDE environment

– An efficient back-annotation of mathematical analysis results to the system model.

In the following sections we investigate how these requirements can be fulfilled with the help of well-known approaches of computer science.

3.1 Visual Languages in Model Transformation

The system model in our field is based on object-oriented UML diagrams such as Class Diagrams or Statecharts. These diagrams provide in the form of graphical objects and diagrammatic techniques a uniform way for system engineers to describe their ideas. The use of visual languages and graph transformation for describing the diagrams of object-oriented tools is a favourite candidate as several approaches concerned with related tasks (as e.g. [EEHT97]) already exist in the literature.

On the other hand the tools used for software verification (e.g. SPIN model checker [Hol97] or Petri Nets) also use visual objects in their mathematical model therefore the usage of visual languages and graph grammars for the purpose of describing the rules of a model transformation is rather straightforward.

When a visual language is transformed from another visual language this process might be called **visual language translation** according to the notation of mathematical linguistics (moreover to avoid confusion by overloading the word 'transformation').

However, the same notation (graph translators) was used in [Sch94] for describing a translation from syntax diagrams to control flow diagrams by triple graph grammars. In his article Schürr pointed out several advantages of his approach:

- Triple graph grammars (with a correspondence graph in the kernel) allow the same specification for a bidirectional transformation.
- The traditional common supergraph approach enforces the introduction of embedded superstructures even in the case when the input and output of translation are viewed as separate graphs and where fine–grained correspondences are of no importance afterwards.

Despite these disadvantages a common supergraph structure has been chosen for *translation rule description* as our purpose is quite different since the correspondence between the source and target language is of crucial importance due to the need for an efficient back–annotation of analysis results.

Moreover, the translation is usually tightening in a sense that UML system models usually contain several attributes which are irrelevant for a specific mathematical analysis.

As a conclusion, in our desired model transformation system both the rules constructing the source and target models and the TRDs are well–formed graph grammar rules, however, (as it is mentioned in Sec. 3.3) the process of these rules are quite different from the traditional context–sensitive graph parsing approach ([RS95]).

3.2 Planner Algorithms in Model Transformation

Planner algorithms (most important lectures e.g.: [Wel94],[PW92], are complex (and usually hierarchical) problem solving procedures of artificial intelligence subdividing the original problem into smaller parts before trying to solve them (according to the "divide and conquer" principle). Finally, these partial solutions are merged yielding the 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 stand before performing the specific operation while action part describes the necessary changes to the next state of the world.

One might notice that *a graph transformation rule can also be regarded as a planner operation* where the left hand side (LHS) of a rule is responsible for the preconditions and the right hand side (RHS) (or rather the difference set of LHS and RHS) describes the next state.

When a user defines his TRD the questions of correctness and completeness arise. After having considered graph grammar rules for both the source and target model and TRDs these problems gain a novel interpretation.

- Correctness: A translation is correct if the target–like model generated by transforming a source model by translation rules (TRDs) can also be generated from scratch by the original grammar rules of the target model.
- Completeness: A translation is complete if the TRD covers all the possible source models defined by their grammar rules.

In such an environment as HIDE a constructive proof was required, which also discloses those parts of the system where correctness and completeness do not stand therefore the traditional algebraic double pushout approach was insufficient (especially when regarding the questions of completeness).

Our novel approach uses a planner for constructively proving correctness and completeness of a given TRD for a specific source and target model. (Each grammar and translation rules are interpreted as a planner operation as described above.)

Under certain conditions correctness and completeness of a translation can be proven independently from input models by regarding only TRDs and the grammar rules of input model, which might provide TRD libraries of a proven quality on one hand and a cheaper proof generation on the other.

Moreover, since the automatic generation of transformation code requires an appropriate ordering of the application of translation rules, planner algorithms can be applied for this purpose as well, however, an explicit ordering (i.e. given by the user) of translation rules usually simplifies the occurring problems (such as termination of transformation algorithm).

In our opinion, the integration of planner algorithms and graph transformation provides an engine for proving correctness and completeness of a TRD with an easy–to–comprehend mathematical background even for those who are not particularly skilled in the traditional underlying mathematics of algebraic graph transformation. However, a formal description and proof of the integration is still needed.

3.3 Deductive Databases in Model Transformation

As it was previously stated in Sec. 2, grammar rules of large complexity manipulate the source and target models (which are stored in a traditional relational database) therefore intelligent and effective database search algorithms are needed.

Therefore we have implemented a deductive database (theoretical background in e.g. [Rei84]) with a so-called tightly coupled architecture where the original relational database calls are covered hence these searches are defined in higher–level query language. Prolog has been chosen for this purpose due to its poweful pattern–matching and unification methods.

Deductive databases are composed of facts (as e.g. 'parent' relation describing a pedigree) and deduced relations(as e.g. 'grandparent' relation). This kind of structuring can also be interpreted in the HIDE environment in a sense that an automatically generated transformation algorithm is a set of translation rules (as deduced relations) built upon basic model elements (as facts).

A description of a visual language usually consists of several layouts (e.g. graphical or logical as in [BT97]) where the logical layout of the (source or target) grammar rules is suitable for an automatic derivation of an adequate Entity–Relationship (ER) diagram describing the storage of a specific model.

The models and rules are stored in the same way since there is no real difference in their syntax as a pattern (a rule) is a special instance with unbound variables as attributes.

Finally, the model transformation algorithm itself is also a Prolog program performing deductive database calls and structured as planner operations (with precondition and action part) to increase legibility and verification of the code.

Summarizing our reasons for integrating visual languages and deductive databases, obviously, the implementational questions played the main role. After having transformed a model into a deductive database the pattern–matching algorithms (within their limits) are cheaper than those sophisticated graph parser algorithms.

4 Conclusion and Future Work

In this current paper our research efforts towards an implementation of a general purpose transformation engine performing 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.

As the transformation algorithms arising in our environment are usually structure–driven therefore an automatic transformation code generation (based on our integrated method) is a real opportunity.

A sample transformation has already been implemented and investigated as a benchmark of our research environment, 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.

The implementation of our model transformation system is still in an early phase, therefore further future work is needed at least in the following areas:

- A graphical framework supporting
  - model and rule definitions
  - the generation of transformation code
  - an interface to existing planners
- Optimizing the transformation code by
  - query re–ordering
  - supporting complex (SQL) database queries
- A more formal description of our system

However, a demonstration and a first implementation of the graphical framework is already in progress.

References

[BCLP99] A. Bondavalli, M. Dal Cin, D. Latella, A. Pataricza: High-level Integrated Design Environment for Dependability. Invited paper to WORDS'99, 1999 Workshop on Real-Time Dependable Systems (1999).

[BJR99] G. Booch, I. Jacobson, J. Rumbaugh: The Unified Modeling Language Reference Manual. Addison-Wesley, (1999).

[BMM98] A. Bondavalli, I. Majzik, I. Mura: Automatic dependability analyses for supporting design decisions in UML, (1998).

[BT97] R. Bardohl, G. Taenzer: Defining visual languages by algebraic specification techniques and graph grammars. Technical report, Technical University of Berlin, (1997).

[CHK99] M. Dal Cin, G. Huszerl, K. Kosmidis: Evaluation of safety–critical system based on guarded statecharts. In Proc. HASE'99 4th IEEE International Symposium on High Assurance Systems Engineering, (1999).

[EEHT97] H. Ehrig, G. Engels, R. Heckel, G. Taenzer: A view–oriented approach to systme modelling based on graph transformation, (1997).

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

[LMM99] D. Latella, I. Majzik, M. Massink: 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).

[PW92] J. S. Penberthy, D. Weld: UCPOP: A sound, complete partial order planner for ADL. In Proc. 3rd Int. Conf. on Knowledge Representation and Reasoning. pages 103–114, October, (1992).

[Rei84] R. Reiter: Towards a logical reconstruction of relational database theory. In On Conceptual Modelling, Springer-Verlag, (1984).

[RS95] J. Rekers, A. Schürr: A parsing algorithm for context–sensitive graph grammars. Technical report. Leiden University, (1995).

[Sch94] A. Schürr: Specification of graph translators with triple graph grammars. Technical Report, RWTH Aachen, Fachgruppe Informatik, Germany, (1994).

[Wel94] D. Weld: An introduction to least commitment planning. AI Magazine, (1994).