Dependability – a Byproduct of Model-Driven System Synthesis?

Dr András Pataricza, Budapest University of Technology and Economics, Dept. Measurement and Information Systems, Hungary

Abstract

The main trend in modern system design is the model-driven development of the designated target application. This strategy uses a gradually refined set of semi-formal specifications starting from the initial requirements to define the IT implementation. UML, the Unified Modeling Language, offers a standard graphical notation covering the entire design life cycle. During the last five years, it became obvious that the main advantage of using a semi-formal modeling paradigm is the possibility to analyze the model from the point of view of correctness and conformance to the requirements. Recent standardization efforts at OMG aim at such extensions of UML that can serve as a basis for the mechanized proof of correctness of UML models. However, the modeling of dependability attributes is still in its initial phase of development. The lecture will present an overview on the state-of-the-art dependability modeling techniques.

1 Introduction

One of the most important developments in the field of computer-aided software engineering (CASE) is the more and more widespread use of the model driven approach in IT system synthesis. The main advantage of this approach is that instead of describing the *solution* of the problem to be solved, the synthesis is based upon a gradually refined sequence of *problem formulations*.

The new objective of the Object Modeling Group (OMG) is to define such a synthesis process that enables a high-level automation of refinement steps by defining mappings between the individual problem formulations.

In more details, the proposed *Model Driven Architecture (MDA)* based approach [1] consists of the following main steps:

- *requirement formulation*,
- definition of the *platform independent model (PIM)* which describes the behavior of the designated system,
- definition of the *platform specific model (PSM)* introducing the architectural details needed for a manual or automated implementation.

These models can be constructed by using different notational elements of UML.

However, a pure design model is insufficient for in-depth analysis purposes. For instance, a proof of correctness necessitates the exact formulation of the application-specific requirements. The check of timeliness in real-time applications requires a faithful model of the underlying platform with a main focus on the use of resources, services, and workload [2].

This way the automated analysis requires an enriched view of the target design.

2 Modeling Dependability

Although a complete paradigm for modeling dependability attributes and dependability-increasing components is far away from having a canonized formulation, MDA provides a good potential for integrating design for dependability into the workflow.

- For instance, a PIM can be enriched by assertions serving as a basis for algorithm-based fault tolerance.
- Similarly, PSMs can be extended towards checks and exception handling.
- Detailed models used for defining the interaction between the target application can be extended to include both the qualitative and the quantitative dependability attributes. The General Resource Model (GRM) [2] introduces scenarios for modeling resource utilization and its associated parameters (like performance) to enable a quantitative analysis.

Obviously, this notation can be extended by introducing the corresponding dependability attributes and parameters like faulty scenarios (fault-dependent behavior of the individual resources) and/or quantitative dependability characteristics like reliability and availability measures, respectively. It has to be mentioned that essentially the same approach was already used in [3] before the submission of this standard proposal.

3 Mathematical Model Generation

3.1 Transformation Methodologies

In order to analyze a semi-formal model usually the following main steps has to be carried out:

1. the parts relevant to a specific analysis task have to be filtered out of the complete and usually very complex UML model of the application;
2. the reduced UML model has to be transformed to a model corresponding to the mathematical analysis tool;
3. the analysis has to be performed;
4. the results have to be back-annotated to the original UML model for presentation to the designer.

Obviously, steps 1,2, and 4 require the implementation of complex transformation algorithms. Ad hoc techniques based on a direct program-based transformation frequently result in an intolerably large code, and easily become a quality bottleneck.
Alternatively, model transformation can have a solid mathematical foundation. The most promising approach is the use of graph transformations[1]. Graph transformations are a generalization of the traditional Chomsky grammars to graphs [4]. Simple rules define the elementary steps transforming the fragments of the source graph (i.e. the UML model of the application) to the target graph (e.g. a Petri-net subnet). This approach is even able to support the design of UML transformation rules in UML itself [5]. Fortunately, any of the above mentioned approaches may generate mathematical models of a good quality in the terms of the size and redundancy of the model. Additionally, the model generation time is usually negligible compared to the model analysis time [6].

3.2 Typical analysis tasks

The first and most basic task in the assurance of dependability is the assurance of the fault freedom of the software design, as according to the statistics a portion as large of 75% of all operational failures has software-related origins.
The first factor influencing the quality of the software is the specification formulated as the initial model of the design flow. Several tools like Paradigm of Computer Associates offer a built-in-check feature for the basic syntax of the UML model.
Obviously, the correctness of the syntax does not guarantee alone the correctness of the design, especially in the case of mission-critical applications. Here a basic requirement is the completeness and consistency of the design [7] resulting in a predefined, deterministic reaction of the system to arbitrary input sequences. These checks can be performed on UML models in the same form [8].
The dynamic correctness of the design is one of the most crucial points in the assurance of the functionality. First industrial strength tools are available with an integrated check for embedded systems [9].
Academic research projects include the integration of qualitative and quantitative fault models into the functional UML model and their formal analysis.

4 Conclusions

Obviously, UML and automated code generation will drastically increase the productivity of IT design. Formal methods can increase the quality of the design and a systematic reuse of the best practice ideas from the field of dependability can lead to a seamless integrated environment for dependability.

5 Literature

[1] Miller, J.; Mukerji J.: Model Driven Architecture (MDA). OMG document No ormsc/2001-07-01

[2] Response to the OMG RFP for Schedulability, Performance, and Time. Revised Submission. OMG document No ad/2001-06-14

[3] A. Bondavalli, I. Majzik and I. Mura: Automated Dependability Analysis of UML Designs. Proc. ISORC'99, the 2nd IEEE International Symposium on Object-oriented Real-time Distributed Computing, 1999, pp 139-144.

[4] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe:. Handbook of Graph Grammars and Computing by Graph Transformations, World Scientific, 1997.

[5] Varró, D.; Varró G.; Pataricza, A.: Designing the Automatic Transformation of Visual Lan-

[1] This approach is in the main focus of the DFG Priority Programme *Software Specification - Integration of Software Specification Techniques for Applications in Engineering* (http://tfs.cs.tu-berlin.de/SPP)

guages Journal of Science of Computer Programming (in press).

[6] 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. Int. J. of Computer Systems - Science & Engineering, Vol. 16, No. 5, 2001, pp 265–275.

[7] N. G. Leveson. SAFEWARE: System Safety and Computers. Addison Wesley, 1995.

[8] Z. Pap, I. Majzik, and A. Pataricza. Checking general safety criteria on UML statecharts. In Computer Safety, Reliability and Security (Proc. 20th Int. Conf., SAFECOMP-2001, .pp.

[9] Bienmueller, T.; Damm, W.; Wittke, H.: The STATEMATE verification environment – making it real. Proc. 12th Int. Conf. on Comp. Aided Verification, LNCS-1855, pp. 561–567. Springer, 2000.