High-level Integrated Design Environment for Dependability (HIDE)

Andrea Bondavalli*, Mario Dal Cin**, Diego Latella* and Andras Pataricza***

*CNUCE Istituto del CNR, Via S. Maria 36, 56126 Pisa, Italy
** IMMD3, Univ. Erlangen-Nürnberg, Martensstraße 3, D-91058 Erlangen, Germany
*** Technical University of Budapest - Dept. of Measurement and Information Systems, Muegyetem rkp 9, H-1521 Budapest, Hungary,

Abstract

For most 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 prior of checking whether the system fulfils its objectives or needs some re-design. Although a thorough system specification surely increases the level of confidence that can be put on a system, it is insufficient to guarantee that the system will adequately perform its tasks during its entire life-cycle. The early evaluation of system characteristics like dependability, timeliness, and correctness, is thus necessary to assess the conformance of the system under development to its targets. This paper presents some activities currently performed towards an integrated environment for the design and the validation of dependable systems.

1 Introduction

Computer controlled systems are used in many application fields, with different levels of criticality requirements. A common characteristic of such systems is the increasing complexity in intrinsic terms (management of distribution, redundancy, layering of functionalities, etc.) and of the in-the-field operation (interfaces towards the environment, timing constraints, criticality of the controlled applications, etc.). The increasing need for effective design has contributed to push for the development of standardized and well-specified design methods and languages, which allow system developers to work with a common platform of design tools. The Unified Modeling Language (UML) [19] is a general-purpose visual modeling language that is designed to specify, visualize, construct and document artifacts of a software system. It provides a series of diagrams with a fine granularity to specify object models and has been widely accepted as an object oriented software design language in the software engineering community. In this respect, UML is expected to become a de-facto standard for the design of a variety of systems from small embedded control systems to large and complex open systems.

An effective design process should also include an early validation of the concepts and architectural choices underlying system design. The early evaluation of system characteristics like dependability[12], timeliness and correctness, necessary to assess the compliance of the system under development to its targets, becomes especially important for designing systems supporting critical applications. The simultancously increasing complexity and dependability requirements of computer controlled systems have, in fact, exposed the limits of the validation techniques traditionally used in industry, like code review, testing, Fault Trees or Failure Mode Error Analysis. Moreover, new technologies such as Object-Oriented Design and Programming, Advanced User Interfaces, Hardware-Software Codesign, the use of COTS (Commercial Off the Shelf) software components, all present new challenges for the validation process. The traditional validation techniques are being more and more complemented with advanced validation techniques, such as Formal Verification, Model based Dependability Evaluation, Schedulability Analysis, Fault Injection. These techniques are not aimed to replace the traditional validation techniques, but should rather be integrated with them.

The validation of designs described using UML is the main objective of the European ESPRIT project HIDE. Parterns involved are FAU (Erlangen, Germany), CPR-PDCC (Pisa, Italy), TUB (Budapest, Hungary), Mid GmbH (Germany), Intecs Sistemi (Pisa, Italy). 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. Formal verification, quantitative and timeliness analysis tools will be made available to the designer within the design environment, and the models will be derived automatically from the UML specification.

The rest of this paper is structured as follows. Section 2 describes the HIDE structure and the overall project, Section 3 details a part of the work currently performed and the results achieved so far, in the directions of formal verification and of quantitative analysis, where two complementary approaches are followed. Last section 4 draws some conclusions and identifies the next steps.

2. The HIDE project

While a relevant effort is being devoted to the development of standardised design languages and methods, such as the recently created UML, much less attention has been dedicated up to now to the integration of the design technologies with the validation techniques.

However, it is clearly understood that good product implementation alone does not assure a proper quality of the services (QoS) delivered by the product. Thus, high QoS also has to be assured during the design process. In spite of this necessity, even the most advanced system design methodologies, such as CASE and HW-SW co-design, lack the proper support of QoS assurance for the entire design process and product life cycle.

At present, the validation of system design is usually performed as a separate project phase postponed to the end of the design process. This practice however suffers from the following weak points:

- Typically, simulation based validation is used for checking the conformance of the system model to the initial specification. However, experimental validation assures only a high likelyhood of correctness, but no proof of it.
- No integrated support is currently offered for quantitative validation of the system concepts, thus leaving the quantitative aspects uncovered. In the past, numerous practical projects suffered significant delays and cost overhead due to post-upgrades.
- There is a trend to use COTS (commercial off-the-shelf) products in order to reduce both design and manufacturing costs. This trend raises the question how to assure dependability in a system built of average quality, non ultra-reliable components. Moreover, dependability is a crucial cost factor in the after-manufacturing phases of the product life cycle (e.g., maintenance). Pure functional specification leaves the system behaviour undefined in the presence of faults.

Despite of the practical importance and the results achieved during the past decades, there are major obstacles in the assurance of the QoS by formal methods and mathematics. The need of a deep knowledge of sophisticated techniques is one of the most serious obstacles to performing the necessary checks on the systems design at early phases of design and development. In fact, while manufacturers of critical systems are prepared and have a considerable experience on the validation of their **products** (driven by the need of certifying and having their systems to be accepted by customers), there are major difficulties and much less expertise in the early validation of system **design**. The large set of analyses and techniques and their level of sophistication require a huge investment in staff skill and time. This discourages designers and builders to apply such analysis methods in the phase of the design and implementation process in which they are most effective.

The HIDE project aims at proposing a convincing and general answer to the need for early validation of system design. It aims at integration among the design, validation and verification techniques for complex software/hardware systems, through a transformational approach that targets the most common analysis tools. Formal verification, quantitative and timeliness analysis are the main validation techniques identified so far. The use of formal methods for the specification and verification of properties of systems is one methodological improvement of the system development process, which, together with other techniques, allows to reach high quality standards. The class of systems targeted by HIDE are characterised by various, often severe real-time constraints. The Project draws results from the methods that have been developed for the fulfilment of hard real-time constraints. The architecture of the software has to be precisely described, by some interaction with the designer so that a tool will be able to perform the timing (or schedulability) analysis. The feedback is then provided to the designer, highlighting timing failures, bottlenecks and identifying to a certain level the possible causes for the design failures and the corresponding recovery actions.

Amongst the approaches commonly adopted to evaluate dependability attributes, analytical modelling has proven to be very useful and versatile. Especially during design, models show their usefulness and potentialities. They allow to compare different architectural and design solutions and to run sensitivity analyses identifying both dependability bottlenecks and critical parameters to which the system is highly sensitive.

The transformations will adhere to UML as much as possible. The evaluation results must have tangible value to the modeller working with UML and the input/output of evaluations must clearly correspond to UML modelling elements. The estimation and the prediction of the system properties performed must obviously have an acceptable level of confidence. The larger model size and run-time requirements of the automatically derived models should not strongly limit the target field of applications in comparison with the manually composed ones. The entire background mathematics will be completely hidden to the designer, thus eliminating the need for both a specific expertise in abstract mathematics and the tedious re-modelling of the system for mathematical analysis. The results gained will be automatically back-annotated for presentation into the same UML model, providing the designer with a source of precious information for driving the subsequent design choices.

The project aims also to supporting features for the design and validation strategy of embedded real-time dependable systems. These features are 1) a support for inclusion of redundancy by means of a library of UML stereotyped classes supporting class based redundancy in object oriented methodologies, and 2) the definition and formalisation of a process modelling method, intended to be used to guide the process of designing and validating embedded real-time dependable systems and specialised for these purposes. UML is a language for different kind of modelling purpose. Since it is not a process for software development there is the need to have a guidance for the system modelling. The model for the evaluation process has the task to deliver a knowledge base, guidelines for proven best practices, assistant tools for conducting an evaluation experiment etc.

The HIDE framework will thus integrate in a user-friendly way the de-facto standard design language UML with a set of validation, verification and evaluation techniques for assuring the QoS of the system still during the early design phases without the need of special skills or extreme efforts by the designer. This will allow to shorten the necessary verification and validation cycle, with a consequent saving in the associated costs. Design refinement will be driven by the information gained during the validation process, thus allowing adequate system designs to be produced before implementation and experimental (in the field) validation take place.

The HIDE software platform will be based on the commercial UML-based tools, provided by the industrial project partners, but at the same time will be general enough to allow an easy integration with other UML-based toolsets. The openness of the approach allows the further integration of other validation and analysis tools.

3. Activities performed

UML is a standard modeling language [8, 20]. It may be used for visualizing, specifying, constructing and documenting several aspects of - or views on - systems. It is based on the object-oriented paradigm and it is heavily graphical in its nature. Different diagrams are used in order to describe different views on a system. For instance, Class Diagrams show sets of classes and their relationships thus addressing the static design view on systems. On the other hand, Statechart Diagrams show state machines thus addressing the dynamic behavior of systems and their components.

At present UML is merely a semiformal notation, with a well-defined syntrax but with little formal semantics attached to the individual diagrams nor is there a formally defined semantics for the integration of the diagrams. Typical instances are diagrammatic techniques with precise rules that specify conditions under which constructs are allowed and textual and graphical descriptions with limited checking facilities.' [16]. Semiformal techniques help to move from initial natural language to formal specifications. It is impossible to apply rigorous analysis techniques to evaluate a UML model as it is. In other words, in order to evaluate a visual UML-model the model has to be enriched with additional information and then be transformed to a model with a precise semantics. This enrichment has been performed in the different cases trading-off the adherence.

3.1 Formal verification

Formal verification is a hot topic nowadays in the field of system engineering, especially for the development of critical dependable systems. Formal methods are mathematically-based techniques that can offer a rigorous and effective way to model, design and analyze computer systems. They have been a topic of research for many years and the question now is whether these methods can be effectively used in industrial applications. Tool support is necessary for a full industrialization process and there is a clear need for improved integration of formal method techniques with other software engineering practices. Several approaches to the application of formal methods in the development process have been proposed. They differ in the degree of involvement of the method within the development process, the simplest being the mere use to write rigorous specifications. Further steps are the (more or less automated) generation of code from the formal specification, and the use of formal verification as an additional validation technique aimed to reach a high level of confidence of the correctness of the system.

Within the HIDE Project, a mapping of a subset of UML Statechart Diagrams to Kripke Structures has been formally defined [14]. This translation defines a reference formal operational semantics for UML Statechart Diagrams within HIDE. Formal semantics are obviously necessary whenever formal verification is an issue; in particular, the Kripke Structure resulting from the translation of a Statechart Diagram can be conveniently used as a basis for model checking. To that purpose it is of course necessary to specify the requirements against which the model has to be checked. Such requirements are usually expressed by a temporal logic formula or by another automaton transformed to a Kripke Structure. In the first case the formula is not part of the UML model, since the UML does not provide an explicit notation for temporal logic. In the second, the requirement can be expressed again as a (simple) Statechart Diagram and its resulting semantics can be used for model checking the (semantics) of the original Statechart Diagram.

A nice aspect of the translation proposed in [14] is that it is *parametric* on some aspects which are not (yet) completely defined for UML in order to offer a unique notation for specific run-time systems. In particular, parametricity of the semantics definition w.r.t. transition priorities, makes it suitable for describing the behavior of systems under different priority schemes.

Based on the above mentioned semantics, a translation from UML Statechart Diagrams to PROMELA, the modeling language of the SPIN verification tool [11], has been formally defined and proven correct [13]. This allows the user to apply to our subset of UML Statechart Diagrams the verification techniques supported by SPIN, including Linear Time Logics model-cheking. Two implementations of this translation are available: one has been developed in CNUCE and is written in Standard ML [9]. The other is running at the Technical University of Budapest and uses database technology [5]. Several experiments have been performed on this last implementation, including the modeling and verification of the version of the production cell presented in [4]. The interested reader is referred to [17].

In [10] a different approach to UML Statechart Diagrams model-checking based on the semantics proposed in [14] has been investigated. It uses Branching Time Logics as opposed to the SPIN experiment. An implementation of this second prototype is in progress.

Several other approaches have been proposed in the literature for the definition of a formal semantics of UML Statechart Diagrams, e.g. [6, 15, 21], and much more has been done for classical statecharts.

To our understanding, transition priorities are dealt with neither in [21], where also state refinement is not allowed, nor in [6], where model checking is addressed. Both transition priorities and state refinement constitute main issues in our work.

The approach we followed in [14] is similar to that proposed in [18] for classical statecharts but it takes into consideration the peculiarities of the UML Statechart Diagrams relevant for the considered subset of the notation. On the other hand, it shares the relative simplicity of the work proposed in [18].

In [15] all interesting aspects of UML Statechart Diagrams semantics are covered. Unfortunately, no correctness result for the proposed semantics is provided. More emphasis is put on implementation related issues as the work constitutes a basis for a PROMELA/SPIN based model-checker for UML Statechart Diagrams. In [15] a ``flat'' representation of UML Statechart Diagrams is used and the authors claim that such a representation is better suited for model-checking purposes than the hierarchical one used in [14].

Using a hierarchical representation for UML Statechart Diagrams (syntax), not only has no negative impact on tools development, but, rather, it helps very much in carrying on correctness proofs; all interesting results presented in [14] and in [13] are proven inductively and such proofs heavily exploit the hierarchical structure of our representation, which is also the basis of the structure of our semantics deduction system.

3.2 Quantitative analysis

Two compementary approaches have been followed for providing quantitative analyses of dependability attributes. Two translations from UML subsets to Timed and Generalized Stochastic Petri Nets [1] have been defined. On one hand, this approach allows to use the elaborate and well established Petri Net tools for the quantitative analysis of UML-models. On the other hand, it integrates the use of Petri Nets into the object-oriented modeling paradigm of UML. For example, the generated Petri Nets models can be extended by modeling aspects like the integration of fault models, difficult to express directly in UML. A trade-off has to be made between the degree of details in modeling and the degree of possible automation of the analysis process which, of course, depends on the size of the state space of the model.

Both these two trasformations use PANDA [2], as Petri Net analysis tool. It allows to annotate transitions with guards and to use state dependent capacities for arcs. Moreover, PANDA accepts not only exponential distribution functions, but also non-exponential ones (Erlang-k, Gamma, Hyperexponential, Normal, Lognormal, Weibull, etc.). Dependability measures can be specified by reward functions. Reward functions are built from so-called characterizing functions like: Mark(place). This function delivers the number of tokens in a place. PANDA computes the expectation value of a reward function at a point in time (e.g. availability or throughput) as well as accumulated rewards. PANDA is available for shared and distributed memory platforms as well.

A first quantitative translation maps UML Structural Diagrams (use case, class, object, and deployment diagrams) to Timed and Generalized Stochastic Petri Nets for dependability assessment [3, 4]. This translation copes with the state explosion by starting from simple models, and making them more and more complex and detailed by refining the relevant parts of the system. It tries to capture only the features relevant to dependability neglecting all other information. It:

- allows a less detailed but system-wide representation of the dependability characteristics of the analyzed systems.
- provides early, preliminary evaluations of the system dependability during the early phases of the design. This way, a designer can easily verify the compliance of the system under design to the predefined requirements on dependability attributes.
- deals with various levels of detail, ranging from very preliminary abstract UML descriptions, up to the refined specifications of the last design phases. Higher level UML models (structural diagrams) are available before the detailed, low levels ones. The analysis on these rough models provides indications about the critical parts of the system, which require a more detailed representation. In addition, by using well defined interfaces, such models can be augmented by inserting more detailed information coming from refined UML models of the identified critical parts of the system. These might be provided by other transformations dealing with UML behavioral and communication diagrams as the transformation described next.

The structural UML diagrams that form the input of such transformation do not have a formal semantics; moreover the specification this set of diagrams provides might be incomplete or ambiguous, so formal correctness of this transformation cannot be provided.

Another quantitative transformation focuses on a UML-dynamic model comprising statecharts and sequence diagrams [7]. A rigorous mapping from the semiformal language to a mathematical language possessing a precise semantics has been worked out. We have developed a framework for deriving Stochastic Reward Nets (an extension of Generalized Stochastic Petri Nets) from the subset of the UML comprising use cases, sequence diagrams and a specific sub-class of statecharts comprising so-called Guarded Statecharts (GSC) . Guarded Statecharts are suited for modeling dependable embedded systems in which also non-deterministic behavior can be modeled.

Interactions between tasks of the control software of embedded systems are modeled by guarded state transitions: this corresponds to an asynchronous synchronization pattern between tasks. This pattern is inherently multithreaded, because it models a message being passed to another object without the yielding of control. Guards are annotated with state transition probabilities and/or branching probabilities (weights).

For a dependability analysis the GSC-models are directly transformed to Stochastic Reward Nets. State transitions with time delay are transformed to timed transitions, those without time delay to immediate transitions. Guards become guards of Petri Net transitions. The resulting nets are amenable to a rigorous analysis.

Since our GSCs model mainly embedded systems, where the communication between several components is of importance, we made it possible by the transformation to model communication errors explicitly within the Petri Nets. Within the statecharts communication errors are simply modeled by substituting (with certain probabilities or failure rates) guards by True (e.g., the sensor signal is stuck-at-active) or False (e.g., the actuator signal is stuck-at-inactive or not observed by the hardware). This way, lost or spurious signals can be modeled. State perturbations are modeled by additional states and/or additional state transitions guarded by fault-tree like guards.

4. Concluding remarks

This paper reports some ongoing activity within the HIDE project. The HIDE project aims at proposing a convincing and general answer to the need for early validation of system design, supporting also features for the design and validation strategy of embedded real-time dependable systems. It intends to contribute to the integration among the design, validation and verification techniques for complex software/hardware systems, through a transformational approach that targets the most common analysis tools. Part of the work performed so far has been described, mainly concerning the transformations for achieving formal verification and quantitative analysis, where two complementary approaches are followed. Work is currently in progress for merging these two approaches and some investigation started for dealing with timeliness analyses in such context. Moreover demonstrators and first implementations of the HIDE repository and of transformations are underway.

References

[1] M. Ajmone Marsan, G. Balbo and G. Conte, “A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems,” ACM TOCS, Vol. 2, pp. 93-122, 1984.

[2] S. Allmaier and S. Dalibor, “PANDA - Petri net analysis and design assistant,” in Proc. Performance TOOLS’97, Saint Malo, France, 1997.

[3] A. Bondavalli, I. Majzik and I. Mura, “Automated Dependability Analysis of UML Designs,” in Proc. ISORC’99 - 2nd IEEE International Symposium on

Object-oriented Real-time distributed Computing, Saint Malo, France, 1999, pp. 139-144.

[4] A. Bondavalli, I. Majzik and I. Mura, "Automatic Dependability Analysis for Supporting Design Decisions in UML," in Proc. HASE99 - 4th IEEE High Assurance System Engineering Symposium, Washington D.C., USA, 1999, pp. 64-71.

[5] A. Borchet, M. Dal Cin, J. Javorskky and C. Szasz, "Specification of the HIDE environment," ESPRIT LTR Project 27439 - HIDE (High-Level Integrated Design Environment for Dependability) Technical Report HIDE/D3/TUB/1/v2, 1998.

[6] J.M. Broersen and R.J. Wieringa, "Interpreting UML-statecharts in a modal μ-calculus," Unpublished manuscript, 1997.

[7] M. Dal Cin, G. Huszerl and K. Kosmidis, "Evaluation of Safety-Critical Systems based on Guarded Statecharts," in Proc. HASE'99 Fourth IEEE International Symposium on High Assurance Systems Engineering, Washington DC, USA, 1999, pp. 37-45.

[8] M. Fowler and K. Scott, "UML Distilled. Applying the Standard Object Modeling Language," Addison-Wesley, ISBN 0-201-32563-2, 1997 1997.

[9] E. Giusti and D. Latella, "Implementazione in SML di un traduttore da automi gerarchici a PROMELA," Consiglio Nazionale delle Ricerche, Istituto CNUCE Technical Report, CNUCE-B4-1998-018 (In italian), 1998.

[10] S Gnesi, D. Latella and M. Massink, "Model checking UML statechart diagrams using JACK," in Proc. HASE99 - 4th IEEE High Assurance System Engineering Symposium, Washington D.C., USA, 1999, pp. 46-55.

[11] G. Holzmann, "The model checker SPIN.," IEEE Transactions on Software Engineering, Vol. 23, pp. 279--295, 1997.

[12] J.C. Laprie, "Dependability-Its Attribues, Impairments and Means," in "Predictably Dependable Computing Systems", B. Randell, J. C. Laprie, H. Kopetz and B. Littlewood Ed., Springer-Verlag, 1995, pp. 3-24.

[13] D. Latella, I. Majzik and M. Massink, "Automatic verification of UML statechart diagrams using the SPIN model-checker," Consiglio Nazionale delle Ricerche, Istituto CNUCE, Technical Report CNUCE-B4-1999-008, 1999.

[14] D. Latella, I. Majzik and M. Massink, "Towards a Formal Operational Semantics of UML Statechart Diagrams.," in Proc. IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Oriented Distributed Systems, Florence, Italy, Feb. 15-18, 1999.

[15] J. Lilius and I. Paltor Porres, "The semantics of UML state machines," Turku Centre for Computer Science Technical Report 273, 1999.

[16] M.D Fraser, K. Kumar and V.K. Vaishnavi, "Strategies for incorporating formal specifications in software development," Communications of the ACM, Vol. 37, pp. 74-86, 1994.

[17] I. Majzik and J. Javorszky, "Formal verification of UML statecharts: Case studies," Dept. of Measurement and Information Systems - Technical University of Budapest, Technical Report MITUB-TR-99-05, 1999.

[18] E. Mikk, Y. Lakhnech and M. Siegel, "Hierarchical automata as model for statecharts," in Proc. Third Asian Computing Science Conference. Advances in Computing Sience - ASIAN'97, Lecture Notes in Computer Science n. 1345, 1997, pp. 181--196.

[19] * Rational Software, * Microsoft, * Hewlett-Packard, * Oracle, * Sterling Software, * MCI Systemhouse, * Unisys, * ICON Computing, * IntelliCorp, * i-Logix, * IBM, * ObjecTime, * Platinum Technology, * Ptech, * Taskon, * Reich Technologies and Softeam, "Object Constraint Language Specification," version 1.1, 1997.

[20] J. Rumbaugh, I. Jacobson and G. Booch, "The Unified Modeling Language Reference Manual," Addison-Wesley, ISBN 0-201-30998-X 1999.

[21] R.J. Wieringa and J. Broersen, "A Minimal Transition System Semantics for Lightweight Class and Behavior Diagrams.," in Proc. ICSE98 Workshop on Precise Semantics for Software Modeling techniq.