

Report on the 2nd Workshop on Model Development and Validation – MoDeVa

Benoit Baudry¹, Christophe Gaston², and Sudipto Ghosh³

¹INRIA, France

benoit.baudry@irisa.fr

²CEA/LIST, France

christophe.gaston@cea.fr

³Colorado State University, USA

ghosh@cs.colostate.edu

1 Introduction

Rigorous design and validation methods appear to be more and more necessary in an industrial context. Software systems are becoming increasingly large and complex, and run the risk of serious failures from unpredictable behaviors resulting from interactions between sub-systems. Without proper standardization of modeling notations and approaches, human beings find it difficult to understand the systems.

Object-oriented and component-oriented design approaches in general, and the Model Driven Architecture (MDA) approach in particular attempt to overcome this problem. Formal methods have been intensively applied to evaluate the reliability of systems. These methods generally require adequate specification and structuring languages to describe the parts of the system under validation.

A major problem encountered when trying to combine design and validation features is that structuring languages suitable for one feature are generally not suitable for the other. For example, the object-oriented paradigm is suitable for large scale system design, since it allows anthropomorphic design based on service exchanges of basic entities. However, this paradigm is not suitable (without restriction) for validation activities, since any enrichment of a system is likely to cause loss of global properties. In the opposite way, the modular paradigm ensures properties preservation but the price to pay is a higher level of design difficulty.

The Model Design and Validation (MoDeVa) workshop aimed at being a forum for researchers and practitioners with varying backgrounds to discuss new ideas concerning links between model-based design and model-based validation. Topics of interest included design processes that support complex system modeling and formal or semi-formal refinement mechanisms. Model-based testing, languages to describe models (e.g., UML), approaches such as model-driven engineering, model driven architecture, algebraic languages, automata-based language, first order language, and propositional languages were considered. The first edition of MoDeVa took place in Rennes in France in 2004. MoDeVa was a satellite workshop of the ISSRE conference. This year MoDeVa was a satellite workshop of MODELS. This paper is a report on this second edition.

The workshop had two parts – presentation of position papers followed by focused discussion by two separate groups. Section 2 presents summaries of the 9 papers selected for presentations. Section 3 summarizes the conclusions of the workshop.

2 Paper Summaries

The workshop selected 9 papers out of 17 submissions. One of the main selection criteria was that the papers clearly demonstrate a step forwards using formal approaches within a software development methodology. The use of formal approaches may incorporate the use of formal tools (proving tools, model checkers, formal testing tool) and include formal definition of semantics to deal with structuring or refinement mechanisms.

- [1] proposes a formal testing methodology dedicated to the Common Criteria ISO standard.
- [2] describes a taxonomy of faults that occur in UML design.
- [3] proposes a model based testing approach for UML specifications.
- [4] presents a rigorous and automated based approach for the behavioral validation of control software systems.
- [5] describes an approach towards increasing the robustness of the UML refinement machinery.
- [6] suggests a systematic modeling method for embedded systems.
- [7] explores the problem of ensuring correctness of model transformations.
- [8] describes a round trip engineering process that supports the specification of UML models and focuses on the analysis of specified natural language properties.
- [9] proposes an interaction-based approach for use case integration.

[1] Test Generation Methodology Based on Symbolic Execution for the Common Criteria Higher Levels – Alain Faivre, Christophe Gaston

In the field of security software, the Common Criteria (CC) constitutes an ISO standard for the evaluation of products and systems from Information Technologies. The international recognition of the Common Criteria justifies the investment undertaken by the manufacturers to obtain the certification of their products. The evaluation criteria are defined according to the Evaluation Assurance Level (EAL). There are seven EALs: EAL1 to EAL7, in an increasing order of security demand. For the upper levels of evaluation, the use of formal methods is mandatory. In that case, supplies intended to realize evaluation activities must contain components associated to modeling, proof and test. This contribution proposes a methodology and a tool (AGATHA) which allows covering the requirements associated to test generation for the upper levels of the Common Criteria. In that case, the criterion used to stop the test generation activity is defined by the standard for EAL7 as follows: the generated test case set covers all functions of the reference model. Each function must be covered “complete” way (although the term complete remains ambiguous in CC definitions). The strategy presented in the paper provides a formal meaning to this criterion and associated test generation techniques.

[2] A Taxonomy of Faults for UML Designs – Trung Dinh-Trong, Sudipto Ghosh, Robert France, Benoit Baudry, Franck Fleurey

As researchers and practitioners start adopting model-based software development techniques, the need to rigorously evaluate design models is becoming apparent. Evaluation techniques typically use design metrics or verification and validation approaches that target specific types of faults in the models. Fault models and taxonomies may be used to develop design techniques that reduce the occurrence of such faults as well as techniques that can detect these faults. Fault models can also be used to evaluate the effectiveness of verification and validation approaches. A taxonomy of faults that occur in UML designs was presented along with a set of mutation operators for UML class diagrams.

[3] Generating Test Data to test UML Design Models – Trung Dinh-Trong, Sudipto Ghosh, Robert France, Anneliese Andrews

This paper presents an approach to generating inputs that can be used to test UML design models. A symbolic execution based approach is used to derive test input constraints from a Variable Assignment Graph (VAG), which presents an integrated view of UML class and sequence diagrams. The constraints are solved using Alloy, a configuration constraint solver, to obtain the test inputs.

[4] Using Process Algebra to Validate Behavioral Aspects of Object-Oriented Models – Alban Rasse, Jean-Marc Perronne, Pierre-Alain Muller, Bernard Thirion

This paper presents a rigorous and automated based approach for the behavioral validation of control software systems. This approach relies on meta-modeling, model-transformations and process algebra and combines semiformal object-oriented models with formal validation. Validation of behavioral aspects of object-oriented models is performed by using a projection into a well-defined formal technical space (Finite State Process algebra) where model-checkers are available (e.g., LTSA; a model checker for Labeled Transition Systems). The approach also targets an implementation platform which conforms to the semantics of the formal technical space; in turn, this ensures conformance of the final application to the validated specification.

[5] On the Definition of UML Refinement Patterns – Claudia Pons

This paper describes an approach towards increasing the robustness of the UML refinement machinery. The aim of this work is not to formalize the UML notation itself, but to substantiate a number of intuitions about the nature of possible refinement relations in UML, and even to discover particular refinement structures that designers do not perceive as refinements in UML.

[6] A Modeling Method for Embedded Systems – Ed Brinksma, Angelika Mader, Jelena Marincic, Roel Wieringa

This paper suggests a systematic modeling method for embedded systems. The goal is to derive models (1) that share the relevant properties with the original system, (2) that are suitable for computer aided analysis, and (3) where the modeling process itself is transparent and efficient, which is necessary to detect modeling errors early and to produce model versions (e.g. for product families). The aim is to find

techniques to enhance the quality of the model and of the informal argument that it accurately represents the system. The approach is to use joint decomposition of the system model and the correctness property, guided by the structure of the physical environment, following, e.g., engineering blueprints. The approach combines Jackson's problem frame approach with a stepwise refinement method to arrive at provably correct designs of embedded systems.

[7] Model Transformations Should Be More Than Just Model Generators – Jon Whittle and Borislav Gajanovic

Model transformations are an increasingly important tool in model-driven development (MDD). However, model transformations are currently only viewed as a technique for generating models (and, in many cases, only code). Little is said about guaranteeing the correctness of the generated models. Transformations are software artifacts and, as such, can contain bugs that testing will not find. This paper proposes that, in fact, model transformations should do more than just generate models. In addition, they should generate evidence that the generated models are actually correct. This evidence can take the form of precise documentation, detailed test cases, invariants that should hold true of the generated models, and, in the extreme case, proofs that those invariants do actually hold. The hypothesis is that there is enough information in the definition of a transformation to provide evidence that certain properties of the generated model are true. Such information is usually left implicit. By making that information explicit and annotating the generated model, a consumer of the model increases his/her confidence that the model does what it is supposed to do.

[8] Automated Analysis of Natural Language Properties for UML Models – Sascha Konrad, Betty H.C. Cheng

It is well known that errors introduced early in the development process are commonly the most expensive to correct. The increasingly popular model-driven architecture (MDA) exacerbates this problem by propagating these errors automatically to design and code. This paper describes a round trip engineering process that supports the specification of a UML model using CASE tools, the analysis of specified natural language properties, and the subsequent model refinement to eliminate errors uncovered during the analysis. This process has been implemented in SPIDER, a tool suite that enables developers to specify and analyze a UML model with respect to behavioral properties specified in terms of natural language.

[9] Interaction-Based Scenario Integration – Rabeb Mizouni, Aziz Salah, Rachida Dssouli

This paper proposes an interaction-based approach for use case integration. It consists of composing use cases automatically with respect to interactions specified among them. A state-based pattern is defined for each of these interactions. A use case interaction graph is synthesized, which serves the detection of not only unspecified, but also implied use case invocations. Additional constraints are added to the system in order to remove such illicit interactions, called interferences.

3 Group Discussions

The audience of the workshop was completely representative of the topics of the workshop. There were people working in the research field of design and people working in the research field of formal methods. The discussion session aimed at helping to bridge the gap between those two communities. Therefore, the attendees formed two groups. One group had to discuss and provide hints to designers about the challenges in the scope of formal treatment for the UML. The second group had to isolate particular aspects of the UML language, for which a formal treatment would be useful: this group chose to discuss issues related to the defining, building, using UML profiles and capturing their semantics.

3.1 Formal Treatment of UML Models

The UML is used in various ways by software developers. Some use it informally, mainly for the purpose of sketching and communicating system requirements and design. Their main requirement is flexibility to enable the representation of mental model of the system to be implemented. They generally do not intend to use these models for any form of rigorous analysis and hence, formal treatments do not apply to them.

Formal methods will be useful for development environments that focus on critical systems. Currently a number of companies use existing methodologies, languages, and tools such as B, SCADE, and SDL. They would like to use a uniform notation that would enable them to distribute models to different groups for implementation. They have considered the UML, which gives them a rich syntax for model development. However, the development of critical systems requires formal approaches for analyzing model properties. The lack of completely formal semantics in the UML prevents them from using it as it stands. For this reason, researchers have developed mappings from UML to various formal notations which are input languages for existing analysis tools. This leads to: 1) lack of uniformity in the expression of semantics; 2) use of similar models with different and hidden semantics. We need to define a formal UML semantics independent of any particular tool.

The UML is huge and deals with a lot of industrial aspects. Some of these aspects clearly go beyond software development. If we want to deal with critical system design, we should be able to restrict the UML to views that are relevant to this purpose. This restriction must be as small as possible. Indeed, the more a language introduces keywords and views, the more providing it with a formal semantics may lead to inconsistencies. Once the relevant parts of the UML are identified, an interesting approach would be to develop denotational semantics for them. We propose to follow a denotational approach because the UML is complex. We believe that providing the UML with only an operational semantics would again raise the problem of inconsistency between views. This is due to the fact that the UML allows the management of several views of the same problem. Links between those views need to be clearly stated. Thus, in order to provide a consistent semantics to the UML, we believe that a rigorous framework, such as set theory or category theory, is mandatory. Moreover the use of a denotational semantics limits the risk of interpretation errors when using formal tools to treat UML specifications. Indeed,

having a denotational semantics for (a part of) the UML and a denotational semantics for the entry language of a formal tool implies to define the bridge between the two semantics by means of relation and mathematical proofs. Contrarily to such an approach, in an operational semantics approach, the bridge between two semantics is generally made by means of a translation and possibly with no hints about the correctness of the translation. Thus, a denotational approach should provide a good framework to define semantics independently of any tool.

3.2 UML Profiles

Profiles tailor the UML to specific areas - some for business modeling; others for particular technologies. For example, the Object Management Group has standard profiles for CORBA, EDOC, and patterns.

Discussions underlined the importance to have a well defined methodology to build profile in order to better understand its objective, role, use and semantics. Such methodology is already used by some users, namely for defining the OMG standard profiles, but has to be widespread in the whole community. According to the UML2 standard and the current practices, the main points are the following:

- a) Profiles are based on the domain meta-model, so first:
 - Build the model of the concepts required by the domain (i.e.: the domain meta-model) with the modeling formalism you want. UML is very often used to create this domain meta-model that could facilitate the next step of mapping the domain model to UML meta-model.
 - Describe the semantics of the meta-model (either with informal text or any formalism that seems useful)
- b) Profiles are implemented in UML through two steps:
 - Identify the mapping between the profile domain meta-model and the UML meta-models;. Mappings target to identify already existing concepts in the UML meta-model and the standard UML profiles that fit with the domain concepts or that could be extended, specialized to fit with the domain concepts.
 - Implement the profile by formalizing the mapping through definitions of stereotypes, tagged values, constraints, notations, semantic variation points choices, etc. Provide its UML implementation in XMI (as a UML model of the profile implementation).

A profile may contain new standard elements, such as stereotypes and tagged values, and common model elements from the UML library of predefined elements. OCL constraints define notations and can be used to understand the semantics of the new standard elements.

The semantics of a profile must be compliant with the semantics of the meta-model of UML 2.0. Additional well-form ness rules or constraints can never violate these existing in UML 2.0.

Question of which kind of formal semantics is provided by these profiles definitions has conclude that it is centered on static semantics and not covers the dynamic semantics.

The discussion group agreed that the semantics provided by these profile definitions are not sufficient from a formal point of view to capture all that is needed to allow connection to validation tools and automatic code generation.

Profiles may be combined in different manners depending on the granularity and scope. Approaches need to be developed to check the levels of abstractions of the profiles, automatically perform profile combination, and check the consistency of the combination. We need to define development processes that incorporate the use of profiles. Developers need systematic ways to determine which profile must be used on which parts of the model. Appropriate tool support can then be developed.

In addition to ongoing works on defining a profile for embedded systems (MARTE), two subjects have been identified as not sufficiently covered by the existing standard profiles:

1. Reliability: more particularly concerning dynamic behavior (e.g., transition of scenarios)
2. Traceability: general subject, partially treated by SysML for requirement traceability, but not supported for any elements, model evolutions as required in Model Driven Development.

Finally, the main open issue in the context of defining and using profiles seems to be the definition of their dynamic semantics. Several approaches can be used to define the semantics from totally informal to totally formal. They are the following:

1. Develop the semantics in natural language (this one remains mandatory, even if more formal information is given).
2. Use correspondence style rules with examples.

4 Conclusion

The content of discussions led us to draw the following conclusions. First of all, the usage of formal tools to treat UML specifications is really meaningful when dealing with critical system specifications. This is due to the fact that potential users in the field of critical system design require having a simple, totally formally grounded semantics to a subpart of the UML. Using formal tools in a different context makes less sense. Secondly, in order to be compliant with the norm, defining a subpart of the UML to be mathematically grounded could be done using a profile approach. But profile themselves should be provided with a semantics. In the next edition of MoDeVa we propose to concentrate on these issues: What subpart of the UML should be considered in the field of formal treatment? Are there several subparts (possibly overlapping) of the UML to be considered depending on the system design domains considered? How this subpart(s) should be described? How to provide and describe a formal semantics in a way which would be acceptable for the OMG?