Testing and Debugging UML Models Based on fUML

Tanja Mayerhofer
Institute of Software Technology and Interactive Systems
Vienna University of Technology, Austria
mayerhofer@big.tuwien.ac.at

***Abstract*—Model-driven development, which has recently gained momentum in academia as well as in industry, changed the software engineering process significantly from being code-centric to being model-centric. Models are considered as the key artifacts and as a result the success of the whole software development process relies on these models and their quality. Consequently, there is an urgent need for adequate methods to ensure high quality of models. Model execution can serve as the crucial basis for such methods by enabling to automatically test and debug models. Therefore, lessons learned from testing and debugging of code may serve as a valuable source of inspiration. However, the peculiarities of models in comparison to code, such as multiple views and different abstraction levels, impede the direct adoption of existing methods for models. Thus, we claim that the currently available tool support for model testing and debugging is still insufficient because these peculiarities are not adequately addressed. In this work, we aim at tackling these shortcomings by proposing a novel model execution environment based on fUML, which enables to efficiently test and debug UML models.**

***Keywords*-model execution; model debugging; model testing; foundational UML; MDD**

I. Introduction

Over the past decade, *Model-Driven Development (MDD)* gained momentum in academia as well as in practice. Models enable to raise the level of abstraction and therewith to handle the complexity of today's systems [1]. The de-facto standard for modeling software systems is the *Unified Modeling Language (UML)*, which is developed and standardized by the *Object Management Group (OMG).*

With the application of MDD, the software engineering process becomes more model-centric and less code-centric. In other words, models are considered as the main artifacts in the software development process and constitute the single source of information. As a result, the success of the whole software development process relies on these models and their quality. Consequently, there is a need for methods to ensure the quality of models [2], [3].

Methods for ensuring the quality of models can be divided into two fields: static analysis of models, and dynamic analysis of models. Whereas static analysis methods verify the correctness of models by assessing static properties of a model, dynamic analysis methods verify the quality of models by executing them. The work proposed in this paper is concerned with dynamic analysis methods. However, UML models are not executable per se, because UML has no precise and completely specified semantics [4]. Its semantics is informally defined in English prose and this definition is scattered throughout the standard with approximately 1000 pages. Thus, ambiguities arise which lead to the situation that UML models are diversely interpreted and executed. To overcome this problem, various formal semantics definitions have been developed. Examples are the *System Model* defined by Rumpe et al. [5] and *Executable UML (xUML)* [6]. Recently, OMG released a new standard called *Semantics of a Foundational Subset for Executable UML Models* or *foundational UML (fUML)*, which aims to provide a precise definition of the execution semantics of a subset of UML 2 focusing on UML activity diagrams [7]. Therefore, fUML builds the basis of our work.

Whereas the operational semantics of a subset of UML is now clarified by OMG, it is not clear, which techniques are needed to enable the quality assurance of UML models by executing them. Methods nowadays used for assuring the quality of code may be adopted for ensuring the quality of models. Two concrete techniques that may be adopted are *debugging* and *testing*, which are essential for ensuring the quality of code. Similar techniques are necessary in order to support systematic testing and debugging of models. Therefore, lessons learned from testing and debugging of code may serve as a valuable source of inspiration. However, the peculiarities of models in comparison to code, constitute a major challenge which has to be tackled to adopt state-of-the-art techniques in programming for testing and debugging models. Such a peculiarity is for instance that models may be defined on different *levels of abstraction* [2]. Another distinctive feature of models is that often critical aspects of a system are modeled very detailed whereas others remain imprecise leading to *incomplete models* [3]. Further, UML models constitute a *multiple view* specification of a system, whereas some views describes the static aspects and others the dynamic aspects of the modeled system; consequently the consistency, completeness, and unambiguousness of models are important issues. Also *semantic variation points* of UML stemming from practice and semantic variation points introduced by the UML standard and specified by the fUML standard have to be considered when executing UML models.

Existing approaches for executing UML models address these peculiarities insufficiently. This means that the required abstraction level of models to be executable is precisely prescribed and incomplete models cannot be executed. Furthermore, the supported diagram types and their interactions are restricted and semantic variation points are scarcely handled. Therefore, existing approaches are too restrictive for meeting the requirements arising from practice and therewith insufficient.

In this paper, we propose a model execution environment based on fUML that enables to efficiently test and validate UML models by providing debugging capabilities as well as a test framework for automated model testing which overcome the aforementioned shortcomings of existing approaches.

II. Research Questions

We identified the following three main research questions which we aim to answer in the proposed thesis.

Model execution. The first research question is concerned with the issue of how existing models can be made executable. UML models are in most cases not created with the purpose of being executed. Thus, models are often defined on a high level of abstraction meaning that some elements of a model may not be executable. Moreover, models are often refined in the consecutive steps of the development life cycle implying that model execution has to consider incomplete models. Furthermore, depending on which systems are modeled and which development process is used, different UML diagram types and different modeling concepts are used. The identification of the used diagram types and the relationships between them, as well as the used elements is also an issue that has to be answered in the context of this research question. Although fUML constitutes a new industry standard for the semantics of UML, other semantics may be applied in practice. Providing the possibility of preserving the semantics of existing models, is also a problem we aim to examine.

Model testing. The second research question is concerned with techniques for automated testing of UML models. Testing serves different purposes. It can be used to find unknown problems which is known as *testing for validation* and it can be used to reproduce known problems which is known as *testing for debugging* [8]. Furthermore, testing is an important instrument to ensure that problems are resolved and will not reoccur. Thus, lots of testing techniques exist in programming such as system testing, integration testing, unit testing, white box testing, black box testing, regression testing and acceptance testing. The investigation of how testing techniques and concepts known from programming can be adopted for models is part of this research question. Also the aforementioned issues for model execution have to be considered in model testing. How can test cases be defined for models on different abstraction levels? How can we cope with incomplete models in defining test cases? How can test cases be defined across separate models of different types? Also the used semantics for execution has a high impact on testing a model. In the course of this research question we also aim to address the question how test cases for models can be specified and represented as well as how test cases can be derived from existing models. Therefore, we intend to investigate the applicability of similar approaches as in model-based testing in programming [9].

Model debugging. The third research question deals with debugging techniques applicable for models. Programming without debugging support is unimaginable nowadays. Debugging a program enables a better understanding of the source code and allows to locate errors more efficiently. Therewith, it serves as important technique in software quality assurance. By adopting code debugging techniques for models, we expect similar benefits for modeling. The *TRAFFIC principle* for debugging introduced by Zeller [8] will lead the investigation of debugging techniques applicable for models. This principle decomposes debugging into seven steps. The initial letters of these steps form the word TRAFFIC. Whereas the first step (**T**rack the problem) is largely an organizational issue that can be handled by existing bug tracking tools and the second and third steps (**R**eproduce the failure, **A**utomate the test case) are mainly addressed in the research question model testing, the steps four to seven (**F**ind infection origins, **F**ocus on likely origins, **I**solate the infection chain, **C**orrect the defect) constitute the challenges of this research question. These steps are about understanding the defective artifact (in our case the model) in order to locate and correct defects. To provide these capabilities, debugging techniques known for programming shall be investigated, such as techniques for visualizing the model execution, investigating and controlling the execution process, as well as logging and tracing of information about the execution. Again, the issues of dealing with incomplete models, models on different abstraction levels, semantic variation points, and the relationships between separate but interdependent views on the models have to be considered.

III. Research Methodology

The following methods will be applied in order to answer the aforementioned research questions and to evaluate appropriate solutions. The methodological approach is conform to the *Design Science* approach presented by Hevner et al. [10].

Survey on model execution in practice. By conducting a survey on model execution, we want to identify the requirements arising from practice regarding model execution and its application for testing and debugging models. The obtained requirements will lead the whole research to ensure that only important and relevant problems are addressed. Therefore, we developed a questionnaire to investigate general modeling habits in practice, such as the used modeling languages, diagram types, modeling tools, the applied modeling process, and to obtain requirements regarding model execution and its application in model testing and debugging.

The results of this questionnaire will be further deepened in expert interviews. In parallel to the questionnaire and the expert interviews, existing UML models from practice will be reviewed to gain more insights into which kinds of applications are actually modeled with UML and how UML modeling concepts are employed in practice. In order to ensure the validity of this survey, we work closely with our industry partner. Participants of the questionnaire and expert interviews will be carefully selected in cooperation with our industry partner and our industry partner will provide representative models from practice.

Evaluation of existing model execution approaches. Academia as well as industry developed approaches for model execution and its utilization for ensuring the quality of models. By evaluating model execution approaches emerging from academia, we aim at establishing a theoretical basis for our research. The evaluation of existing model execution tools will serve two purposes. First, we will identify, which of the elaborated requirements from the survey are already fulfilled by these tools, which are insufficiently handled, and which are not addressed at all. Second, we will investigate differences between the operational semantics implemented by existing model execution tools and the operational semantics defined by fUML.

Extension of the fUML model execution engine with respect to debugging and testing models. In order to provide a powerful model execution environment for testing and debugging models, the model execution engine provided by the fUML standard has to be extended because of several shortcomings. These shortcomings were identified while conducting my master's thesis [11]. In the course of this thesis a prototype of a model interpreter was developed based on the fUML standard in order to answer the research question whether the semantics definition of the fUML standard is sound and applicable for building tools that enable the execution of UML activity diagrams. Thereby, the following shortcomings of the fUML model execution engine have been identified. The main obstacle for directly building model debuggers based on the fUML engine is that the fUML execution model is neither observable, nor controllable. Thus, the state of a model execution is not explicitly available. Another problem is that the key semantic elements time, concurrency, and inter-object communications mechanisms are unconstrained in the fUML standard, but no defined adaptation points of the execution model exist in order to address them. Furthermore, modifying the semantics of modeling concepts or introducing new modeling concepts (e.g., UML profiles) are not addressed by the fUML standard. We intend extensions enabling the investigation and manipulation of the runtime model in order to observe and control the state of a model execution and to enable the modification of the model during execution. Furthermore, we want to enable the customization of the model execution engine and to introduce extension points to allow the incorporation of new modeling concepts. This extension of the fUML model execution engine is the prerequisite for implementing a model execution environment for testing and debugging UML models based on fUML.

Implementation of a model execution environment for testing and debugging UML models. Many techniques for debugging and testing code have proven successful. As already mentioned, it is an open question how these techniques are applicable and adaptable for models under consideration of their peculiarities. The investigation of these techniques will be based on a search process [10], i.e., alternative solutions will be investigated. Once we have identified the applicable techniques and adapted them for models, we will incorporate them in a prototype of a model execution environment based on the extended fUML model execution engine. This execution environment shall support testing and debugging of models and meet the requirements identified in the survey on model execution in practice.

Evaluation of the implemented model execution environment for testing and debugging UML models. In order to ensure that the collected requirements on model execution systems are met, an extensive evaluation of the developed model execution environment for testing and debugging UML models will be performed. This evaluation will be carried out using controlled experiments. The participants of these experiments will again be carefully selected in cooperation with our industry partner.

IV. Related Work

Related work can be divided into three fields: model execution, model debugging, and model testing. Due to space limitations we only provide some representative examples.

Model execution. In order to debug and test models, models have to be executable, i.e., they need to have an operational semantics. A number of approaches have been developed for executing UML models. Riehle et al. [12] developed a virtual machine for UML that interprets UML models without transforming the model into another form. *Executable UML* introduced by Mellor and Balcer [6] use model-compilers to produce executable UML models. Rumpe et al. [5] defined a system model serving as basis for formalizing the semantics of UML, including actions, state machines and interactions. IBM Haifa Research Lab [13] developed a generic model execution engine that can be used to simulate models, and implemented a UML simulator on top of it. Other approaches generate code from models in order to execute them. Fujaba [14] for instance, allows the generation of executable Java code from UML class diagrams, and a combination of UML activity diagrams, state machines and collaboration diagrams. Since fUML is a standard provided by OMG that specifies the execution semantics of UML precisely, our work will be based on this new standard.

Model debugging. Approaches for the adoption of debugging primitives for models including step-by-step execution, setting of breakpoints, inspection, and manipulation of variable values as well as the visualization of the execution exist. For instance, the UML simulator developed by IBM Haifa Research Lab [13] allows to execute UML models step-wise, set breakpoints, inspect variables and it visualizes the execution by animating the model. Similar functionality is provided by ACTi [15], which is an interpreter for UML actions and activities and was developed based on the system model defined by Rumpe et al. [5]. Recently, commercial UML modeling tools took first steps to support model execution by providing basic debugging functionalities. Examples are Enterprise Architect from Sparx Systems, IBM Rational Rhapsody, and Magic Draw from No Magic, Inc. However, the existing approaches for model debugging make simplifying restrictions for executable models and their execution. For instance, incomplete models cannot be executed, the relationships and interactions between models are restricted, semantic variation points are scarcely handled, and the level of abstraction is prescribed.

Model testing. Only few approaches exist that are concerned with testing of UML models. Dinh-Trong et al. [16] present an approach for testing UML design models by transforming them into an executable form and executing tests which are generated under the consideration of defined test adequacy criteria. Gogolla et al. [17] proposed an approach for testing UML class diagrams and OCL models using snapshots. Whereas considerable research work is going on in model-based testing [9], defining and assessing test cases for models similar to unit testing of code is an open issue.

V. Conclusion

In this paper, we proposed a model execution environment based on fUML which enables to efficiently test and debug UML models. The following contributions are expected from the proposed work. A comprehensive empirical study will provide insights into how UML is used in practice and will build the basis for the remaining research work. We will extend the fUML model execution engine in order to enable the implementation of practicable applications of model execution. Finally, debugging and testing techniques from programming will be adopted for modeling and incorporated into the envisioned model execution environment.

Acknowledgment

This work has been partly funded by our industry partner LieberLieber Software GmbH.

References

[1] J. Bézivin, "On the unification power of models," *Software and Systems Modeling*, vol. 4, pp. 171–188, 2005.

[2] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering*, 2007, pp. 37–54.

[3] B. Selic, "The pragmatics of model-driven development," *Software, IEEE*, vol. 20, no. 5, pp. 19–25, 2003.

[4] R. France, A. Evans, K. Lano, and B. Rumpe, "The UML as a formal modeling notation," *Computer Standards & Interfaces*, vol. 19, no. 7, pp. 325–334, 1998.

[5] M. Broy, M. V. Cengarle, H. Grönniger, and B. Rumpe, *Definition of the System Model*. John Wiley & Sons, Inc., 2009, pp. 61–93.

[6] S. J. Mellor and M. J. Balcer, *Executable UML: A foundation for model-driven architecture*. Addison Wesley, 2002.

[7] Object Management Group, *Semantics of a Foundational Subset for Executable UML Models*, Object Management Group Std., Rev. 1.0, 2011. [Online]. Available: http://www.omg.org/spec/FUML/1.0

[8] A. Zeller, *Why Programs Fail: A Guide to Systematic Debugging*. Elsevier, 2005.

[9] P. Baker, Z. R. Dai, J. Grabowski, O. Haugen, and I. Schieferdecker, *Model-Driven Testing: Using the UML Testing Profile*. Springer, 2008.

[10] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, pp. 75–105, 2004.

[11] T. Mayerhofer, "Breathing New Life into Models: An Interpreter-Based Approach for Executing UML Models," Master's thesis, Vienna University of Technology, 2011. [Online]. Available: http://publik.tuwien.ac.at/showentry.php?ID=196765

[12] D. Riehle, S. Fraleigh, D. Bucka-Lassen, and N. Omorogbe, "The architecture of a UML virtual machine," in *Conference on Object-oriented programming, systems, languages, and applications*, 2001, pp. 327–341.

[13] A. Kirshin, D. Dotan, and A. Hartman, "A UML simulator based on a generic model execution engine," in *Conference on Models in Software Engineering*, 2006, pp. 324–326.

[14] U. Nickel, J. Niere, and A. Zündorf, "The FUJABA environment," in *Conference on Software engineering*, 2000, pp. 742–745.

[15] M. L. Crane and J. Dingel, "Towards a UML Virtual Machine: Implementing an Interpreter for UML 2 Actions and Activities," in *Conference of the Center for Advanced Studies on Collaborative Research*, 2008, pp. 96–110.

[16] T. Dinh-Trong, N. Kawane, S. Ghosh, R. France, and A. Andrews, "A tool-supported approach to testing UML design models," in *Conference on Engineering of Complex Computer Systems*, 2005, pp. 519–528.

[17] M. Gogolla, J. Bohling, and M. Richters, "Validation of UML and OCL Models by Automatic Snapshot Generation," in *The Unified Modeling Language. Modeling Languages and Applications*, 2003, vol. 2863, pp. 265–279.