Moliz: A Model Execution Framework for UML Models*

Tanja Mayerhofer, Philip Langer
Business Informatics Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology, Austria
{mayerhofer, langer}@big.tuwien.ac.at

ABSTRACT

In model-driven development models are the primary artifacts in the software development process. Consequently, the quality of those models affects the quality of the developed system significantly. Therefore, adequate methods for ensuring a high quality of models are needed. Model execution can serve as the crucial basis for such methods by enabling to debug and test models. In this paper, we present ongoing research towards a model execution framework based on fUML that enables to test and validate UML models efficiently by providing debugging capabilities, as well as a test framework for automated model testing.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*debugging aids, testing tools*

General Terms

Design, Languages, Reliability

1. INTRODUCTION

With the application of model-driven development, models become the main artifacts in the software development process constituting the single specification of the software. Thus, the quality of the models influences the quality of the developed system significantly and, as a result, affects the success of the whole software development process. Therefore, adequate techniques for ensuring the quality of models are crucial. Model debugging and testing can serve as such techniques requiring the executability of models. For enabling the executability of models, the behavioral semantics of a modeling language has to be defined formally. Unfortunately, the semantics of many modeling languages is specified only informally leading to imprecision and ambiguity. This is also true for UML, the most adopted modeling language in industry [3], which impedes the execution of UML models. Thus, various approaches to specify the behavioral semantics of UML formally have been developed in the past. The latest attempt to standardize the behavioral semantics of UML was made by the OMG with the release of the fUML[1] standard. fUML defines a key subset of UML comprising parts of class diagrams and activity diagrams and specifies its behavioral semantics precisely and completely in terms of a virtual machine for executing compliant models. In the research project Moliz[2], we aim at developing a model execution framework based on this standardized virtual machine enabling to debug and test UML models.

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

2. MODEL DEBUGGING AND TESTING

In this section, we discuss the three main research challenges, which we aim to address in the project Moliz.

Model execution. The standardization of a virtual machine for executing UML models in the fUML standard comprises an important step towards the utilization of model execution for UML models. UML models can now be used not only for informal design sketching but also for completely building executable systems. However, the full potential of UML model execution cannot be exploited yet, as the virtual machine lacks in providing the means for runtime observation, analysis, and execution control. As a result, important applications of model execution, such as controlling, observing, and adapting the behavior of a system at runtime, cannot be realized using fUML so far. To overcome these limitations, we extended the reference implementation of the fUML virtual machine in terms of a dedicated trace model, an event model, and a command API.

A *trace model* enables to record the execution of a system, which constitutes an abstract representation of the system's runtime behavior. Thus, it provides the information necessary to analyze the execution serving as the prerequisite for several applications, such as testing and dynamic adaptation. Thus, we elaborated a dedicated trace model for recording the execution of UML models carried out by the fUML virtual machine. This trace model provides information about the chronological execution order of activity nodes, input/output relationships between the nodes, as well as the occurred token flows. Based on this trace model, it is possible to analyze the runtime behavior of UML activities. To enable also the observation and control of the activity execution during runtime, we further extended the fUML virtual machine with an event model and a command

[1] `http://www.omg.org/spec/FUML`
[2] `http://www.modelexecution.org`

API. We developed an *event mechanism* that notifies listeners about the state of an activity execution whenever the runtime state of the execution changes. This includes the starting and ending of the execution of activities, the starting and ending of the execution of activity nodes, as well as the creation, destruction, and modification of objects. Further, the *command API* introduced into the fUML virtual machine enables to suspend the execution of an activity at a certain activity node, to perform single execution steps comprising the execution of exactly one activity node, and to resume the execution. The suspension of an execution is also indicated by a dedicated event.
These extensions build the basis for implementing debugging facilities, as well as a testing environment for UML models based on the fUML virtual machine. We implemented these extensions using the aspect-oriented programming language AspectJ, i.e., we weave the extensions into the reference implementation without altering the code to retain compatibility in case of revisions in the reference implementation.

Model debugging. Programming without debugging support is unimaginable nowadays. Debugging a program enables a better understanding of the 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 model-driven development [4].
Based on the above discussed extensions of the fUML virtual machine, we implemented a debugger for UML models which is integrated with the Eclipse Debug framework. This debugger enables the debugging of UML activities created with the Papyrus UML editor and provides capabilities known from debugging program code: users can step-wise resume and suspend the activity execution using the commands step into, step over, step return, resume, and terminate. Further, it enables adding breakpoints. The state of the execution is visualized by means of animating the model directly in the Papyrus editor, displaying the values of the input pins of the currently executed activity node, as well as the current runtime model (i.e., existing objects, their instance values, as well as links between them), and the call hierarchy of the executed activities including the nodes enabled for being executed next. Besides this integration of a model debugger into Eclipse, we are also working on an integration into the widely-used modeling tool Enterprise Architect.
Ongoing work regarding debugging UML models concerns the development of more sophisticated debugging functionalities, such as empowering the user to modify an activity during its execution requiring the impact analysis of such changes, the justification of the current execution state based on trace information about the actual execution, as well as trace-based debugging enabling debugging backwards. A further issue we want to address in future work is the dealing with incomplete models as this would allow to support the user not only in understanding existing models and locating errors but also in creating the models, as they are usually refined in the consecutive steps of the development process and not created all at once.

Model testing. 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* [5]. Furthermore, testing is an important instrument to ensure that problems are resolved and will not reoccur.
Thus, we aim at adopting the concept of unit testing for UML models providing modelers with means for ensuring a high quality of their models. Therefore, we are currently developing a dedicated *testing language* that enables the definition of test cases for UML activities which can be assessed using the fUML virtual machine. For executing the test cases we make use of the trace model, event model, and command API discussed above. Test cases shall enable the user to evaluate the execution order of the nodes of an activity, the inputs provided to as well as the outputs produced by the nodes and the executed activities, as well as the runtime model consisting of created objects and links between them. These evaluations shall be possible at an arbitrary state of the execution and not only after the execution of the entire activity. Further, we plan do develop an approach for automatically deriving test cases from the models as proposed by the model-based testing paradigm [1].

All these efforts in developing a model execution framework based on fUML that provides debugging capabilities, as well as a test framework for automated model testing are initially targeted at UML activities, which are directly part of fUML. However, the long-term goal of our research is to provide these functionalities also for the remainder of UML. Further, we are elaborating on using fUML for specifying the operational semantics of domain-specific modeling languages, enabling the automatic generation of debugging and testing facilities for these languages [2].
Besides the development of a model execution framework based on fUML, we are also conducting *empirical studies* on the usage of UML and requirements regarding model execution. For this purpose, we developed a *questionnaire* which we distributed via mailing lists of research communities in the field of software development. Invitations for participating in the questionnaire were also sent out by our industry partner who provides tools for model-driven development. 96 people participated in the questionnaire. Further, we are developing *metrics* on the usage of UML modeling concepts in real-world models. These metrics are intended to determine which modeling concepts and diagram types of UML are actually used and which combinations are used. We are implementing a script that extracts this information from models created with the modeling tool Enterprise Architect.

3. REFERENCES

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

[2] B. R. Bryant, J. Gray, M. Mernik, P. J. Clarke, R. B. France, and G. Karsai. Challenges and directions in formalizing the semantics of modeling languages. *Computer Science and Information Systems*, 8(2):225–253, 2011.

[3] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. Empirical assessment of MDE in industry. In *Proc. of ICSE'11*, pages 471–480, 2011.

[4] R. Mannadiar and H. Vangheluwe. Debugging in domain-specific modelling. In *Proc. of SLE'10*, pages 276–285, 2010.

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

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Vienna University of Technology
Institute of Software Technology and Interactive Systems
Business Informatics Group
Advisor: O.Univ.-Prof. Mag. Dipl.-Ing. Dr. Gerti Kappel

Moliz: A Model Execution Framework for UML Models

Tanja Mayerhofer, Philip Langer

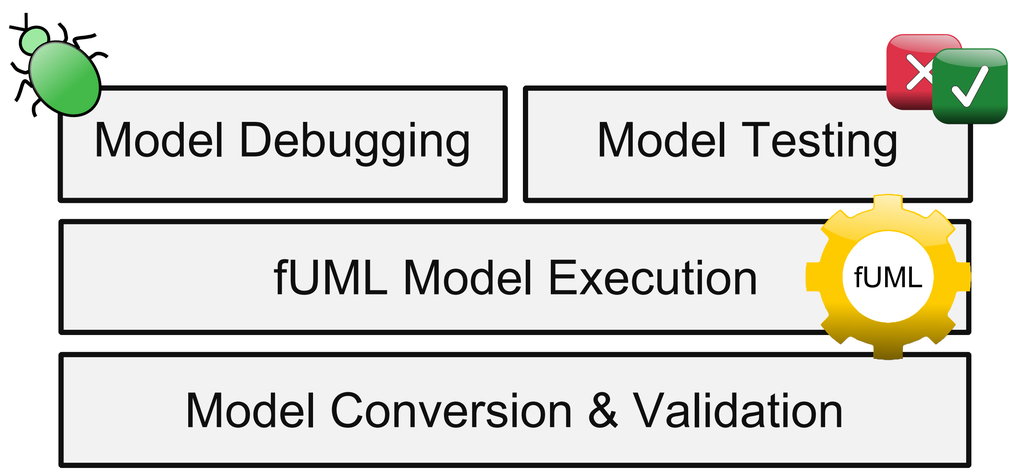
Motivation

- High quality of models is important as they become the primary artifact in software development (MDD)
- Methods for ensuring high quality of models are necessary
- Model execution can serve as basis for quality assurance methods by enabling model debugging and testing

Objective

- Provide model execution environment based on fUML, enabling efficient debugging and testing of UML models
- Preliminary focus: Execution of fUML compliant activity diagrams
- Medium-term objective: Execution of selected modeling languages
- Long-term objective: Execution of domain-specific languages

Moliz Model Execution Framework



Model Execution

Extension of the fUML reference implementation

- Enabling runtime analysis, observation, and execution control
- Incorporation of trace model, event model, and command API

Model Testing

- Domain-specific language to specify assertions on traces
- Automatic derivation of test cases
- Generation of executable test scripts

Model Debugging

Basic functionality

- Step-wise execution
- Breakpoints
- Visualization of execution
- Access to runtime model

Advanced functionality

- Impact analysis of changes
- Justification of execution state
- Trace-based debugging
- Dealing with incomplete models

Tool Integration and Empirical Study

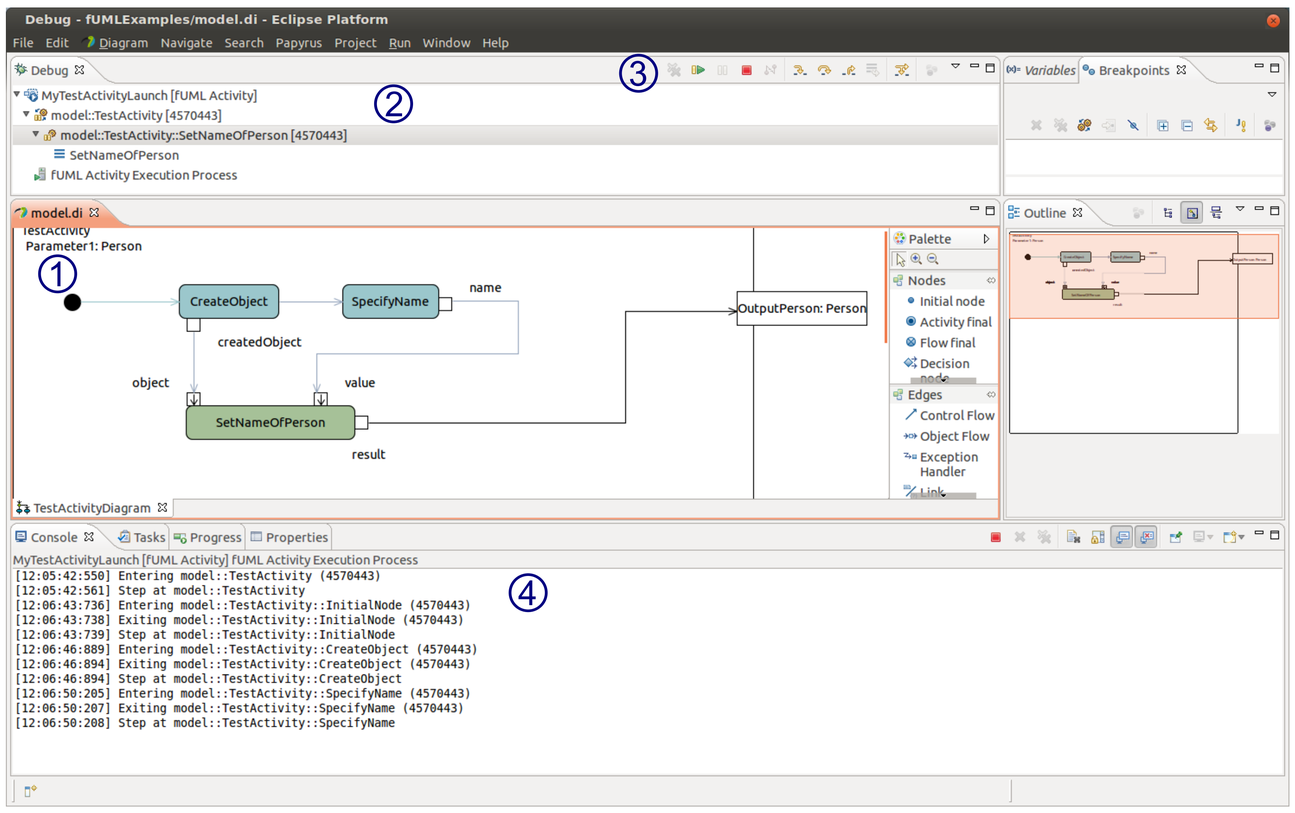
Tool integration

- Papyrus UML editor
- Enterprise Architect

Empirical studies

- Questionnaire about modeling habits and model execution
- Metrics on usage of UML in Enterprise Architect models

Eclipse Debugger for fUML Compliant Activities



① Executed activity diagram modeled with Papyrus UML editor

② Debug view displays the current execution status
Example: Action "SetNameOfPerson" is enabled and will be executed in the next step

③ For debugging activities the following operations are available: Resume, Terminate, Step Into, Step Over, Step Return

④ Log messages are displayed in the console of the respective activity execution

Debugging and Testing of Models Based on fUML
www.modelexecution.org

Contact: mayerhofer@big.tuwien.ac.at