Tracing of state machine execution in the model-driven development framework

Anna Derezinska, Marian Szczykulski
Institute of Computer Science
Warsaw University of Technology
Warsaw, Poland

***Abstract*—Framework for eXecutable UML (FXU) supports a model-driven approach for creation of reliable applications in C#. Basing on UML classes and their state machines, a source code is generated and executed using a run-time library. All modeling concepts of UML behavioral state machines can be used in the FXU. Tracing of program execution in terms of elements of origin state machines assists program comprehension and verification. This paper presents a new component of the framework for observing traces of program execution. The state machine tracing improved model-driven engineering of an application in comparison to the direct analysis of textual logs generated during the program execution. We discuss system requirements, steps of tracing process and advantages and disadvantages of the solution. The approach was applied in the development of an application simulating a presence server for the status service of a social network model.**

Keywords-state machine; UML; Model-Driven Development

I. INTRODUCTION

Model-Driven Development (MDD) promotes using of models through the whole software lifecycle [1]. Models act as primary artifacts of the software development and can be used to reflect system entities or processes. A code generation can be treated as a kind of the model transformation, in which the source and target models reside at different abstraction levels and use different notations. Providing artifact traceability relationships between program behavior and model elements is a challenging problem.

State machines, among different Unified Modeling Language (UML) diagrams, constitute a widely accepted formalism for the specification of concurrent reactive systems. They extend classical finite-state machines by incorporating hierarchy, orthogonality, compound events, and a broadcast mechanism for the communication between concurrent components [2].

A model driven approach for program development is realized in a *Framework for eXecutable UML* (FXU) [3]. FXU performs transformation from a UML class and state machine model into a C# implementation. Experience gathered during applying FXU [4,5] was exploited in the framework evolution. The development of an application modeled using cooperating complex state machines requires a user-friendly tool support. Specifically, the construction, modification, and revision management of state machines tend to become increasingly troublesome for larger models [6].

A semantic variation point is a semantic aspect of a model element that the UML specification allows a user to define [2]. It is especially characteristic for the interpretation of state machines. Selecting implementations of semantic variation points of state machines and the observation of subtle differences in the program behaviors is of major importance.

Therefore, a new version of FXU was equipped with a graphical user interface and *Application Wizard* to improve the process of building an executable application. Moreover, *FXU Tracer* was designed to enable tracing of the execution of state machines in a generated C# code. Its usage supports the model comprehension that can help better understanding of state machines created by someone else. It assists verification of correctness of the execution of an application reflecting a behavior of state machines and facilitates detection of some mistakes.

The next Section gives the main features of FXU and the related work. Section 3 discusses *FXU Tracer* and its application. Information about advantages and disadvantages concludes the paper.

II. FXU OVERVIEW

The FXU tool transforms UML models in a C# source code and supports execution of an application reflecting the behavioral model. The generated and executed code corresponds to a structural model specified in class diagrams and its behavioral model described by state machines of these classes. All elements of class diagrams (classes, interfaces, attributes, operations, relations between classes, etc.) are transformed using a code template [3].

The distinguishing feature of FXU is that all single concepts of behavioral state machines included in the UML 2.x specification are taken into account, including all kinds of states such as simple states, composite states, orthogonal states, entry-, do-, exit- activities and submachine states. Also all possible pseudostates are considered, including: initial pseudostate, deep and shallow history, join, fork, junction, choice, entry and exit points, terminate, as well as different kinds of events (including deferred events). The FXU framework was the first tool that supported generation and

The original publication is available at https://ieeexplore.ieee.org/document/5553380
Proceedings of the 2nd International Conference on Information Technology, ICIT 2010 • 28-30 June 2010, Gdansk, Poland, pp. 109-112

execution of all elements of the behavioral state machine UML 2.0 using the C# language [3].

The framework consists of two components: *Code Generator* and *Runtime Library*. *Runtime Library* contains realization of different UML meta-model elements, especially referring to behavioral UML state machines. In order to obtain a specified behavior several semantic variation points of state machines have to be resolved [4].

Verification of an input UML model is based on a set of hard coded rules [5]. The rules originate from the UML specification [2] and can be applied for any object-oriented language, or take into account features of the target language, in this case C#. Static verification is performed during transformation of class and state machine models into a corresponding code. During execution of the code corresponding to the given state machines other rules are verified dynamically.

Transformation of UML models in a programming code and development of an executable application are realized using the FXU framework in the following steps:

1) A model, created using a CASE modeling tool, is exported to a defined serialized form.

2) The model (classes with all relations and their behavioral state machines) is transformed by *FXU Generator* that creates a corresponding code in C#.

3) Using *Application Wizard* associated with the *Generator*, a Microsoft Visual Studio project is created. It comprises the generated code of classes and required libraries. Additionally, a class with the *main* method is created, in which a set of objects that can cooperate is declared, state machines are initialized and can be launched.

4) The generated code can be modified, if necessary, including completion of operations bodies. Compiled and linked with *Runtime Library* the final application is created.

5) The application that reflects the model behavior of all cooperating state machines can be executed. Concurrently executed class objects as well as substates in composite orthogonal states are run as parallel threads. Information about the application run is stored in a log file.

6) The application run in reference to the model elements is traced using *FXU Tracer*.

The current version of *FXU Generator* works with EMF 2.4.1 (Eclipse Modeling Framework) [7]. It accepts UML models stored in the UML 2.1 format of Eclipse (*.uml* extension). In experiments, UML models were built using the IBM Rational Software Architect [8] – a CASE tool that is run within the Eclipse platform.

One of MDE-based approaches is a code generation process that introduces an intermediate model level between UML models and code generations [9]. FXU assumes a direct model transformation to the target code.

In some solutions, the code created as the target of a model transformation includes the mapping of the state machine structure as well as the logic supporting model execution [10]. The FXU framework is based on another approach, applying a run-time library and providing an engine for the state machine execution

Twelve tools that could also generate a code from behavioral state machines were compared. Only few of them took into account more complex features of state machines, like choice pseudostates, deep and shallow history pseudostates, deferred events or internal transitions. The most complete support for the state machines UML 2.x is implemented in the IBM Rational Rhapsody tool [11] (formerly Telelogic and I-Logix). It supports tracing of an application. Items to be traced can be identified with trace commands. Tracing is proceeded for one or more steps, or run to an inserted breakpoint. However, Rhapsody does not consider the C# language.

III. FXU TRACER

A. Requirements

The main task of *FXU Tracer* is observation of application execution in terms of elements of an original model. Observed events refer to all cooperating objects declared in the application. There are two possible modes to trace state machines: step by step and run to the next breakpoint.

Model diagrams are visualized in a condensed tree-like form (Fig. 1). The root node of the tree is the model name. Its children are nodes representing packages. A package node contains class nodes, which have nodes representing their state machines. A state machine node has information about all its regions. A region comprises its states, pseudostates and incoming and outgoing transitions also grouped in tree nodes. This hierarchy represents the real hierarchy in the UML model.

Currently processed nodes are indicated in the tree. It is possible to set a breakpoint on a specific node. The tracing process of state machines stops at the moment this node is reached and the process can be continued if it is requested. The nodes are marked using colors and graphical signs. Four kinds of meanings that label tree nodes can be distinguished:

- A node has not been processed yet and no breakpoint has been set.
- A node has not been processed yet, but a breakpoint has been set.
- A node is being processed at the moment and no breakpoint has been set.
- A node is being processed at the moment and a breakpoint has been set.

Each executed step is commented with detailed textual information. It informs about entering a state or a pseudostate, completing an *entry*, *do* or *exit* action, traversing a transition as a result of a certain trigger, etc.

The main window of the *FXU Tracer* is composed of two panels (Fig. 1). The left panel is designed to contain a tree, which represents the UML model. The right panel has a text area designed to contain some textual information about the tracing process.

The original publication is available at https://ieeexplore.ieee.org/document/5553380
Proceedings of the 2nd International Conference on Information Technology, ICIT 2010 • 28-30 June 2010, Gdansk, Poland, pp. 109-112

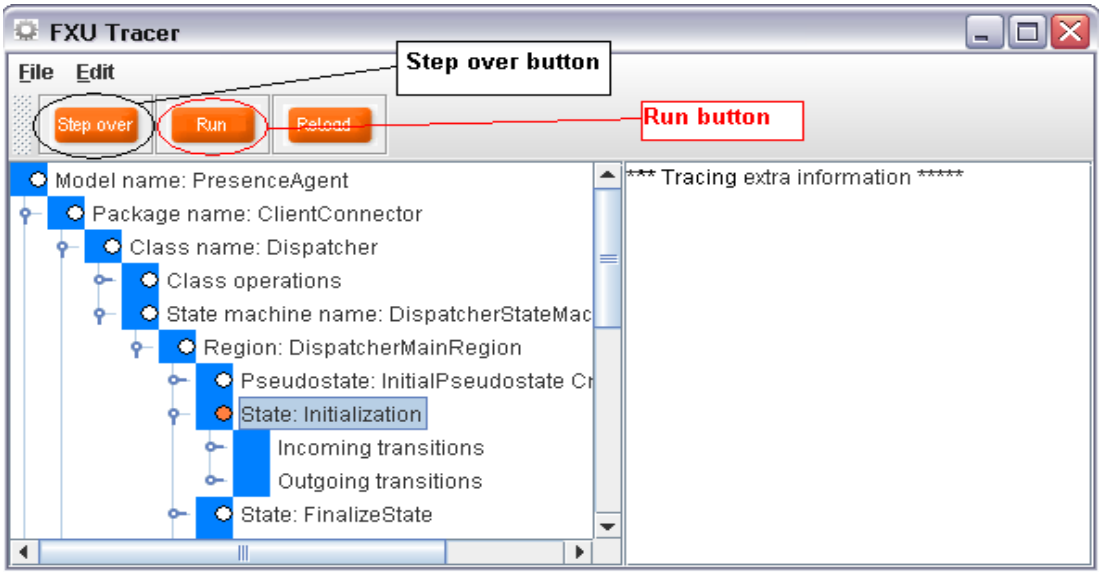

Figure 1. FXU Tracer - main window

B. *Tracing Process*

This section describes the basic steps of a tracing process using *FXU Tracer*.

1) Load a UML model serialized in the uml2 format of Eclipse. The loaded UML model is visualized on the left part of the main panel as a tree. The tree has hierarchically grouped information about the whole UML model.

2) Load a file with trace logs. The trace logs are created by the FXU environment during the application execution.

3) Start tracing. The tracing process is visualized by the model tree as well as by the text information in the text area.

a) Follow steps visualized by the tree, which represents the UML model. Every node of the model tree has its graphical label indicating its meaning. A breakpoint can be set on a selected node. The tracing process of state machines stops when this node is reached. The process can be continued on demand. Two trace modes (step by step or run to the next breakpoint) can be used.

b) Follow steps in the text area. During the tracing process textual information messages are continuously displayed. These information messages are more detailed and contain elements, which visualization on the tree is difficult, such as an end of an *entry*, *do* or *exit* action execution of a state or transition *effects.*

4) Restart the tracing process. If required, the whole tracing process can be restarted using the same UML model and the trace log file.

C. *Application*

The presented approach was verified in experiments. The FXU framework was used in the development of an application simulating a presence server for the status service of a social network model [12].

The presence server was modeled in UML. It is the central part of the system and realizes the business logic. The server completes three main tasks: subscription of a status of another user, publication of a new status with given rules and notification another user about a status. A status constitutes information about a user accessibility and its context. Users are associated with relations, which can be divided into categories, like family members, friends, etc.

The whole model consists of about twenty classes and interfaces and about seventeen state machines. The behavior of each class of the server was specified by its state machine. State machines of main classes consisted of a dozen or more states. Various types of complex states and pseudostates, like fork, join, history, etc. were used. An example of a simple state machine is shown in Fig. 2. A part of its model was visualized in Fig. 1.

The model was transformed into C# and supplemented with the implementation of basic service operations. The functionality of the logic responsible for handling requests for publishing, subscription and notification was the main goal of tests.

The detailed analysis of a system behavior was accomplished with *FXU Tracer*. The behavior of handling basic requests of a server user was validated.

IV. CONCLUSIONS

Observation of a program behavior using *FXU Tracer* helped to verify correctness of the execution of behavioral state machines in a complex application. It assists also a comparison of consequences of different implementations of selected semantic variation points encountering in state machines. The execution of state machines is presented in a visual and textual form that is helpful especially for bigger applications. Another advantage is possibility to start tracing in any state, omitting

The original publication is available at https://ieeexplore.ieee.org/document/5553380
Proceedings of the 2nd International Conference on Information Technology, ICIT 2010 • 28-30 June 2010, Gdansk, Poland, pp. 109-112

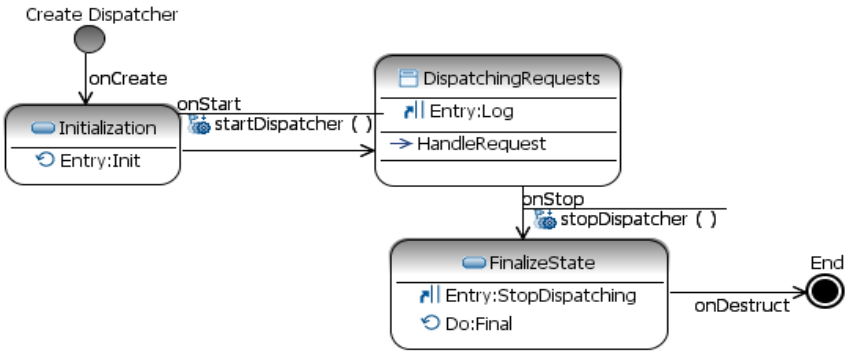

Figure 2. Exemplary state machine of Dispatcher class from Presence Agent model

unimportant steps. The same execution can be traced any number of times. At the moment, the most of events – but not all of them - are logged and can be traced. The tool can be easily extended to take into account all possible events, if necessary. The main disadvantage is tracing after application execution using an appropriate model and log files, and not simultaneously to it.

REFERENCES

[1] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," Future of Software Engineering at ICSE'07, IEEE Soc., 2007, pp. 37-54.

[2] Unified Modeling Language Superstructure v. 2.2, OMG Document formal/2009-02-02, 2009, http://www.uml.org

[3] R. Pilitowski, and A. Derezińska, "Code generation and execution framework for UML 2.0 classes and state machines," in T. Sobh (eds.) Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, Springer, 2007 pp. 421-427.

[4] A. Derezińska, and R. Pilitowski, „Event processing in code generation and execution framework of UML state machines," in L. Madeyski at al. (eds.) Software Engineering in Progress, Nakom, Poznań, 2007, pp. 80-92.

[5] A. Derezińska, and R. Pilitowski, "Realization of UML class and state machine models in the C# code generation and execution framework," Informatica vol. 33, no 4, Nov. 2009, pp.431-440.

[6] S. Prochanow, and R. von Hanxleden, "Statecharts development beyond WYSIWIG," in G. Engels et al. (eds.) MODELS 2007, LNCS 4735, Springer, Berlin Heidelberg, 2007, pp. 635-649.

[7] Eclipse Foundation, UML2, http://www.eclipse.org/uml2/ (visited 2010).

[8] IBM Rational Software Architect, http://www-01.ibm.com/software/awdtools/swarchitect/ (visited 2010).

[9] T. Haubold, G. Beier, and W. Golubski, "A pragmatic UML-based meta model for object-oriented code generation," in Proc. of 21st International Conference on Software Engineering & Knowledge Engineering, SEKE'09 , 2009, pp. 733-738.

[10] A. Niaz, and J. Tanaka, "Mapping UML statecharts into Java code", In Proc. of the IASTED International Conference on Software Engineering, Acta Press, Anheim, Calgary, Zurich, 2004, pp. 111-116.

[11] IBM Rational Rhapsody, http://www-01.ibm.com/software/awdtools/rhapsody/ (visited 2010).

[12] A. M. Dziekan, "Context-aware services in the IP Multimedia Subsystem: social networks modeling and implementation," MSc Thesis, Warsaw University of Technology, Institute of Telecommunications, 2008 (in polish).

The original publication is available at https://ieeexplore.ieee.org/document/5553380
Proceedings of the 2nd International Conference on Information Technology, ICIT 2010 • 28-30 June 2010, Gdansk, Poland, pp. 109-112