DEMOCLES: A Tool for Executable Modeling of Platform-Independent Systems

Christian Glodt
University of Luxembourg
christian.glodt@uni.lu

Pierre Kelsen
University of Luxembourg
pierre.kelsen@uni.lu

Elke Pulvermueller
University of Luxembourg
elke.pulvermueller@uni.lu

Abstract

The main goal of model-driven architecture is the generation of the full implementation of a system based on a precise description of a platform-independent model and a platform model. Such a description must accurately specify the static structure as well as the dynamic behavior of the system. We present a tool – called DEMOCLES – that realizes a hybrid approach to platform-independent modeling. It describes the static structure using a modified UML class diagram that separates query operations from modifier operations. The former are defined in the class diagram via OCL constraints, while the latter are defined using a MOF-based metamodel that contains modifier operations and properties as first-class entities and augments them with associations and OCL expressions. The tool is an Eclipse-plugin that offers overlay views of the structure and behavior with visual editing capabilities and permits execution of a platform-independent system.

Categories and Subject Descriptors D.1.1 [*Programming Techniques*]: Applicative (Functional) Programming; D.1.5 [*Programming Techniques*]: Object-oriented Programming; D.1.7 [*Programming Techniques*]: Visual Programming; D.2.6 [*Programming Environments*]: Graphical Environments; D.2.6 [*Programming Environments*]: Integrated Environments; D.3.2 [*Language Classifications*]: Multiparadigm languages

General Terms Design, Languages

Keywords executable models, model-driven architecture, platform-independent model, visual programming, code generation, eclipse

1. Introduction

Model-driven architecture aims at describing a system using a platform-independent model in sufficient detail so that the full implementation of the system can be generated from this model and a platform model. This implies that the platform-independent model must describe the static structure as well as the dynamic behavior of the system.

Although UML provides some means of expressing the behavioral aspects of a system, these are either incomplete (e.g., sequence diagrams) or apply only to certain types of systems (e.g., systems with a finite number of states) thus restricting their use for full code generation.

OOPSLA'07, October 21–25, 2007, Montréal, Québec, Canada.
ACM 978-1-59593-865-7/07/0010.

One way of supplementing dynamic information is via an Action Language based on Action Semantics [1]. Action languages (e.g., ASL [4]) follow an imperative style: they are reminiscent (although more abstract than) traditional programming languages such as Java or C++. Furthermore, while the action semantics have been standardized by the OMG, the actual concrete syntax is not part of the standardization. This brings about the prospect of a plethora of action languages being used for describing systems, further complicating the task of understanding and sharing the underlying models.

In an attempt to overcome some of these shortcomings we have developed a small behavioral modeling language, named EP [2, 3]. This language is based on two main types of elements: *events* and *properties*. Additional related elements and OCL code snippets augment these basic elements in order to provide an executable specification of the system. It provides a declarative solution to behavioral modeling (as opposed to the imperative style of action languages) while maintaining executability.

2. Modeling Platform-Independent Models

2.1 Structural Modeling

Although our main focus is on behavioral modeling, we need to concern ourselves also with structural modeling since structural elements will be reused in the behavioral model. For the structural modeling we make use of standard UML class diagrams with the following modifications:

- We list as operations only query operations, i.e., operations that do not modify state.
- We add a fourth compartment named "events" . We may think of events as modifying operations whose semantics will be detailed in the behavioral model. Note that adding named compartments is a facility provided by UML.
- We define initial values of a property using an OCL "init" constraint - essentially an OCL expression that determines the initial value of the property.
- We define the body of a query operation using an OCL "body" constraint that describes, using an OCL expression, what a query operation returns.

These modifications can be expressed more formally using an UML profile (omitted).

2.2 Behavioral Modeling

For behavioral modeling we depart from UML by introducing a new executable language for modeling the behavior of a system: this language expresses the behavior of the system by using events and properties (from which we derive the name – EP – of the

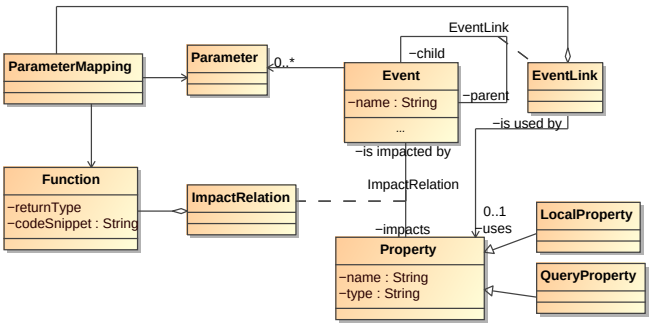

Figure 1. Behavioral Metamodel

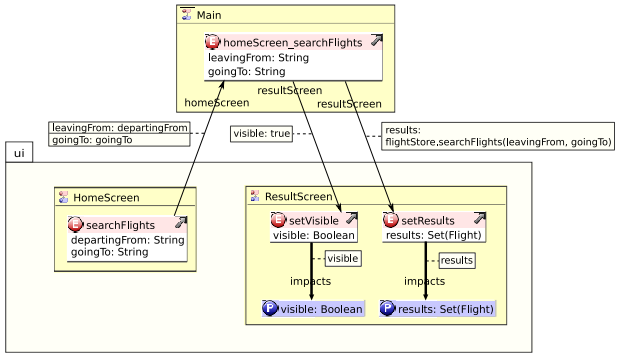

Figure 2. View of the EP-model for the *searchFlights* event

language) from the class diagram as first-class entities. They are connected among each other using two types of associations, with additional OCL code snippets for specifying the detailed behavior.

In figure 1 we describe the abstract syntax of the language via its UML metamodel. From hereon we shall use the name *EP-model* to denote an instance of this metamodel.

The main entities are events, properties and functions. An event can have parameters, each parameter having a name and a type. An event link connects an event to a child event via a link property which is either a property or a parameterless query operation (denoted by the *QueryProperty* class in the metamodel). Each event link carries one parameter mapping per parameter of the child event. The parameter mapping is a code snippet (represented by a *Function* object) that expresses the value of the parameter of the child event in terms of the parameters of the parent event and in terms of query operations and properties of the model containing the parent event. An event can modify the state of the system by impacting a property in its class: the *impact* link carries a code snippet that expresses the new value of the targeted property.

In figure 2 we illustrate the concrete syntax by presenting a partial view of the EP-model of a system for flight reservation.

We describe the dynamic semantics by defining what happens to the system state when an event is triggered on an instance. Let the *old state* denote the system state just before an event is triggered and let the *new state* stand for the system state right after an event has occurred. The value of a query operation on an instance is determined by computing the value of the OCL expression on the old state. When an event is triggered, all properties impacted by this event are set to the values of the OCL expressions attached to the impact links. For each event link leaving this event we evaluate the link property to determine the target instance referred to by this link property. We also evaluate the value of each parameter of the child event by evaluating the OCL expressions in the corresponding parameter mappings. We then recursively trigger the event on the target instance with the computed arguments.

3. The Tool

The DEMOCLES tool allows the user to manipulate the structure and behavior of a platform-independent model. It provides several notable features that assist the user in this task:

- A structural system view presents the UML class diagram of the system, including its package structure. The user can edit classes directly on this diagram and perform a manual or automatic layout.
- Facilities are provided for the comprehension of the dynamic behavior of a system. In particular an *event navigation view* supports the quick discovery of all external events that may affect the state of a system. Furthermore the user can quickly switch between the structural and behavioral view, the latter one being presented as an overlay view of the class diagram.
- The user may execute a system using an interactive debugger that displays the instance graph with the possibility of triggering events on this graph and observing the resulting changes.
- The tool provides full support for writing OCL expressions, both in the structural and behavioral views, with OCL code-completion and syntax-highlighting.
- Immediate visual feedback is provided on the syntactic validity of both model elements and user-supplied code snippets when the user applies any changes to a model.

4. What the Audience will be Seeing

We demonstrate the basic features of the tool using a simple flight reservation system called *FlightFinder*. The goal is to show how to do static and behavioral modeling.

Next we will highlight those features of the tool that are related to comprehending and inspecting platform-independent models. We show how the static and behavioral models of the application can be explored. In particular we will demonstrate how to use our tool for quickly grasping the main events that a system offers and their effect on the system state. Next we will execute the platform-independent model for the *FlightFinder* application using our interactive debugger that visualizes snapshots of the system state and the effect of triggering individual events on this system state.

References

[1] Alcatel, I-Logix, Kennedy-Carter, Inc. Kabira Technologies, Inc. Project Technology, Rational Software Corporation, and Telelogic AB. Action semantics for the UML. In *Document ad/2001-03-01. OMG*, 2000.

[2] Pierre Kelsen. A simple static model for understanding the dynamic behavior of programs. In *IWPC*, pages 46–51, 2004.

[3] Pierre Kelsen. A declarative executable model for object-based systems based on functional decomposition. In *ICSOFT (1)*, pages 63–71, 2006.

[4] C. Raistrick, P. Francis, and J. Wright. *Model Driven Architecture with Executable UML.* Cambridge University Press, 2004.