Model-Integrating Software Components

Mahdi Derakhshanmanesh[1], Jürgen Ebert[1], Thomas Iguchi[1], and Gregor Engels[2]

[1]University of Koblenz-Landau, Institute for Software Technology, Germany
{manesh,ebert,tiguchi}@uni-koblenz.de
[2]University of Paderborn, Department of Computer Science, Germany
engels@uni-paderborn.de

Abstract: In a model-driven development process the problem arises that over time model and code may be not aligned. Thus, in order to avoid this steadily increasing distance between models and code, we propose the integration of (executable) models and code at the component level. Redundancy – the source of inconsistencies – is reduced by interpreting models directly. Moreover, variability and adaptivity can be achieved by querying and transforming the embedded models. As the basis for such Model-Integrating Components (MoCos), we introduce a component realization concept that is compatible with existing component technologies.

Overview

In *Model-Driven Development* (MDD) [BCW12, SV06] models are used to describe views of a system. At some point in the development process, code is generated from these models. Despite all efforts (e.g., round-trip engineering), this generation step is often a source of inconsistencies between model and code artifacts as they evolve. Currently, models and code artifacts are kept separately and are – at most – connected by links, e.g., to maintain traceability or a "causal connection" at runtime. Thus, understanding and reusing associated parts of models and code may become tedious.

Modularization concepts proposed in *Component-Based Development* (CBD) are well-established to manage the development of complex software and to achieve reuse [SGM02]. Yet, component concepts for realizing software architectures have been traditionally targeted at the programming language level. Carrying executable models as first-class constituents of components has not been in their focus. Even if models are executed, they are usually not software components.

In this presentation, a realization concept for combining models and code in the form of *Model-Integrating Components* (MoCos) is introduced. A MoCo is a non-redundant, reusable and executable combination of logically related models and code in an integrated form where both parts are stored together in one component. MoCos enable the interplay of code with (i) design-time models of software (e.g., feature models, documentation), (ii) reflective models@run.time [BBF09] as well as (iii) stand-alone, non-reflective and possibly executable models. A sketch of the core idea is given in Figure 1.

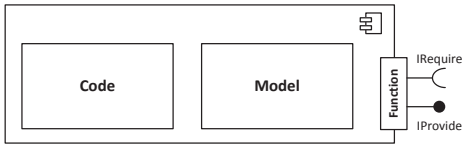

Figure 1: High level sketch of a MoCo

The *code part* of a MoCo allows better performance than model execution for performance-critical functionality. Moreover, it supports the use of existing software libraries and enables the connection to third-party middleware. The *model part* of a MoCo supports flexibility and comprehensibility of the component as all models can be queried and transformed (using dedicated languages) and may be directly executed by model interpreters. This supports software engineers when evolving components, and system administrators when observing and managing a running system.

The *component realization concept* proposed here supports the modular development of software systems where users of components cannot differentiate between MoCos or other, more traditional, components. Code and models are both *first-class entities* with equal rights inside the component. We present this realization concept in the form of an abstract template that is compatible with existing technologies. In addition, we provide a reference implementation using (i) Java for programming, (ii) OSGi for components and (iii) TGraphs [ERW08] for modeling.

In order to study the feasibility of the introduced MoCo template and its reference implementation, we apply them in the fictional context of an insurance company that equips its field staff with an assistive *Insurance Sales App* (ISA) for the Android™ mobile platform.

Acknowledgements. This ongoing work is supported by the Deutsche Forschungsgemeinschaft (DFG) under grants EB 119/11-1 and EN 184/6-1.

References

[BBF09] Gordon Blair, Nelly Bencomo, and Robert B. France. Models@run.time. *Computer*, 42(10):22–27, 2009.

[BCW12] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool, 2012.

[ERW08] Jürgen Ebert, Volker Riediger, and Andreas Winter. Graph Technology in Reverse Engineering, The TGraph Approach. In Rainer Gimnich, Uwe Kaiser, Jochen Quante, and Andreas Winter, editors, *10th Workshop Software Reengineering (WSR 2008)*, volume 126, pages 67–81, Bonn, 2008. GI.

[SGM02] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley, second edition edition, 2002.

[SV06] Thomas Stahl and Markus Völter. *Model-Driven Software Development*. Wiley, 2006.