Round-Trip Engineering of Eclipse Plug-Ins Using Eclipse Workbench Part Interaction FSML

Demonstration of the Prototype

Michał Antkiewicz and Krzysztof Czarnecki
University of Waterloo
{mantkiew, kczarnec}@swen.uwaterloo.ca

Abstract

A Framework-Specific Modeling Language (FSML) is a kind of Domain-Specific Modeling Language that is used for modeling framework-based software. FSMLs enable automated round-trip engineering over non-trivial model-to-code mappings and thereby simplify the task of creating and evolving framework-based applications. In this demonstration, we present a prototype implementation of Eclipse Workbench Part Interaction, a FSML capturing an aspect of Eclipse plug-in development. We walk through an example Eclipse plug-in development scenario and demonstrate the round-trip engineering capabilities of the prototype.

Categories and Subject Descriptors D.2.1 [*Software Engineering*]: Requirements/Specifications—Tools; D.2.2 [*Software Engineering*]: Design Tools and Techniques—Computer-aided software engineering (CASE); D.2.4 [*Software Engineering*]: Software/Program Verification

General Terms Documentation, Design, Languages, Verification

Keywords Object-oriented application framework, domain-specific modeling, Framework-Specific Modeling Language, FSML, round-trip engineering, Workbench Part Interaction, WPI, Eclipse

1. Description

What problems are addressed? Using models in software engineering requires establishing and maintaining consistency between the models and the implementation code. Round-trip engineering is an approach to model-driven software development, where the models and the code are synchronized by *reconciling* the differences rather than just performing forward engineering to produce code from models and reverse engineering to produce models from code. Reconciliation propagates individual changes among related artifacts by updating them rather than recreating and replacing previous versions of the artifacts.

Framework-Specific Modeling Languages (FSMLs) were recently proposed as a means to aid the framework instantiation process [3]. FSMLs are defined on top of object-oriented application frameworks and are used to express models showing how framework-provided concepts are used in framework-based applications. FSMLs enable automated round-trip engineering over non-trivial model-to-code mappings.

What will the audience be seeing? In this demonstration, we present a prototype implementation of Eclipse Workbench Part Interaction (WPI) [2], a FSML capturing an aspect of Eclipse plug-in development. WPI offers explicit definitions of concepts provided by the Eclipse platform such as *view*, *editor*, *listens to selection*, *requires adapter*, and *provides selection*.

During the demonstration, we first create a sample Eclipse plug-in using an Eclipse-provided wizard that generates the necessary implementaton code. The code implements a single *view*, i.e., contains an instance of the *view* concept. We then automatically reverse engineer the plug-in code to create its WPI model. The WPI model is then manually edited by removing or modifying the properties or *features* of the concept instances that were detected in the code or by adding new concept instances to the model. The resulting model is shown in the upper part of Figure 1. The model contains several instances of framework-provided concepts, including the view that was detected in the code by the reverse engineering process and an instance of the *requires adapter* concept. The *Properties* view shows features of the selected view. While the model is being modified, the corresponding code can also be modified in a way that amounts to adding or removing concept instances or modifying their properties.

After each change, we execute a synchronization procedure which compares the model and the code and computes *synchronization states* describing the discovered changes. The *Model-Code Synchronization* view in Figure 1 contains example synchronization states: *modification* for `SampleView`, *reverse addition* for `extendsPageBookView` feature, and *forward modification* for `PartId` feature. The new and old values are indicated for the `PartId` feature. A new instance of the *listens to parts* concept has been also recognized in the code. We reconcile each changed concept automatically using the *reconcile* action and then show the results of the reconciliation. We also remove an implementation of a mandatory feature from the code and show how broken concept instances can be recognized and automatically be fixed.

An on-line version of the demonstration is available [1].

What makes the software relevant to the OOPSLA community? *Object-oriented application frameworks* are one of the most effective and widely used software reuse techniques. However, the creation of framework-based applications (i.e., *framework completion* or *framework instantiation*) is often challenging. The developers need to know what the framework-provided concepts are and how to instantiate them in order to achieve the desired effect. Furthermore, the developers also need to know which concepts have already been instantiated and how they were instantiated in order

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

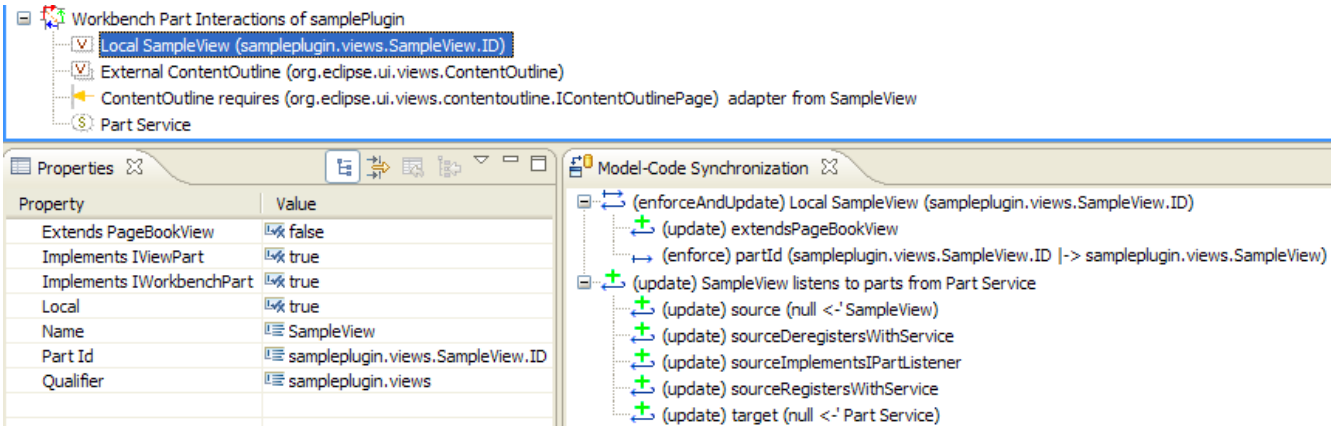

Figure 1. Example WPI model and result of synchronization shown in Model-Code Synchronization view

to understand the application. These challenges can be addressed by using models to aid framework instantiation and round-trip engineering to keep the models and the code consistent. A FSML defines a language for expressing such models and FSML concept definitions encode the framework instantiation knowledge and enable automatic creation and recognition of concept instances in the code.

What is unique about the design or implementation? WPI is a practical and non-trivial FSML. WPI *concepts* are decomposed into *atomic features* that correspond to basic implementation steps or choices and *composite features* that represent higher-level implementation choices. The semantics of every concept and every feature is defined using *forward* and *reverse mappings*. Forward mappings define how a *concept instance* or a feature can be created, updated and modified in the code. Examples of forward mappings include creating a class, implementing an interface, adding a class member, modifying XML configuration file, weaving a before advice, and changing the value of an argument of a method call. Reverse mappings define how a concept instance of a feature can be recognized in the code. Examples of reverse mappings include verifying structural constraints, such as `A extends B`, behavioral constraints, such as `A calls B.m()`, retrieving values of method call arguments, and handling XML configuration files.

FSMLs enable round-trip engineering by allowing for a fine-grained execution of the mappings. The prototype supports *agile round-trip engineering* where both the model and the code can be created and modified independently and synchronized whenever desired, provided that a model can be completely retrieved from the code using static analysis.

What underlying technologies are used? Abstract syntax, including well-formedness constraints, is implemented using Eclipse Modeling Framework [4] and its model validation framework. Reverse mappings use the Java Model, AST, query, and pattern matching APIs of Eclipse's Java Development Tools (JDT) [5] and the type inference engine of the *Infer Generic Type Arguments* refactoring [6]. Forward mappings use JDT's Java Model and AST rewriting APIs. Details of the WPI FSML design and prototype implementation are available in the technical report [2].

What techniques were used to build the software? The prototype consists of three Eclipse plug-ins. The metamodel and abstract syntax editor were generated using Eclipse Modeling Framework. The mappings were implemented in Java.

What are the interesting technical details and challenges? The reverse mappings are limited by availability and effectiveness of static analysis techniques. Implementation of the *addition*, *modification*, and *removal* aspects of the forward mappings is particularly challenging as it involves direct code manipulation in different places and needs to be flexible enough to take different coding styles into account.

2. About the Authors

Michał Antkiewicz received the MSc (2003) degree in Computer Science from the Wroclaw University. Currently, he is a Ph.D. candidate in Electrical and Computer Engineering department at the University of Waterloo. He is a member of the Generative Software Development Lab and a Ph.D. Fellow with IBM Centers for Advanced Studies in Ottawa. Previously, he worked on feature modeling and feature-based model templates. Currently, he works on Framework-Specific Modeling Languages. His main research interests include software product line engineering, domain-specific modeling, and code generation and static analysis in the context of round-trip engineering.

Krzysztof Czarnecki received the Dipl.-Inf. (1995) from the Ilmenau Technical University, MSc (1994) from California State University at Sacramento, and Ph.D. (1998) from the Ilmenau Technical University. Currently, he is an Assistant Professor in Electrical and Computer Engineering department, at the University of Waterloo, where he heads the Generative Software Development Lab. Before coming to Waterloo, he spent eight years at Daimler-Chrysler Research working on the practical applications of generative programming. He co-authored the book "Generative Programming" (Addison-Wesley, 2000). His main research interests include generative software development, model-driven software development, software product lines, and software design.

References

[1] M. Antkiewicz. Eclipse Workbench Part Interaction FSML on-line demo. `http://gp.uwaterloo.ca/files/WPIDemo/`.

[2] M. Antkiewicz and K. Czarnecki. Eclipse Workbench Part Interaction FSML. Technical Report 2006-09, ECE, University of Waterloo, 2006. `http://gp.uwaterloo.ca`.

[3] M. Antkiewicz and K. Czarnecki. Framework-Specific Modeling Languages with Round-Trip Engineering. In *MoDELS*, 2006.

[4] Eclipse Foundation. Eclipse Modeling Framework (EMF). Available at `http://www.eclipse.org/emf`.

[5] Eclipse Foundation. Java Development Tools (JDT). Available at `http://www.eclipse.org/jdt`.

[6] F. Tip, R. Fuhrer, J. Dolby, and A. Kieżun. Refactoring techniques for migrating applications to generic Java container classes. Technical Report RC 23238, IBM T.J. Watson Research Center, 2004.