Abstract. We propose Framework-Specific Modeling Languages (FSMLs) as a special category of Domain-Specific Modeling Languages that are defined on top of an object-oriented application framework. They are used to express models showing how framework-provided abstractions are used in framework-based application code. Such models may be connected with the application code through a forward and a reverse mapping enabling round-trip engineering. We also propose a lightweight and iterative approach to round-trip engineering. Furthermore, we present a proof-of-concept FSML for modeling the interaction of workbench parts within Eclipse. Finally, we identify a number of challenges, opportunities, and directions for future research on FSMLs.

1 Introduction

Object-oriented application frameworks are one of the most effective and widely used software reuse technologies today. The creation of framework-based applications is often called framework completion. The resulting framework completion code implements the difference in functionality between the framework and the desired application. A framework provides a set of abstractions, referred to as framework-provided concepts, and means of instantiating them in the framework completion code. The concepts are instantiated by writing the completion code.

Unfortunately, framework completion can be challenging. The application programmers need to know which framework-provided concepts are available and how to instantiate them in order to get the desired effect. The instantiation, which usually involves steps such as implementing interfaces or invoking framework services, is challenging since the implementation choices provided by the framework are not always compatible. Furthermore, the developers need to be able to see how the framework-provided concepts are instantiated in the application code. The latter is challenging since some concepts instances, such as collaborations among objects, are usually scattered in the completion code.

In this paper, we identify the challenges of framework completion and characterize framework-based application development as a mixture of concept configuration and open-ended programming with restrictions. As a main contribution, we show how the challenges of framework completion can be addressed by explicitly capturing the framework-provided concepts as a Framework-Specific Modeling Language (FSML) with round-trip engineering. Furthermore, we propose
an agile round-trip engineering approach, which is inspired by the Concurrent Versioning System (CVS) and its Eclipse user interface [1] and can operate over non-trivial abstraction gaps thanks to mappings enabled by FSMLs. Finally, we describe a proof-of-concept prototype implementation of a FSML with round-trip engineering for an aspect of Eclipse plug-in development and discuss the merits and limitations of our approach.

2 Running Example: Eclipse Workbench Part Interaction

Eclipse [1] is a universal, open-source platform for building and integrating tools, which is implemented as a set of Java-based object-oriented frameworks. In this paper, we consider a particular part of the Eclipse Application Programming Interface (API), which is concerned with workbench parts and their interactions. Workbench parts are the basic building blocks of the Eclipse Workbench, which is the working area of an Eclipse user. The parts can interact in various ways, for example, by exchanging events.

In this paper, we only consider two kinds of workbench parts, namely editors and views. An editor is used for displaying and editing the contents of input resources. An example of an editor is the Java editor included in the Eclipse Java Development Tools (JDT) [1]. A view is also used for displaying and editing information, but unlike an editor, a view is not associated with any particular input resource. An example of the standard workbench view is Content Outline, which is used to display the outline of an input resource opened in an active editor. Editors and views have to be contributed to the Workbench by declaring them in a plug-in manifest files. The Workbench scans manifest files upon startup and makes contributed workbench parts available to the user.

Workbench parts interact in various ways. In this paper, we consider two kinds of part interactions, namely listens to parts and requires adapter. For example, the Content Outline view listens to part activation events by registering itself as a listener with the Workbench Part Service and, therefore, it participates in the listens to parts interaction. When an editor, such as the Java editor, is activated, the view will receive an activation event. In response to this event, the view will ask the editor for its IContentOutlinePage adapter, which is used to display the outline of the editor’s input resource. Therefore, the view and the editor participate in the requires adapter interaction, with the view as a source and the editor as a target. For a detailed description of the example see [2].

3 Challenges of Framework Completion

Framework completion is often difficult due to the extensive knowledge about the framework design that is needed in order to write and understand the completion code. In particular, application developers face the following challenges.

Knowing how to complete a framework. The developers need to know what are the framework-provided concepts and how the concepts are instantiated in