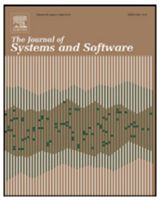
The Journal of Systems and Software 84 (2011) 2234–2252

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss



ReuseTool—An extensible tool support for object-oriented framework reuse

Toacy C. Oliveira[a,b,*], Paulo Alencar[b], Don Cowan[b]

[a] PESC/COPPE Federal University of Rio de Janeiro, Brazil
[b] David Cheriton School of Computer Science, University of Waterloo, Canada

ARTICLE INFO

Article history:
Received 3 November 2010
Received in revised form 6 June 2011
Accepted 14 June 2011
Available online 21 June 2011

Keywords:
UML
Object-oriented framework
Software process
Software reuse

ABSTRACT

Object-oriented frameworks have become a popular paradigm used to improve the software development lifecycle. They promote reuse by providing a semi-complete architecture that can be extended through an instantiation process to integrate the needs of the new software application. Instantiation processes are typically enacted in an ad-hoc manner, which may lead to tedious and error-prone procedures. This work leverages our previous work on the definition of RDL, a language to facilitate the description of instantiation process, and describe the ReuseTool, which is an extensible tool to execute RDL programs and assist framework reuse by manipulating UML Diagrams. The ReuseTool integrates a RDL Compiler and a Workflow Engine to control most of the activities required to extend a framework design and, therefore, incorporates application-specific needs. This work also describes how the tool can be extended to incorporate new reuse activities and provides information of its use based on an exploratory Case Study.

1. Introduction

With the popularity of object-oriented languages such as Java and C++, and the pressure for improving software development productivity, the concept of object-oriented frameworks gained momentum and became the de facto approach to develop complex software systems. In essence, frameworks are reusable assets in the form of quasi-complete and flexible software components, specially assembled to reduce the effort to develop new applications within a specific domain (Bosch et al., 1999). An object-oriented framework defines a set of permanent features known as frozen-spots to deliver unchangeable functionality (Pree, 1995). It also uses typical object-oriented techniques to incorporate flexible features, the hot-spots (Pree, 1995), which must be extended appropriately to integrate application-specific needs (Markiewicz and Lucena, 2001) (Mattsson and Bosch, 2000). Examples of major current object-oriented frameworks are: J2EE (J2EE, 2010), DotNET (DotNet, 2010), Hibernate (Hibernate, 2010), JUnit (Junit, 2010), Eclipse (Eclipse, 2010), Struts (Struts, 2010) and MooTools (MooTools, 2010).

In contrast to Software Product Lines (Atkinson et al., 2001), where reuse occurs from putting together a set of pre-defined software components, developers reusing object-oriented-frameworks need to engage in a more difficult reuse procedure, which typically involves understanding the framework design rationale and programming. For example, a developer must be knowledgeable about the order in which hotspots are refined since they may need to be refined in a specific implicit sequence. Considering frameworks may contain several hotspots, such reuse order should be clearly specified to avoid faulty reuse processes (Fayad et al., 1999) (Kirk et al., 2005) (Hou et al., 2005). Moreover, it is expected that the reuse processes of complex frameworks are supported by tools to assist handling all the constraints that need to be satisfied when reusing a framework.

In this work we describe the ReuseTool as a way to assist object-oriented frameworks reuse. The tool rationale is to orchestrate reuse actions within a reuse process that is specified by the framework developer and executed interactively by the framework reuser. The reuse process is specified using the RDL (Reuse Description Language) (Oliveira et al., 2004, 2007, 2002), which is a special language that allows the specification of process flows such as sequencing, loops and branches, and also supports commands to manipulate UML-based framework designs. As a result, the ReuseTool is capable of tailoring the UML (2010) framework design with new model elements that will represent the needs of the application under development.

This work is organized as follows. In Section 2 we describe the approach overview using a fictitious reuse project. In Section 3 we describe the ReuseTool Architecture and the associated Software Reuse Process. Section 4 illustrates the approach with a detailed example and also brings information about our experience using the ReuseTool. As RDL is a major component in our solution, Section

* Corresponding author at: PESC/COPPE Federal University of Rio de Janeiro, Brazil. Tel.: +55 21 2562 8672.
E-mail addresses: toacy@cos.ufrj.br, toacy@csg.uwaterloo.ca (T.C. Oliveira), palencar@csg.uwaterloo.ca (P. Alencar), dcowan@csg.uwaterloo.ca (D. Cowan).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.06.030

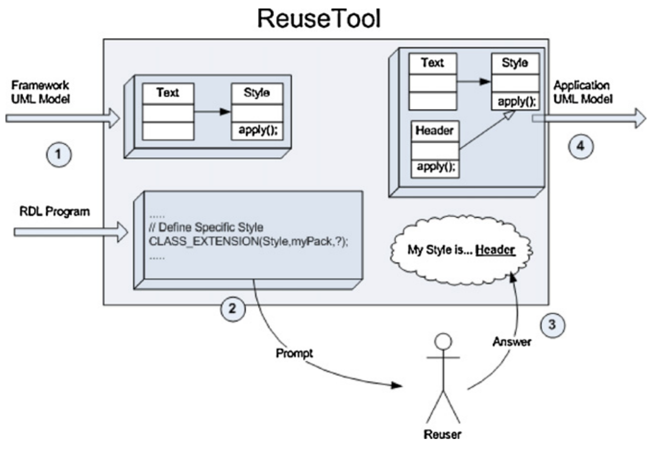


Fig. 1. ReuseTool operation.

5 describes how RDL was enhanced to cope with the tool needs. Section 6 describes how the ReuseTool can be extended by exposing the tool hotspots and the associated RDL program.

2. Approach overview

As pointed out by Krueger (Krueger, 1992), most reuse processes involve four dimensions: Abstraction, Selection, Specialization and Integration. The Abstraction and Selection dimensions deal with the cognitive aspects of identifying a set of possible reusable assets and selecting the most suitable to implement the requirements associated with the new application. The cognitive nature of these two dimensions relies on the fact that requirements are typically expressed with natural language, which demands human interpretation. The Specialization dimension appears once the reusable artifact is chosen and needs to be tailored to incorporate the application specific increments. Last but not least, the Integration dimension deals with combining several reusable artifacts into a single system.

In this scenario, the ReuseTool aims at facilitating the Specialization dimension with possible effects to Selection and Integration. The Specialization dimension for object-oriented frameworks deals with adding new design elements and code to the original framework design, where each new element connects to a framework's hotspot. The connection between the new design element and the hotspots extends the framework with information that is specific to the application under development.

In order to orchestrate the framework specialization, the ReuseTool takes an RDL Program and the Framework UML Model as input and produces the Application UML Model. The Framework UML Model represents framework's classes and relationships and also provides useful information on how the framework code can be obtained. The RDL program details how the Framework UML Model should be manipulated to accommodate new design elements related to the new application and the Application UML Model is the final application design.

As illustrated in Fig. 1 the process of executing the ReuseTool is quite straightforward. In Fig. 1-1 (Circle 1) the Framework UML Model and the RDL Program are passed to the tool. The framework model has classes *Text* and *Style* to indicate that the hypothetical framework is capable of applying styles to a given text such as in a word processor. The framework developers understand they cannot provide all types of styles so, in this case, they have decided to leave the style feature as a hotspot that must be configured by the application developer.

Fig. 1-2 (Circle 2) shows how the framework developers have exposed the *Style* hotspot and how to extend it. The rationale is based on the fact that the framework design has the class *Style* and this class is responsible for the style feature. As a result, the specialization of hotspot *Style* requires the specialization of the *Style* class, which is represented in the RDL Program by the statement “CLASS_EXTENSION (Style,myPack,?);”.

The CLASS_EXTENSION command indicates the class *Style* must be specialized with a new class that will represent the new style. The question mark (?) passed as parameter is a RDL feature called *Reuser Interaction* to indicate a placeholder for a name that should be given at runtime (i.e. reuse-time) by the framework reuser. The Reuser interaction is shown in Fig. 1-3 (Circle 3), where the reuser indicates the new style will be called *Header*.

The result of the reuse process is an extended model called the Application UML Model (Fig. 1-4), which contains the design elements that represent application specific requirements. In our example, the new model has the class *Header* as a sub-class for class *Style*, satisfying the reuser requirements from Fig. 1-4 (Circle 4).

Besides orchestrating reuse actions along the Specialization dimension, the ReuseTool can also impact the Selection and Integration dimensions. Selection can take into consideration the effort needed to reuse a framework measured in terms of the complexity to execute RDL programs. Frameworks with complex RDL programs may be avoided as a higher complexity may indicate more reuse effort is needed. The ReuseTool can also assist with Integration, making it more straightforward since some integration actions can be incorporated into the RDL language as a new type of reuse action. Further impact on the Selection and Integration dimensions are still under investigation.

3. Reuse tool

The ReuseTool aims at helping developers to follow a strict process when working on the Specialization Dimension as discussed in Section 2. By a strict process we mean a process that guides the framework reuser to extend all the required hot-spots in a sequence that is pre-defined by the framework engineer, thus following the same rationale used in the framework implementation.

In order to create a process-based execution environment we have established a set of requirements for the tool.

Adopt current trends. Besides the use of RDL, the tool should follow current trends in the Information Technology scenario to avoid a steep learning curve and allow further integration with other approaches.

Handle object-oriented frameworks. The object-oriented programming paradigm has been widely adopted, and has vast framework examples we can leverage on. It also provides a potential market for our approach.

*Provide ways to pause and resume the process***.** Since a reuse process can be lengthy, the ability put the process in a suspension mode and resume with no loss of context is an important feature.

Provide infrastructure for a multi-“reuser” scenario. Nowadays frameworks have to tackle multi-faceted scenarios, being typically complex and requiring different types of expertise during customization. As a result, the framework reuse process became multi-user by nature and our tool should provide means to support it.

Extensibility. RDL provides the building blocks to represent the reuse process and manipulate UML Models. However it's important to leave space for improvements, such as handling forthcoming UML characteristics and new programming paradigms such as Aspect-Oriented Programming. In this context, the tool must be open for extension, and become itself a framework that can be extended to handle other reusable assets.