UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Faculty of Electrical Engineering – Computer Science – Mathematics
Heinz Nixdorf Institute and Department of Computer Science
Software Engineering Group
Zukunftsmeile 1
33102 Paderborn

ProtoCom3: Design and Implementation

by
Thomas Zolynski and Sebastian Lehrig
Zukunfstmeile 1
33098 Paderborn

Paderborn, August 2013

Contents

1 Introduction

ProtoCom is a model-to-text transformation that generates performance prototypes from Palladio Component Model (PCM) models. It was first introduced by Steffen Becker in his dissertation [Bec08] and later improved ("ProtoCom2") by Sebastian Lehrig and Thomas Zolynski [LZ11].

1.1 Motivation

As of today, ProtoCom is realized as an Xpand transformation. This transformation shares a common transformation code basis with SimuCom that is used for performance simulations. Both, the usage of Xpand and the common code base, are now causing difficulties.

While ProtoCom's and SimuCom's generator output share similarities in structure and behavior, they differ at many places. For instance, one major difference is the communication between components. While SimuCom is simulating systems locally, ProtoCom's performance prototypes need to communicate in distributed systems. In consequence of Java's nontransparent implementation of RPC, this required several small changes scattered over the code base. Owing to the lack of inheritance mechanisms in Xpand, template methods in combination with aspect-oriented programming had to be used to implement these divergences in transformations. Over the time and due to several people extending and patching the transformation code, it became the purest incarnation of spaghetti code in which new features usually break other transformations.

With regards to the current goal to use ProtoCom for more deployment targets than the current Java SE target, adding more transformations would lead to even more maintainability issues. Therefore, we propose a new transformation solely for ProtoCom. In the same breath, we base this new transformation on Xpand's successor Xtend2 to cope with modernization issues. We refer to our new ProtoCom transformation as "ProtoCom3".

1.2 Scenario: The Alice&Bob-System[1]

As example scenario, we consider the *Alice&Bob* system as shown in Fig. 1.1. This system consists of two server instances, for example two Glassfish servers. On one server, the *Alice* component is deployed that provides the interface *IAlice*

[1] Note: This section is taken from Daria Giacinto's and Sebastian Lehrig's "EJB ProtoCom: Architecture" guide.

with the method *callBob()*. The other server deploys the *Bob* component that provides the interface *IBob* with the method *sayHello()*. The *IAlice* interface is provided to a user who can invoke this method through a client-side technology like a browser. This invocation can be received by a component on the server side. In our case, the particular component is the *Alice* component implementing the *IAlice* interface.

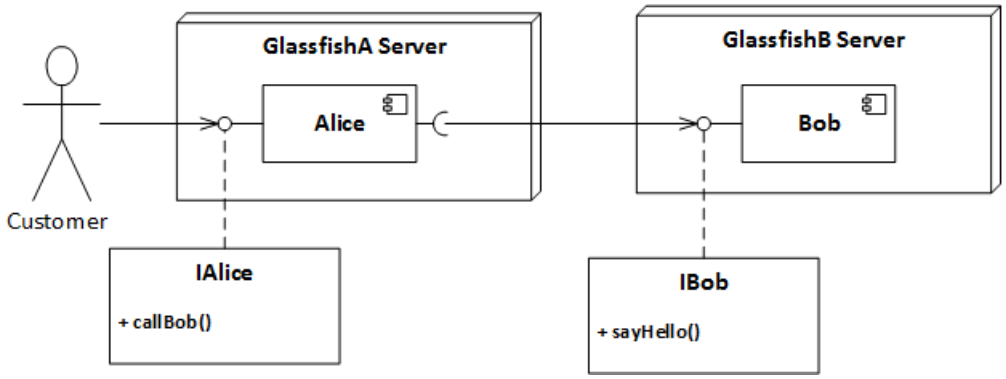

Figure 1.1: Alice&Bob-System

2 Concept

2.1 Structure

To give an idea of how transformations are processed in ProtoCom3, this section gives an overview of the steps taken during code generation. First, take a look at the class diagram in Figure 2.1. This diagram shows a simplified excerpt of classes used during transformation. Please note that the boxes above classes are the inserted generic types that are only valid in this excerpt. For instance, the metaclass `PCMRepresentative` binds `BasicComponent`. Illustrating bindings like this is **not** the correct UML notation, however, it should be easier to understand this way.

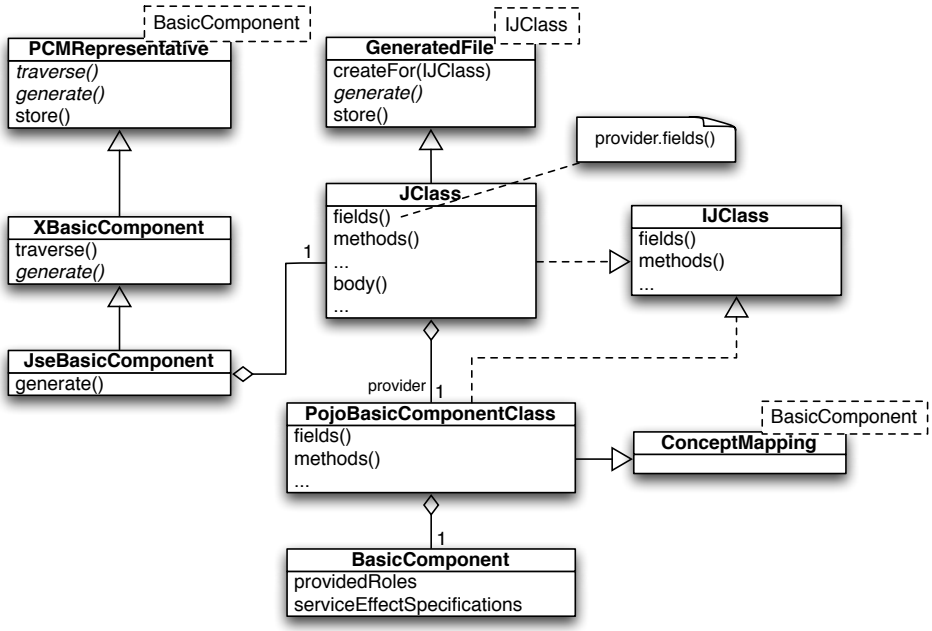

Figure 2.1: Transformation classes (excerpt).

- The PCM model is being traversed, starting from the root elements like `Repository`, `System`, etc. For each PCM entity type, one class of the form `XEntityName` is specified. This class specifies how the model is further traversed. In this example, `XBasicComponent` is such a class.

- Depending on the target technology (Java SE, Java EE, etc.), subtypes for these `XEntity` classes are specified. These subtypes describe which files are generated for this particular PCM entity type. In this example, `JseBasicComponent` creates one Java class per `BasicComponent` for the core implementation of the corresponding component.
- `GeneratedFile` is an abstract type that encapsulates the generation and storage code. All subtypes of this type are templates for a specific data or compilation unit file. The generic type specifies an interface for the content provider used by its subtypes' templates.
- `JClass` is one subtype of `GeneratedFile`. It specifies templates to create Java classes (`body()`, etc.), but it does **not** specify how these templates are filled. Instead, it calls its content provider (`PojoBasicComponentClass` in this example) using delegation. Both, the subtype of `GeneratedFile` and the content provider, have to implement the same interface because this interface specifies the delegated calls.
- The provider classes have a reference to one PCM entity instance whose information are used to fill in the template's calls.

2.2 Packages

This section briefly describes most of the source code packages, starting at `de.upb.swt.pcm.protocom`. Don't bother reviewing the workflow package `de.uka.ipd.sdq.codegen.protocom`!

.lang Language templates and `GeneratedFile`.

.lang.java / .lang.java.impl Templates for Java. Including `JClass`, `JInterface` and their common supertype `JCompilationUnit`, as well as their interfaces.

.lang.java.util Collection of Java template snippets used by several providers.

.tech Content provider for different target technologies.

.tech.rmi Content provider for Java POJO+RMI files.

.traverse.framework Classes for traversing PCM models.

.traverse.jse Java Standard Edition subtypes of the traversing classes. Defining which files using which content providers should be used to generate a POJO+RMI performance prototype.

2.3 Technologies

ProtoCom3 uses Xtend and Google Guice:

Xtend. "Xtend is a statically-typed programming language which translates to comprehensible Java source code." In contrast to Xpand, which is only a template language, Xtend also features a complete programming language. Documentation can be found on
http://www.eclipse.org/xtend/documentation.html.

Google Guice. Guice is a dependency injection and configuration framework created by Google. It is used to configure the file system access objects in `GeneratedFiles` and to configure the traversing classes without subclassing them. Documentation can be found on
https://code.google.com/p/google-guice/.

3 Installation Guide

While ProtoCom might already work with the newest version of Eclipse (Keplar), currently Juno is used for development. Due to recent changes in the workflow engine, the nightly build of PCM is necessary. For getting a functional ProtoCom development environment, the following steps need to be executed:

1. Install *Eclipse Juno Modeling Tools* (`eclipse-modeling-juno`[1]).
2. Add `http://sdqweb.ipd.kit.edu/eclipse/palladio/nightly/` as an update site. Install all packages (without source codes).
3. Install `Xtend 2.4.2` and all necessary dependencies.
4. From our SVN repository, check out the `Code/ProtoCom` and `ProtoComWorkflow` folder.
5. Create a new *Run Configuration* of the type *Eclipse Application*.
6. Start the new run configuration.
7. From our SVN repository, check out the `Code/AliceAndBob` PCM example.

Now you can create ProtoCom run configurations for the current and new version. Note that the new version does not create runnable performance prototypes yet!

[1] http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/junor

Bibliography

[Bec08] Steffen Becker. *Coupled model transformations for QoS enabled component-based software design.* PhD thesis, Carl von Ossietzky University of Oldenburg, 2008. http://d-nb.info/989923983.

[LZ11] Sebastian Lehrig and Thomas Zolynski. Performance prototyping with protocom in a virtualised environment: A case study. In *Proceedings to Palladio Days 2011, 17-18 November 2011, FZI Forschungszentrum Informatik, Karlsruhe, Germany*, 2011.