UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

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

ProtoCom 3.1 Code Review

by
Christian Klaussner

Paderborn, May 2015

Contents

1 Motivation

Palladio allows performance engineers to model performance-relevant aspects of a software architecture. In order to conduct performance measurements according to these models, ProtoCom[1] is used to automatically generate performance prototypes for the desired platforms, e.g., Java SE and Java EE. Earlier versions of ProtoCom focused only on Java SE but with ProtoCom 3 the focus has shifted to provide extensible transformations in order to support multiple platforms, especially in the context of cloud computing.

With ProtoCom 3.1, we added support for generating performance prototypes for Java EE Servlets running on the SAP HANA Cloud. This addition required substantial changes in the ProtoCom framwork and transformations. Therefore, we want you to review the code of the extension to ensure its quality.

This code review guide conists of three parts. First, Chapter 2 describes how Palladio, ProtoCom, and their dependencies can be installed in order to test the code being reviewed. Second, Chapter 3 gives an overview of the framework's architecture and the locations of the corresponding source code, and states instructions for the code review. Finally, Chapter 4 gives a short overview of a new concept in the ProtoCom transformations.

[1] `https://sdqweb.ipd.kit.edu/wiki/ProtoCom`

2 Installation

This chapter describes how to install Palladio (which includes ProtoCom) and all dependencies required to build performance prototypes for the SAP HANA Cloud. If you already have an Eclipse installation with SVN, you can skip the first two steps. **It is important that you install the packages in the same order as described here. Otherwise, Eclipse's package management may be unable to resolve some dependencies.**

1. Download and install the *Eclipse Modeling Tools (Luna)* (`https://www.eclipse.org/downloads/`). You can use other pre-packaged Eclipse versions but then most of the modeling components need to be downloaded while installing Palladio, which takes longer than downloading the bundle.

2. Go to *Help* → *Eclipse Marketplace...* and install *Subversive*. Restart Eclipse after the installation is finished. Change to the SVN perspective and install the latest version of SVN Kit. Restart Eclipse again.

3. Go to *Help* → *Install New Software...* and paste the URL `http://download.eclipse.org/webtools/repository/luna/` in the update site text field and install the latest version of the *Web Tools Platform*. Restart Eclipse when the installation is finished.

4. Go to *Help* → *Install New Software...* and paste the URL `https://tools.hana.ondemand.com/luna` in the update site text field to load the repository of the SAP HANA Cloud Eclipse plugin. Expand *SAP HANA Cloud Platform Tools* and select *SAP HANA Cloud Platform Tools for Java*. Click *Next* and restart Eclipse when the installation is finished.

5. Go to *Help* → *Install New Software...* and paste the URL `http://sdqweb.ipd.kit.edu/eclipse/palladio/nightly/` in the update site text field to load the repository of the Palladio Nightly build. Select everything from *Palladio Component Model* and click *Next*. After the dependencies are resolved, click *Next*, accept the license agreements, and click *Finish* to install the components. Restart Eclipse when the installation is finished.

6. Go to `https://tools.hana.ondemand.com/#cloud` and download *Java Web Tomcat 7* and the *SAP JVM (Version 7)* for your platform. Unpack both archives to a (fixed) location of your choice.

7. Add the Palladio repository in Eclipse: `http://www.palladio-simulator.com/de/tools/source_code/` and check out all projects that are in the folders *Core/trunk/ProtoCom* and *Core/trunk/Workflow*.

8. Go to *Run* → *Run Configurations...*, create an *Eclipse Application* configuration, and run it. After the new Eclipse instance is started, change to the Java perspective in the upper right corner.

9. Go to *Preferences* → *Java* → *Installed JREs* and add the previously downloaded SAP JVM (as *Standard VM*). Make sure to select the *jre* subfolder as *JRE home*.

10. Open the *Server* view (*Window* → *Show View* → *Other...*) and create a new *SAP/Java Web Tomcat 7 Server*. After clicking *Next*, select the directory of the previously downloaded *Java Web Tomcat 7* and use *sapjvm_7* as JRE.

11. Download and run MongoDB as described here: `http://bit.ly/1bRMtnX`

12. Go to *File* → *New* → *Other...* and create new Palladio project. Use the *Minimum Example Template*. Open the file `default.usagemodel_diagram` and change the *Think Time* in the yellow box to a *1*.

13. Go to *Run* → *Run Configurations...* and create a *ProtoCom Generator* configuration. Select the *Allocation File* and *Usage File* from the project and change the *Transformation Target* in the *Analysis Configuration* tab to *JavaEE Servlet Performance Prototype* and click *Run*.

14. Right click on the generated project and select *Run As* → *Run on Server* and select the created Tomcat server. After the server is started, the running prototype can be accessed under `http://localhost:8080/org.palladiosimulator.temporary`.
Don't use the browser that is integrated in Eclipse. It can't display the page correctly.

15. After running the calibration with the default values, start the containers and the system, download the JMX file and open it in JMeter[1]. Press the play button in JMeter to run the performance measurements. After the simulation is finished, the results appear on the prototype's web interface and can be downloaded.

16. Go to *Window* → *Show View* → *Other...* and open the *Experiments View*. Click on the button next to the plus button (*Open a Data Source*), select *File Datasource* and check *Link to folder in the file system* in the *Advanced*

[1] `http://jmeter.apache.org/`

section. Select the extracted results archive that you downloaded previously and click *Finish*. Investigate the entries in *Default Experiment* → *Experiment Runs* by double-clicking *accountingInterface_chargeForEspresso0 [ID:0]*. Choose *JFreeChart Response Time TimeSeries* in the pop-up.

Troubleshooting

In rare cases, the server throws a `MalformedURLException` on Mac OS X: "Local host name unknown: java.net.UnknownHostException: MacBook-Pro.local: MacBook-Pro.local: nodename nor servname provided, or not known" or similar.

Solution

```
MacBook-Pro:~$ scutil --get LocalHostName
MacBook-Pro:~$ scutil --set LocalHostName "localhost"
MacBook-Pro:~$ scutil --get LocalHostName
```

3 Framework

The refactoring of the framework was planned to be accomplished in two steps. First, shortcomings of the existing framework implementation should be identified by implementing a new transformation (for Java EE Servlets) and by trying to reuse as many components as possible in the process. Second, the insights gained from this implementation should be transferred to all existing transformations. With ProtoCom 3.1, only the first of these steps was implemented. Hence, there exist two separate frameworks: one for Java SE with RMI and one for Java EE Servlets). Assessing to what extent these frameworks can be integrated is one subject of this code review.

However, the main focus lies on the Java EE Servlets framework, whose architecture is illustrated in Figure 3.1. The diagram is separated into two parts, the framework components on the top and the generated, model-dependent prototype on the bottom.

Note: The packages referred to in this chapter are relative to the package *org.palladiosimulator.protocom.framework.java.ee*.

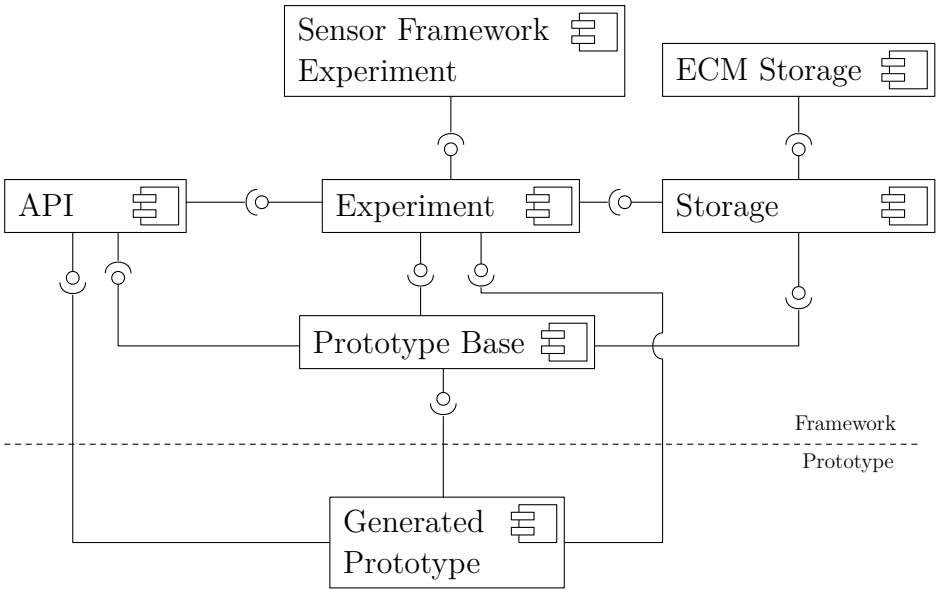

Figure 3.1: Architecture of the Java EE Servlets framework

One important insight of the refactoring was that we needed a more modular design of the components to make the framework more extensible. Therefore, we

created two framework components *Experiment* for takings measurements during simulations and *Storage* for persisting data and configuration files. These components provide platform-independent functionality for the rest of the framework, whereas the platform-dependent functionality for Java EE Servlets is realized in the components *Sensor Framework Experiment* and *ECM Storage*. The source code for these components can be found in the packages *experiment* and *storage*.

The *API* component provides a user interface from which the performance prototype can be managed. For example, it is used to start resource containers and present log output. The code for this component can be found in the *api* package.

All remaining packages of the framework are combined in the *Prototype Base* component. They provide other basic functionality, e.g., a RPC protocol for distributed components (*protocol* package) and the entry point for the prototype.

Review Tasks

1. Does the code follow the Palladio coding conventions? Instructions on how to set up Checkstyle can be found under `https://sdqweb.ipd.kit.edu/wiki/Palladio_Coding_Conventions`.

2. Start the review with the only-providing components, i.e., *Experiment* and *Storage* and their platform-dependent components *Sensor Framework Experiment* and *ECM Storage*. Does their purpose match the description above? Do the platform-independent components provide a suitable level of abstraction such that they can be extended for other platforms?

3. Investigate the *API* component. Does the API provide enough functionality to allow the implementation of other front-ends instead of the web interface, e.g., a console?

4. Investigate the *Prototype Base* component and its interaction with the generated prototype (including the RPC protocol). Are there any issues regarding the component's quality?

5. Is the architecture illustrated in Figure 3.1 suitable? Does it provide enough reusability with respect to other transformation targets, e.g., Java SE?

4 Transformations

In addition to changes in the framework, a new adapter concept was introduced in the transformations of ProtoCom. Previously, each transformation had direct access to the input model, as illustrated on the left side of Figure 4.1. However, the format of the PCM is inconvenient for most of these transformations because the underlying EMF variable names make the code difficult to understand. Furthermore, the model has to be pre-processed in some cases, e.g., to find entities in the model. As more transformations were developed, these pre-processing functions were not made accessible in a separate module, which led to duplicated code.

The Java EE Servlet transformations solve these problems by introducing a simple adapter layer between the PCM and the transformations, as illustrated on the right side of Figure 4.1. Instead of accessing the PCM directly, the transformations get all relevant data from the adapter layer, which is responsible for common tasks like finding entities in the model and translating entity names to valid Java identifiers.

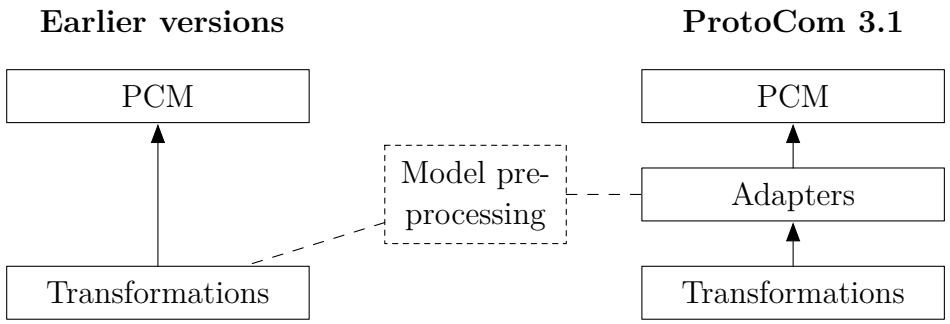

Figure 4.1: Adapters for PCM transformations

Currently, this adapter concept is used only in the Java EE transformations. The transformations for Java SE (both prototype and code stubs) and Java EE for EJBs still use the old approach. The code of the adapters can be found in the package *org.palladiosimulator.protocom.model*.

Review Tasks

1. Is the motivation for the supplied solution clear?
2. Is the solution suitable for this motivation?
3. Assess the quality of the adapter package.