Enhancing Decoupling in Portlet Implementation

Salvador Trujillo, Iñaki Paz, and Oscar Díaz

University of the Basque Country, San Sebastian (Spain)
{struji,jibparei,oscar}@si.ehu.es

1 Introduction

A Portlet is a Web component that processes requests and generates dynamic content. Portlet consumers (e.g. a portal) use Portlets as pluggable user interface components. Typically, a Portlet is rendered as a window in the portal, the Portlet being a main building block for portal construction. The recent delivery of standards, namely, *Web Services for Remote Portlets*[1] and *Java Specification Request* 168[2] promise to bring interoperability to the Portlet realm. Nothing is said about how the Portlet should be implemented. However, Portlet implementation could be quite an issue. Presentation and even the navigation logic could need to change not only for maintenance reasons, but also to cater for the idiosyncrasies of distinct Portals. Therefore, variability, and thus decoupling, is a main issue in Portlet development. In contrast to servlet techniques (i.e. modularisation), a single Portlet needs to implement (1) **control** code to determine which action is being requested, (2) what **action** needs to occur, (3) what **state** to leave the Portlet in, and (4) what **view** (fragment) to render back to the user. Without appropriate decoupling patterns, this code can mix up different concerns, making the separate evolution of each concern a real maintenance problem. This work[3] revises distinct approaches that gradually achieve higher levels of decoupling.

2 Decoupling

The Action from the State. Cleaner Portlet code can be obtained by using the state pattern as proposed in [1]. A Portlet is modeled as a state machine where a state is defined for each Portlet fragment, and arrows are labelled with the actions that achieve the transition between fragments. A state has an associated view which in turn, embeds the potential set of actions. Unfortunately, the control-flow is hard-coded in both actions and fragments. This pattern facilitates the introduction of states but offers no help to weave the new states into the flow of the existing states.

The Action from the View. A step forward is the use of the Model-View-Controller paradigm, an evolution of the former where the control logic is decoupled in a new element, the *controller*. No information about the Views is introduced in the action but

[1] WSRP standardizes the interface between the Portlet consumer and the producer.

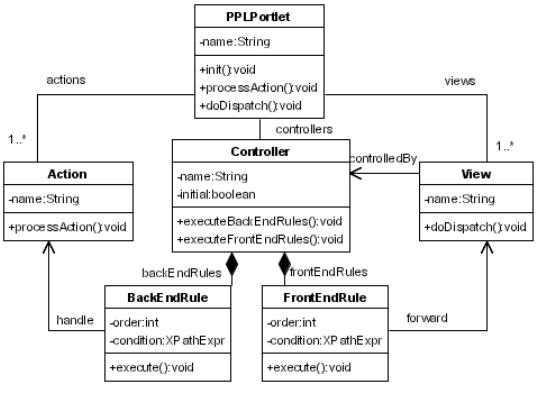
[2] JSR168 standardizes the interface between the Portlet container and the Portlet itself in Java.

[3] This work was partially supported by the Spanish Science and Technology Ministry (MCYT) under contract TIC2002-01442. Salvador Trujillo enjoys a doctoral grant for the MCYT.

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 587–588, 2004.

rather on the controller definition. Moreover, the flow is described as a set of *Front-end* rules: the antecedent expresses a predicate over the result of an action (i.e. *success* or *error*) whereas the consequent states the next fragment to be rendered (e.g. *flightSearch.jsp*). As an example, this approach is followed by the eXo Portlet Framework (http://www.exoplatform.org). However, the coupling between the view and the action still exists. The view contains the distinct actions the End-user can execute through it, but still a direct mapping from the views to the actions exists. Indeed, the action-view dependence is only moved to the controller, and still exists. No real independence between the Actions and the Views exists.

The View from the Action. For Portlets to become real components, Portlet implementation should be engineered to cope with the changes and variations the Portlet needs to cope with during its life time. That is, the Portlet should be built in such a way that the impact and cost of performing changes will be minimal. To attain this goal, we propose to decouple also the view form the action. In addition to *Front-end* rules, we introduce *Back-end* rules that dictate the actions to be executed based on the interaction achieved through the previous view. The controller comprises both *Front-end* and *Back-end* rules, which describe the whole flow, and neither action nor view contain a reference to each other, clarifying the development and reducing the impact of a change.



In brief, as with any other component, Portlets need to be engineered for variability. This work describes distinct approaches that gradually achieve higher levels of decoupling, and hence, enhance Portlet variability. This effort is part of a wider endeavour in attempting to apply a product-line approach to Portlet implementation.

Reference

1. T. Hanis, S. Thomas, and C. Gerken. Applying the State Pattern to Websphere Portal Portlets, 2002.
http://www-106.ibm.com/developerworks/websphere/library/techarticles/0212_hanis/hanis1.html.