Chapter 4
Using WSRP 2.0 with JSR 168 and 286 Portlets

Jana Polgar
Next Digital, Australia

ABSTRACT

WSRP—Web Services for Remote Portlets—specification builds on current standard technologies, such as WSDL (Web Services Definition Language), UDDI (Universal Description, Discovery and Integration), and SOAP (Simple Object Access Protocol). It aims to solve the problem of traditional data oriented web services which required the applications to be aggregated prior to any specific presentation logic could be applied for presenting the content. Portlet standard (Java Community Process, 2005) complements WSRP mechanism by defining a common platform and APIs for developing UI in the form of portlets. WSRP enables reuse of an entire user interface. One of the advantages is that only one generic proxy is required to establish the connection.At present, portlets based on JSR 168 (Java Community Process, 2005) as well as JSR 286 (Java Community Process, 2008) specification are often used in portal applications. This paper examines the relationship of WSRP specification with the portlet specification JSR 168 and evaluate some and shortcomings of WSRP specification 1.0 (OASIS, 2003). We discuss the impact of WSRP 2.0 (OASIS, 2009) and portlet specification JSR 286 (Java Community Process, 2008) on "on glass" integration paradigm.

INTRODUCING WSRP SPECIFICATIONS

The WSRP specification (*WSRP specification version 1* introduced two complimentary the concepts of *Producer* and *Consumer*. The WSRP 1.0 (*WSRP specification version 1* specification requires that every *producer* implements two required interfaces, and allows optional implementation of two others:

1. **Service Description Interface (required):** This interface allows a WSRP *producer* to advertise services and its capabilities to consumers. A WSRP *consumer* can use this

DOI: 10.4018/978-1-4666-0336-3.ch004

interface to query a *producer* to discover what user-facing services the *producer* offers. Furthermore, the description also contains additional metadata and technical capabilities of the producer. The producer's metadata might include information about whether the p*roducer* requires registration or cookie initialization before a *consumer* can interact with any of the remote portlets. For the *consumer*, this interface can be used as a discovery means to determine and localize the set of offered remote portlets.

2. **Markup Interface (required):** This interface allows a *consumer* to interact with a remotely running portlet supplied by the *producer*. For example, a *consumer* would use this interface to perform some interaction when an end-user submits a form from the portal page. Since this interface supports the notion of the state, the portal might obtain the latest markup based on the current state of the portlet (for example when the user clicks *refresh* button or interaction with another portlet on the same page takes place).
3. **Registration Interface (optional):** This interface serves as a mechanism for opening a dialogue between the *producer* and *consumer* so that they can exchange information about each others' technical capabilities. The registration interface allows a *producer* to ask *consumers* to provide additional information before they start interaction with the service through the service description interface and markup interfaces. This mechanism enables a producer to customize its interaction with a specific type of *consumer*. For example, a *producer* may use a filter and reduce the number of offered portlets for a particular *consumer*.
4. **Portlet Management Interface (optional):** This interface gives the *consumer* control over the life cycle methods of the remote portlet. A *consumer* acquires the ability to customize a portlet's behaviour, or destroy an instance of a remote portlet using this interface.

Processing user interaction When the user clicks on a link or submits form data, the *consumer* application controls the processing and invokes the performInteraction() method (). When the *producer* receives this call, it processes the action and returns the updated state. To redraw the complete page, the *consumer* then invokes the getMarkup() call to receive the latest markup fragment. Because the state of the *produce*r has changed since the previous getMarkup() call, the markup fragment returned is typically different from the one previously returned. The end user can then perform another action, which starts a new interaction cycle. (See Figure 1)

Handling customization and initialization: In a typical interaction a single centrally hosted services are used by multiple consumer applications and/or multiple individual users. The WSRP protocol supports multiple configurations of a single service. Good example is a lookup in a remote list of the course offerings and subjects offered within a particular course to international students. The list can be configured to display different offerings per semester, different currencies for subject fees or both depending on the *consumer* country and language prerequisites.

The WSRP protocol provides a set of function calls which allow *producers* to expose multiple versions of the same service each with different preconfigured interface. Furthermore, *consumers* can create and manage additional configurations of the same service, and end users can customize their configurations. However, such configurations are static (predefined) only in the current version of WSRP 1.0 (*WSRP specification version 1*.

SERVICE DESCRIPTION INTERFACE

Service Description interface enables the *consumer* to determine what services are available