WSRP Specification and Alignment

Jana Polgar
Monash University, Australia

Tony Polgar
Sensis Pty. Ltd, Australia

INTRODUCTION: WSRP SPECIFICATION OVERVIEW

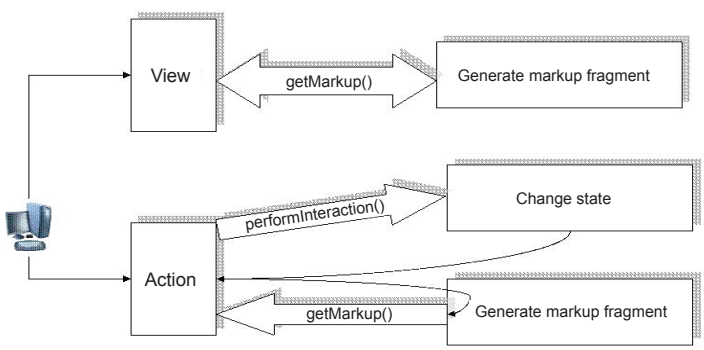
The WSRP specification (WSRP specification version 1, 2003) requires that every *producer* implement two required interfaces, and allows optional implementation of two others:

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 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 behavior, 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 (Figure 1). When the *producer* receives this call, it processes the action and returns the updated state. To redraw the complete page, the *consumer* then

Figure 1. Remote portlet interaction in View and Action modes



invokes the getMarkup() call to receive the latest markup fragment. Because the state of the *producer* 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.

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. A 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 that 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. dynamic configuration is planned for the future versions of WSRP, starting with the WSRP 2.0.

A CLOSER LOOK ON SERVICE DESCRIPTION INTERFACE

Service description interface enables the *consumer* to determine what services are available, and also provides information about the service capabilities. The services can be discovered through UDDI or other public registry. The access to the *producer* metadata is provided through getServiceDescription()method. All *producers* must provide the service description. This is important because it affects the decision whether the portal can display the markup, cookies handling, registration requirements, and so forth. In addition, the service description also includes the access to the information about portlet capabilities: supported portlet modes, window states, and list of locales the portlet supports. The service description structure supports also an extension field. This is an Array of objects that allow both client and server to support custom features. The ServiceDescription object contains useful information for the *consumer,* such as whether the registration is required, list of offered portlets, need to initialize cookies, and list of resources. The list of allowed types for ServiceDescription structure is in Figure 2.

Figure 2. ServiceDescription structure

```
ServiceDescription type (structure) details:
   boolean requiresRegistration
   PortletDescription offeredPortlets[]
   ItemDescription userCategoryDescriptions[]
   ItemDescription customUserProfileItemDescriptions[]
   ItemDescription customWindowState Descriptions[] 20
   ItemDescription customModeDescriptions[]
   CookieProtocol requiresInitCookie
   ModelDescription registrationPropertyDescription
   String locales[]
   ResourceList resourceList
   Extension extensions[]
```

ItemDescription is a set of arrays used to describe custom items the *consumer* is allowed to use in interaction with the portlets at the *producer* location. Each of these arrays provides the description of different types of extended data (e.g., custom modes). For those areas where this information is provided, portlets are not allowed to use extended values the producer has not described. This restriction allows the administrator of the consumer to determine a mapping of these values to those supported by the *consumer* implementation.

The information about portlets that the *producer* hosts is available as an array of PortletDescription(s), each of which describes single offered portlet. The description of each portlet listed in offeredPortlets[] array can be obtained by invoking getPortletDescription() method. This interface allows the *consumer* access to the following information:

- The consumer references this portlet using portletHandle;
- Markup types this portlet can generate in the array markupTypes. For each markup type, the supported modes, window states, and locales are specified;
- The portlet functionality is stored in the description field;
- Title describing this portlet is stored in the shortTitle type;
- Possible (x)html forms generation availability; and
- Information about the usage of URL templates.

The *producer* must expose one or more logically distinct ways of generating markup and handling interaction with this markup (Figure 3).

The boolean field usesMethodGet was added to this metadata due to the difficulties introduced by means in which browsers handle query string in GET request method. It suggests that the portlet will or will not generate any (x)html forms using GET methods to submit the input data. The query