WSRP Architecture

The OASIS WSRP standard defines pluggable, user-facing, interactive web services with a common, well-defined interface and protocol for processing user interactions and providing presentation fragments suitably for aggregation by portals. WSRP standardizes web services at the presentation layer on top of the existing web services stack, builds on the existing web services standards and will leverage additional web services standards efforts, such as security efforts now underway, as they become available. The WSRP interfaces are defined in the Web Services Description Language (WSDL). In addition, WSRP defines metadata for self-description for publishing and finding WSRP services in registries. All WSRP services are required to implement a SOAP binding and optionally may support additional bindings.

In the definition of the WSRP standard and the JSR 168, the OASIS Technical Committee and the JSR 168 Expert group have closely collaborated to make sure that that both fit together well in portal architectures. JSR 168 compliant portlets can be exposed as WSRP compliant web services and conversely, WSRP services can be integrated through generic portlet proxies written to the Portlet API (see Figure below).

Portal Overview

The WSRP4J project provides the WSRP4J Producer, which allows implementing such WSRP compliant services based on a free, open source software stack consisting of Tomcat, Axis and WSRP4J which in turn includes Pluto, the JSR 168 reference implementation. In addition, the WSRP4J project provides a generic proxy portlet written to the Portlet API, the WSRP4J Consumer (see Figure below).

WSRP4J Components

WSRP Consumer Architecture

The Producer and Consumer are provided with a very modular architecture enabling an easy exchange of module implementations. All modules excel by interfaces based on the WSRP object model hiding the portal's object model and thus gaining independence of changes in the portal implementation or design.

Consumer Architecture

1. Protocol Handler

The Protocol Handler is a standalone Swing based application that implements the consuming portal and the browser functionality. It aggregates the integrated WSRP portlets and forwards all invocations together with relevant context and request information to the remote WSRP service. The Swing consumer thereby uses the ConsumerEnvironment to collect all data required for a WSRP call.

2. WSRP Object Model

Most of the WSRP object model is being generated from the WSRP specification's WSDL types.

3. PortletDriver

The PortletDriver is the task oriented abstraction of the generated WSRP stubs for markup and action invocation.

The PortletDriver is the task oriented abstraction of the generated WSRP stubs for markup and action invocation.

4. WSRP service (stubs) ()

The stubs generated by the axis SOAP implementation. The stubs are also implementing the cookie handling - for more details see SessionHandler below.

5. ProducerRegistry

The ProducerRegistry stores and manages details about producer portals from which portlets were integrated or shall be integrated.

6. SessionHandler

The WSRP consumer implementation session handling is based on the SOAP stack's session handling. The generated SOAP stubs are doing the cookie handling and are caching the cookies in instance variables. This means that one stub object instance is equivalent to one HTTP session which we map to one WSRP session (initCookie wise). When a WSRP session did time out an InvalidCookie exception is being thrown and a new session must be established.

7. URLHandler

The URLHandler is responsible for the URL rewriting of the WSRP URLs embedded in the markup received from the remote portlet.

8. PortletRegistry

The PortletRegistry stores and provides access to WSRP specific data and descriptions of a remote Portlet.

9. UserRegistry

The UserRegistry is in the Swing consumer case very simplistic. As the browser is integrated there is always only one "dummy" user being managed.

WSRP Producer Architecture

Producer and consumer are provided with a very modular architecture enabling an easy exchange of module implementations. All modules excel by interfaces based on the WSRP object model hiding the runtime environment's (portal's) object model and thus gaining independence of changes in the environment's implementation or design.

WSRP Producer Architecture

1. WSRPEngine

The WSRPEngine is the WSRP implementation endpoint. This class must be deployed in the app server (Tomcat) as a web service. The WSRPEngine implements the WSRP protocol specific ports (=interfaces) and does the corresponding protocol handling. There are four WSRP ports:

Markup
deals with Portlet invocation and Session handling
PortletManagement
covers lifecycle and properties of portlets
Registration
enables a consumer to register at the producer
ServiceDescription
enables a consumer to discover the services that a producer provides

Note:
To be able to reuse Portal functionality regarding session and request handling when invoking a portlet the WSRPEngine must be able to access the HttpServletRequest.

2. ConsumerRegistry

This component manages and provides access to the registered Consumers.

3. HandleGenerator

This component is responsible for generating IDs / handles required for the WSRP protocol handling.

4. WSRP Object Model

The WSRP object model is being generated from the WSRP specification's WSDL types.

5. Provider

The Provider is the access point for the WSRPEngine to the Provider components which hide the provider (portal) implementation's components required to handle and invoke portlets. All Subcomponents wrap corresponding provider components and map the WSRP object model to the provider object model.

6. Portlet Invoker

The PortletInvoker wraps the Provider's invocation mechanisms and provides the Provider with the required environment.

7. Description Handler

The DescriptionHandler manages and provides the description of the provider regarding configuration properties like registration or session handling policy, etc. It moreover provides the descriptions of the provided portlets depending on the registration.

8. PortletPool

The PortletPool manages the portlet instances and is responsible for the portlets' lifecycle management (clone, destroy).

9. Portlet State Manager

The PortletStateMananger enables a Producer to access a portlet's state as a blob that than can be delegate to the consumer to be stored on consumer side.

10. Session Handler

No additional session handling implementation for the Pluto provider is required as the session handling concept is completely HTTP (cookie) based and relies on consumer' cookie handling.

11. URL Composer

The URLComposer must be used for WSRP triggered portlet invocation to create WSRP

URLs instead of the portal's URL handling implementation. Therefore the URLComposer is being used by WSRP's version of the DynamicInformationProvider which is being used by the Portlet API implementation to generate portlet URLs. There are two ways how URLs can be composed in a WSRP environment:

1. Via templates that represent URLs that are valid on Consumer side and contain placeholders for all portlet specific URL components.
2. Via URL rewriting. In that case WSRP specific URLs are being composed that will have to be rewritten by the Consumer.