A Deep Look at Web Services for Remote Portlets (WSRP) and WSRP4J

Xiaobo Yang, Rob Allan
CCLRC e-Science Centre, Daresbury Laboratory, Warrington WA4 4AD, UK
{x.yang, r.j.allan}@dl.ac.uk

Abstract

Web Services for Remote Portlets (WSRP) extends today's data-centric web service paradigm by providing markup fragments that can be used directly for rendering. WSRP alleviates some complex tasks of building web portals by enabling remotely-hosted portlets to be accessed directly. This illustrates a promising approach for portal federation. However, according to our early investigations on selected open-source portal frameworks, WSRP implementations are still immature. In this paper, a deep look at WSRP, in particular, WSRP4J the Apache implementation of the OASIS WSRP 1.0 specification will be reported with further discussions aiming at providing a full solution for WSRP.

1. Introduction

Web portals are gateways to many kinds of information systems including e-Learning, e-Research and e-Business. Today the second-generation web portals based on portlet technique are widely adopted and deployed. For example, in the grid community many grid portals are now portlet based such as the OGCE Portal Toolkit [2] and the UK National Grid Service (NGS) Portal [17]. Acting as web components, portlets encapsulate relatively independent functionalities with markup fragments provided to portals for composition of integrated web pages. While the Java Portlet Specification 1.0, JSR 168 [1], standardises portlet development using Java, there is still need to re-deploy the same portlet in different portal frameworks so that it can be re-used. The Web Services for Remote Portlets (WSRP) specification 1.0 [8] proposed by OASIS defines presentation-oriented web services aiming to remove the re-deployment of portlets by solving the interoperability issues among different portlet containers. Now portlet providers can develop and maintain their own portlets individually without requiring upgrades on their consumer side (normally web portals).

The difference between traditional data-oriented and WSRP services is that the latter provide markup fragments generated by remote portlets that can be used directly by their clients for rendering, while data-oriented services provide only raw data. These data normally require post-processing, e.g. using XSLT [11] transformation before being presented to end users. WSRP makes it possible for service providers to develop and maintain independent services which provide all the rendering information needed by the client side UI. This greatly alleviates the hard work needed for constructing end user focused applications like web portals. Now web portals, including grid portals, can be federated as required. As shown in Fig. 1, Portal A has two local portlets and two remote portlets, one from Portal B, the other one from Portal C. Altogether, four portlets are combined together and rendered on one portal page.

Although WSRP depicts a beautiful blueprint for future portal development, according to our investigation on open-source portal frameworks there are many interoperability issues [18]. We looked at eXo platform, Liferay, StringBeans and uPortal plus WSRP4J. Even with producer and consumer from the same vendor almost none of

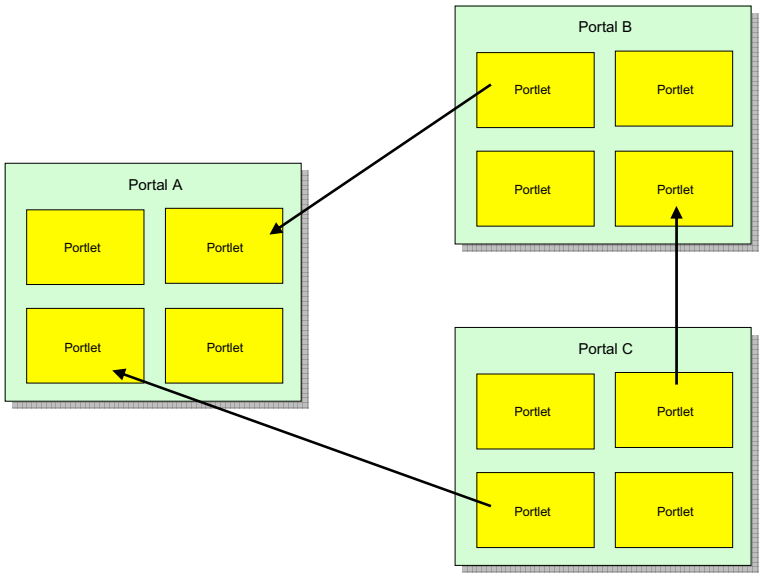

Figure 1. Portal federation: building up portals by plugging in local and remote portlets.

the WSRP implementations are fully functional. Therefore in this paper, a deep look at WSRP4J, the Apache WSRP 1.0 implementation will be reported. This will form the basis of our WSRP consumer developed for the UK JISC funded Sakai VRE Portal Demonstration project [12].

Initiated by IBM, WSRP4J from Apache is an open-source project which is now still in incubation stage. Currently WSRP4J is functional although some WSRP features are missing [10]. However, during our WSRP support test [18], the WSRP4J producer gave the best performance, especially for handling interactions between end-user and remote portlet. WSRP4J makes use of Pluto [3], a JSR 168 reference implementation by Apache, as its portlet container. Producer within WSRP4J has been developed based on a very modular architecture. In theory, Pluto should be able to be replaced by other portlet containers, although in practice, this will involve a considerable amount of work.

In this paper, we will first give a brief introduction to the mechanism of WSRP. Different aspects of several missing bits in WSRP4J will be discussed followed by further discussions on providing a fully functional WSRP solution to portal developers. Finally we provide a summary and concluding remarks.

2. Web Services for Remote Portlets (WSRP)

Besides *End-user*, the WSRP 1.0 specification defines *Producer* and *Consumer*. A description of the sequence flow is given below for better understanding on how producer and consumer work together to complete portlet's publication and consumption.

Fig. 2 illustrates a simplified sequence flow of communications in WSRP. In this figure, communications such as *clonePortlet* and *deregister* are omitted to make it more readable. A detailed description of WSRP communications between consumer and producer can be found in the WSRP 1.0 Primer [9] and other introductory materials [13]. As depicted in Fig. 2, a consumer first talks to a producer's description interface to retrieve metadata. This includes a description of the producer itself and the portlets it holds. During this stage, the consumer may be asked to register itself with the producer. According to the consumer's capabilities, the producer may then provide markup fragments in different formats. The consumer can now select a portlet it wants to access. A unique portlet handle is used for communications between producer and consumer so that the portlet can be uniquely identified. The

producer will first provide the default view of the portlet. Now the consumer is able to make use of the markup fragments provided by the producer so that a full (HTML) web page can be constructed and rendered to end-users. The consumer also needs to collect requests, such as form input, and re-direct them to the producer. This is the most important task of the consumer. Only when a user request is transferred correctly, can a correct response be created. As reported in [18], many WSRP consumers could not handle such a re-direction correctly which makes them currently unusable.

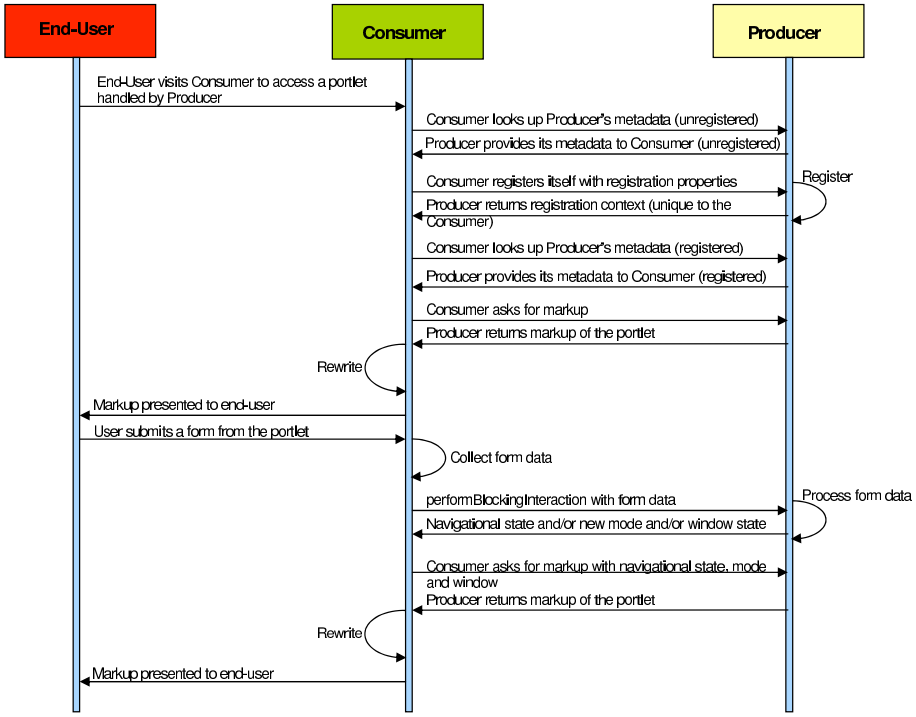

Figure 2. Simplified WSRP sequence flow.

As mentioned above, WSRP4J is still under development. The TODO list [10] highlights a good starting point for investigation of WSRP4J. In this paper, we will not touch on all of the items in the TODO list but with several points we noticed during our investigation. These include templates, mode/ windowState change, and file upload support in WSRP4J.

3. Experiences on WSRP4J

3.1. Templates support plus templatesStoredInSession

In WSRP the consumer acts as the intermediary between the producer and the end-user. Among the communications involved, one key task the consumer has to handle is to generate correct URLs as many of them are relative URLs on the producer side. Furthermore, the consumer may want to add its own fragments to those generated by the producer. There are basically two approaches for doing this: a) consumer rewrites URLs in markup fragments generated by the producer; and b) consumer provides templates to the producer so that URLs are generated according to the consumer requirements. The second one is a more flexible approach for consumers to communicate with producers. This is because if a consumer provides templates to a producer, the latter will

be able to process markup fragments using these templates. Hence such a producer will not ask the consumer to re-write URLs within the markup fragments. The consumer can now handle everything such as links and form actions according to requirements on its own side. If *templatesStoredInSession* support is used, the consumer will only need to provide templates once, which reduces network traffic. For example, the BEA WSRP producer (*http://wsrp.bea.com/portal/producer?wsdl*) requires its consumers to support both templates and *templatesStoredInSession* while the one from NetUnity Software (*http://wsrp.netunitysoftware.com/WSRPTestService/WSRPTestService.asmx?Operation=WSDL*) only requires templates support.

The only missing bit of templates support in WSRP4J producer is *resourceTemplate*. We have implemented this together with *templatesStroedInSession* support in WSRP4J producer and our ProxyPortlet based WSRP consumer developed for Sakai, an open-source collaboration and learning environment for education [4]. This consumer has been observed to successfully consume portlets held by our modified WSRP4J producer with *templates* and *templatesStoredInSession* support, the aforementioned BEA and NetUnity WSRP producers. Fig. 3 gives a screenshot of a remote portlet from BEA running inside Sakai through WSRP. In [19], this WSRP consumer for Sakai has been described with examples given for consuming grid portlets developed for the UK NGS Portal [17].

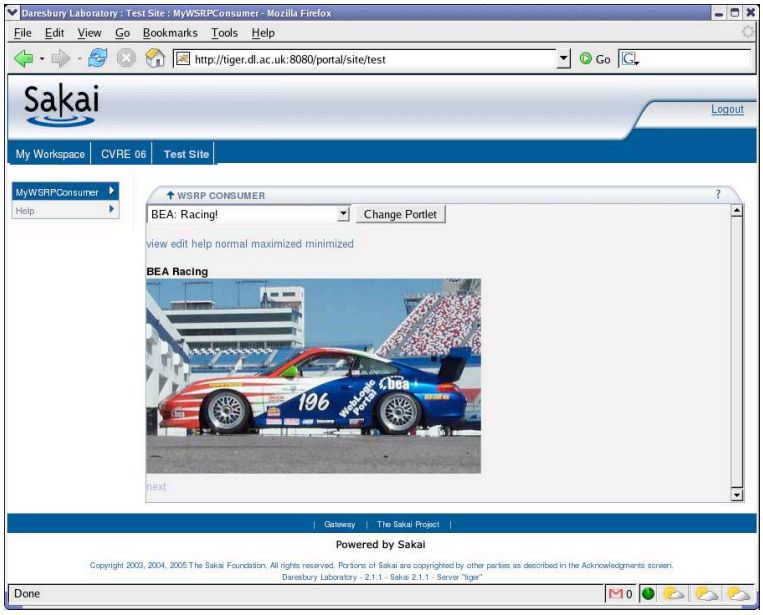

Figure 3. Consuming BEA remote portlet in Sakai through WSRP.

Namespace support is also currently not available in WSRP4J but can be realised in a similar approach as templates support.

3.2. Mode/ WindowState change support

It was observed that WSRP4J does not support *Mode/ WindowState* change in portlets. For example, a portlet may have a button in its *view* page to go to *help* page and vice verse. This involves a *mode* change. There was no response observed during our WSRP support test of mode change requests. The same thing happened to *WindowState* which implies WSRP4J does not support window state change, for example from *normal* to *maximized*. As one of the key features portlets provide, *Mode/ WindowState* change support has been added to WSRP4J producer during our investigation. In Fig. 3, a line of URL links between the ”Change Portlet” button and the ”BEA Racing” title includes different modes and window states we support.

3.3. File upload support

To support file upload, it is practical for consumers to extract data first then set up correct WSRP request parameters to transfer the data to producers. A producer can then retrieve data transferred and set particular HTTP request parameters that the portlet is aware of, so that the file upload can be finished. It was found not practical for the producer to construct a HTTP request object with ”enctype” attribute set as ”multipart/form-data”. Therefore file the upload portlet had to be modified to be aware of some particular parameters the producer sets as mentioned above.

If uploaded data is encoded then included in the SOAP message from consumer to producer, we observe severe performance problems when transferring large amounts of data. This issue is not mentioned in WSRP 1.0 specification. Therefore extensions are needed to increase the performance of transferring large datasets between consumer and producer. Howes [14] reported that BEA has extended the WSRP 1.0 specification (Custom Data Transfer Extension) to allow *”WSRP consumers and producers to create, view, modify, and control concurrent access to shared, scoped data in a scalable, reliable, and highly performant manner”* with the help of Tangosol Coherence.

On the other hand, file download support is also missing in WSRP4J. Normally file download is redirected to a servlet which does the real file download task. This is currently not available in WSRP4J as redirect is not supported.

4. Further discussions

4.1. Redirect and file download

Support of redirect is missing in WSRP4J. As mentioned above, file download could be realised by redirecting to a servlet. Current version of WSRP4J producer simply sets *redirectURL* to *null*.

4.2. Customisation

One of the key benefits portlets bring is the ability to set preferences for each user, however in a remote portlet environment, this becomes an issue. Because producer-offered portlets are accessible to all consumers, they are not allowed to be customised. WSRP uses the concept of ”cloned portlet” to let consumers customise portlets for end-users. Two approaches are listed in the WSRP 1.0 specification: 1) using *edit* mode of a portlet; and 2) using property definitions within the portlet’s metadata. In WSRP4J, all methods related to properties are empty thus it is not implemented.

According to our test with WSRP4J, once a portlet handle is changed (e.g., edit a cloned version of portlet A, then switch to portlet B, come back to portlet A) or connection to a WSRP4J producer is closed, the preference settings are lost. This is because such a portlet will be cloned during those actions. Preferences of a standard JSR 168 portlet are managed by the portlet container. Inside a WSRP4J producer, Pluto uses an XML file (portletentityregistry.xml) to store portlet preferences. These preferences are stored with a link to the portlet’s ID which is in fact the portlet handle. Since such a handle for cloned portlets is always changing and the WSRP4J producer does not keep it, portlet preference support is not persistent.

To solve the issue, a WSRP4J producer may record additional information such as consumer’s name and user’s name together with the portlet handle when a user edits preferences of a cloned portlet. Then next time the same user from the same consumer accesses the same portlet and tries to edit preferences, these values can be retrieved and set by the producer.

4.3. Rich user interface

The tide now in web development is to develop Rich Internet Applications (RIA) that have features and functionalities similar to desktop applications. AJAX (Asynchronous JavaScript And XML) is today's hottest topic in web development. Although AJAX can be applied to standard JSR 168 portlets, there will be problems to publish these as remote portlets without modification. Early investigation shows that applying AJAX in a remote portlet is not straightforward. AJAX makes heavy use of JavaScript especially *XMLHttpRequest* to contact a remote server while in a remote portlet scenario there exists a consumer acting as a broker. Normally all requests from end-users should be sent to this consumer first and then redirected to the producer by the consumer. After that, the producer handles requests from the consumer and passes them to the portlet container. It is the portlet container which does the real job of talking to a particular portlet. Responses from portlets are collected by the container, and then passed to the producer and the consumer accordingly. AJAX breaks this design pattern by contacting remote services directly behind the WSRP consumer portal server. For example, an HTTP request may be sent from the user agent (browser) to services deployed on a WSRP producer server.

Due to security concerns, requests sent by AJAX are limited to the server where the web application resides. Internet Explorer will pop up a warning message once such a call is made while Firefox simply blocks it without warning. A proxy servlet can be utilised to re-direct HTTP requests to remote services [15] so that this issue can be worked around. Fig. 4 gives a screenshot of such a portlet, an AJAX Invoice Viewer portlet [20] running under our servlet-based WSRP producer. When a new invoice number is selected (in Fig. 4 the invoice number is "439091"), AJAX will contact a proxy servlet sitting on the consumer web server. The proxy servlet will then re-direct the request to a WSRP producer (WSRP4J) server where the target resides. The response will then be returned from the remote service. All this is done behind the web browser so that performance is greatly improved.

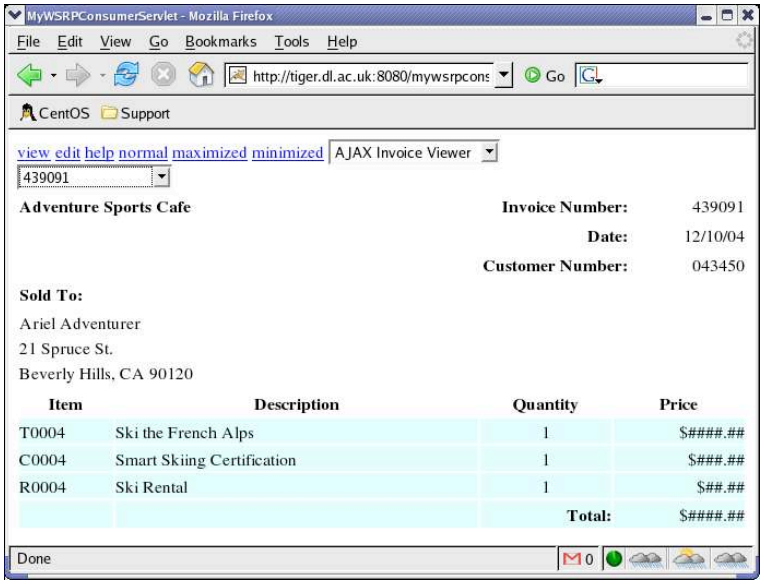

Figure 4. AJAX-enabled remote portlet.

4.4. Producer/ portlet publish and discovery

End users today have to remember too many URLs, a central registry for web services can alleviate the pain. UDDI [6] which is widely adopted for web services registry, discovery and integration can be adopted since WSRP

producers are web services. Although a consumer can then obtain a producers' interface URLs and get a list of their portlets, end users are only interested in searching for particular portlets rather than producers. Portlets should therefore also be registered, for instance, in a UDDI registry. We have set up a test registry for this purpose [16] which makes it possible for end users to discover portlets and then transparently access them (a user simply selects th required portlet; the consumer will then connect to the producer to access it).

4.5. Security

Another area which is not detailed in the WSRP 1.0 specification is security. Because WSRP is based on web services, the specification does not address the security issue in detail but simply guides developers to follow existing security mechanisms, such as WS-Security [7] and SAML [5]. WSRP 1.0 emphasises the usage of transport-level security standards, e.g. SSL/ TLS between consumer and producer, to achieve message integrity and confidentiality.

5. Conclusions

WSRP is a promising specification for building up presentation-centric web services that can be plugged into web portals without requiring portlets to be deployed locally. Unfortunately as revealed by our investigations of WSRP support in several open-source portals frameworks, WSRP is still not fully functional. Therefore in this paper, WSRP4J, the best performer in our tests, has been selected to dig out the missing bits including templates, mode/ windowState plus file upload support. Further discussions include redirect support, customisation, AJAX which is now prevailing for building up Rich Internet Applications, producer/ portlet publication and discovery and security issues. With these issues tackled in the future, WSRP could play a key role in portal federation and shared portal services.

Acknowledgements

The authors would like to thank the anonymous reviewers for their insightful comments helped to improve this paper. This work was undertaken at the CCLRC e-Science Centre, Daresbury Laboratory supported by UK JISC (The Joint Information System Committee) as part of the Sakai VRE Demonstrator project.

References

[1] JSR 168. *http://www.jcp.org/aboutJava/communityprocess/ final/jsr168/*.
[2] OGCE. *http://www.collab-ogce.org/ogce2/*.
[3] Pluto. *http://portals.apache.org/pluto/*.
[4] Sakai: Collaboration and learning environment for education. *http://sakaiproject.org/*.
[5] SAML. *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security*.
[6] UDDI. *http://www.uddi.org/*.
[7] Web Services Security. *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss*.
[8] WSRP 1.0. *http://www.oasis-open.org/committees/ download.php/3343/oasis-200304-wsrp-specification-1.0.pdf*.
[9] WSRP 1.0 Primer. *http://www.oasis-open.org/committees/download.php/10539/wsrp-primer-1.0.html*.
[10] WSRP4J TODO List. *http://wiki.apache.org/portals/WSRP4J/TodoList*.
[11] XSLT. *http://www.w3.org/TR/xslt*.
[12] R. Allan, D. Chohan, X. Wang, X. Yang, R. Crouchley, A. Fish, M. Gonzalez, M. Baker, H. Ong, M. Dovey, and F. Pinto. Virtual research environments: Sakai demonstrator. In *Proc. UK e-Science All Hands Meeting 2005, available on CDROM*, pages 208–215, Nottingham, UK, September 2005.
[13] R. Gupta. WSRP: Dynamic and real-time integration - an introduction to wsrp, its usage, and implementation. *Web-Services Journal*, 5(8):10–19, 2005.

[14] J. Howes. Sharing data among federated portals: Using WebLogic Portal, Tangosol Coherence and WSRP. *http://dev2dev.bea.com/pub/a/2005/11/federated-portal-cache.html*, November:2005.

[15] J. Margaglione. AJAX programming in BEA WebLogic Portal 8.1, part 2. *http://dev2dev.bea.com/pub/a/2006/03/ajax-portal-2.html*, April 2006.

[16] X. D. Wang, X. Yang, and R. Allan. Plug-and-play remote portlet publishing. In *GCE05: Portals Workshop*, Seattle, USA, November 2005.

[17] X. Yang, D. Chohan, X. D. Wang, and R. Allan. A web portal for the National Grid Service. In *Proc. UK e-Science All Hands Meeting 2005, available on CDROM*, pages 1156–1162, Nottingham, UK, September 2005.

[18] X. Yang, X. D. Wang, and R. Allan. Investigation of WSRP support in selected open-source portal frameworks. *Concurrency Computat.: Pract. Exper.*, 2006, in press.

[19] X. Yang, X. D. Wang, R. Allan, M. Dovey, M. Baker, R. Crouchley, A. Fish, M. Gonzalez, and T. van Ark. Integration of existing grid tools in Sakai VRE. In *International Workshop on Collaborative Virtual Research Environments (CVRE06), to be held in conjunction with GCC 2006*, Changsha, China, October 2006; accepted.

[20] G. Ziebold and J. Suri. Asynchronous rendering of portlet content with AJAX technology. *http://developers.sun.com/prodtech/portalserver/reference/techart/asynch_rendering.html*, June 2005.