

Adapting existing search tools for use with the JSR 168 and WSRP portlet standards: the CREE Project

Jonathan Hunter, EDINA, University of Edinburgh, and Chris Awre, University of Hull

Introduction

The CREE (Contextual Resource Evaluation Environment) Project was funded by the Joint Information Systems Committee (JISC) in the UK to, in part, investigate the use of the recently released JSR 168 and WSRP (Web Services for Remote Portlets) portlet standards. Emerging institutional environments such as institutional portals offer the opportunity to present search tools locally to the end-user rather than through diverse websites. CREE sought to investigate the adaptation of a range of distinct search tools using the portlet standards.

Full details of the investigations undertaken can be found on the CREE website at <http://www.hull.ac.uk/esig/cree/>. This article focuses on the experiences of one partner, at the EDINA National Data Centre at the University of Edinburgh¹, and the adaptation of a bibliographic cross-search tool and related OpenURL resolver for use within a conformant portal framework.

Background

Viewed from the perspective of a portal host, the JSR 168 and WSRP standards have a similar goal: they aim to make the aggregation of content into portals a simple task. Assuming the portal is compliant with the standards, the portal host should be able to select any conformant portlet and configure their portal to deploy it. One of EDINA's main roles as a Data Service Provider is to provide online services to the UK academic community. For an organisation like EDINA the use of the portlet standards offers the possibility of a realistic means of embedding national services within institutional portal frameworks through the provision of portlets compliant with one or both of the standards.

For this case study, the two standards are first briefly described, followed by an overview of the services for which portlets were developed. The technicalities of portlet creation are described, and finally a summary and conclusions drawn from the experience of partaking in the project are provided.

JSR 168 and WSRP

The JSR 168 specification defines a set of APIs, or a contract, between the portal (more specifically the portlet container) and a Java portlet. A compliant JSR 168 portlet is a software module that should work in any portal framework that supports the specification. The APIs in the specification address the areas of personalisation, security and content aggregation. The portlet container provides the portlet with a runtime

environment in which the portlet can process requests and generate markup. The two most significant methods defined in the interfaces are:

1. `processAction` – the portlet container invokes this method every time a user performs an action in a portlet window.
2. `doView` – called every time the portal window is refreshed. Thus, every individual portlet currently in display has this method invoked by the portlet container.

The specification does not address which protocol a portlet should use to communicate with external services, or even whether it should have a reliance on an external service: these attributes are beyond the scope of the specification, which is focused solely on the contract between portal and portlet.

The WSRP standard specifies the interaction between an endpoint (the user's browser), a WSRP consumer (a compliant portal will provide this) and a WSRP producer. Communication between the consumer and producer is done using the SOAP protocol, thus allowing the developers to use their language of choice for their implementation. WSRP defines four interfaces:

1. Markup Interface – defines operations for obtaining the markup from a Portlet as well as processing user interactions with that markup.
2. Service Description Interface – defines an operation for acquiring the producer's metadata.
3. Registration Interface (optional) – defines operations for establishing, updating and destroying a registration.
4. Portlet Management Interface (optional) – defines operations for obtaining Portlet metadata and for cloning Portlets for further customization.

Only the first two of these interfaces are mandatory. The two operations in the Markup Interface `render` and `performNonBlockingInteraction` are analogous to the two method calls `doView` and `processAction` in JSR 168. This is significant; it allows a Java portlet to be wrapped with web service calls. The relevance of this with respect to the Apache WSRP4J project is discussed later in this article.

Those interested in a thorough understanding of the JSR 168 specification and WSRP v1.0 standards should read the relevant documentation from the Java Community Process², and the OASIS technical committees³.

The GetRef and GetCopy Services

The original project plan for EDINA's role in the CREE project was to create portlet versions of two EDINA services, namely `GetRef`⁴ and `GetCopy`⁵, which play a key role in document discovery and location within the JISC Information Environment. The potential of embedding these services in other contexts, particularly institutional portals, was the main reason for their selection.

GetRef is a discovery tool; it offers fully functional cross-searching across multiple remote Abstract and Indexing databases to which an institution subscribes. The user enters a single search and the GetRef search engine returns individual result sets with real time updates for each target. Although GetRef is frequently referred to as a portal, it is only a portal in the sense that it brings together multiple resources in the same subject area. It is a subject portal; it does not make use of portal framework software. Most users of GetRef are only aware of the web interface, but there is also a machine-to-machine (M2M) interface available, and it is this interface that the portlet uses to aggregate the searching of multiple targets.

GetCopy is an OpenURL resolver with a limited knowledgebase. It is intended for users whose institution does not have a dedicated OpenURL resolver, or for an individual who is not a member of an institution. It is typically invoked by bibliographic services which having assisted a user in discovering an article then wish to assist the user in obtaining a copy of the discovered article. The GetCopy service is usually seen by a user when clicking on an OpenURL link from a record in a bibliographic online service, such as BIOSIS, and displays as the opening of a new browser window showing possible locations of an article. It should be noted that the GetCopy service is not directly 'searched' by a user as such; it is invoked by an existing service. This has implications for attempts to portalise it.

Resources required for portlet development

Software

The software required for development of the GetCopy and GetRef portlets was:

- Java - It is implicit that creation of a portlet complying with the JSR 168 specification requires the use of Java as the implementation language.
- uPortal⁶ (v2.4.2) - A product of the Java Architectures Special Interest Group (JA-SIG⁷). It includes a JSR 168 compliant portlet container and a WSRP consumer. Any portal framework supporting the respective standards could have been chosen to host the portlet.
- Tomcat⁸ - A web application server supporting the Java servlet specification. Tomcat is a product of the Apache Jakarta project⁹ and is available under the Apache software license. The uPortal software (web application) was hosted on a Tomcat installation. Any servlet container compliant with the Java servlet specification could have been used, including JBoss, JRun, Oracle JServer, IBM WebSphere or BEA WebLogic.
- Ant¹⁰ - A tool used to simplify the maintenance, building and deployment of applications written in the Java language (although not in fact exclusive to the Java language).
- MySQL¹¹ - a relational database, required by the uPortal application. Used to host the profile and channel information. As with the servlet container, the underlying database can be abstracted from the portal and other similar relational databases used, e.g., Oracle or Ingres.

- JAFER¹² - Java Access For Electronic Resources. JAFER was developed at the University of Oxford, who are also participants in the CREE project. The package includes a Z39.50-client suitable for embedding into a portlet in order to query the M2M interface presented by the GetRef service.
- WSRP4J¹³ – Web Services for Remote Portlets for Java. This software was originally created by IBM and donated to the Apache foundation to try to increase uptake of WSRP. It acts as a WSRP interface to a Java portlet.

Hardware used in development

Most of the development was performed on a desktop machine. This was a machine with 512Mb memory and a Pentium IV processor, running a Linux operating system. The choice is arbitrary; the platform independence of the software required by CREE frees the engineer to opt for whichever development environment they wish.

The MySQL database used for storage of channel and profile information was resident on a Solaris platform. An instance could have been installed on the same desktop machine used for the portlet development but it seemed sensible to take advantage of existing resources in the EDINA workspace. The development portlet made use of the GetRef M2M interface, which also resides on the same Solaris machine as the MySQL database.

Technical development

At the outset of the project EDINA had to decide with which specification to begin implementation with. Discussions in the early project meetings and during an initial desk-based investigation had flagged that uPortal was a suitable tool for hosting a JSR 168 compliant portlet. At that time, the only open source software available for assisting with WSRP development was WSRP4J. The goal of this tool is to make a JSR 168 portlet available as a remote Web Service. It supplies the necessary SOAP wrappings, maintains state information, and maps the web service calls to the appropriate Java portlet API request. Successful use of the WSRP4J tool requires a working JSR 168 portlet, thus deciding what to develop first was simple. The GetRef portlet was created first. The GetRef service is substantially more complex than GetCopy and it was felt that a good knowledge of the standards would have to be acquired to portalise it, helping to build up experience and expose any shortcomings. The major aspects and issues associated with the development are described below.

Model-View-Controller design pattern

An initial portlet was written to prove the concept. It extended the `GenericPortlet` abstract class and made use of the JAFER Z-Client package to communicate with the Z39.50 M2M interface of the GetRef service. This was found to be adequate for demonstration purposes, but was difficult to develop due to a lack of a formal design. The portlet was subsequently re-engineered using a simple Model View Controller (MVC) design pattern¹⁴. The Controller was the main portlet class extending the `GenericPortlet` abstract class. The Model was essentially a wrapper for the JAFER

Z-Client, and the View consisted of an XSLT transformation class which made use of the Xalan transformation engine¹⁵, and several stylesheets to transform the XML returned by the GetRef interface to HTML for return to the user's browser. Use of the MVC design pattern greatly simplified the structure of the code and eased its maintenance.

GetCopy portlet

The initial project plan envisaged creating portlets for both the GetRef and the GetCopy services. However, whilst the GetCopy service is intended to locate an article, it does not require direct user input. The service is called directly by other services using an OpenURL encoded metadata string. Unfortunately, the JSR 168 specification does not allow for communication between services in this fashion (which would require portlets to communicate with each other – inter-portlet communication). More specifically, it is not possible to create a new portlet instance from an existing portlet instance. Provision of the functionality of the GetCopy service was thus moved into the GetRef portlet and presented as an additional available query (“Attempt to locate article”) of the GetCopy locate function (with an OpenURL string) and then transforming the returned XML with an additional stylesheet for presentation in the GetRef portlet window.

Viewed from the perspective of a software implementor a standalone GetCopy portlet would be the preferred means of providing the GetCopy service functionality. It would allow all necessary software to be isolated and would spare the engineer from having to do his or her own implementation. It should be noted that the release of the WSRP 2.0 specification does allow for inter-portlet communication, thus allowing for the future possibility of a standalone GetCopy portlet.

Portlet URLs

In the early stages of development the portlet passed in a single portlet URL (created by the portlet container in response to an API call from the portlet code) to a stylesheet. The stylesheet appended the necessary parameters required by the consuming portal to create the various HTTP links and form inputs seen by the end point (the consumer's browser). This was found to be problematic. The URLs returned by the portlet container included a fragment descriptor; a fragment descriptor acts as a navigation element within an HTML page allowing a browser to display, or scroll to, the particular part of the page that is referenced by the descriptor. Appending parameters to a URL that already contains a fragment descriptor creates an invalid URL and results in the appended parameters being ignored by the consuming portal.

The correct way to generate portlet URLs is to use the portlet API to add parameters to the URL and then request the complete URL in a string format for passing as a parameter to the stylesheet. To keep the code simple, the initial approach was to pass in only one portlet URL to the stylesheet (having removed the fragment descriptor – this is desirable rather than necessary). This worked as intended for a JSR 168 portlet hosted locally to the portal.

On trying to deliver the portlet as a web service using WSRP4J this was again found to be problematic - all parameters appended to URLs in the stylesheet were being ignored. The JSR 168 specification does not describe the syntax and semantics of the URLs created by the portlet container in response to calls on the API (which the container is obliged to provide). However, the distributed nature of WSRP means that it is necessary to specify how the portlet URLs should be constructed for passing between the consumer (the portal framework) and the producer. Two means of URL construction are specified in section 10.2 of the WSRP v1.0 specification. Details of these are beyond the scope of this article but they do reinforce the need to use the provided portlet API to generate the URLs required by the specification.

The problems described above may seem relatively trivial but there is a significant issue here for the portlet developer. EDINA had recommended in its early project reports that a software engineer may wish to design a service such that any render request would initially generate XML. The XML can then be transformed using XSLT to the desired output format. However, if there are values in the XML document which must be included in dynamic links to be presented at the end point (the user's browser, in this context) then there is a problem. This is best explained by example. The GetRef portlet requests a record of a document matching the user's search criteria from the M2M interface of the GetRef service. The XML record returned includes an OpenURL query string as an attribute. It is necessary to submit this parameter to the portlet if the user wishes to locate an article. This is most easily achieved by its inclusion as a value in a HTTP link seen by the user in the portal window. Unfortunately, this is not straightforward. Three means of doing this are listed below:

- Use XSLT extensions such as those made available by the Xalan parser to allow Java processing within the stylesheet. This was untried but should allow the portlet URLs to be constructed in the correct manner from within the stylesheet code.
- Preprocess the XML, to extract the required parameters which can then be added to the URLs (using the portlet API) to be passed to the stylesheet - this seems to defeat the purpose of using XSLT in the first place.
- Include the parameter (in this example, the OpenURL query string) as a HTML form input in a POST request. It should be noted that this option is not always convenient since it requires a button (for submission of the form) on the user interface.

The third of these options was chosen for the GetRef portlet. It required a minor change in the interface. The "Attempt to locate article" query link was replaced by a HTML button to submit the form and the all important OpenURL query string parameter. Aesthetic concerns may dictate that a HTTP link is preferable to a button, but this was not the case with the GetRef portlet, so the easiest implementation solution was chosen. Should the appearance be considered to be of greater importance then the first option (Xalan extensions) would be the recommended means of including XML attributes in the HTTP links seen in the portlet window.

WSRP4J

Delivery of the GetRef portlet through a JSR 168 compliant portal framework was a necessary first step towards using WSRP4J to deliver a remote portlet via Web Services. Use of WSRP4J reduces the implementation effort to the extent that correct configuration of the consuming portal and the WSRP4J tool is all that is required to deliver an existing JSR 168 portlet. Matthew Dovey, the head of the University of Oxford CREE technical development team, delivered a presentation to the 'GridSphere and Portlets' workshop in Edinburgh in March 2005¹⁶, which was found to be very useful in assisting with the set up of the consuming portal and the WSRP4J web application. A summary of the necessary steps is listed below¹⁷:

- Install WSRP4J on a web application server (tomcat, major version 5, was chosen by EDINA). Scripts are supplied with the WSRP4J software to do this.
- Deploy the portlet as a separate web application on the same web server. This is most easily done using a tool such as the uPortal deployment tool.
- Configure the WSRP4J web application (the portletentityregistry.xml file) to publish the portlet via WSRP.
- Configure a WSRP consumer channel for the chosen consuming application. This requires the portal administrator to assign values to the four WSRP endpoints (markup interface URL, service description interface URL, registration interface URL and portlet management interface URL). These URLs are systematically constructed by appending the correct namespace to the host and port number of the host web server.

Overall Review

The tangible result of the project has been the creation of a single portlet combining the essential functionality of both the GetRef and the GetCopy services. This portlet is compliant with the JSR 168 specification, and can be included as a component in any portal framework compliant with the JSR 168 specification. Alternatively, the WSRP4J tool may be used to make this portlet available as a Web Service.

At the time of writing no decisions have been taken as to how the portlet and software developed during the course of the CREE project should be used after the project. The points listed below will impact on this decision.

- Whether to support the distribution of portlet code (a JSR 168 portlet for installation in an institutional framework), or whether developed portlets shall be made available only as a Web Service (WSRP).
- The implications on user support of portalised services are unclear. In particular, the increased complexity of the delivery mechanism would make fault diagnosis and correction more difficult.
- It is uncertain how widespread the use of institutional portals that support the specifications is, and more information would be required on possible take-up to build the business model for supporting portlets.

- It is also unclear how the academic and wider world would find out that portlets are available? Increased development of the portlet registries available¹⁸ would be required to support this and encourage use.
- Release of the portlet as a service would also necessitate greater development of the design of the portlet to bring its presentation more in-line with the aesthetics of the native GetRef service and so as not to confuse users.
- Testing the portlets has been limited to the uPortal framework. It will be necessary to test portlets in other frameworks before they can be accepted as a means of service provision.

Conclusions and recommendations

- We suggest reading the two portlet specifications ahead of portlet service implementation if there is the possibility of a future requirement for a portlet version of a service. The portlet specifications make specific demands on how a service should be developed. The service must be designed to operate in two steps: one step to change the operational "state" of the service (`performBlockingInteraction` or `processAction`) and the next step to output the HTML (`render` or `getMarkup`) corresponding to that state.
- The tools available for portalising a service are mainly Java based. A prospective service implementer should take this into consideration when deciding on the development language for a service.
- A software engineer should consider developing a service with a two-stage process for rendering of output. In the first step the service should output XML and then transform the XML depending on the consuming application (portal framework or dedicated browser window) in the second step.
- It is essential that a software developer gives careful consideration to the construction of portlet URLs – this may be difficult if XSLT is being used to render output.
- The WSRP 1.0 specification states that security should be achieved using transport layer encryption, but does not make any firm recommendations (this is also being addressed in WSRP 2.0) and security is an untested factor in the delivery of services using portlet standards in general.

Ideally, the work undertaken by CREE has revealed that search tools should be designed with delivery via a portlet in mind. However, the work carried out by EDINA and other CREE technical partner sites (University of Oxford, Archaeology Data Service at the University of York and instructional media + magic, inc.) has shown that adapting existing search tools for use with JSR 168 and WSRP is possible. The CREE investigations have highlighted that institutional portals are prospective locations for presenting search tools to end-users within local environments and made accessible where needed.

¹ EDINA National Data Centre, <http://www.edina.ac.uk>

² Introduction to JSR 168 – the portlet specification, <http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/>

³ OASIS Web Services for Remote Portlets (WSRP) Technical Committee, <http://www.oasis->

open.org/committees/tc_home.php?wg_abbrev=wsrp

⁴ GetRef, <http://www.edin.ac.uk/getref/>

⁵ GetCopy, <http://www.edina.ac.uk/getcopy/>

⁶ uPortal, <http://www.uportal.org/>

⁷ Java Architectures Special Interest Group (JA-SIG), <http://www.ja-sig.org/>

⁸ Apache Tomcat, <http://tomcat.apache.org/>

⁹ Apache Jakarta project, <http://jakarta.apache.org/>

¹⁰ Apache Ant, <http://ant.apache.org/>

¹¹ MySQL open source relational database, <http://www.mysql.com/>

¹² JAFER toolkit, <http://www.jafer.org/>

¹³ WSRP4J toolkit, <http://ws.apache.org/wsrp4j/>

¹⁴ Model-View-Controller design pattern, <http://java.sun.com/blueprints/patterns/MVC-detailed.html>

¹⁵ Xalan XSLT transformation engine, <http://xml.apache.org/xalan-j/>

¹⁶ This presentation is available via the CREE presentations website,

<http://www.hull.ac.uk/esig/cree/presentations/>

¹⁷ See also a step-by-step guide to the process (developed by Matthew Dovey),

<http://www.oucs.ox.ac.uk/portal/developers/environment.xml>

¹⁸ For example the JA-SIG Clearinghouse at <http://jasigch.princeton.edu/> or the Portlet Open Source Trading site (POST) at <http://portlet-opensrc.sourceforge.net/>