CREE
Contextual Resource Evaluation Environment

Investigation of JSR 168 and WSRP standards
Part B: Experience of use in building portlets

*CREE Deliverable S2D1B*

Jonathan Hunter, EDINA, University of Edinburgh
Stewart Waller, Archaeology Data Service, University of York
Matthew Dovey, University of Oxford
Justin Tilton and Jonathan Allen, instructional media + magic, inc.
Chris Awre, University of Hull
April 2005
Introduction

CREE Deliverable S2D1A described the initial assessments of the JSR 168 and WSRP standards by the technical partners within the CREE project. These assessments were made on the basis of desk research and reflected the environments from which each technical partner was starting.

The initial investigation revealed a preference to use JSR 168 over WSRP, at least to start with, as the former appeared to better suit the purposes of the CREE project to develop portlets around existing search services rather than new tools being built specifically for use within portals. The use of the WSRP4J toolkit further allowed development of JSR 168 portlets to then be used to produce WSRP portlets at a later stage. The assessment, though, suggested that presenting search interfaces through portlets would be possible; to what depth the existing search functionality could be re-produced required implementation.

This paper, the second part of Deliverable S2D1 on investigation of the JSR 168 and WSRP standards, covers experience and feedback from use of the standards to develop portlets. Each technical partner has addressed particular issues within their work and these are reflected within the respective reports from the partners. The majority of the experience described is with the use of JSR 168, the starting point agreed as a result of the initial assessment. Both EDINA and the ADS provide valuable information into the environment required to make use of JSR 168 and the skills and knowledge used to develop the portlets. Matthew Dovey took the first steps towards using WSRP amongst partners and has developed a procedure that can be used to transform a JSR 168 portlet for use with WSRP. Testing of this procedure with other portlets, both inside and outside CREE, has taken place and the results of this will be reported in a later deliverable. im+m, in their work to develop a Google portlet, have also developed a useful means of using XSLT to affect how results are displayed from a search according to a user’s role, established a method for presenting multilingual versions of the portlet, and created a structure for contextual help files that can be associated with a portlet.

Overall, the experience in using the standards to develop portlets has been a good one. Working versions of all portlets have been produced and implemented at a third party site, addressing the issues of moving portlets between portal installations. It is clear that bending existing search tools to fit within the portlet standards has implications, particularly if the tool is not Java-based originally. It is also a serious commitment and not something that can be achieved lightly. It would appear that functionality beyond that of basic search is possible to include. This raises the issue of how much the portlet is attempting to re-produce what is available elsewhere, or whether the portlet is there to provide an initial interactive point of entry to searching, with the user being passed off to the native interface for more detailed usage. How thin or fat should the portal be?

This report demonstrates that portlets can be built. A future report will address some of the further functionality that the partners have sought to incorporate within the portlet environment.
Contents

Introduction 2

Contents 3

GetRef – JSR 168 Portlet Development

1. Introduction 5
2. Resources 5
   2.1. Hardware 5
   2.2. Software 5
   2.3. Skills 6
3. Engineering constraints 7
   3.1. Presentation issues 7
   3.2. Limitations of portlet API 7
   3.3. Language of implementation 8
4. Future progress 8
5. Summary 9
Appendix A: References 10

HEIRPORT – Portlet Development and Design

1. Questions and answers on developing portlets 11
   1.1. Local requirements for development 11
   1.2. Resources 12
   1.3. Software 12
   1.4. Assistance 13
   1.5. Difficulties 14
   1.6. Next steps 15
   1.7. Portlet details 15
2. ArchSearch portlet design 17
3. The link between the ArchSearch portlet and HEIRPORT portal 20
3.1. The HEIRPORT Framework
3.2. ArchSearch Portlet ← → HEIRPORT
4. Points to consider
Jafer – Transforming JSR 168 to WSRP
Google – The Google Portlet
  Executive Summary
1. Introduction
2. The CREE Google Portlet
  2.1. Modes and Roles
  2.2. View Mode
  2.3. Roles
  2.4. Edit Mode
  2.5. Help Mode
3. Portlet comparison
4. Internationalization
  Appendix A: Google Search API Overview
  Appendix B: Help Mode Input XHTML
  Appendix C: XLIFF Option for Internationalization
JSR 168 Portlet Development

Jonathan Hunter, EDINA, University of Edinburgh

1. Introduction

EDINA has continued to develop a portlet, based on the JSR 168 specification, for accessing the GetRef service. An initial version of the portlet was delivered to the CREE evaluation team as a proof of concept in late August. The reasons for choosing to base the development around the JSR 168 specification rather than WSRP were outlined in the document “Notes on the initial investigation of the WSRP and JSR 168 standards” (see Appendix A). The goal of this document is to more precisely inform a developer about the environment, the resources, the skills and the constraints on the creation of software enabling integration of the target service (GetRef) with a conformant portal.

2. Resources

2.1. Hardware

Most of the development was performed on a desktop machine. This was a machine with 512Mb memory and a Pentium IV 1800 MHz 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 portal channel (the uPortal term for portlets) and profile information is resident on a large Solaris box. 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 makes use of the GetRef machine-to-machine (M2M) interface, which also resides on the same Solaris machine as the MySQL database.

2.2. Software

So far development has focused on Java portlets and the JSR specification. Consequently this section primarily describes the software utilised in developing a JSR 168 portlet rather than a WSRP producer.

- Java - It is implicit that creation of a portlet complying with the JSR 168 specification requires the use of Java as the implementation language.
- uPortal - a product of the Java Architectures Special Interest Group (JA-SIG). Version 2.3 onwards includes a JSR 168 compliant portlet container, and was the selected portal framework in which to embed the test portlet. Any portal framework supporting the respective standards could have been chosen to host the
portlet.

- Tomcat - a web application server supporting the Java servlet specification. Tomcat is a product of the Apache Jakarta project and is available under the Apache software license. The uPortal software, or web application, was hosted on a Tomcat installation, on the desktop machine described in the previous section. Any servlet container compliant with the Java servlet specification could have been used, including JBoss, JRun, Oracle JServer, WebSphere, or WebLogic.

- Ant – 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).

- MySQL – a relational database, required by the uPortal application. It is used to host the portal profile and channel information.

- JAFER – Java Access For Electronic Resources. JAFER was developed by a team at the University of Oxford Computing Services (who are also participants in the CREE project). The package includes a Z-client suitable for embedding into a portlet in order to query the M2M interface presented by the GetRef service.

2.3. Skills

Before work commenced on the implementation it was necessary to carry out the paper based investigation of the two standards. This investigation may be found in a previous project report and is not repeated here (see Appendix A).

It would not have been possible to make a contribution to the technical aspects of the CREE project without substantial programming experience. This is not to say that no new skills were learnt. This section of the report attempts to give an accurate reflection of the knowledge required by the project, and acquired during the project.

2.3.1. Required skills

- Java – experience with Java is a pre-requisite for working with JSR 168.

- Tomcat – Installation of a Tomcat web server is fairly straightforward, and is not a time consuming task. Administration and configuration, however, is learnt through experience. EDINA has used different versions of Tomcat for many years in the area of geo-services. One of the software implementers, who is involved in CREE at EDINA, has hands-on experience of using Tomcat in a service environment, namely e-MapScholar (currently hosted within the Digimap service).

- Ant – experience of this tool and Java usually go hand in hand, particularly when developing web-based applications.

- SQL – useful for setting up a uPortal implementation.

- MySQL – uPortal is supplied with in-built support for multiple relational databases, one of which is MySQL. Some database administration is required but the existence of a dedicated relational database administrator within the EDINA workplace freed the CREE development team to work on the more challenging aspects of the implementation.
2.3.2. Skills acquired

- uPortal – It seemed sensible to develop the portlet in the portal container to be used by the test and evaluation team. Although the development team have had some past exposure to uPortal, most knowledge of this particular web application has been acquired during the project.
- JAFER – The JAFER (Java Access For Electronic Resources) team at Oxford have implemented a Z-client as part of the JAFER project. This utility was found to be suitable for embedding into the GetRef portlet. The EDINA team have had previous experience using this package through its association with the d+ project (see Appendix A) but much experience was gained directly through the CREE project.

3. Engineering Constraints

Many of the complexities in creating 'toolkits' or portlets to services are related to the failure to take into account the need to create a portlet when the service was initially devised. It is difficult to retrofit portlet features into a mature service. Reasons for this are explained in the paragraphs below.

3.1. Presentation Issues

Services intended for presentation in different user agents (a desktop browser or hand held device) can make use of server side (or client side) XSLT to transform the output to the appropriate format. A portlet window can be thought of a separate user agent. GetRef was only ever intended to consume the whole of the display window presented by a browser, so XSLT was not considered necessary. With hindsight, this would have been a useful feature. If the initial output from the service had been XML it would have been a relatively simple task to apply a different transform depending on the required display device.

3.2. Limitations of the Portlet API

The API between the portlet container and portlet is shaped by the interaction between the portal framework and the user. To explain, every time an action is performed on any portlet in a portal, every portlet currently displayed in the current portal window is required to refresh. The action performed by the user is, however, only captured by one portlet (via a processAction method call). This portlet performs the action determined by the user's behaviour, and then on completion of the action the framework will make a refresh request (via a doView method request) to all the visible portlets. The implication of this for the GetRef portlet is that the action of performing a search and the presentation of the results from that search are performed in two sequential steps. This does not fit well with the single client (browser) request followed by server response observed by users of the dedicated web interface. The approach to this has been to connect to the
M2M interface presented by the GetRef service. This bypasses all the intricacies of the GetRef user interface but has the failing that all the markup seen in the portlet has to be generated from within the portlet software, rather than from the server software.

The behaviour of the Z39.50 protocol (see Appendix A) when searching is to some extent analogous to the interaction of the portlet with its container. With the Z39.50 protocol a search may be instigated using the QUERY verb. This returns a results identifier that is then used in the subsequent PRESENT request to obtain the records. The two stage QUERY - PRESENT logic of the protocol is easy to map to the interaction between the portlet and its container, and is useful in simplifying the implementation logic of the portlet.

It should be noted that it is unlikely that the GetRef portlet will be able to give real time feedback on the state of the searches to the individual targets. This is due to a limitation in the servlet specification – a page will only be rendered by a servlet once it is complete. The GetRef web interface is dependent on Apache ASP (see Appendix A) technology, which has no such limitation.

3.3. Language of Implementation

If it is the intention from the outset that a service is to be portalised, then the engineers should look at the available resources to reduce the engineering task. WSRP4J is such a tool; it handles much of the low level SOAP messaging and portlet registration, leaving time for the implementers to concentrate on the higher level functionality. The tool acts as a wrapper to a JSR portlet enabling it to be accessed as a remote portlet using the WSRP standard. The task of mapping the API presented by the JSR 168 specification to a service written in the Java language seems preferable to trying to reconcile the API with a service written entirely in Perl. However, the decision to implement the GetRef and GetCopy services in Perl was taken several years ago. If it was known that portlet versions of these services were desired, then, given the lack of a tool such as WSRP for Perl, and no standard equivalent to JSR 168 in Perl, then the need to portalise these services may have been a factor in determining the language of their implementation!

4. Future Progress

In the short term, EDINA will continue to develop the GetRef portlet. Work will continue on improving the functionality and to a lesser extent the aesthetics. This is sensible as the priority of CREE is to demonstrate the creation of a prototype for the GetRef service; there is more to learn in trying to coalesce the portlet API presented by the specification with the underlying function of the current service.

EDINA is currently reliant on the Z-client included in the JAFER software package to communicate with the M2M interface of the GetRef service. This was always intended to be a temporary measure. Once the GetRef service has a SRW interface the portlet will query the GetRef database using this interface. The switch of protocol will make little
appreciable difference to user of the portlet.

As the project progresses EDINA will focus more on the WSRP standard. Although WSRP and JSR 168 appear similar in their objectives (the delivery of portlet services into compliant portal frameworks), the reality is somewhat different. WSRP is focused on the delivery of remote services. With JSR 168, the software is hosted locally to the portal - it may make use of a remote service, but this is beyond the scope of the specification.

EDINA, as a national data service provider, does not currently distribute software; it would seem less than ideal that development to date has focused on JSR 168 at the expense of WSRP in this context, though is largely due at this stage to the desire to investigate emerging technologies.

The WSRP standard has been released, but there is currently a lack of tools available to support WSRP development. In particular, the current release of uPortal does not have a WSRP client, and the WSRP4J project is still relatively nascent. It is hoped that a mature WSRP4J service will allow service providers like EDINA to retain autonomy over their portlets. The portlets will be hosted local to the service, but will be made available via web services through the use of a WSRP4J wrapper discussed in the previous section.

5. Summary

In the early stages of the project, 'toolkit' was the term used to refer, loosely speaking, to the documentation and software requirements to enable integration of the target service with a conformant portal. Integration is, though, perhaps not the best term to describe the work necessary to create a portlet for the GetRef service. In reality the work involved in ensuring compliance with the JSR 168 standard for this service is on a par with the creation of a completely new service. Had the GetRef service not been mature, integration may have been more appropriate. It is better to allow for the possible portalising of a service when it is still in the development stages rather than trying to retrofit the necessary functionality required by the specification.
Appendix A – References

The references found below are in addition to those found in the references section of the first document listed.

Notes on the initial investigation of the WSRP and JSR 168 standards with respect to EDINA’s involvement in the CREE project.
*This is available as Appendix A to CREE Deliverable S2D1 Part A: Initial investigation of the JSR 168 and WSRP standards, [http://www.hull.ac.uk/esig/cree/documents.html](http://www.hull.ac.uk/esig/cree/documents.html)*

d+ (Brokerage for Deep and Distributed e-Learning Resources Discovery),

National Information Standards Organisation – Z39.50 Resource Page,
*[http://www.niso.org/z39.50/z3950.html](http://www.niso.org/z39.50/z3950.html)*

Apache ASP - Active Server Pages port to the Apache Web Server,
Portlet Development and Design

Stewart Waller, Archaeology Data Service, University of York

1. Questions and answers on developing portlets

1.1. Local requirements for development

What hardware was required?

Development took place on two platforms:

- **Windows 2000** running on an IBM-PC compatible,
- **Solaris 8** running on a networked Sun station.

What skills were required to enable development?

As a portlet is essentially an extension of the Java Servlet technology, a working knowledge of Java Servlets and JSP’s was required.

In addition, knowledge of the build/deployment processes for open source software was essential for working within the Unix environment. Tools such as Apache Ant & Maven were used for this purpose.

What learning was required (if any)?

The JSR 168 API defines the portlet specification. A study of the various packages and methods defined in this API was necessary. Also, the differences/similarities between the portlet specification and the servlet specification needed to be understood.

Did the tool being adapted for delivery through a portlet have any effect or influence on the way development was taken forward, i.e., did the language or the platform of the tool affect portlet development, and how?

The ADS ArchSearch Portlet was developed to provide basic search functionality for the ADS HEIRPORT Z39.50 target database. The existing HEIRNET Portal servlet is based on the ZavaX portal framework, which in turn relies on the Apache Cocoon publishing framework to transform the result set. It was decided that the portlet would send a query to the existing HEIRPORT servlet, which in turn would perform the query on the Z39.50 target. Therefore, the ADS Portlet would essentially act as a ‘remote form’ with the ability to handle the request/response mechanism in a more sophisticated manner.
1.2. Resources

*What external resources were used to enable development?*

See elsewhere (software/hardware)

*What ‘hoops’ had to be jumped through in order to enable development?*

A relatively steep learning curve for JSR 168 and related technologies

*What internal resources were required to develop the portlets, i.e., staff time, technology?*

Staff time taken for portlet development was approximately 30 days

1.3. Software

*What software was installed to enable portlet development?*

Two environments have been used for development and test purposes as described below:

**Windows:**
- Windows 2000,
- Java 1.4.2,
- Tomcat 5.0.28,
- uPortal 2.4,
- HSQL 1.7.2, MySQL
- Pluto 1.0.1

**Unix:**
- Solaris 8,
- Java 1.4.2_05,
- Tomcat 5.0.28,
- uPortal 2.3.4,
- Oracle 8i,
- Pluto 1.0

In both cases, the NetBeans IDE 3.6 was used for development of the portlet. The introductory tutorial for developing JSR 168 compliant portlets using NetBeans was found to be very helpful, particularly in constructing JSR-168/Pluto compliant portlet templates. See [http://www.netbeans.org/kb/articles/NBAndPortlets.html](http://www.netbeans.org/kb/articles/NBAndPortlets.html) for details.

*How easy was it to carry out this installation?*

**Windows:**

1) uPortal quick-start / HSQL. Successfully installed and configured this set up to run on local PC.

2) uPortal stand alone / MySQL. Successfully installed and configured this set up on local PC. No problems with deployment. MySQL database worked fine.
UNIX:

iii) uPortal stand alone / Oracle. This configuration was deployed to the Solaris development box. However, initial population of the Oracle database tables caused problems, mainly hanging during processing of the pushfragments.xml file.

This problem was fixed by updating the environment from JDK 1.2 → JDK 1.4 and uPortal 2.3.3 → uPortal 2.3.4.

What lessons were learnt from installing the software?

Variations between existing Xerces and Xalan Java libraries seem to cause major problems amongst various web applications running under Tomcat. It was found that the versions supplied with the uPortal distribution were required.

At the time of set-up of the above environments there were no distributions of the Jakarta Pluto container available to download from the web. To overcome this, the source had to be downloaded from the CVS tree and compiled on the respective platform.

However, there are now pre-compiled Windows binaries and source code available from the Jakarta Portals website at http://portals.apache.org/pluto/mirrors.cgi.

What options were available to the software being used, and why was the software used chosen?

The only specification was to develop a JSR 168 compliant portlet. However, the majority of CREE developers used JA-SIG’s uPortal as the primary test-bed/target portal framework as a result of the interest in this platform. uPortal is based on the Pluto JSR 168 reference portal container.

1.4. Assistance

Were you able to find any assistance as part of your portlet development, i.e., lists, websites, people etc? Were they helpful?

The initial chapter in Richardson et al. (2004) “Portal Development With Open Source Tools” was found to contain some useful information regarding the basic structure of JSR 168 compliant portlets. Additionally, Chapter 9, “Developing Applications and Workflow for Your Portal” contained some interesting reading regarding Portal architecture and in particular the MVC (Model-View-Controller) paradigm for Portlet design.

Useful information was found on the NetBeans website, in particular the tutorial for using the NetBeans IDE to develop portlets at http://www.netbeans.org/kb/articles/NBAAndPortlets.html.
Additionally, the following weblogs/newslists were found to be very helpful:

- http://portlets.blogspot.com/
- http://groups.yahoo.com/group/portlets

Finally, the obligatory ‘googling’ for ‘portlets’, ‘JSR-168’ etc. also provided some interesting links to relevant information. This alone is indicative of the growing interest in portal solutions.

**What tips would you give to others adapting an existing tool for delivery as a portlet?**

i. Templates help when developing for target Portals – such as those described in the NetBeans tutorial.

ii. Test your portlet in a number of portal containers! Although there are many ‘JSR-168 compliant’ portal solutions emerging, they all seem to have a slightly different approach to implementing this support.

**1.5. Difficulties**

**What were the major barriers you perceived and encountered in your development?**

Computing Services at the University of York employ a strict firewall policy that affects development requirements. Significantly, the development server has a limited number of ports available to the outside world. Therefore, a test portal framework had to be set up off-campus to test external functionality of the portlet.

As previously mentioned, the procurement of the Pluto source/binaries was initially an obstacle. However, once a CVS environment had been set up this was no longer an issue. Also, the source/binaries are now available for public download at the Jakarta portals website.

Signing portlets for different Portals (uPortal, GlueCode, Pluto etc.) was definitely an issue. For example, each portal container needs to make specific modifications to the web.xml file bundled within the .war file. This means that unless the portlet deployment mechanism has been designed in a suitable manner, the administrator of the portal system would need to manually open and edit this file themselves for use in their portal system. The creation of web.xml templates for specific portal containers, as mentioned re: NetBeans etc., can to a certain extent help with this.

Until portal frameworks address this issue, It may be advisable to provide several web.xml files for specific portal platforms.
1.6. Next steps

Bearing in mind progress to date, what would see as the natural next steps, in terms of what you could now do?

This stage of development was primarily a test to establish communication using an existing application framework (HEIRPORT/Z39.50). However, if the ADS were to develop a portlet for public use / academic institutions etc., a revised portlet design would be essential. This would be based upon an MVC architecture and would perhaps require the use of JavaBeans to help manipulate the result set more efficiently.

For example, rather than relying on preformatted HTML from Cocoon that is displayed ‘as-is’ by the portlet through JSPs, a custom Cocoon TagLib could be created to supply the Bean with pure XML. This in turn could be accessed and translated by the portlet and displayed to the user by the various JSPs etc. The net effect would be to decrease the pressure on the ADS Server and place more processing onto the portlet.

As the ADS is host to a number of collections and also has access to a number of resource targets, there is no reason why a suite of portlets could not be designed. Although this could be unnecessary if the user was given the ability to choose from various datasets through the preferences functionality of a single portlet.

1.7. Portlet details

Which services have been specified within the portlet?

The ZavaX system we are using can accept the following search criteria:

**CIMI 4Ws:**
- Who
- What
- When
- Where

**CIMI Spatial Area:**
- British National Grid (OSGB)
- Irish National Grid (OSI)
- Latitude & Longitude (LL)

The Where option can be configured using place names (CIMI 4W's - Where) or by using X, Y map coordinates (CIMI Spatial Area). However, using a coordinate system would be difficult to implement in a portlet context, as the majority of users wouldn't know which to use. A map interface could be used but would be difficult to implement, not only due to the nature of portlet client/server communication but also due to the licensing issues surrounding use of map images etc.
How is the mark-up being used within the presenting search results being generated?

The mark-up is being generated from Cocoon. Cocoon is an integral part of our Zavax system, providing the ability to transform results using XSLT.

These transformations are specified using a custom TagLib. This has been based upon the ADS ARENA service TagLib for convenience.

The final portlet will have its own TagLib that will leave one of two options for mark-up and presentation:

- no server side formatting - portlet will receive pure XML encoded result sets.
- minimal formatting - results will be received as an HTML fragment.
2. ArchSearch Portlet Design

The ArchSearch Portlet will act as a JSR-168 compliant remote search tool for searching the HEIRPORT Z39.50 targets.

The following views will be displayed using the VIEW portlet mode:

**viewSearch(): viewSearch.jsp**

methods:

- viewSearch()
- viewResults()
- editBookmarks()
- editPrefs()

**dispForm()**

**doSearch()**

The ‘Search’ view is basically a form that will send query parameters to HEIRPORT. The ‘Where’ option will be based upon place names and the ‘When’ option will be based upon a controlled drop-down box of common period terms as used by the ArchSearch database schema.

**viewResults(): viewResults.jsp**

methods:

- saveRecords()
- dispRecords()

- nextTen()
- previousTen()
- saveRecords()

<record1> ...
... ...
...</record1>

<record2>
... ...
... ...
...</record2>

<record3>
... ...
... ...
...</record3>
The ‘Results’ page will present the user with a list of results, allowing the user to step through x number of results per page. Using the Save option will allow the user to save the results (or a subset of the results) to a text-file (TXT, XML etc.) for use offline.

The following views will be displayed using the **EDIT** portlet mode:

**editBookmarks(): editBookmarks.jsp**

<table>
<thead>
<tr>
<th>METHOD</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewSearch()</td>
<td>viewSearch.jsp</td>
</tr>
<tr>
<td>viewResults()</td>
<td>viewResults.jsp</td>
</tr>
<tr>
<td>editBookmarks()</td>
<td>editBookmarks.jsp</td>
</tr>
<tr>
<td>editPrefs()</td>
<td>editPrefs.jsp</td>
</tr>
<tr>
<td>dispQuery()</td>
<td>dispQuery.jsp</td>
</tr>
<tr>
<td>saveQuery()</td>
<td>saveQuery.jsp</td>
</tr>
</tbody>
</table>

Upon selecting the ‘Bookmarks’ option, the user can choose to Save the existing query buffer as a pipe-delineated string which can then be stored as portlet preference. This could then be recalled using the Load Bookmark option which would either switch to the Search view with form values preloaded from the Bookmark values, or automatically perform a search based upon the bookmarked values. There would need to be an option to add a ‘description’ of the bookmark/query to the end of the stored query string for identification purposes.
editPrefs(): editPrefs.jsp

The ‘Prefs’ option will allow the user to select various preferences such as Target(s) and formatting options. Formatting options are discussed later in this document.

A view for HELP and ABOUT will be displayed using the corresponding portlet modes.

viewHelp(): viewHelp.jsp

viewAbout(): viewAbout.jsp

The 'Prefs' option will allow the user to select various preferences such as Target(s) and formatting options. Formatting options are discussed later in this document.

A view for HELP and ABOUT will be displayed using the corresponding portlet modes.

viewHelp(): viewHelp.jsp

viewAbout(): viewAbout.jsp
3. The link between the ArchSearch portlet and HEIRPORT portal

3.1 The HEIRPORT Framework

The implementation of this framework is based on two purpose-designed Java packages, Zava, a Z39.50 API providing client/server features, and ZavaX, a Web Client API providing access to Zava from the Web. The system makes extensive use of XML and RDF for configuration and communication, and of XSLT for transformation and delivery of content.

As shown above, the Portal accepts search and retrieval parameters from the web and passes them to the Mediator. The Mediator selects the data sources according to their capabilities (e.g., Spatial Referencing System) and based on the profile configuration, converts the parameters to the Z39.50 protocol, launches parallel searches over all of the elected Z39.50 servers through their target access points, and finally presents the results as soon as they arrive using a TagLib.

3.2. ArchSearch Portlet ➔ HEIRPORT

The following diagram shows how the ArchSearch Portlet will communicate with the HEIRPORT Portal:
A typical HEIRPORT query is spread over two processes. Firstly, a request is made to the portal to identify the targets and number of hits. The response of this query is then displayed to the user. If the user wants to view the results, they must choose one of the fruitful targets as displayed in the initial results screen.

If the user decides to view a specific targets result set, a second query is then carried out by HEIRPORT and returns the results on an ‘$X$ number of results’ per page basis.

This two-stage query process is illustrated below:
4. Points to consider

4.1. How will the ArchSearch Portlet deal with multiple targets?

It may be worth limiting searches to a single target. This target could be set using the preferences options. By searching a single target, the reply to the initial response (as mentioned above) could be automated to either return a ‘no results found’ message to the portlet, or automatically perform the second query and retrieve the result set.

Alternatively, if a search across multiple targets is implemented, the user would need to be presented with the initial targets results summary and be directed to choose which target to perform the second query on (as is the case with HEIRPORT). This would introduce an additional element/process to the ‘results’ view as outlined above.

4.2. Client or Server side formatting?

The ability to transform the result set using Cocoon into formatted HTML is useful for design purposes on a standard web page. However, within a portlet context it is often the case that any formatting/design is left to the Portal framework to handle. This maintains an element of branding consistency and overall Portal usability. However, the Cocoon XSLT will still be needed to transform the internal Z39.50 based XML into an XML structure suitable for manipulation by the portlet.

If the portlet receives an XML formatted result set, stepping through query results one at a time or ten at a time becomes possible. Also, without the padding of HTML markup, the result set should be smaller, saving on bandwidth etc. Further, to cut down on Client/Server requests, there is no reason why 100 results can’t be pulled from HEIRPORT at a time. Then the user could step through 10 records per page without having to request the next 10 from HEIRPORT until they have reached records 90-100 etc.
Transforming JSR 168 to WSRP

Matthew Dovey, Oxford University Computing Services

Information on Developing Portal Channels is available on the OUCS website at http://www.oucs.ox.ac.uk/portal/developers/. An environment to allow JSR 168 portlets to be transformed for use with WSRP using the WSRP4J toolkit has also been developed through work to build portlets around the JAFER toolkit. Details can be found at http://www.oucs.ox.ac.uk/portal/developers/environment.xml. The steps required to build this environment are also presented below.

1. Overview

This will set up an environment for testing WSRP and JSR 168 portlets using uPortal and WSRP4J. The setup will resemble the following diagram

![Diagram of the setup for testing WSRP and JSR 168 portlets using uPortal and WSRP4J](image)

Figure 1. Overview
The JSR 168 portlet is deployed such that it is accessible within a local installation of uPortal. The JSR 168 is also deployed within the WSRP4J producer: this allows the JSR 168 portlet to be accessed via WSRP either from a local or remote uPortal installation.

2. Prerequisites

1. You will need JDK 1.4.2
   Remember to set path and JAVA_HOME accordingly
2. You will need Apache Ant (preferably 1.6.2)
   Remember to set path and ANT_HOME accordingly
3. You will need CVS
   Remember to set path accordingly

3. Setup environment

1. Download build.xml
2. build.xml has a number of targets for downloading the various components needed for creating the JSR168/WSRP Environment
   1. ant download-uportal will download the latest uPortal from cvs into sub-directory portal
   2. ant download-wsrp4j will download the latest Apache WSRP4J from cvs into sub-directory ws-wsrp4j
   3. ant download-tomcat will download the Apache Tomcat 5.0.28 into sub-directory jakarta-tomcat-5.0.28
   4. ant download-hsqldb will download the Hypersonic SQL Database 1.7.2 into sub-directory hsqldb
   5. ant download-all will download all of the above
   6. On Unix platforms you should chmod the various shell scripts to allow execute access.

4. Build WSRP4J

1. Edit build.properties. Copy ws-wsrp4j/build/build.properties.example to ws-wsrp4j/build/build.properties
   Change TOMCAT_HOME to give the fill path to the tomcat downloaded above
   The TOMCAT_PORT can be left as 8080 (unless you are using a non-standard Tomcat setup)
2. Whilst in the ws-wsrp4j/build directory, run build.bat install-provider-pluto or build.sh install-provider-pluto
   Note errors about deploying wsrptest can be ignored.
3. Edit jakarta-tomcat-5.0.28/webapps/wsrp/WEB-INF/config/services/ConfigService.properties
   Change the host.name to the DNS name of the machine, if you want external consumers to access the WSRP portlets

5. Start Hypersonic

   1. Move to the directory hsqldb/demo
   2. run runServer.sh -port 8887 -database uPortalDb or runServer.bat -port 8887 database uPortalDb
   3. Leave running!

6. Build uPortal

   1. Edit portal/build.properties
      Change server.home to point to the full directory for tomcat downloaded above
   2. In portal subdirectory run ant deploy
   3. Still in portal subdirectory run ant initportal

7. Deploy Portlet

   This will deploy the RSS portlet supplied with uPortal - however, you can substitute your own (some of these steps will have already been done by ant initportal above, but are safe to repeat)

   1. In the portal subdirectory run
      ant deployPortletApp -DportletApp=<path>/<portlet.war>
      For the RSS portlet this is
      ant deployPortletApp -DportletApp=lib/portlets/RssPortlet.war
      The reason for this step beyond normal war deployment is that the portlet-guid in web.xml needs to be set to {context}.{servlet} where {context} is the portlet's servlet context, and {servlet} is the portlet's servlet mapping. Obviously the former may vary.
   2. At this point we have a JSR 168 portlet deployed under the id {context}.{servlet} (RssPortlet.RssPortlet for the RSS portlet); next, to deploy this via WSRP
   3. Edit jakarta-tomcat-5.0.28/webapps/wsrp/WEB-INF/data/portletentityregistry.xml. You need to add a new application using the following template where {context}.{servlet} is the portlet-guid as above:

            <application id="{context}"/>
            <definition-id>{context}</definition-id>
<portlet id="{serlvet}">
  <definition-id>{context}.{serlvet}</definition-id>
</portlet>
</application>

For the RSS portlet this is:

<application id="RssPortlet">
  <definition-id>RssPortlet</definition-id>
  <portlet id="RssPortlet">
    <definition-id>RssPortlet.RssPortlet</definition-id>
  </portlet>
</application>

4. We now have a WSRP portlet deployed under the handle \{application id\}.\{portlet id\}. For simplicity the above template sets the handle and portlet-guid to be the same.

8. Run Tomcat

  1. In the jakarta-tomcat-5.0.28 subdirectory run bin\startup.bat or bin\startup.sh

9. Configure uPortal to access the JSR 168 portlet

This can only be done in the uPortal instance running on the local machine (in the same Tomcat as the JSR 168 has been deployed). The following steps are a template (e.g., you may wish to add the portlet to different groups)

  1. uPortal should be accessible on the URL http://\{hostname\}:8080/uPortal where \{hostname\} is the machine on which you are running Tomcat
  2. Logon (default is admin, admin)
  3. Click on the Channel Manager icon
  4. Select publish new channel
  5. Select channel type portlet. Press next.
  6. Enter name, details, timeout etc. Press next.
  7. Portlet id is \{context\}.\{serlvet\} as in the portlet-guid in web.xml file above, e.g. RssPortlet.RssPortlet
  8. Skip over parameters (unless your portlet needs them)
  9. Add channel controls as required by the portlet (skip if unknown)
  10. Add to the ‘development’ group (check it and press select marked) so that you can find the portlet. Press next.
  11. Add to the ‘everyone’ group (check it and press selected marked) so that you can find the portlet. Press next.
  12. Press finish to save, and then the home icon to exist the Channel Manager
  13. Click on ‘turn on preferences’ and add the channel content to the portal
10. Configure uPortal to access the WSRP portlet

This can be done in uPortal running on any machine

1. uPortal should be accessible on the URL http://{hostname}:8080/uPortal
   where {hostname} is the machine on which you are running Tomcat
2. Logon (default is admin, admin)
3. Click on the Channel Manager icon
4. Select publish new channel
5. Select channel type WSRP Consumer. Press next.
6. Enter name, details, timeout etc. Press next.
7. The URLs needed are as follows, where {host} is the IP address of the
   machine running the WSRP4J producer.
   Markup Interface URL: http://{host}:8080/wsrp4j/WSRPBaseService
   Service Description Interface URL:
   http://{host}:8080/wsrp4j/WSRPServiceDescriptionService
   Registration Interface URL:
   http://{host}:8080/wsrp4j/WSRPPortletManagementService
   Portlet Management Interface URL:
   http://{host}:8080/wsrp4j/WSRPPortletManagementService
8. Handle is {application id}.{portlet id} as entered in the
   portletentityregistry.xml file above, e.g., RssPortlet.RssPortlet. Ensure that
   user can modify is checked. Press next.
9. Skip over parameters (unless your portlet needs them)
10. Add channel controls as required by the portlet (skip if unknown)
11. Add to the ‘development’ group (check it and press select marked) so that
    you can find the portlet. Press next.
12. Add to the ‘everyone’ group (check it and press selected marked) so that you
    can find the portlet. Press next.
13. Press finish to save, and then the home icon to exist the Channel Manager
14. Click on turn on preferences and add the channel content to the portal

References

1. JDK 1.4.2., http://java.sun.com/j2se/1.4.2/download.html
3. CVS, https://www.cvshome.org/
4. This file has been developed in the context of use at Oxford University.
   However, the actions it carries out can be applied at any location without
   editing. The file can be downloaded from
   http://www.oucs.ox.ac.uk/portal/developers/build.xml
The Google Portlet

Justin Tilton and Jonathan Allen, instructional media + magic, inc.

Executive Summary

Preface

The CREE Google portlet prototype was developed as part of the Contextual Resource Evaluation Environment (CREE) project. Oxford University, University of Edinburgh, University of York, and Newark and Sherwood College participate in the project led by the University of Hull. instructional media + magic, inc. provides additional resources and perspective. The project is one of those funded by the Joint Information Systems Committee as part of the JISC Information Environment and Portals areas.

The portlet development is one of several search tools or software—JAFER toolkit, BALSA, HEIRPORT, Google APIs, cross-search—important to higher education research and instruction. The project redevelops the search services in three steps: The first is to create an open-standards version using the recently adopted JSR 168 portlet specification. Following this specification enables the search channels to be used with almost all commercial portal software and a number of open source portals, including uPortal from the JA-SIG Collaborative. The second step is to extend the functionality of the portal by adding Help instructions, supporting internationalization, and providing for alternative role-based or user-preferred presentations. The third step—based on project research and usability testing—is a redesign of the user interface to better meet user needs. Based on the final project specifications, the prototype will be redesigned. The user presentation may change significantly at this time.

There have been suggestions for further extending the project to support the Web Services Remote Portlet (WSRP) service, to incorporate federated searches—where a search combines results from several information sources, and consolidating several of the different search services into a single multi-protocol search portlet. This is speculation on how the research itself and how it will influence design. Early reviews of the work suggests standardization and focus on design will produce software portlets that are easier to use, more productive and less difficult to maintain than the many now separate Web-based search applications.

We hope the results will meet those early expectations.

Background of Plumtree’s Google Portlet

The word “portal” was used describe a Web service that simultaneously displays information from several independent data sources. Yahoo became a symbol of this aggregation as news, advertisements, weather, and other information was displayed on a
single Web page in separate “rectangles.” Each rectangle represented the display of a different information source, and the user could control what information was displayed and where. Portals became a distinct category of software and subsequently were used in business to simultaneously display from several different business systems. For example, the portal page could simultaneously display financial data, e-mail, customer information, product catalogs and inventory, and general news.

Software firms such as Plumtree and Epicentric—subsequently purchased by Vignette—developed enterprise portal software. The move to standard “portlets”—the separate “rectangles”—was led by Epicentric and immediately followed by Plumtree. In February 2002 IBM and Sun Microsystems, Inc. initiated a request through the Java Community Process that led to the JSR 168 Portlet Specification. The public review specification became available April 16, 2003 and the final specification was approved October 27, 2003. Plumtree Software, Documentum, BEA Systems Inc., and Sun Microsystems announced the Portlet Open Source Trading Site (POST). The Source Forge site (http://sourceforge.net/projects/portlet-opensrc/) aims to help companies kickstart their portal deployments, leading to faster time to value for all portal customers by providing open source portlets and a forum to exchange and learn about how these emerging new standards.” December 2, 2003 Plumtree contributed the Google portlet. It is described: “[The] Google Portlet is a simple portlet that searches Google using the Google Search API. It supports view and help portlet modes, and normal and maximized window states.” This was one of four portlets contributed that month; no more have been contributed.1

Plumtree’s goal was to advance the JSR 168 specification by having a portlet that could immediately be recognized—almost all Web users are familiar with Google—and used an existing SOAP-message based XML data exchange. The Plumtree portlet demonstrates the capability well, is simple to install and use, and requires no integration with other components. But there are limitations to the number of entries and the fixed display appropriate to a demonstration portlet.

The CREE Google Portlet

The CREE project similarly elected to use the Google Portlet to demonstrate capabilities that improve the usefulness and productivity of the original portlet that features the Google API. The enhancements include: Implementing the full capabilities of the Google Portlet, extending the number of entries from 100 to unlimited, supporting multiple languages, adding content language filtering, adding a feature that uses the JSR-168 Help mode, provide alternative presentation formats and item content, and support multiple languages for the portlet itself. In addition, user preferences are persisted so they are available to the user the next time they logon to the portal. It may be useful if the default presentation formats were related to the role of the user. This depends upon a subsequent

---

1 At the same time Plumtree contributed the RSS Portlet described: “Rss Portlet views Rss 0.91 and 2.0 newsfeeds. It includes edit mode for adding or eliminating additional newsfeeds.” Redeveloping and enhancing this portlet is outside the scope of the original CREE proposal.
design of an authorization service and standardized roles.\textsuperscript{2}

The resulting portlet is called the CREE Google portlet prototype.\textsuperscript{3} The prototype will be made available for use and review. The CREE project will recommend changes to the user interface based on emerging research and usability testing and subsequent user interface design specifications.

\textsuperscript{2} The uPortal installations used for this development and testing using “Groups and Permissions” from Columbia University. How roles will be defined and made available to the portlet may depend upon the subsequent development of an authorization service. This lack of definition prevents a useful implementation of role-based display. The essential different may be “student” that display more items with less information than the same query for a “faculty” member.

\textsuperscript{3} The project has advised Google, Inc. of this project and requested permission to use the Google logo on the channel. These discussions continue. There is one significant difference between using the Google API and using the Google Web page: advertisements or advertisement URLs are not included in the current Google API XML message.
1. Introduction

The Google portlet was developed for Contextual Resource Evaluation Environment (CREE) by instructional media + magic, inc. The purpose of the Google portlet is to demonstrate a functional, JSR-168 compliant portlet in uPortal which utilizes WSRP.

The CREE google portlet uses XSLT for creating the presentation layer of the portlet. This presentation is based on the role of the user, altering the presentation depending on if the user is a student, faculty member, etc. The portlet is also designed with parameters for internationalization and localization, allowing the user to change the language settings used in the presentation of the portlet when available. It has a flexible preferences settings, Edit Mode, that gives the user control over the display language as well as the number of results they see on each screen. The total number of results is unlimited, scrolling with that many per screen for as long as the Google API continues to return new pages. It also contains a Help mode that uses XHTML for its content and a hyperlinked index to move between chapters. Providing this function to the portlet has required some framework changes to uPortal, for allowing parameters to be constructed - which will be useful to all future uPortal portlet developers.

2. The CREE Google Portlet

2.1. Modes and Roles

The JSR-168 Specification defines that the function and rendered GUI of a portlet should use 'modes' to control any changes in function caused by user interaction. Three of the most commonly expected modes, View, Edit, and Help are predefined within the specification. An extensible Custom mode is also provided for.

2.2. View Mode

Figure 1 shows the initial Google portlet interface in View mode. The only actions available to the user in view mode are to perform a search, or to switch to a different mode with one of the icons.
Figures 2 and 3 show the results after a search has been performed, and then when a different results page has been selected. The only additional action that a user can perform beyond the initial search GUI is to select one of the results links. Otherwise, they can still perform a new search or change modes with one of the icons.
2.3. Roles

In uPortal v2.4 and later, portlets are able to access information about the user. Since the portlet is aware of the user, the function and GUI of the portlet can be changed accordingly. The CREE Google Portlet provides an example of this by changing the look of the search results for a student.

Typical Google Search Results

![Google Search Results](image)

A typical Google search will return results similar to the image above. Each item that is returned has three main parts. The **Title** is name of the item or page. The **Snippet** is a small portion of the text from the page, similar to an annotation. The **Link** is the URI where the item is actually located. The **Title** and the **Link** are usually both active links to this URI.

Most users of the CREE Google portlet will see results similar to those illustrated above.

If, however, the user is logged in with the role of a 'student' they might see a return on their search more like the image below:
Google Search Results – Student Role

The actual function of the CREE Google Portlet is not affected in this example of roles in uPortal. The same set of results is returned. The only difference is in the GUI of how these results are displayed. The Student Role removes the Snippet and Link parts of each result, returning just the Title. As with the typical results, the Title is an active link to the URI for each item.

2.4. Edit Mode
When selecting the edit button, the portlet changes to **Edit Mode**. In this mode, the user can change the values for the runtime parameters that affect the function of the portlet. In the case of the Google portlet, these parameters are the Google Key (the unique 'permissions' key that allows use of the Google API), the number of results that appear on each page and the locale for changing the language this portlet displays. Any changes made to these parameters in Edit mode will instantly appear when selecting the 'Submit' button.

### 2.5. Help Mode

As mentioned briefly above, the **Help mode** uses an XHTML document to determine its content and menu. XHTML was chosen for the ease in which it can be
created and edited. Several simple text and HTML editors can create XHTML, and no special knowledge is needed to make the Help input document.

The function of the Help mode is similar to a hypertext manual. The list on the left, under the heading “Help Menu” is an index of the chapters in the help content. Each item in the menu represents a chapter in the help content, and links to the first page of that chapter.

The right side of the help window contains the help content. It uses well-formed XHTML fragments, and can therefore include any kind of media resource that the writer wishes to include. Within each chapter, the user can step forward or back one page at a time with the arrows provided below the content. Alternately, the help content can contain links that can go to other help pages, or to any viable URL. It would be useful, though not required, that any link to an external page target a new window.

Appendix B discusses the input XHTML and lists the entire XHTML document used in the CREE Google Portlet.

3. Portlet Comparison

Plumtree released a JSR-168 compliant portlet that uses the Google search API, and can be published in uPortal. While this portlet does operate in uPortal and does use Google to perform searches, it is limited in both its ease of use and completeness of its function. The following comparison details the Plumtree Google portlet with the one created by CREE.

Initial View

Plumtree

CREE

This is a JSR-168 compliant portlet that uses WSRP, SOAP messaging, and SSL-7 along with the Google Web API to perform web based searches. It has a rule based output that will show different search results dependent upon the permissions of the user. It also contains an edit mode for changing the preferences. The edit mode is accessed by clicking the edit button for this portlet.

{%image Aberdeen - Edinburgh.png%}

{%image Aberdeen - Edinburgh.png%}
Both Google searches start with a very similar interface, giving the user a text input box to perform the search. The two words above the text input on the Plumtree are actually links, and it has the appearance of an unfinished beta version of the portlet.

**Search Result – First Page**

**Plumtree**

**CREE**

The results of the search begin to show the limitations in the function and GUI of the Plumtree version of the Google search. First, it displays only the title line of each returned item, opting not to include the description and link provided by Google. Additionally, there is no mechanism provided to display an item that does not have a title. It is merely shown as a blank line, and is thereby an inaccessible link to the user. As a demonstration of role based presentation, the CREE portlet will display the results GUI dependant on the 'role' of the user. A user who is logged in as a student will get a result very similar to the Plumtree output, showing just a list of the returned item titles with a bullet next to each one. Any item that does not have a title will be designated at “No title provided.” If the user is logged in as a faculty or administrator, however, the results will return the title, description and link URL, as seen in the example above.

Changing the output GUI was done as an example of how 'roles' can be used in a portlet. The actual GUI output of any role can be easily customized, as the interface design is constructed using XSL-T. XSL-T is a fairly simple to use, yet powerful tool designed specifically for providing UI from XML input. Since the Google API passes XML in its SOAP payload, XSL-T is extremely well suited to work with the API data. The Plumtree portlet makeup is constructed directly by the business logic in JAVA classes. Since the
logic layer is combined with the presentation layer, it is rather complicated to make any modifications to the Plumtree portlet.

Search Results – Final Page

Plumtree

CREE

The Plumtree search has a maximum of 10 pages with 10 items each, for a total maximum of 100 items returned. It mentions 54000 results in the first line, yet they will only give the user access to the first 100 of them, and those only if the item returns a title. The CREE Google portlet will display all of the results returned from Google, regardless of the number.
Help Mode

Plumtree

The Plumtree help mode consists of just one screen, a description of the portlet and a message to contact the administrator for any help with or changes made to the portlet.

CREE

The CREE help mode uses an XHTML input document to create an interactive help interface. This mode includes a hot linked menu that connects to chapters and pages of the help content.

Edit Mode

Plumtree

None Provided. Any changes to the Portlet must be done by the portal Administrator.

CREE

The CREE portlet provides for editing of the portlet parameters during run time. These include the key for use of the Google search engine, the number of results shown on each page, and a locale setting. If you look at the Plumtree help screen above, it states that the Google API to only 10 limits the number of results that are returned. The API actually limits the maximum results to be 10. The CREE portlet provides a run time parameter to change the number of results listed to be any number from 1-10, allowing the user control
over the screen real estate that is used by the portlet. Any changes to these run time parameters will be instantly reflected in the UI, even if the user has just completed a search.

4. Internationalization

The CREE Google portlet is designed to be used with multiple languages, as the issue of internationalization becomes more prominent. Generally, uPortal developers employ the resource bundle model used in JAVA. The figure above shows a section of the XSL that builds the View mode of the portlet. The first highlighted line writes the text “Search:” in front of the input textbox in the initial view of the portlet. This XSL line itself does not determine what text is written there, but uses whatever value is in the variable portlet.view.form.textbox.label. The second highlighted line is the default value for that variable.

Each user has a preferred language specified in their settings. For any portlet that has a resource bundle matching this language setting, the uPortal framework code will import the values for any needed translatable units or variables that are included in the properties file for that language. If a variable is not included in the properties file, the default will be used. The text below shows a section of a resource bundle from a translation of the Google portlet View mode text that has been translated into Czech. The JAVA will import the value for the variables listed in this resource, thereby replacing the text “Search:” with “Hledat:” when the portlet is rendered.

portlet.view.description=Tenhle portlet je v souladu s JSR=168 a používá WSRP, SOAP messaging a XSLT spolu s Google Web API k provádění webového vyhledávání. Výsledky vyhledávání jsou zobrazeny na základě rolí a přístupových práv jednotlivých uživatelů. Volby je možné měnit v edit mode. Tyto změny jsou možné uskutečnit zvolením edit tlačítka pro tento portlet. ... portlet.view.form.textbox.label=Hledat:
Appendix A: Google Search API Overview

The following is a summary from the Google API Overview found on their website:


The API itself can be downloaded at:

http://www.google.com/apis/download.html

A Google Search request, or Operation, submits a query string ('q'), along with a set of parameters, to the API. The parameters include the Google Key for authentication, maximum results, language restrictions, encoding and other such filters. The request is submitted via a SOAP request/response pair. A sample SOAP pair taken from the Google API files:

- **doGoogleSearch** is a query for the string "shrdlu winograd maclisp teletype"
- **doGoogleSearchResponse** returns three search hits

uses the following XML:

**Request XML** (note that the key is false and 'q' is the actual query string):

```xml
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doGoogleSearch xmlns:ns1="urn:GoogleSearch" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <key xsi:type="xsd:string">00000000000000000000000000000000</key>
      <q xsi:type="xsd:string">shrdlu winograd maclisp teletype</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
      <filter xsi:type="xsd:boolean">true</filter>
      <restrict xsi:type="xsd:string"></restrict>
      <safeSearch xsi:type="xsd:boolean">false</safeSearch>
      <lr xsi:type="xsd:string"></lr>
      <ie xsi:type="xsd:string">latin1</ie>
      <oe xsi:type="xsd:string">latin1</oe>
    </ns1:doGoogleSearch>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Response XML** (note: 'item' is the root for the actual list of results that users see):

```xml
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
      <return xsi:type="ns1:GoogleSearchResult">

```
<documentFiltering xsi:type="xsd:boolean">false</documentFiltering>
<estimatedTotalResultsCount xsi:type="xsd:int">3</estimatedTotalResultsCount>
<directoryCategories xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/" xsi:type="ns2:Array" ns2:arrayType="ns1:DirectoryCategory[0]"></directoryCategories>
<searchTime xsi:type="xsd:double">0.194871</searchTime>
<resultElements xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/" xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[3]">
  <item xsi:type="ns1:ResultElement">
    <cachedSize xsi:type="xsd:string">12k</cachedSize>
    <hostName xsi:type="xsd:string"></hostName>
    <snippet xsi:type="xsd:string">... on a simple dialog (via teletype) with a user, about a ...</snippet>
    <directoryCategory xsi:type="ns1:DirectoryCategory">
      <specialEncoding xsi:type="xsd:string"></specialEncoding>
      <fullViewableName xsi:type="xsd:string"></fullViewableName>
    </directoryCategory>
    <relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>
    <directoryTitle xsi:type="xsd:string"></directoryTitle>
    <summary xsi:type="xsd:string"></summary>
    <title xsi:type="xsd:string">SHRDLU</title>
  </item>
  <item xsi:type="ns1:ResultElement">
    <cachedSize xsi:type="xsd:string">12k</cachedSize>
    <hostName xsi:type="xsd:string"></hostName>
    <snippet xsi:type="xsd:string">... on a simple dialog (via teletype) with a user, about a ...</snippet>
    <directoryCategory xsi:type="ns1:DirectoryCategory">
      <specialEncoding xsi:type="xsd:string"></specialEncoding>
      <fullViewableName xsi:type="xsd:string"></fullViewableName>
    </directoryCategory>
    <relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>
    <directoryTitle xsi:type="xsd:string"></directoryTitle>
    <summary xsi:type="xsd:string"></summary>
    <title xsi:type="xsd:string">SHRDLU</title>
  </item>
  <item xsi:type="ns1:ResultElement">
    <cachedSize xsi:type="xsd:string">32k</cachedSize>
    <hostName xsi:type="xsd:string"></hostName>
    <snippet xsi:type="xsd:string">... man and woman through teletype and has to ... human diseases) 1970* Terry Winograd's SHRDLU (Natural Language Processing Lisp Machine Lisp, MacLisp, NIL, S-1 ...</snippet>
    <directoryCategory xsi:type="ns1:DirectoryCategory">
      <specialEncoding xsi:type="xsd:string"></specialEncoding>
      <fullViewableName xsi:type="xsd:string"></fullViewableName>
    </directoryCategory>
    <relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>
    <directoryTitle xsi:type="xsd:string"></directoryTitle>
    <summary xsi:type="xsd:string"></summary>
    <URL xsi:type="xsd:string">http://hci.stanford.edu/cs147/examples/shrdlu/</URL>
    <title xsi:type="xsd:string">SHRDLU</title>
  </item>
</resultElements>
The Google API will also accept two other Operations, Cached Page and Spelling Suggestion, that are handled in a similar way. A Cached Page Operation will accept a URL as the input and return the most recent version of that page Cached in the Google crawler. The Spelling Suggestion Operation receives the same input as a Search Operation, but returns any available suggestion for spelling along with the search results.

The important factor in any Google Operations is that a request is made using SOAP request response pairs in the form of standardized XML.
Appendix B: Help Mode Input XHTML

The help mode of the CREE Google Portlet uses an XHTML document as its input. The help.xsl document then extracts the content from this document to build the help pages and menu index. All of the tags in the help input xml are standard XHTML 1.0 tags. No special tags were created. Instead, a specific structure is used to discriminate the sections of the document. This method was chosen for the ease of creating new help input documents. As long as someone adheres to the structure, any standard XHTML editor, such as Dreamweaver, can be used to write a new help input file. Once a new file has been written and placed in the appropriate directory, the portlet will use this document to build the new menu and pages the next time the help mode is used.

Below is the entire XHTML document used for the CREE Google Portlet help mode input. Note that the nested <div> tags with different “id” attributes are used to differentiate the parts of the document. That is the document “structure” referred to above. The XSL uses those attributes and the position of the node to determine which is the correct text to use. There are no other restrictions to the content of the document. Any, well-formed XHTML fragments can be used as page content.

CREE Google Portlet Help Mode Input Document

<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="text/html; charset=ISO-8859-1" http-equiv="content-type"/>
<title>portlet.help.xhtml</title>
</head>
<body>
<div id="helpChapter">
<div id="chapterName">
<p>New Search</p>
</div>
<div id="helpPage">
<div id="pageName">
<p>Search Basics</p>
</div>
<div id="pageContent">
<p>How you choose the Keyword(s) for your search will effect the results that you get. Below are a few tips that should make it easier for you to find what you are looking for in your search:</p>
<ul>
<li>Simple searches are the best. Don't get too fancy trying to second guess the search engine.</li>
<li>Search for what you really want to find. If you want to find Mozart's birthhome, search for &quot;Mozart birthhome,&quot; and not just &quot;Mozart.&quot;</li>
<li>Many words are common. Try to combine two or more Keywords that will give you a search with fewer results. &quot;Mozart&quot; will give you links for everyone with the name Mozart who appears somewhere on the web. Narrow it down with something like &quot;Mozart composer.&quot; The more specific you can be, the better your results will be.</li>
</ul>
</div>
</div>
</body>
</html>
For further search techniques, try the Google help page.

The Google API Key is for programmers to gain permission to use the Google database in applications such as this portlet. A Google Key was obtained when the portlet was first written, and a new one is not needed when using the portlet for searching. The Google API is free for developers to use in non-commercial projects. For more information, see the Google API page.

The Locale setting determines what language you would like to use with the Google portlet. The locale setting will change all of the text in the portlet interface. This setting does not have any effect on content that is brought into the portlet from outside, such as the results of a search. Google does have the ability to return results in a variety of languages, but that setting is independent of the portlet Locale setting.

Google will display from 1-10 results at a time. This drop down menu lets you choose any number in that range. Note that this is the maximum number of results on a page. If your search does not return the maximum number of results, than you will only see what is available. Any change to this will be updated instantly.
Appendix C: XLIFF Option for Internationalization

XLIFF is an XML dialect that has been designed specifically to assist in translating text from one language into another. The text that is used in a portlet, and the language that it has been written in, is part of the interface for the portlet. The XSLT step of rendering a portlet is where the interface is determined and created. By using XLIFF in conjunction with the XSL stylesheet, the language copy is dealt with at the proper stage and does not interfere with the java business logic. The original version of the Google portlet handles language in the java, because XLIFF has not yet been adopted by the uPortal community as the standard for language translation. It is our hope that this portlet will show that XLIFF is a very easy way to perform language translations in JSR-168 portlets.

The text copy of an XLIFF document is written in pairs, the first being the source, or original, language, and the second as the target language for translation. An English/Czech source/target pair from the Google Portlet XLIFF document is shown below:

```xml
<trans-unit id="7" retype="text" resname="portlet.view.message.noTitle">
  <source xml:lang="en">Did you mean:</source>
  <target xml:lang="cs">Myslel jsi:</target>
</trans-unit>
```

The complete XLIFF document, then, would be a series of these <trans-unit> pairs of source and target text, along with the necessary XML and XLIFF headings. The English/Czech XLIFF document can be seen in its entirety at the end of this appendix.

When using XLIFF to translate the text in a portlet, the translation is done completely by the XSL stylesheet and independent of the java business logic. A separate XLIFF document, “gportletXLIFF-xx_XX.xml,” is made for each language that is to be available to the portlet. The XSL stylesheet then imports the proper document as a node set, and places the text as the output XHTML is written. Making a new language for the portlet merely requires translating a new copy of the XLIFF document, naming the document correctly, placing it in the appropriate directory and adding the language to the drop down box of the edit mode.

---

4 The first version of the CREE Google Portlet translates all of the text for the portlet through the java business logic, as resource bundles. The portlet has a “gportlet_xx.properties” file for each language. These files match the variable names with the proper text for that language. All of the text is then passed to the XSL stylesheet as parameters.

5 xx_XX is a combination of the two letter language code from ISO 639 and the two letter country code from ISO 3166. Thus, en_GB refers to English as it is spoken in Great Britain.
This is a JSR-168 compliant portlet that uses WSRP, SOAP messaging and XSL-T along with the Google Web API to perform web based searches. It has a role based output that will show different search results dependant upon the permissions of the user. It also contains an edit mode for changing the preferences. The edit mode is accessed by selecting the 'edit' button for this portlet.

No web pages containing your search terms were found.

Your search - did not match any documents.

Check the spelling of the words used in your search, consider different keywords, or fewer keywords. Additional information about the use of this portlet can be found by clicking on the help icon.
This Google Web APIs service will allow you to obtain Google search results in a portlet window. It is made available to you for your personal, non-commercial use only. You may only create a single account and must provide accurate identification, contact, and other information required as part of the registration process. You may not use the search results provided by the Google Web APIs service with an existing product or service that competes with products or services offered by Google. To obtain a Key please visit.


48