uPortal 3.0 WSRP Producer Development Project

The Need

The implementation of the Web Services Remote Portlet standard permits an organization's portal to display a WSRP portlet from a remote site as if it were running locally.

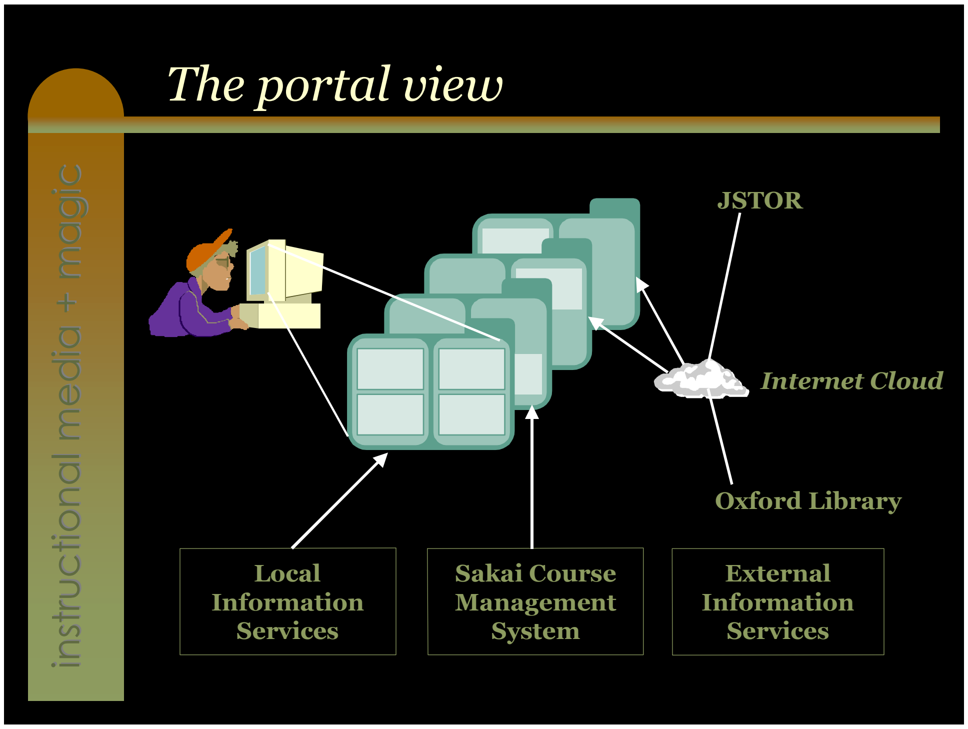
The portal view
instructional media + magic
JSTOR
Internet Cloud
Oxford Library
Local Information Services
Sakai Course Management System
External Information Services

Figure 1 – The Portal View

Figure 1, adapted from a uPortal presentation, shows a student seeing four windows on his computer screen. One, the lower left hand corner, is from the university's computer center and could be a typical administrative application. For example, a list of current course enrollments and their schedules. The Sakai Course Management System includes a WSRP producer. The user's portal must have a WSRP consumer in order to use the Sakai Course Management System.[1] Two other examples are WSRP consumer portlets connected via Web Services and the Internet displayed in the two other windows shown in Figure 1. The local portal shown in Figure 1 must have WSRP consumer installed to display these WSRP "feeds." Each of the WSRP data sources, the two libraries and the

[1] There is a standalone system, but it has limited integration with other systems. Likely this version will disappear sometime in the future.

Sakai course management system need an application that produces WSRP or have WSRP producer that creates a WSRP data stream from an installed portlet.[2]

The most frequent implementation of WSRP producer will be in conjunction with a JSR 168 compliant portal.[3]

This project will implement WSRP producer in uPortal so the current installations of uPortal can support remote WSRP installations. An example of how this would be used in a proposed Miami-Dade Community College District implementation of uPortal shown in Figure 2.

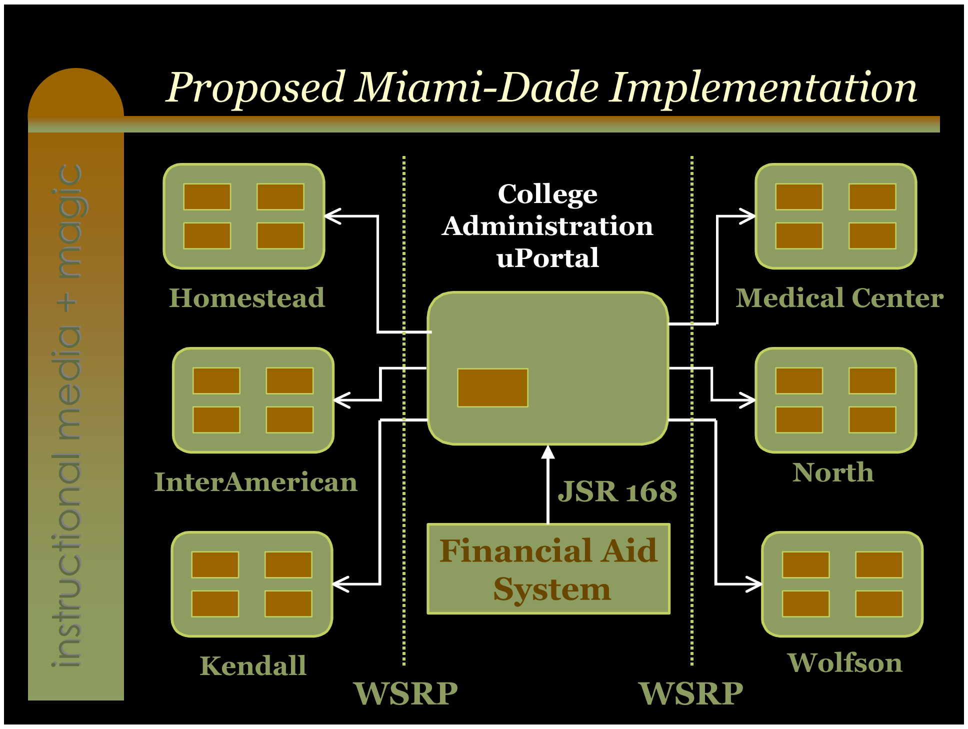
Proposed Miami-Dade Implementation
instructional media + magic
College Administration uPortal
Homestead
Medical Center
InterAmerican
North
JSR 168
Financial Aid System
Kendall
Wolfson
WSRP
WSRP

Figure 2 – uPortal and WSRP

In this example the Financial Aid System is a JSR 168 compliant portlet installed on uPortal at the College administrative computer center. This portlet can be seen by authorized users of the College portal.

The WSRP producer in the College Administration uPortal installation produces the WSRP Web services messages that permit authorized users at each of the six campuses to

[2] Sakai uses WSRP producer code from the Apache Software Foundation's WSRP4J project. uPortal's WSRP portlet type also is based on WSRP4J code.

[3] JSR 168 is a specification of the Java Community Process used to develop a standard portlet application in Java that can be installed, without modification, in any JSR 168 compliant portal. Oracle, IBM, Sun, PeopleSoft (now Oracle), uPortal version 2.4 or later, and the next version of SCT's Luminus product are all now JSR 168 compliant. Plumtree and Vignette also were early supporters of the JSR specification. There are still some implementation differences that are being resolved with experience.

see the financial aid portlet on the campus portal exactly as it would be seen on the college administrative portal.

WSRP and uPortal 3.0

WSRP Consumer has been implemented in uPortal 2.4. WSRP producer will be implemented in uPortal version 3.0. Based on experience with uPortal 2.x., there are three major enhancements and refactoring that are needed. The new architecture will be designed to reduce maintenance costs and enhance performance. This is possible as the standards and uses of portals have become more stable. The new architecture is flexible designed to support the way that users author, control, and display content. And, with the number of users increasing in both earlier and recent implementations, a focus on performance improvement and administration.

Portals began as an aggregated display of data from several sources. This was generally achieved using complex Web programming. Portals made this aggregation less difficult and more stable during changes in content and sources. Then portals became the presentation for administrative systems. The displays were used to cross boundaries between administrative systems so the display matched the business process. Now the implementations are beginning to serve faculty that control the content being “pushed” to students and control communication between faculty and students. This requires capabilities that do not now exist in commercial portals; businesses tend to centralize rather than delegate control of content and presentation that is needed for faculty to be responsive to student needs.

uPortal 3.0 will also introduce “services”—an architecture that makes functions independent of the portal itself, yet are used by portlets. Sakai services—based on the MIT Open Knowledge Initiative Interface Service Definitions and the IMS Global Learning Consortium’s Tool Interoperability Work Group scheduled to be complete “early 2005.” These services may be needed to supplement some WSRP and JSR 168 portlet developments. (They are required for the WSRP producer used in Sakai. The proposed implementation of WSRP producer is consistent with and supplements the Sakai and uPortal integration).

The Work

There are eight subtasks in developing and unit testing the code for WSRP producer in uPortal 3.0. They begin with interfaces to the database management system used by uPortal. Then the Java Data Objects must be designed and programmed. The WSRP4J code from the Apache Foundation has some code specific to their demonstration portal. This will be replaced with more specific and robust code and integrated with uPortal code. Currently uPortal uses the Pluto 1.0 portlet container; WSRP producer will be integrated with Pluto 1.0.[4] The WSRP producer will also have to be integrated with the

[4] The authors of Pluto 1.1 have advised the uPortal development team not to use Pluto 1.1 since development has just begun. The code is both incomplete and likely to change during development.

uPortal 3.0 portlet and windows registries. (Registries are being introduced in uPortal version 3.0). uPortal's URL interpretation will have to be extended to accommodate the WSRP Producer attributes.

Throughout this process, the work will be unit tested using JUnit. Further implementation testing involving multiple portals, WSRP producer and consumer, and networking, will be done in Task 9.

For configuration, ANT Command Line is proposed as Task 8. A graphical user interface could be developed and incorporated into portal administration. This further development is estimated as Tasks 13 and 14. This should be done only if a need is established for this additional capability.

	Task	Person-days
	Programming	
1.	Abstract persistence layer for the WSRP producer	5
2.	JDO implementation of the WSRP persistence	10
3.	Integration with the uPortal 3.0 domain object model	8
4.	Implementation of Pluto 1.0 providers/factories	15
5.	Integration with uP3.0's portlet entity/window registries	7
6.	Integration of uPortal's URL	5
7.	Test cases with JUnit	5
8.	Ant-based command line utility to manage the WSRP producer registry	10
	Total programming	65
	Testing	
9.	Configure an implementation and test using CREE JSR 168 portlets	5
10.	Install and test in the CREE Project environment	5
	Total implementation testing	10
	Documentation	
11.	Installation and User's Guide	10
12.	Programming Documentation	12
	Total documentation	22
	Further Development	
13.	Graphical user interface for managing the WSRP producer registry (based on the command line version from Task 8).	15
14.	Additional documentation of the Graphical user interface to be incorporated into the Installation and User's Guide.	5
	Total further development	20
	All projects total	117

Table 1 – Project Tasks and Work Estimates

Consistent with this advice, the uPortal team did some early work and found Pluto 1.0 was more complete and stable.

The tasks listed in Table 1 exceed the scope of the current WSRP project—Task 1 through 8. The additional information is provided to plan for future enhancements that are projected to be useful in the future. However, experience with WSRP may suggest other alternatives or change in the scope of the tasks.

Work Assignments

The work is done under the direction of the University of Hull. Cris Aware would be assigned as project manager.

Tasks 1 through 8 would be assigned to the current uPortal developers. im+m's Michael Ivanov would be the lead developer; some tasks may be assigned to others. Unicon's Peter Kharchenko is the chief architect for version 3.0; all of the work would be done consistent with his direction.

Schedule

The work for Tasks 1 through 3 will be completed by the uPortal Developers' Meeting March 17th. Tasks 4 through 6 and some of 7 will completed within two weeks of the Developers' Meeting; The work for Tasks 1 through 8 should be completed by the end of April or early May.

Appendix
Cost Estimates

Cost Estimates

The estimates in Table 1 represent recent experience with JSR 169, Pluto, and uPortal 3.0 development. In general the early work has taken additional time because of the new technology. The work here may be done in less time than shown in Table 1 because of the "learning" that has taken place.

The Table includes broader testing that may be useful to other projects. Typically the JA-SIG developers will test new versions and provide corrections. Similarly an estimate is given for product-level documentation should that be needed I the future. And, as uPortal administration turns to graphical interfaces, the possibility of such an interface after there has been sufficient experience to determine requirements is also included.

	Task	Person-days	Estimated Cost in GBP
	Programming		
1.	Abstract persistence layer for the WSRP producer	5	£2,000
2.	JDO implementation of the WSRP persistence	10	4,000
3.	Integration with the uPortal 3.0 domain object model	8	3,200
4.	Implementation of Pluto 1.0 providers/factories	15	6,000
5.	Integration with uP3.0's portlet entity/window registries	7	2,800
6.	Integration of uPortal's URL	5	2,000
7.	Test cases with JUnit	5	2,000
8.	Ant-based command line utility to manage the WSRP producer registry	10	4,000
	Total programming project	65	£26,000
	Contributed services[5]	25	10,000
	Net cost estimate	40	£16,000

[5] The estimates are based on recent version 3.0 experience and exceeds the early estimate. The contributed service includes some work that has been done on the tasks before this estimate was made. The net effort is consistent with the early estimate. Because of the Learning Curve associated with new technology, it is likely the final results will be more similar to the early estimates because of the recent JSR 168, WSRP consumer, and Pluto experience.