

Web Portal Research with uPortal

CS 526 Project Report

Spring 2005

Lee, Austin

Narayan, Sujeeth

Viswanathan, Arun

<u>1 INTRODUCTION</u>	4
<u>1.1 MOTIVATION</u>	4
<u>1.2 PORTAL INTRODUCTION</u>	4
<i>Categorization of Content</i>	5
<i>Content Search & Indexing</i>	5
<i>Content Management</i>	6
<i>Personalization</i>	6
<i>Application Integration</i>	6
<i>Development Tools</i>	6
<i>Mobile & Wireless Support</i>	6
<i>Single Sign-On</i>	7
<i>Security</i>	7
<u>1.3 AVAILABLE PORTALS</u>	7
<u>1.4 PORTALS VS. CUSTOMIZABLE HOME PAGES</u>	8
<u>2 UPORTAL</u>	9
<u>2.1 UPORTAL INTRODUCTION</u>	9
<u>2.2 UPORTAL STRENGTHS AND WEAKNESSES</u>	10
<u>2.3 UPORTAL EXTENSION: SAKAI PROJECT</u>	11
<u>3 UPORTAL ARCHITECTURE</u>	12
<u>3.1 WEB BROWSER REQUIREMENTS</u>	12
<u>3.2 SERVER PLATFORMS</u>	12
<u>3.3 SERVER SOFTWARE REQUIREMENTS</u>	13
<u>3.4 JAVA 2 ENTERPRISE EDITION (J2EE)</u>	13
<u>4 UPORTAL INTERFACE</u>	13
<u>4.1 GUI</u>	14
<u>5 CHANNELS</u>	15
<u>5.1 CHANNEL LIFECYCLE</u>	15
<u>5.2 CHANNEL TYPES</u>	16
<u>6 CHANNEL MANAGER</u>	17
<u>7 CHANNEL SUBSCRIPTION</u>	19
<u>8 PROGRAMMING A CHANNEL</u>	20
<u>8.1 CUSTOM CHANNELS</u>	20
<u>8.2 PORTLETS</u>	21
<u>9 PERFORMANCE ANALYSIS</u>	21
<u>9.1 WEB SERVER PERFORMANCE ANALYSIS BY SUJEETH</u>	21
<u>9.2 UPORTAL PERFORMANCE ANALYSIS BY ARUN</u>	24
<u>10 EXPLORATIONS WITH UPORTAL BY ARUN</u>	27
<u>10.1 CAMPUS NEWS CHANNEL</u>	28
<u>10.2 CUTODAY IMPLEMENTATION</u>	28
<u>10.3 USAGE-TRACKING FUNCTIONALITY</u>	31
<u>11 EXPLORATIONS WITH UPORTAL BY AUSTIN</u>	31

<u>11.1 MYSQL INTEGRATION</u>	32
<u>11.2 PAYPAL INTEGRATION</u>	33
<u>12 EXPLORATIONS WITH UPORTAL BY SUJEETH</u>	35
<u>12.1 CLASSIFIEDS CHANNEL DESCRIPTION</u>	35
<u>12.2 CHANNEL DESIGN</u>	35
<u>12.3 CURRENT IMPLEMENTATION</u>	36
<u>13 REFERENCES</u>	38

1 Introduction

1.1 Motivation

More companies and organizations today are aware of Portal technology, and are implementing portals on their web sites. Portal sites typically have a larger team size behind them, where each member or group maintains a certain channel, and a management group that oversees the entire structure. As a result, the demand for experience with portal technology in the job market is growing.

Currently, University of Colorado at Boulder has uPortal implemented on their web site, with different access levels set up for students, faculty, developers, administration, and so forth. University of Colorado at Colorado Springs plans to implement uPortal and have it deployed by Fall semester of 2005.

1.2 Portal Introduction

Portals wrap an organization's documents and applications in a single web interface that provides distributed access, cross-platform usability, personalization, management, and security features.

In today's world, Portals form the core of any digital nervous system of an organization. They could also serve as home page or normal website to any general user and also provide the specific, advanced services to its members.

There are many **Classes** of portals:

- Information Portals
- Functional Portals
- Industry Specific Portals
- Enterprise Portals

Portal **Features** could be listed as below:

- Categorization of Content
- Content Search & Indexing
- Content management & aggregation
- Personalization
- Robust application integration
- Development tools
- Mobile/wireless support
- Single Sign-On
- Security

Categorization of Content

A portal would allow the organization to organize content and applications in different ways in order to meet the needs of various groups within the organization.

Content Search & Indexing

A portal should provide or integrate with document index and search systems so that users can quickly navigate to the information they need.

Content Management

A portal should provide content authoring systems that allow non-technical staff to create content. It should control access to content to allow only authorized users access to document repositories.

Personalization

A portal should display different views of organizational data based upon user groups. Individual users should be able to further customize their view to only display the content they use most often.

Application Integration

A portal should provide a wrapper around existing web-based applications. Aspects of integration include support for single sign-on, inclusion of external web resources, support for web services, and portal preferences that carry over into the integrated application.

Development Tools

A portal product should provide robust, standards-based development tools that allow IT staff to integrate applications and extend portal functionality.

Mobile & Wireless Support

A portal should provide cross-platform functionality that allow users of different operating systems and web browsers to access the portal. This includes support for mobile phones and wireless PDAs.

Single Sign-On

A portal should integrate with or provide a single sign-on system. In other words, a portal should pull user information from a directory server such as LDAP, NDS, or Active Directory.

Security

A portal should provide robust authentication and authorization systems. Any integration with a single sign-on system should be secure and prevent the unencrypted transmission of user credentials across application domains.

1.3 Available Portals

There are many portal frameworks available. There are also companies customizing the portal for the organization needs on commercial basis. Fig 1.1 illustrates market share of each of the portals listed below.

Some of the **Open Source** Portals with **Java**:

- eXo – by eXo
- JetSpeed – by Apache
- Liferay Portal –by Liferay
- uPortal – by JASIG

Some **Open Source** Portals with **Microsoft** Technologies:

- DotNetNuke
- Rainbow

Others:

- Microsoft Share Point
- Oracle Portal
- Peoplesoft Portal
- Many others

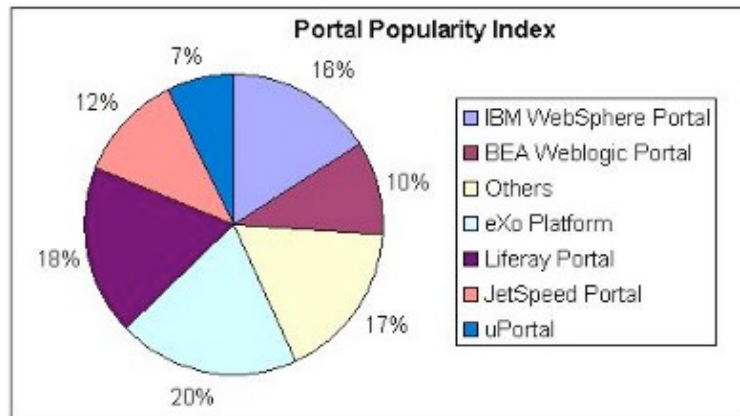


Fig 1.1: Portal Survey by: <http://portlets.blogspot.com>

1.4 Portals vs. Customizable Home Pages

Portals help in customizing the ‘look and feel’ for individual login similar to customizing home pages provided by Netscape or Yahoo. The difference lies in the content management by the organization.

The organization providing the portal service will be able to push data to the users. It also enables them aggregate data or results of several users. Each user data could be tracked in a portal.

Example Scenarios:

uPortal: Portal could be designed for publishing grades of individual students who have subscribed.

DotNetNuke: A forum could be created for all the subscribers where they could post on different threads.

JetSpeed: A special announcement could be pushed to every subscriber of the portal service.

2 uPortal

2.1 uPortal Introduction

uPortal was developed under JA-SIG (Java in Administration Special Interest Group) by institutions of higher-education including Princeton, Yale, and University of Delaware. It is a framework for building custom portals rather than itself being a portal. It enables universities to rapidly incorporate their web-based content into a single point-of-presence. uPortal provides the ability for universities to integrate web-based applications through an open Java framework built on accepted web standards.

uPortal categorizes content into **channels**. Each channel is a service provided to the user. It has over 130 implementing organizations with approximately 12 developer institutions working in tandem. There are hundreds of thousands of users in least 13 countries and also 7 commercial partners.

As of the time this document was written, the current release version of uPortal is 2.4.2, with a version 3.0 milestone 1 release candidate available.

2.2 uPortal Strengths and Weaknesses

The major strengths of uPortal are:

- It is open source and hence **Free**.
- Default user **interface** is very **intuitive**
- Strong support for **industry standard web** and **distributed application technologies** (J2EE, XML, XSL)
- **Built-in** support for **RDS/RSS** channels
- Open source code allows more options for customization and integration
- Support for **multiple databases**, application servers, and web servers
- User interface is very **easy to customize**
- Expert developer community (you can talk directly to the original developers)
- Vendor independent single sign-on
- Quick personalization by user or group
- Can be wrapped in **SSL** (with a web-proxy)
- Can be made to support mobile and wireless browsers as well as new desktop browsers
- Is already in use at many universities

There are also weaknesses that uPortal has.

- Open source products don't have the same support options as commercial software (though one consulting firm close to uPortal offers support plan)
- Developers require a very high level of expertise in Java, XML, XSL, and HTML

- Redundancy, failover and backup capabilities are not integrated with uPortal (but can be configured by a DBA and sysadmin)

2.3 uPortal Extension: SAKAI Project

SAKAI was another program from the joint effort of universities and other organizations. The group, which made this program, includes: University of Michigan, MIT, Stanford, Indiana University, O.K.I (Open Knowledge Initiative), JASIG and supported by Mellon Foundations.

SAKAI is intended to deliver open source Course Management System and research collaboration modules that are completely portable and interchangeable in web and non-web Java environments.

SAKAI portal has the following deliverables list:

- Build an Enterprise Services-based Portal
- A complete Course Management System with assessment tools
- Research Support Collaboration System
- Tool Portability Profile Tool, which has clear standard for writing future tools that can extend this core, set of educational applications.

The current version at the time of writing for SAKAI is 1.5. In this release it has added Syllabus Management Tool. The planned completion of this project as development is December 2005 and thereafter to convert it from project into community.

SAKAI has a special program called SAKAI Educational Partners Program (SEPP). Any organization could join as member and be part of the SAKAI project with

early access to many aspects of the project like the roadmap and developers and exchange tools.

3 uPortal Architecture

uPortal consists of five components: a web browser, web server, Java servlet container, ODBC database, and external web resources. A user needs only a compatible web browser to connect to a web site that has uPortal installed.

3.1 Web Browser Requirements

According to the documentation, the following web browsers are compatible:

- Netscape 4 and above
- Internet Explorer 4 and above
- Mozilla 5 and above

Although not documented, Safari for Mac OS X also works with uPortal.

3.2 Server Platforms

The following operating systems support the installation of uPortal:

- Microsoft Windows
- Solaris
- Linux
- Mac OS X

3.3 Server Software Requirements

In order for uPortal to run on a server, the following software packages must also be installed prior to, or concurrently installed:

- Sun JDK 1.3 or above
- Ant 1.5.3 or above
- Servlet Container/Engine compatible with Servlet 2.2 and JSP 1.1 API
- JDBC compliant database

The quick-start installation of uPortal is packaged with Ant, Tomcat 3.3.2, and HypersonicSQL.

3.4 Java 2 Enterprise Edition (J2EE)

uPortal's architecture can take advantage of the J2EE services provided by the Servlet Container. Java Messaging Services (JMS) and Java Transaction API (JTA) can be used, as available. As a result, applets, servlets, and JSP pages can be used within uPortal.

4 uPortal Interface

Once uPortal has been installed, and its database and web server are running, it is managed entirely through a web browser. By default, there are six accounts preloaded: demo, student, staff, faculty, developer, admin. Only the developer and admin users are allowed to manage user accounts and implement new channels.

4.1 GUI

In the top-right corner of the browser window, every user has a row of buttons, as shown in Fig 4.1.



Fig 4.1 Administration buttons

The house button takes a user to the home page for that user.

The button to the right displays a site map.

The button with a gear takes a privileged user (developer) to the channel creation wizard.

The button with the wrench customizes the portal interface.

Finally, the button with the door logs the user out of the system.

Fig 4.2 illustrates a layout of the uPortal interface. Sections are divided by tabs (shown as Applications) which the user can select. In each tabbed section, there are one or more columns of content. Within each column, there can be multiple channels.

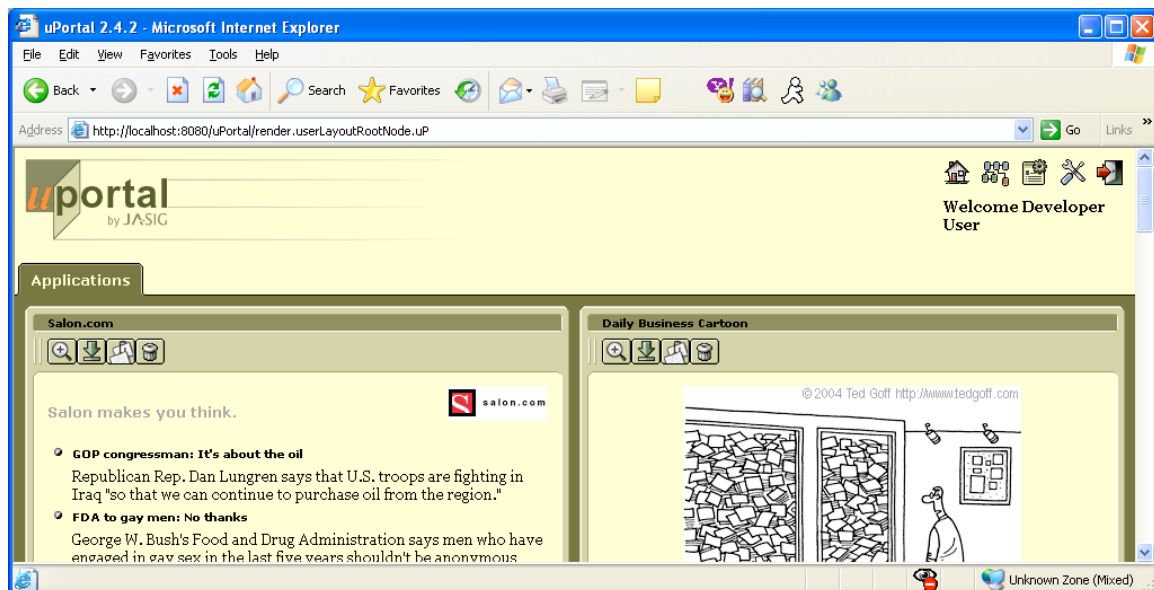


Fig 4.2 uPortal layout

5 Channels

The channel is the most basic source of information in uPortal. It coordinates all related information and provides a framework to display the different contents of a channel in an organized way. Channels can only be created and/or modified by authorized users with administrative privileges (channel admin or developer, for instance). Other non-admin users can subscribe to channels of their choice and add them to their personalized layout.

5.1 Channel Lifecycle

Channels are Java objects that implement the `org.jasig.portal.IChannel` interface. There are several phases in the lifecycle of a channel, the details of which are briefly described below:

1. *Creation*: The creation of a channel involves the definition of a Java class for the channel, specification of its various parameters, description of the purpose of the channel and outlining a workflow for the different steps involved in publishing and subscribing to the channel. All of these details are mentioned in a Channel Publishing Document (CPD file).
2. *Registration*: After a new channel is created, the uPortal system needs to be made aware of its presence. This process is called channel registration, at this stage the new channel is assigned a `channelTypeId`.
3. *Publishing*: A newly created and registered channel needs to be made available for

users to subscribe to it and view it. This process, known as channel publishing, involves the configuration of the channel. The configured channel is then assigned a `channelPublishId` and placed in the taxonomy of the various channel categories.

4. *Subscription*: This is the final stage in which a user can subscribe to the newly created channel. By subscribing to the channel, what the user essentially does is that he adds the selected channel to his personalized layout. Internally, when the user subscribes to a channel, the system assigns a `channelSubscribeId` to that channel which is unique to each user's layout.

5.2 Channel Types

Channels are categorized into various types based on the type of content they are meant to hold. uPortal supports a number of channel types, as listed below:

1. *Applet*: Renders a Java applet in a uPortal channel.
2. *Image*: This channel provides a container to render images (JPEGs, GIFs, PNGs).
Though named an image channel, it supports more than just images. It can be used to render any media that can be played within a browser, such as audio, video, graphics or flash objects.
3. *Inline Frame*: The inline frame channel type is similar to the image channel, in that it also provides a container or a frame within uPortal, but is limited to displaying HTML content alone.
4. *Portlet*: The portlet channel provides an adapter to support JSR-168 portlets. Java Specification Request (JSR) 168 is a specification for Java portlets that provides standards to enable interoperability between portals and portlets.
5. *RSS*: This channel renders content conforming to the popular Rich Site Summary

(RSS) format. This can be used to create news channels.

6. *Web Proxy*: This channel, like Inline Frames, is also capable of supporting HTML content, but in this case the content needs to be in well-formed XHTML. This channel can also render XML content.
7. *WSRP Consumer*: This channel supports content that conforms to the Oasis Web Services for Remote Portlets (WSRP) standards. This channel is used when third party web services need to be used in conjunction with the existing uPortal framework. In such a channel, certain user input may be taken from the uPortal side and processed by the third party web-service. The result is then returned back to the user via the uPortal framework.
8. *XML Transformation*: This channel renders XML content validated by an XSLT stylesheet specified in a uPortal stylesheet list (SSL) file.
9. *Custom*: This is a channel that is completely customizable. The writer of the channel has to provide all the resources and contents required by this channel.

One type not listed in the channel creation wizard is a *Pushed Fragment*. This is a special type of channel which is forced by the channel creator onto other users. The user may not remove or modify this type of channel unless the creator gives them permissions to do so. Currently, uPortal only supports additions of Pushed Fragments through the command line.

6 Channel Manager

The channel manager is used to publish a new channel or modify the settings of an existing channel. Only those users at the top of the hierarchy, the admin users and

developers, can access this functionality. The process of publishing a new channel (a.k.a. channel workflow) involves several steps. These steps are discussed below:

1. *Channel Type*: The first step in publishing a new channel is to select a channel type for it, this is done based on the content that the new channel is intended for.
2. *General Setting*: This is where the general information on the channel, like the channel title (displayed at the top of the channel), channel name and channel description are specified.
3. *Channel Controls*: Each channel has an option of providing the user with some control. There are 3 possible settings under channel controls:
 1. Editable – the channel can be made editable if the user would have any need to modify any part of the channel or provide any input data to it. Enabling this property would generate an 'Editable' button in the channel being created.
 2. Has Help – if the channel being created is going to be complicated for some users, then it is possible to provide a 'Help' dialog explaining the several functionalities the channel provides.
 3. Has About – this is an additional feature that would provide extra information to the user, like who created the channel, when the channel was created, etc.
4. *Categories*: When a user wishes to subscribe to a new channel, he is provided with a list of categories under which the various channels are enlisted. This part of the channel workflow is where the channel admin specifies under which category the new channel should be listed. uPortal comes with 4 basic categories: Applications, Development, Entertainment, and News.
5. *Groups*: This is the part where user-permissions for the channel being created are

specified. For instance, if there is a channel called “Grades” to be accessed by all students, then only the 'student' login must be allowed to subscribe to it. After this permission is granted to the 'student' login, any student will be able to view the “Grades” channel in the category listing of all channels and subscribe to it, if he wishes to.


6. *Review*: This is the final stage in the channel publishing process. In this stage all the settings of the channel are displayed on the screen to be reviewed by the channel admin. Any modifications can be made, if need be. Or else the channel admin can finalize the settings by clicking on the 'Finished' button after which the channel will be published.


The above steps are the general steps in the workflow for publishing a channel. This workflow may change very slightly depending on the channel type chosen in step 1. For instance, for an RSS channel, there will be an additional step called 'RSS URL and Stylesheet' (after 'General Settings') where the channel admin would specify RSS parameters, like the URL for the XML document, among other settings specific to the RSS channel type. In a similar fashion, each of the channel types would have a slightly different workflow based on the parameters required for configuring that channel.

7 Channel Subscription

Once a channel is published, it will be made available to the users for subscription. Users who are authorized to subscribe to a channel can then view the different channels in their channel categories list and subscribe to any channel that they

like. (NOTE: Users who do not have permission to subscribe to a channel will not be able to view that channel. That is, only channels that are “subscribable” by a user are displayed in that corresponding user's channel categories list).

Channel subscription is a simple procedure which involves the following steps. The user will first need to “Turn on Preferences” by clicking on the  button at the top right of the screen. This will turn user preferences on and display all the possible actions in a row just below the tabs. The user can then create a new tab by clicking on “New Tab”, selecting a position for it among all existing tabs in his layout and giving a suitable name to that tab. After creating a new tab, the user will have to add at least one column in that tab (by using the “New Column” action button). Then, the user can add any channel to that new tab by clicking the “Add Content” action button and subscribing to the channel the user wants. The user also has the option of choosing a skin for his or her layout and a language of his or her choice (among the available choices).

After setting up the new tab, the user MUST save his preferences so that these changes would be in effect for all subsequent logins as well. The user can “Save Changes” by clicking on the  button at the top right corner.

8 Programming a Channel

8.1 Custom Channels

To create a custom channel, it must be written in Java, using the API library. The class name must be prepended with the letter ‘C’, such as CHelloWorld. Also, a stylesheet for the webpage using the same name must be created, such as CHelloWorld.ssl. Finally, the webpages themselves must be created, one for Internet

Explorer browsers, and one for Netscape browsers: `normal_explorer.xml`, and `normal_netscape.xml`. Optionally, pages for the about and edit options can be added as well: `about.xml`, and `edit.xml`.

The custom channel will be compiled when it is added using the Channel Manager.

8.2 Portlets

Portlets are similar to custom channels, but are packaged into one file for easy distribution. In order to create one, the Java source files must be compiled into classes using the Portlet library provided with uPortal (`portlet-api-1.0.1.jar`) and a portlet descriptor file (`portlet.xml`). Then, it must be packaged into a JAR file, then renamed to WAR. The Google portlet is provided with the default uPortal installation is a good example of one.

9 Performance Analysis

9.1 Web Server Performance Analysis by Sujeeth

The performance test was carried out on the following machine configuration:

Windows 2000 Server, Pentium 3 399Mhz, 380MB RAM

The simple benchmark test run result is below in Table 9.1.1.

Server Software:	Apache-Coyote/1.1				
Server Hostname:	localhost				
Server Port:	8080				
Document Path:	/uPortal				
Document Length:	0 bytes				
Concurrency Level:	5				
Time taken for tests:	3.64406 seconds				
Complete requests:	1000				
Failed requests:	0				
Write errors:	0				
Non-2xx responses:	1000				
Total transferred:	159000 bytes				
HTML transferred:	0 bytes				
Requests per second:	326.33 [#/sec] (mean)				
Time per request:	15.32 [ms] (mean)				
Time per request:	3.06 [ms] (mean, across all concurrent requests)				
Transfer rate:	50.58 [Kbytes/sec] received				
Connection Times(ms)					
	min	mean	[+/-sd]	median	max
Connect:	0	0	2.5	0	10
Processing:	0	14	6.0	10	50
Waiting:	0	13	6.1	10	50
Total:	0	14	6.2	10	50

Table 9.1.1 : Apache Benchmark tool run –1

This table shows there are no errors or request failures, which indicates a correctly working server. This run has Concurrency Level 5 which means there were 5 simultaneous requests made to the server. The important results are Requests per second and Time per request. The Connection Times table shows less time in making requests with the reason being the server running at local machine. If the server was opened to the Internet, there would have been delays due to Internet traffic.

Another run was benchmarked with keep-alive session configuration. The results are as below in Table 9.1.2.

Server Software:	Apache-Coyote/1.1				
Server Hostname:	localhost				
Server Port:	8080				
Document Path:	/uPortal				
Document Length:	0 bytes				
Concurrency Level:	5				
Time taken for tests:	2.884148 seconds				
Complete requests:	1000				
Failed requests:	0				
Write errors:	0				
Non-2xx responses:	1000				
Keep-Alive requests:	0				
Total transferred:	159000 bytes				
HTML transferred:	0 bytes				
Requests per second:	346.72 [#/sec] (mean)				
Time per request:	14.421 [ms] (mean)				
Time per request:	2.884 [ms] (mean, across all concurrent requests)				
Transfer rate:	53.74 [Kbytes/sec] received				
Connection Times(ms)					
	min	mean	[+/-sd]	median	max
Connect:	0	1	2.4	0	10
Processing:	0	13	5.2	10	30
Waiting:	0	12	5.4	10	30
Total:	0	13	5.4	10	30

Table 9.1.2 : Apache Benchmark tool run –2

The differences noticed between results are reduced time taken for tests. This is probably because the test generated was easily processed from the previous runs in the buffer. Even though there has not been many keep-alive requests, the overall performance has improved. It was believed that the multiple requests were being sent in the same session by avoiding creating multiple sessions. Also with this run, there has been no errors, indicating that the server was running stable.

If there were tests possible on documents with database request or other documents then there would have been more delays. The delays could be in the processing time by the server and the servlet container installed. This test was not possible as the uPortal specific document pages could not be addressed.

In the overall performance of the server, it could be considered good for organizations with medium size concurrent users and large number of requests with high transfer rate. This server could be performing faster because it was run on the local host. The processing time is good with the consideration of the system hardware.

9.2 uPortal Performance Analysis by Arun

I used the ApacheBench benchmarking tool in order to conduct performance tests on uPortal, as it runs on my platform. I ran the benchmark test a number of times to see if it made any difference. Figures 9.2.1 and 9.2.2 below are screenshots of two of the test results I obtained.


```

Server Software:      Apache-Coyote/1.1
Server Hostname:     localhost
Server Port:        8080

Document Path:       /uPortal/
Document Length:     0 bytes

Concurrency Level:   5
Time taken for tests: 4.634 seconds
Sent requests:       1000
Completed requests:  1000
Failed requests:     0
Total transferred:   317000 bytes
HTML transferred:    0 bytes
Requests per second: 215.796288303841
Transfer rate:       66.8041244065602 kb/s received

Connection Times (ms)
      min      avg      max
Connect:    0      0.10    51
Response:   1     21.34   670
    
```

Fig 9.2.1: Benchmark test, first iteration

Figure 9.2.1 above was the first iteration of the benchmark test I ran and Figure 9.2.2 below was the fourth. All the tests were run with the same settings of number of requests = 1000 and number of concurrent requests = 5. I found that the very first iteration of the test took slightly longer to complete (over 4.6 seconds) compared to all subsequent tests (around 1.7 seconds). There is a considerable difference even in the connection times and response times, requests per second and transfer rates. This discrepancy is probably only because the ApacheBench tool needs to warm up before it gives consistent and precise results, because all subsequent runs (after the first) pretty much gave the same result, similar to the one on Figure 9.2.3 below.

```

Server Software:      Apache-Coyote/1.1
Server Hostname:     localhost
Server Port:         8080

Document Path:       /uPortal/
Document Length:     0 bytes

Concurrency Level:   5
Time taken for tests: 1.731 seconds
Sent requests:       1000
Completed requests: 1000
Failed requests:     0
Total transferred:   317000 bytes
HTML transferred:    0 bytes
Requests per second: 577.700751010976
Transfer rate:       178.839002021953 kb/s received

Connection Times (ms)
      min      avg      max
Connect:    0      0.17      5
Response:   2      8.11     262
    
```

Figure 9.2.2: Benchmark test, fourth iteration

From the readings shown in Figure 9.2.2 above, it is obvious that uPortal was serving requests quite reliably (failed requests = 0). The connection time is very small (0.17ms average) and the response time is not too bad either (over 8ms average). The transfer rates and requests per second were also found to be consistent and at a decent level in all runs (barring the first).

10 Explorations with uPortal by Arun

For my part of the ‘explorations’ with uPortal, I set it up on a Linux laptop. The specifications of my setup are as follows:

- < Operating System: Fedora Core 3, 2.6.11 kernel
- < Processor / Memory: Pentium 4, 256 MB
- < Browser: Mozilla Firefox v1.0.3
- < uPortal: uPortal Quick Start v2.4.2
- < Java: JDK v1.4.2 and v1.5.0

The first thing to do in order to get uPortal up and running on the Linux system (as is with any other), is to set the JAVA_HOME environment variable to the Java root directory and add the java/bin path to the PATH environment variable. Then, the HSQL server and the Tomcat server have to be started in 2 separate console windows. (NOTE: Before using uPortal for the first time, it is essential to start up both the servers as root. For all subsequent times, the servers may be started as any user, not necessarily root). Once both the database server and the Tomcat server are running, you can open up a web browser and point it to <http://localhost:8080/uPortal>. This will bring up uPortal’s default page asking for a login and password. For demonstration purposes, you may log in with both username and password as “demo”.

10.1 Campus News Channel

I implemented a channel called CUToday to serve the purpose of a UCCS info-channel. This channel would provide its subscribers with information on what is happening in and around UCCS, providing current affairs, campus news, announcements (if any), among other things. In addition to campus news, there were 2 other parts to this channel – a weather channel and an online monthly events calendar.

10.2 CUToday Implementation

As mentioned in the previous section, the CUToday channel has 3 parts to it: campus affairs, weather channel and an online events calendar. Strictly speaking with regard to implementation, the CUToday channel is actually a conglomerate of 3 different channels bundled in one. Since all 3 components dealt with daily affairs, I thought it to be logical to put them together in a single channel (or tab) called CUToday. But putting them together or not is totally left to each individual user's discretion.

For the campus news portion of this channel, I used the RSS channel type. I wrote a simple XML file (adhering to the RSS specifications) and passed it in to uPortal as an RSS feed. I listed this channel under “Entertainment” and gave permission to “Everyone” (all users of uPortal) to subscribe to this channel.

The weather channel was also done as an RSS channel, but for this I simply had to pass in an RSS feed from a third party weather information resource. I initially intended to implement this weather channel as a WSRP Consumer type channel. Since all the information on CUToday is with respect to Colorado Springs alone (specifically

UCCS and its vicinity), I saw no reason to have a WSRP channel (to accept user input) and use a third party web service. Moreover, the WSRP channel would take much longer to load, since it would need the user to first enter some information (like zip code) and then it would have to contact the third party web service, retrieve the information and deliver it back to user. All of this would take much longer to render and would slow down the system. Hence, I thought that it would be more efficient to render an RSS channel for weather information too. I placed this channel in the “Applications” category, giving permission to “Everyone” to subscribe to it.

The monthly calendar was implemented as an Inline Frame pointing to the location of UCCS’ online events calendar. This calendar would get updated each month, showing all campus activity that was scheduled to take place that month. Just as in the case of the weather channel, I listed this channel under “Applications” as well and again granted permission to “Everyone” to subscribe to it.

All of the above 3 channels have been aggregated into one single tab, called CUToday, for demonstration purposes. You may view this demo-channel (provided the HSQL and Tomcat servers are still running and assuming nobody has taken the liberty to change the channel) by pointing your browser to <http://s50.csnet.uccs.edu:8080/uPortal> and logging in as a “demo” user.

A screenshot of this channel is also shown in Figure 10.2.1 below.

Web Portal Research with uPortal – Lee, Narayan, Viswanathan

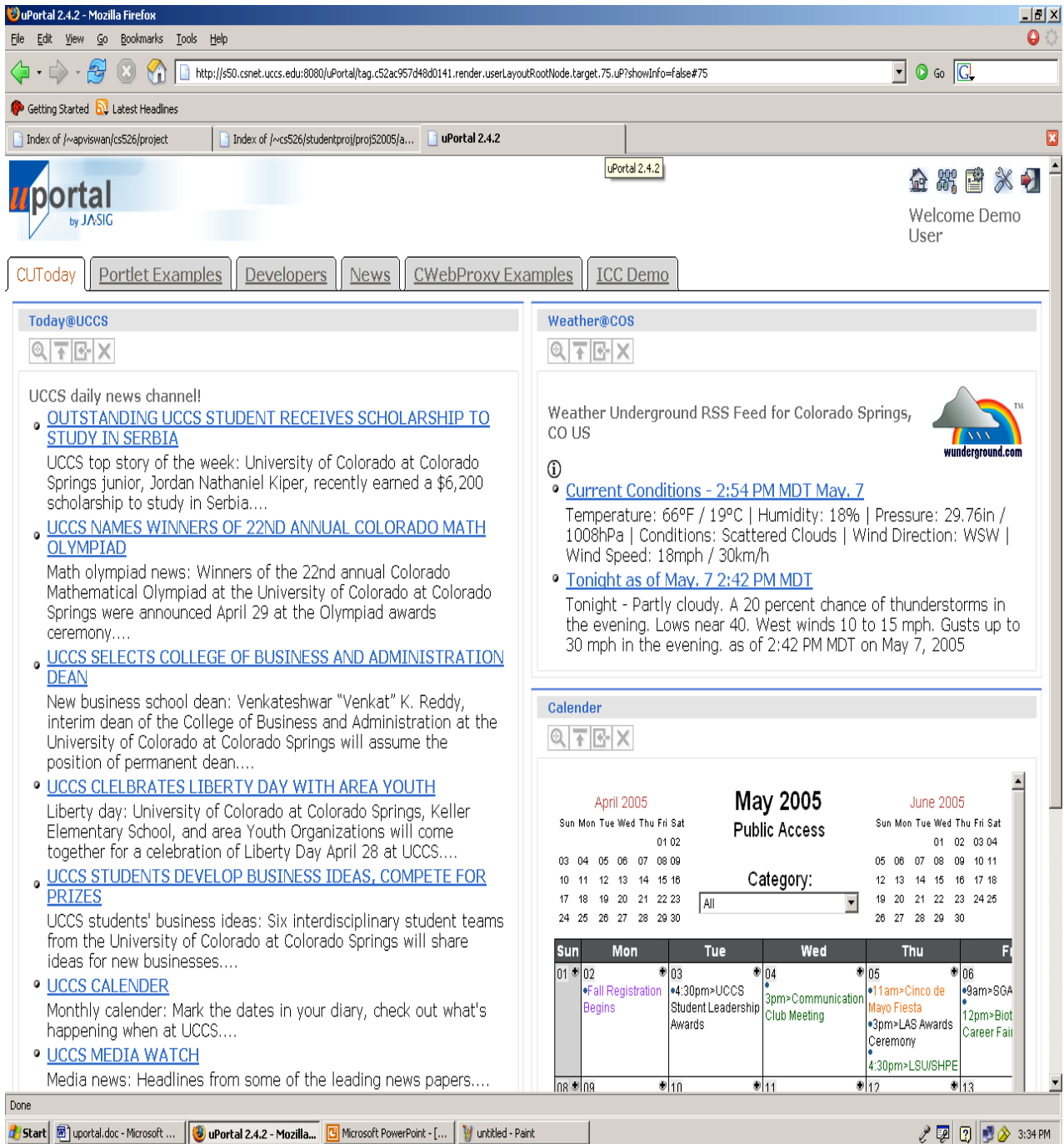


Figure 10.2.1: CUToday channel

10.3 Usage-tracking Functionality

I researched on how to add an extra functionality of tracking the usage of the portal by different users. The intention was to come up with a method to determine who used the portal and what parts of the portal they accessed. There are certain Java classes in the src directory of the HSQLDB root directory called Session.java, SessionInterface.java, User.java and UserManager.java. Session.java implements SessionInterface.java, this handles all session activity, liking opening a session, authenticating it, accessing the respective database and finally closing it. The User.java file creates a User object to hold the name, password, role and access rights for a particular database user. The UserManager.java file manages all users' information. Tracking the type of users who access the portal could be done by adding custom code to the User.java or UserManager.java file. Determining what specific parts of the portal were visited by an individual user is dependent on each user's session. Hence, I believe that the Session.java file should be of help to track this aspect.

11 Explorations with uPortal by Austin

For my contribution to this project, I ran the uPortal 2.4.2-quick start on my Windows XP laptop. The system specifications are as follows:

- Pentium M processor at 1.6GHz
- 512M PC2700 RAM
- 5400RPM Hard drive
- Windows XP Pro SP2
- Java 1.4.2_08

As noted in Arun's exploration, the path for the JDK needed to be set in the JAVA_HOME and PATH environment variables for the system.

11.1 MySQL Integration

Since many organizations already have their own databases in place, it is not efficient to switch to the bundled HypersonicSQL. Moreover, these databases generally perform much faster, on different server machines. MySQL is a free open-source database currently used by many web sites. Documentation on how to integrate MySQL with uPortal is available on uPortal's web site at:

<http://www.uportal.org/administrators/mysql.html>

It is important to note that the document is a bit out of date, and the newer versions of MySQL and the MySQL JDBC connector have different naming schemes.

After verifying that uPortal ran normally on my machine, I downloaded and installed the latest stable release version of MySQL, version 4.1.11. Also, I downloaded and installed the latest JDBC connector, version 3.1.8.

Although not mentioned in the integration documentation, CLASSPATH had to be set to `.;C:\j2sdk1.4.2_08\jre\lib\ext\mysql-connector-java-3.1.8-bin.jar` in the environment variables as well. This was discovered in the JDBC connector documentation.

Since the naming schemes of MySQL and the JDBC connector had changed significantly, and uPortal requires the exact names, I had to write my own Java program to determine this. Searching through the documentation provided by both did not reveal what they were. The program is currently available at:

<http://cs.uccs.edu/~cs526/studentproj/projS2005/ahlee/src/JdbcExample1.java>

At the end of the integration documentation, it refers to a command line command: `ant db`. This command needs to be run inside the uPortal folder itself, and not the quick-start folder where `ant tomcat` (to start the server) is run. In order to make this command work, the `ant.bat` file needs to be copied into uPortal folder, and modified to reference the ant program correctly. Once `ant db` is run, MySQL tables are created, and it is ready to use with uPortal.

Another command in the uPortal folder, `ant deploy`, is also valuable when configuring uPortal. This command forces all of the Java class files to be recompiled with the current JDK installed. This cuts down on JDK incompatibility problems.

It has been observed that MySQL on this machine makes uPortal perform extremely slow, and occasionally time out. It is believed that this slowdown was caused by the small amount of RAM on the machine, along with Symantec AntiVirus running in the background.

11.2 Paypal Integration

Many web sites have ways to collect money from users. Although most use credit card processor companies such as VeriSign, Paypal has become acceptable as a payment method as well. Paypal has great documentation on integrating web sites with their service, located in the Merchant Tools section of their site: <http://www.paypal.com>

The easiest way to use Paypal is to let their service do all the work. To do this, an Inline Frame channel needs to be created, referencing a form that submits to Paypal. Paypal has form generator tools on their website, so no HTML experience is required. The form must be placed on a web server Paypal can access.

Figure 11.2.1 illustrates an Inline Frame form button that launches a Paypal window. The source code for that page is available at:

<http://cs.uccs.edu/~cs526/studentproj/projS2005/ahlee/src/donate.php>



Figure 11.2.1 Paypal Inline Frame

Although I used a PHP extension, this file could very well have been a plain HTML file, as no PHP instructions were used.

For more complex transactions, such as a shopping cart, Paypal also has documentation on how to use Instant Payment Notification (IPN). This feature from Paypal allows the shopping cart to be processed on uPortal's side, leaving only the payment authentication to Paypal. Once the transaction is completed, a notification packet is sent back to uPortal.

12 Explorations with uPortal by Sujeeth

12.1 Classifieds Channel Description

This channel is to help students publish advertisements. We see lot of flyers in the University. With this wide need, students could submit them through a web page. A moderator will control this channel. Users of the portal can subscribe to this channel. Hence it is also in the control of the users and not a pushed fragment. Users could unsubscribe and re-subscribe to the channel whenever needed.

This channel could be improved by adding more information from the user. The classifieds could be categorized to common categories such as Rooms, Cars, Bikes, etc. The user could also submit additional Control information like request dates and the moderator could take appropriate action. A delete request mechanism could also be added to this channel.

12.2 Channel Design

The channel is designed for different roles. The roles are as follows:

- Admin: This user has complete control over the channel publishing in uPortal.
- Channel Moderator: This user group has privileges to add or remove an advertisement in the classified. The user could be considered as having the roles of Channel Content Manager, Time Frame Manager and other related ones.
- Request User: This user will be able to submit a request for placing his advertisement in the classifieds. Some fields in the Submit form are not mandatory.

- General Users: Common users could subscribe to this channel and see the classifieds advertisement channel. The user will have control over the subscription action.

12.3 Current Implementation:

The Classifieds channel has simple user interface for submitting the Classified. The classifieds submitting channel is made as a normal XHTML file, which is added as an Inline Frame to uPortal. That form submits the information to a servlet page. The servlet page then parses it and stores it in the Classifieds XML file.

The Classifieds channel which carries all the advertisements is a Portlet Channel. This portlet channel follows JSR-168 specification to provide functionality. The portlet checks for the user information and checks the user group for Classifieds Moderator group. If it is a match, the user is provided with an interface of selecting the submitted advertisements. For user authentication, uPortal APIs are used.

The XML file is listed as below for the current implementation. This will be changed for new capabilities by the channel.

```
<Classifieds>
  <Classified>
    <Title></Title>
    <Details></Details>
    <cEmail></cEmail>
    <cPhone></cPhone>
    <cAddress><cAddress>
  </Classified>
</Classifieds>
```

Listing 12.3.1: XML Schema for Classifieds Channel

The figure below is a screenshot of the current implementation. The view is of the General user and Request user.

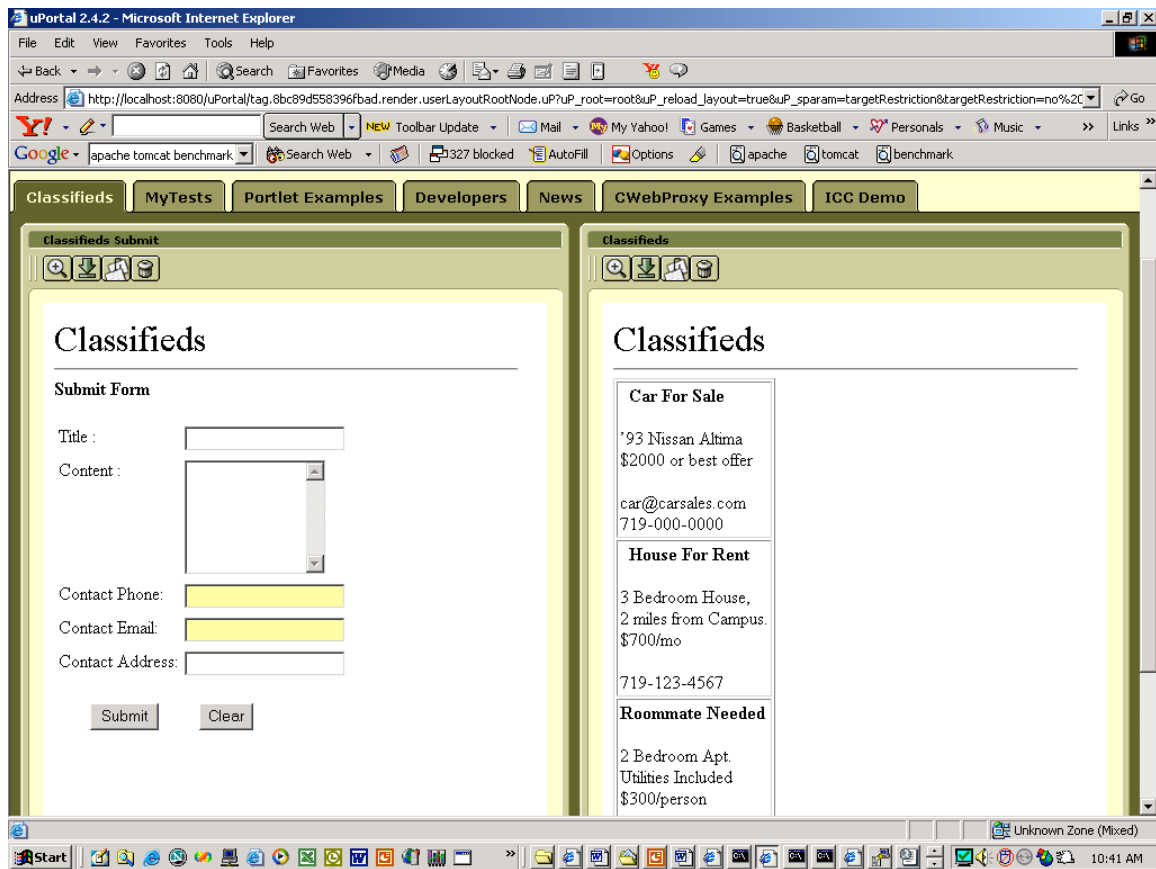


Fig 12.3.1 Screenshot of Classifieds Channel

13 References

<http://www.uportal.org>

<http://www.sakaiproject.org>

<http://www.cuconnect.colorado.edu>

<http://www.portlets.blogspot.com>

<http://www.w3c.org>

http://people.emich.edu/kmanickam/uportal_2_0_1/channel_2_0_1.htm

<http://www.mysql.com>