

Design Features and Technology Evaluations for the PET Online Knowledge Center

Mehmet Aktas, Galip Aydin, Mehmet Necati Aysan, Ozgur Balsoy, Geoffrey Fox, Cevat Ikibas, Ali Kaplan,
Jungkee Kim, Marlon Pierce, Ahmet Topcu, and Beytullah Yildiz
Community Grid Labs, Indiana University
Bloomington, IN 47402

Abstract

This paper presents an overview of the Online Knowledge Center (OKC) web portal's initial phase. We review the overall purpose of the OKC, review some shortcomings of previous PET web sites, and list our identifications of the requirements and desired features for the OKC. We then describe numerous technologies that we are using to build the OKC, based on these requirements and features. These are initially presented in an overview that summarizes the technologies and how they work together. We then present the technologies in some detail in individual sections.

1. Introduction

The Online Knowledge Center (OKC) has been commissioned by the Department of Defense High Performance Computing Modernization Program's Programming Environment and Training (PET) project to serve as a web-based repository and information management environment for the PET program. The OKC will potentially be used to provide access to a wide range of information in various formats, including but not limited to presentation slides, software repositories, contact information, training announcements, and newsgroup postings.

This report is intended to serve as an overview of the OKC's initial phase, in which we identified the basic problems we needed to solve and core features we needed to provide, determined the base technologies and portal architecture to meet these needs, and built, tested, and evaluated the prototypes for the OKC's constituent subsystems. It is the purpose of this report to explain these issues and design considerations.

Readers wishing a broad overview of the OKC's design should review the introductory material of Section 1. Specific topics are covered in more depth in individual sections for readers interested in more detail. Third party technologies involved are numerous, so we focus on their relevance to the OKC. For readers interested in more general information, we provide a list of references at the end.

1.1.OKC Design Goals

1.1.1. OKC Portal Requirements

The various desired capabilities of the OKC require the evaluation of a diverse set of technologies. It is the purpose of this document to present a general overview of technologies that have been considered for incorporation into the OKC, justify our choices and present how the various pieces fit into an overall architecture.

We identify the following as important initial capabilities of the OKC:

1. A distributed, component-based web portal environment that can be easily modified by both OKC administrators and users.
2. Dynamic web page creation “wizards” that can be used in various ways, including a newsgroup management system, training registration management, and automatic page creation for Computational Technology Areas (CTAs).
3. A distributed content management system that allows OKC contributors (such as CTA page developers) to modify, add, and delete the content of their web pages.
4. Support for a wide range of page authoring tools and techniques.
5. Multiple search capabilities, including structured XML searches (for use in newsgroups), unstructured searches (over OKC documents in many different formats) and linked structured/unstructured searches (for use in searching newsgroup postings with attachments)

The OKC has several other general requirements, not all of which we address in the current version. We record them here, though, as an indication of the scope of work remaining in OKC development.

6. The OKC should be easy to update. It is absolutely essential to the success of the OKC that CTAs can easily and quickly post material to their areas for their user community. Otherwise, the OKC’s content will quickly become outdated and irrelevant. Updating is both a technical and managerial issue. Technically, the OKC should provide simple interfaces for the various web content providers to manage their own material. Managerially, it is critical that we define procedures that expedite approval of material to be suitable for general viewing. This can be supported by OKC tools, such as event notifications that email the appropriate reviewer when new OKC material becomes available and search engines that help search for sensitive material.

7. The OKC should be easy for users to navigate and find information. This requires both good base organization and good search engine capabilities. In the production OKC web portal, it will be necessary for us to keep usage metrics to determine how well the site is being used.
8. The OKC should provide various levels of security, from publicly accessible information to SecurID protected areas. This applies to both web content and to newsgroups.
9. The OKC can define automatic updating procedures as options for content providers. Time sensitive information, for example, can be assigned an expiration date, after which it is moved to an archive area. This should address problems with the original PET websites, which became bloated with outdated information.
10. The OKC team should not dictate the style, organization, or content of the web material provided by the CTAs. We should however provide models for web sites and site generation and authoring tools to help CTAs create and organize their sites. This consistent organizational structure will also greatly help OKC users find the material they want.

1.1.2. Distributed Components and Patterns

A distributed component-based user interface is crucial to our vision for the OKC and requires some further explanation. In summary, the idea is to provide an OKC portal that serves as a central framework for loading, managing, and displaying content that can be authored, developed, and displayed. The OKC should provide a skeleton framework, or a container system, into which each CTA can add their web pages without extensive knowledge of the underlying OKC technologies. Users can then modify and personalize their pages, choosing from different content, layouts, and look-and-feel skins.

Let us now be specific about the term “distributed component.” A component in the software design sense is a programming object with a specific set of publicly accessible functions and associated internal data. These interface functions define how the component interacts with other components. Components typically are derived from abstract parent components that define universally required interface methods. Components do not exist independently but instead are placed in some root container, sometimes also called a context. Components may also be containers themselves. These containers control their child components: how they are displayed and arranged, how they communicate with other components, and so on.

Let us now consider an example of this for a web portal and its user interface. Jetspeed [JETSPEED], described more below, acts as the root container and defines a number of portal components, called portlets. These portlets are Java objects that load and display local and remote web sites. Thus an important part of the

XML [XML] interface for a remote portlet is the URL of the remote site that it displays. Within the root container, Jetspeed defines a number of different layout formats for these containers, and this arrangement is hierarchical. For instance, a user can have a container that has a two-columned arrangement. Each column is a portlet container itself and so can contain other portlets with other layouts.

We envision that the OKC in production will eventually consist of a central OKC web server that manages a number of distributed content web servers maintained by HPCMP Major Shared Resource Centers, distributed centers, and partner institutions. The central OKC server manages the publication of this distributed content and also provides options for the arrangement and display. Likewise, features like site searching and reading messages posted to newsgroups will possess web-based user interfaces that will also serve as portlet components.

Using the distributed component model such as the one defined by Jetspeed, we will be able to

1. Extend existing portal components to address specific needs of the OKC (such as in-portlet navigation).
2. Encapsulate local and remote web sites within components that have XML interfaces, thus providing us with a well defined portlet-portlet interactions and portlet-container interactions. This can be used for example to define a consistent linking procedure between different remote web sites running in different OKC portlets.
3. Define a framework into which we can plug in advanced features developed independently of the OKC.
4. Allow users to pick the portlets that they want and arrange their web pages in a way that suits them and simplifies their navigation.

Another, related feature is the use of the Model-View-Controller (MVC) design pattern in the portal. The basic idea of MVC is to keep separate and independent the mechanisms that are responsible for the portal data (the web sites wrapped in portlets), the display (skins and layouts), and the internal communications. This means that the portal components separate data from display and control. Thus for example, the portlets define the data, but the arrangement is completely independent. This design allows one for instance to arrange the display of different components in different ways and easily define new layout schemes without having to modify the displays themselves. Likewise, in Jetspeed (and its underlying Turbine [TURBINE] control servlet) we can separate the control that links various pages in a sequence from the pages themselves.

1.2. Basic Technologies

1.2.1. XML and XML Schemas

XML, the extensible markup language, is a World Wide Web Consortium (W3C) [W3C] standard format for describing tree-structured data. XML defines how one can construct a set of HTML-like markup tags to describe and organize information. For example, as we will discuss below, all remote content contained in the OKC is described using XML metadata. XML documents should be both valid and well-formed. Valid XML documents conform to a particular *XML Schema [SCHEMA]* that defines the rules for a document: which sub-elements and attributes are required for a particular element, which are optional, how many times sub-elements can occur, and so forth. The XML schema for a particular markup language dialect extends from the global XML schema defined by the W3C. Well-formed XML documents are those that maps to a tree structure. XML is useful for creating metadata descriptions that are both human and machine readable. Validity and well-formedness also lead to a particularly interesting connection between XML data structures and JavaBeans. We discuss this in some detail in Section 4 on schema wizards.

1.2.2. JavaServer Pages

JavaServer Pages [JSP] (or JSP) allow HTML web pages to contain Java code for dynamic control. JSPs can thus use Java control structures (if statements, for and while loops) to generate dynamic web page content in reaction to different requests. JSPs also allow web pages to contain non-visual Java components that provide services such as authentication. JSPs pages are run entirely on the web server and so require no special software to be installed on the client and work with standard browsers.

JSP pages are converted into Java code, compiled, and executed by a special web server that runs a Java Virtual Machine. A free, open source version of this server called Tomcat [TOMCAT] is available from Apache. Tomcat servers can be used as standalone web servers or can be run behind other web servers. Commercial versions of JSP servers are available, including WebSphere from IBM, WebLogic from BEA, and iPlanet from Sun.

The OKC makes extensive use of JSP for building the interface to the newsgroup system. Jetspeed, the OKC's web layout management system, uses JSP and related technologies to generate pages dynamically and to maintain session state so that different users can maintain different views of the web site.

1.2.3. Java Messaging Service

The Java Messaging Service [JMS], or JMS, is a publish/subscribe system that can be used to communicate events between Java services running on different hosts. Java programs that generate events register as publishers to a central JMS server. Other applications can register to the same JMS server as subscribers to one or more publication channels. Publishers then send events whenever anything interesting

occurs (such as someone posts a message to a newsgroup) and the JMS server notifies all subscriber applications, which can decide independently how to react to the events.

JMS is an important part of the OKC newsgroup system. When someone posts a message to a particular news group, this generates an event received by two subscribers: a database listener that archives the message and makes it available through the web site and a listener that posts the message to interested users' regular email accounts.

1.3. OKC Architecture

We present an overall view of the OKC in Figure 1. Essentially, the figure illustrates the central OKC control server with content delivered from remote servers. The OKC server contains only the central portal control and display code and OKC specific web content. We use Jetspeed for this portal framework. Other web content is maintained on separate servers that can double as content management servers. Apache-Jakarta's Slide [SLIDE] project is one such content management system that we have evaluated, but other systems are possible. Note in the diagram that we do not currently have display restrictions on the content servers. They can be accessed directly from a browser as well as through the OKC portal. The decision between open and locked-down content servers will have important implications on search technologies that we will use.

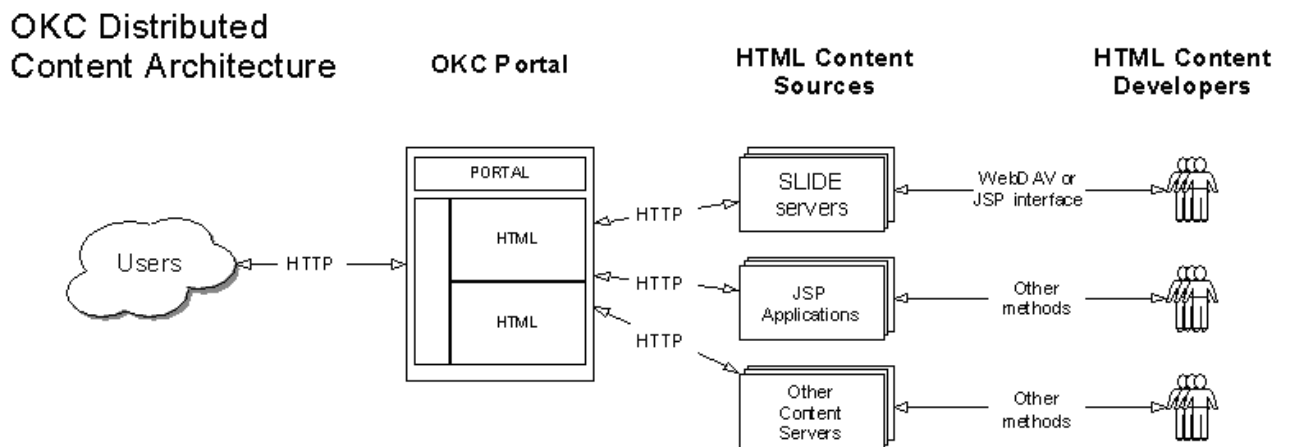


Figure 1 OKC organizes distributed content servers into a single interface.

1.4. Jetspeed Overview: Browser Interface Framework

The OKC Web Portal is based on Jetspeed, a Web portal development environment from the Apache Open Source Group [APACHE], developed under the Jakarta project. Jetspeed defines a number of portlets that encapsulate a web page or set of web pages. Portlets can be either local to the Jetspeed server or can be loaded

from remote servers. The Jetspeed root container manages the display layout of the portlets. Portlets can contain other portlets.

The obvious benefits of the portlet concept for web design have lead to product developments from a number of commercial companies, including BEA, Oracle, IBM, and Sun. Representatives from these companies, together with members of the Jetspeed development team, are developing a Portlet specification as part of the Java Community Process [JSR].

1.5. Slide Overview: Distributed Content Management

Slide [SLIDE] is a web content management system from Apache-Jakarta, the same umbrella group that is also developing many of the other technologies we are using, particularly Jetspeed and Tomcat. In brief, Slide is a Java based implementation of Web-DAV [DAV], a standard extension to the HTTP protocol for distributed content management. As described above, Jetspeed allows us to display content from distributed web servers. We discuss the use of Slide for these content servers, which will allow users to manage their own web material on the remote content servers.

1.6. Newsgroup Overview: Dynamic Content Creation and Page Wizards

The OKC development team has developed an XML and JMS-based newsgroup system. Messages are posted through either a JSP user interface or with a standard email client. Messages are formatted with XML and archived in a database. Newsgroup displays then query the database to create a message list. The technologies underlying the newsgroup system can be reused to build a number of related features, including page wizards for creating CTA pages, conference management systems, and training management systems. The newsgroup system is described in detail in four related sections.

The newsgroup system is summarized in Figure 2. Users can post messages to newsgroups through either an interface in the OKC portal or by using their regular email client. Users have the option of viewing postings through a portal window in the OKC or having them delivered to their regular email.

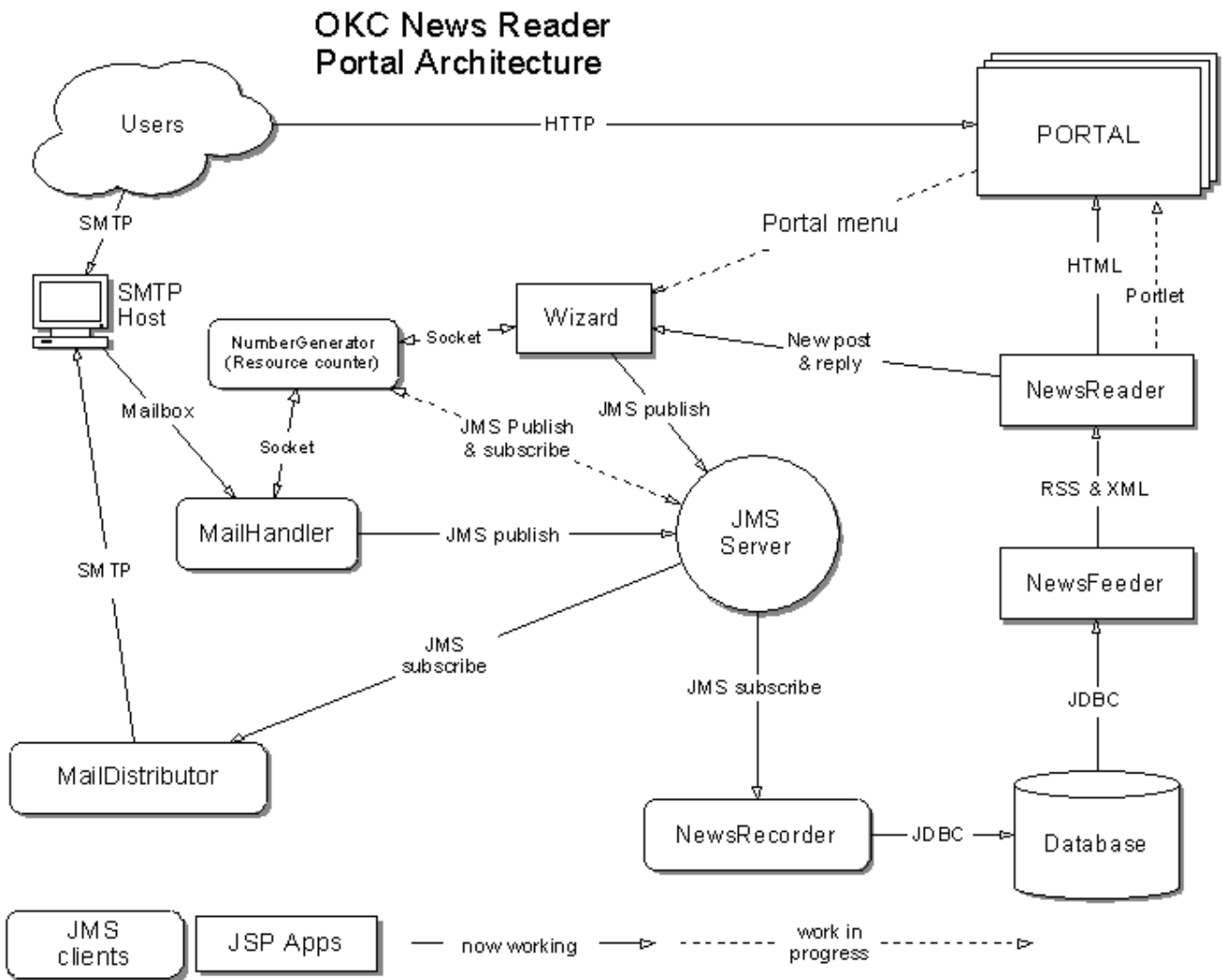


Figure 2 OKC's newsgroup architecture supports both email and JSP clients and readers.

Posted messages are received by a schema wizard that publishes them to a JMS server. The JMS server notifies both the NewsRecorder and the MailDistributor. The NewsRecorder breaks down the message into its XML constituent parts and stores in a database. The NewsFeeder and NewsReader handle the extraction of messages from the database and their display, respectively. The MailDistributor is responsible for forwarding postings to all newsgroup subscribers that have chosen the direct email option.

1.7. Search Capabilities Overview: Structured, Unstructured and Hybrid Searches

The Online Knowledge Center requires a number of different search capabilities, and the final choices will depend upon architectural decisions about content accessibility that need to be made by the sponsoring/hosting MSRCs and the PET program. If content is publicly available or not determines the type of search that will be used in production. We thus have evaluated technologies to support either option.

We address the following search scenarios:

- a. Unstructured searches over publicly available OKC material;
- b. Unstructured searches over private web material;
- c. Simple structured searches of XML documents, such as newsgroup postings; and
- d. Linked structured and unstructured searches, such as over newsgroup postings and their attachments.

Search capabilities (a) and (b) will be part of the initial OKC capabilities. We have reviewed both options and describe how they may be implemented in the section on search. Searches of types (c) and (d) are still being testing at the time of writing. The linked search is an interesting topic of research for us.

2. OKC Browser Interface Management

2.1. Design Considerations

The central feature of the OKC is a centralized user interface management system. For this, we chose Jetspeed, a subproject of Jakarta. This allows us to separate web content from the management system. Jetspeed organizes both local and remote web content into “portlets” that can be arranged and displayed in a user-customizable interface.

For the OKC, for example, we have built a prototype that consists of the main OKC web server, with separate server instances supplying the contents of the portlets. We consider this system also to be a prototype for future integration of HPCMP portal projects.

As far as we are aware, the PET OKC is the most ambitious project so far to be built with Jetspeed. Thus, our requirements are more extensive than originally considered by Jetspeed developers, and there are several problems that we are solving in order to use Jetspeed.

2.2. Jetspeed

Jetspeed is an open source implementation of an enterprise information portal, from the Jakarta Project, which is sponsored by the Apache Software Foundation. Basically, Jetspeed uses Java and XML. We used Jetspeed as a base portal structure for our Online Knowledge Center. Jetspeed defines a standard API for writing vendor neutral portlets that work on any portal framework that supports this API such as Jetspeed itself.

The user can access the portal by a web browser, WAP-phone, pager, and potentially other devices. Jetspeed is like a center where information from multiple sources is available for the end users.

The data presented by Jetspeed is independent of content type. This means that content from for example RSS (Rich Site Summary, an XML dialect, see [RSS]) can be integrated with Jetspeed and the actual presentation of the data is handled via XSL and delivered to the user for example via the combination of Java Server Pages (JSPs) and HTML. Jetspeed provides support for template and content publication frameworks such as Velocity.

Definitions of the numerous terms in this section are provided in an appendix

2.3. Portlet

A portlet, which is a web component managed by a container, is used in Jetspeed portal environment to generate dynamic content. Portlets are actually platform independent java classes which can be dynamically loaded and run by a web server. They implement the `org.apache.jetspeed.portal.Portlet` interface. It is essential to extend new portlets from `org.apache.jetspeed.portal.portlets.AbstractPortlet`, which provides the basic functionalities. The difference of portlets compared with servlets is the indirect communication between web clients and portlets through Jetspeed.

A typical Jetspeed portal screen is a collection of portals. There are pre-defined portlets but it is also possible to write and use by your own portlets. As you see in Figure 3 there are three types of portlets. On the left there is Menu portlet, on the center there is Manual portlet and on the right there is Welcome portlet.

The portlets are defined in the registry by an entry description. These entries can be in any of three distinct types:

1. instance: Instance is the basic entry type. An instance entry should provide all the necessary information to instantiate the portlet. This type of entry is instantiated as is by the Jetspeed engine.

2. abstract: An abstract entry is an entry which cannot be instantiated directly. It acts as a portlet template, useful for defining common properties for a group of related portlets. This type of entry is never instantiated by Jetspeed.

3. **ref** : The ref entry is an entry that defines a new entry derived from some other entry definition. Thus the ref entry definition must reference another registry entry which may be of any type, even other ref entries. The engine will cascade all refs until it finds either an abstract or instance entry. It will then override all the parameters found in the abstract or instance definitions by those found in the ref definition and try to instantiate this portlet.

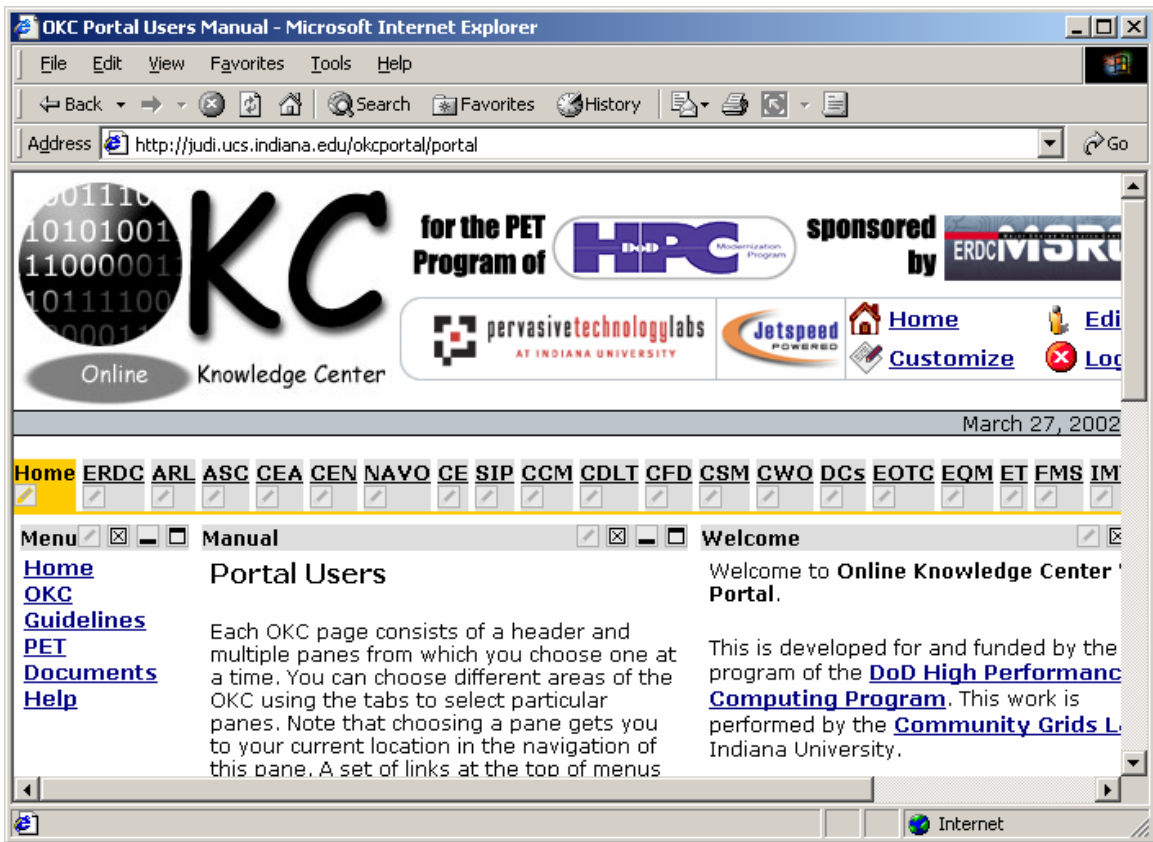


Figure 3 The OKC main page illustrates tab menus, portlets, and customization features.

The screen shot in Figure 3 contains three portlets, Menu, Manual, and Welcome, identified by their title bars. Each portlet has buttons in their title bar on the rightmost. These are (from left to right and linked to title bar icons) customize, maximize, minimize and closed buttons which works with actions. It allows users to do some actions such as maximize, minimize, customize if they are allowed. We now explain each:

Portlets Buttons:

Customize edits the portlet data, customizing the portlet content.

Maximize views the portlet in full screen with no other portlets in view.

Minimize shows only the title bar.

Closed closes the portlet and remove it from the page.

2.4.Jetspeed Features

The following list, taken from Jetspeed web site, summarizes its features.

- Java Portlet API specification
- Template-based layouts including JSP and Velocity
- Supports remote XML content feeds via Open Content Syndication
- Custom default home page configuration
- Database user authentication
- In-memory cache for quick page rendering
- [Rich Site Summary](#) support for syndicated content
- Integration with Cocoon, WebMacro and Velocity so that you can develop with the newest XML/XSL technology.
- Wireless Markup Language (WML) support
- XML based configuration registry of portlets
- Full Web Application Archive (WAR) support
- Web Application development infrastructure
- Local caching of remote content
- Synchronization with [Avantgo](#)
- Fully portable across all platforms that support JDK 1.2 and Servlet 2.2
- Integrated with Turbine modules and services
- Profiler Service to access portal pages based on user, security (groups, roles), media types, and language
- Persistence Service available to all portlets to easily store state per user, page and portlet

- Skins so that users can choose colors and display attributes
- Customizer for selecting portlets and defining layouts for individual pages
- PSML can be stored in a Database.
- User, group, role and permission administration via Jetspeed security portlets.
- Role-based security access to portlets.

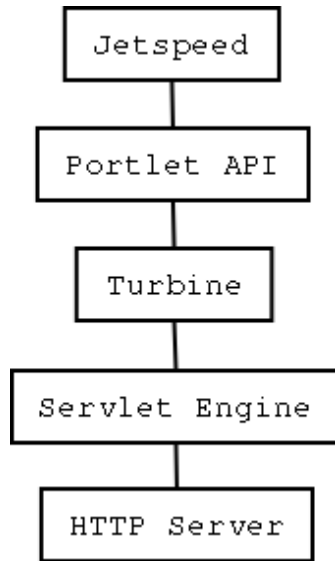
2.5. Factors choosing Jetspeed

The most important factors when we chose Jetspeed as a base portal implementation for OKC are given below.

1. Jetspeed is free and open source from Apache Jakarta Project, so we have the ability to modify and extend the source code to meet our specific requirements.
2. Jetspeed developers are participating in a Java Specification Request with Oracle, IBM, Sun, and BEA to develop a standard portlet API. Thus we can use Jetspeed for development with confidence that our developments will be later compatible with commercial products.
3. The portlet architecture meets several of our basic requirements, as described in the introduction.
4. As an open source project, we have the ability to have direct interactions with and support from Jetspeed developers, and, as importantly, influence and contribute to Jetspeed development.
5. Jetspeed is Java-based.

2.6. Portal Architecture

Jetspeed is built on top of Turbine, which is also another open source application framework from the Jakarta Apache Project. Turbine is a servlet based framework in order to build secure web applications. In Jetspeed architecture Turbine is responsible for user authentication and security.



There are following layers in the order.

1. HTTP server
2. Servlet Engine (Tomcat)
3. Turbine
4. Portlet API
5. Jetspeed

Jetspeed is built on the Turbine framework. The Turbine provides us a Turbine servlet which receives and responds to HTTP requests from clients. The Turbine servlet is configured to call Jetspeed components when there are HTTP requests. It works as event base model. When the Turbine receives an HTTP request, it generates a Turbine event and searches for an action class and invokes a method in the action class. Each portlet that wants to process HTTP requests in the event model must provide an action class with the corresponding named methods to handle the events generated by Turbine.

2.7.OKC Portal Internal Mechanisms

2.7.1. *Internal Mechanism Diagram*

The architecture of a Jetspeed portal is given in Figure 4. We will explain each of the constituent pieces.

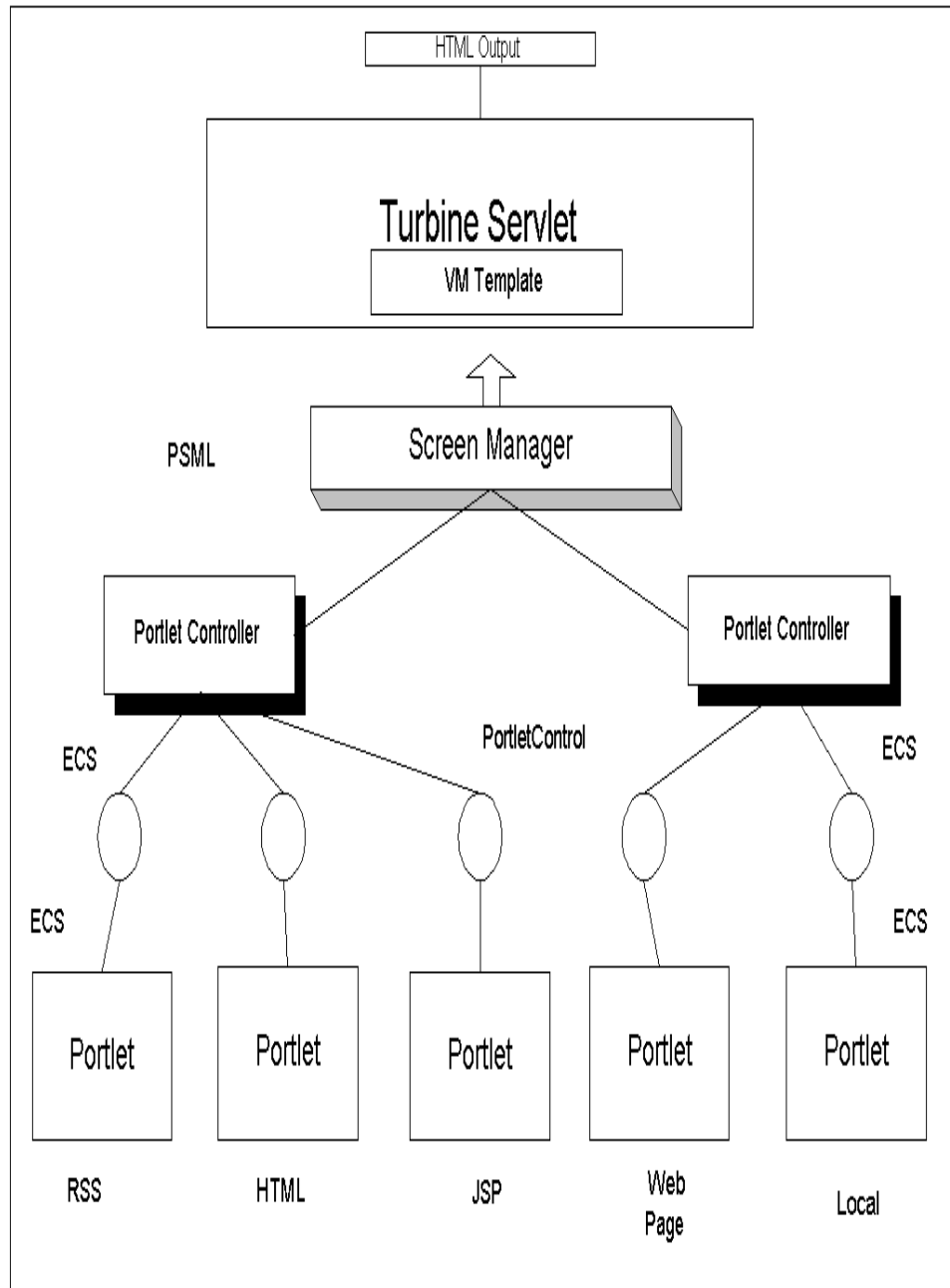


Figure 4 Jetspeed's architecture is used to rewrite local and remote data sources into a single HTML file for display.

2.7.2. Portlets

OKC Portal consists of pre-defined and user defined portlets. There are six main portlets in our portal structure. These are RSS, HTML, XML and XSL, WebPagePortlet and local portlets. As defined before each portlet is actually java classes. In order to use these portlets each of them has to be registered in \$HOME/WEB-INF/conf/okcportlets.xreg file.

We use the following portlets in the OKC:

1. OKCWebPagePortlet
2. HTML
3. JSP
4. GridsJSP
5. RSS + XSL
6. XML + XSL

2.7.3. PortletControl

In the second layer there is the PortletControl (Java) object. The portlet content is rendered by a PortletControl object, and it provides a box which is actually an HTML table which the portlet fits into it. PortletControl handles rendering the title and body of the Portlet then it passes these values by ECS.

2.7.4. Element Construction Set (ECS)

These pre-defined portlets are used to generate object elements in order to build the web page and control goes to ECS, which stands for Element Construction Set [ECS]. ECS is a java API to generate elements for various markup languages such as HTML and XML. In our case ECS allows us to use Java Objects to generate markup code. It takes the parameters from the portlets and builds a java object which consists of HTML tags.

For example a basic ECS will look like similar to the next example.

```
Document doc = (Document) new Document()  
    .appendTitle("Example")  
    .appendBody(new H1("Example Header"))  
    .appendBody(new H3("Sub Header:"))  
    .appendBody(new Font().setSize("+1")  
        .setColor(HtmlColor.WHITE)  
        .setFace("Times"))
```



```
        .addElement("text goes here");
out.println(doc.toString());
```

So this ECS will automatically creates an HTML page for us as shown below.

```
<html>
<head><title>Example</title></head>
<body><h1>Example Header</h1><h3>Sub Header:</h3>
<font size="+1" color="#FFFFFF" face="Times">text goes here</font></body>
</html>
```

2.7.5. PSML

PSML stands for Portal Structure Markup Language. It is in XML format. It allows content structure and abstraction within Jetspeed. PSML is actually composed of two different markups which are registry and site markup.

The registry markup which describes all the available resources to the Jetspeed engine. It supports more than a Portlet Registry. It also has a CapabilityMap Registry, a MediaType Registry, PortletControl Registry, and a PortletController Registry which are registry entries for the portal.

The site markup which describes which portlets, available in the registry, are actually displayed for a given user, as well as how they are organized on screen and what presentation properties they should use.

2.7.6. PortletController

After that it comes to PortletController. A PortletController is responsible from the set of portlets. It combines multiple PortletControls and provides entire page information. The PortletController is configured by XML files and has almost every option seen by the users.

2.7.7. Turbine

At the top of Jetspeed, Turbine is used for portal container implementation. According to PSML files it handles layouts, navigations and screens. It handles user authentication and page layout as well as scheduling. Portlets interact with Turbine services with the provided RunData object. In that sense pluggable template systems, including JavaServer Pages and Velocity provide for dynamic web applications through layouts, XML/XSLT. In OKC we use velocity for template structure.

Turbine consists of five different modules which each serve for a specific service within the Turbine framework. These modules are Action, Navigation, Layout, Screen and Page. The layout structure for a web site with Turbine would like as in Figure 5.

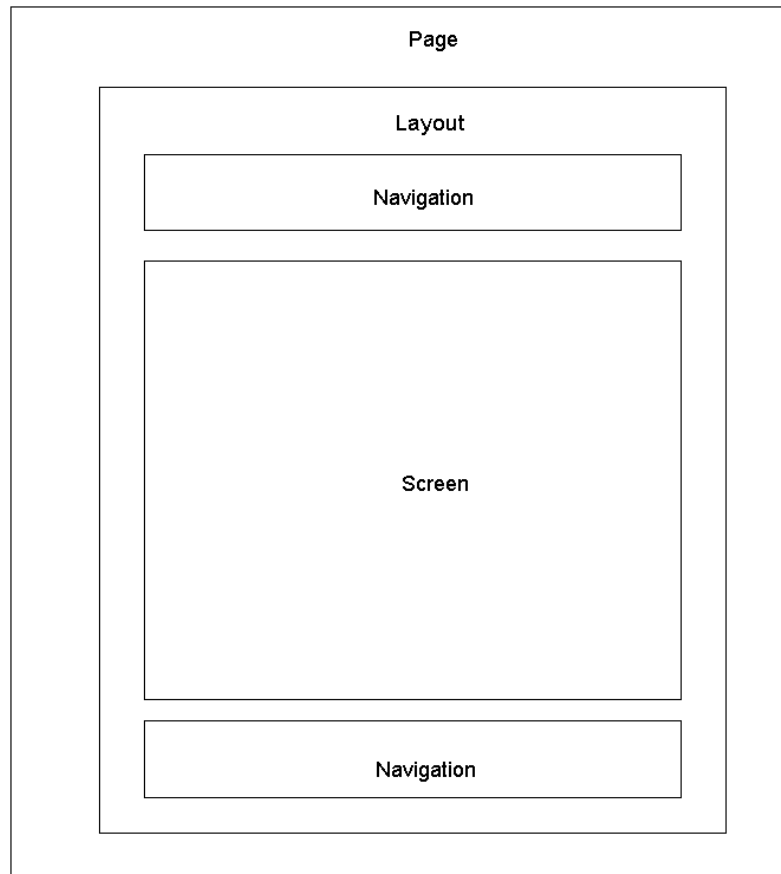


Figure 5 Jetspeed/Turbine portlet containers are mapped into HTML tables for display.

Turbine is responsible from the following structures.

Layout: It controls the layout of the entire web page.

Navigation: Navigation areas are divided as top, bottom, right and left.

Screen: This is the central area of the web page where the PSML file is placed.

2.7.7.1 *Velocity*

Velocity is actually a Java based template engine. It is similar to Java Server Pages (JSP) but with its different syntax it is also another alternative to JSP from Apache. Velocity provides to users template language to reference objects in Java code. Velocity can be used either as a standalone utility for generating source code and reports, or as an integrated component of other systems. Velocity provides template services for the Turbine. Velocity and Turbine together provides a template service that allows web applications to be developed.

Example syntax of velocity is as follows.

```
<table cellspacing="0" cellpadding="0" border="0" width="100%">
<tr>
<table width="100%" border="0" cellspacing="0" cellpadding="0">
  <tr bgcolor="#000000">
    <td></td>
  </tr>
  <tr bgcolor="BDC5CB">
    <td>
      <div align="center">Department of Defense High Performance Computing Modernization Program:
Programming Environment & Training
      </div>
    </td>
  </tr>
  <tr bgcolor="#000000">
    <td></td>
  </tr>
</table>
<td>
</td>
</tr>
</table>
```

In OKC this prints the table on the bottom for the webpage. In OKC directory structure they are located at \$OKCHOME\WEB-INF\templates\vm\ and layouts, navigation, controls, controllers and screen structure are defined here.

2.7.7.2 HTML OUTPUT

As a result of those steps the HTML output has been automatically generated and publishes as nested tables to the user. In figure 5 the output of this internal mechanism can be seen.

2.7.8. *Consequences of Using Jetspeed*

There are some consequences of this portal approach. Primarily, these arise from the rendering of portlets as HTML tables.

1. Netscape Navigator version 4.X has problems with rendering nested tables. NN users might have display problems. A detailed explanation can be found in problems section.
2. Content pages that depend on HTML tags that must be located in the document's <HEAD> will have problems. This includes for example the page refresh meta tag. This again is because the remote content is loaded into an HTML table. The remote web page is no longer an independent HTML document but instead part of a larger single document.
3. In order to display portlets and portlet sets, a default Cascading Style Sheet (CSS) is used, and so the portlet may override content formatting specified in the original web page.
4. Jetspeed does not support the frame structure in a webpage so when building a new page it is necessary not to use frame structures. If a user wants to use menu for the page he or she can use stemlinks.xml and define the webpage in one of the areas. This procedure will explained in OKC Portal section.
5. Jetspeed (through ECS) is less forgiving of poorly form HTML and incorrect Javascript. Browsers either correct or ignore these problems and are very robust when it comes to handling mistakes in content. ECS is not so robust and can fail if given documents with mistakes. Thus a page that can be displayed directly by a browser may not make it through the Jetspeed reconstruction process illustrated in Figure 4.

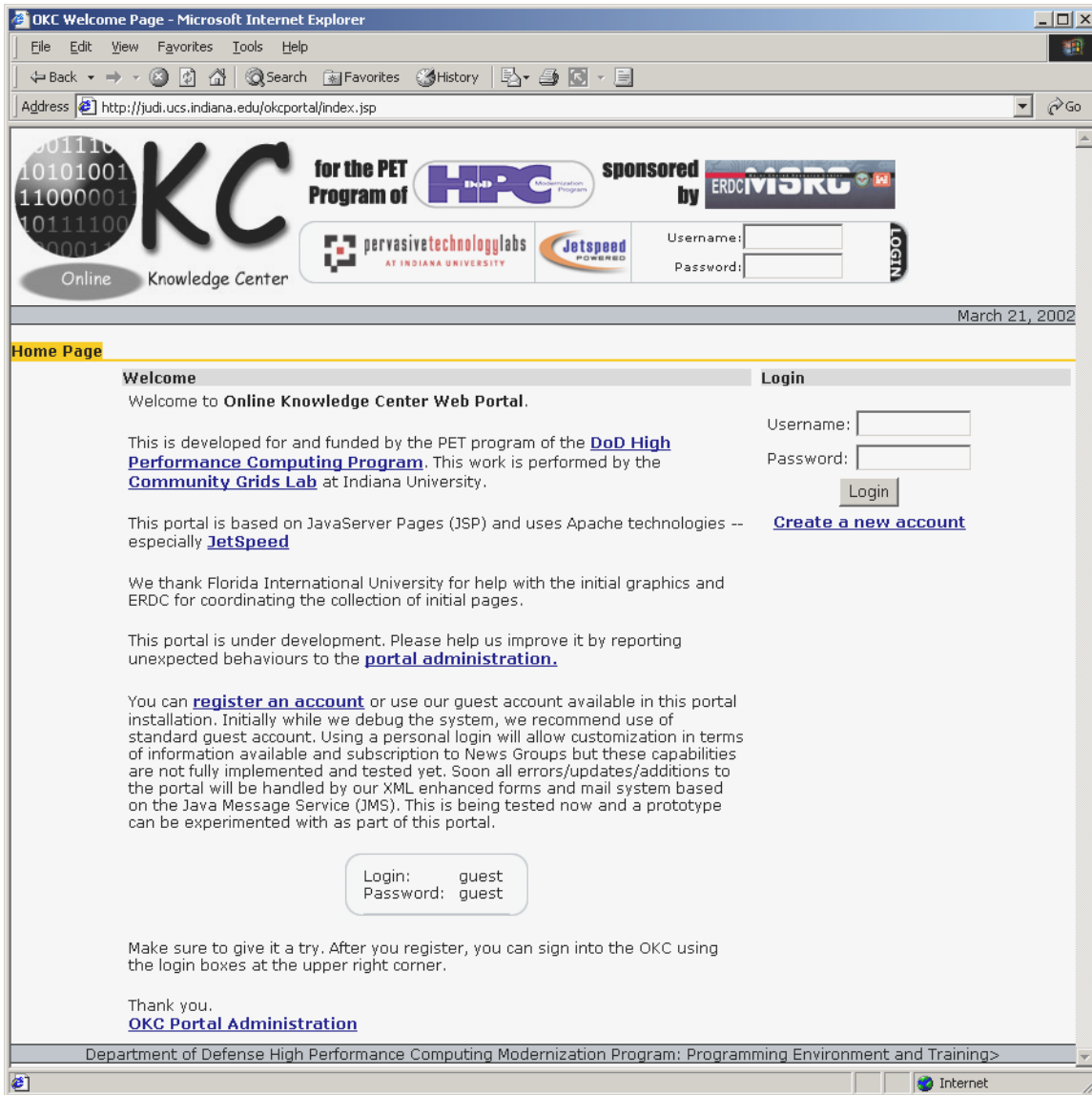


Figure 6 The Online Knowledge Center's login page.

Figure 5 Online Knowledge Center Portal

2.8. OKC Portal

Features and architecture aside, we have to evaluate Jetspeed to see if its portlets can support the wide range of potential web content. The PET legacy web content is a good representative of these different content styles. Before handing over the content management to other developers outside the OKC project team, we must add the content ourselves by hand and find and correct any problems.

The OKC Portal supports plain HTML, CGI, JSP, JavaScript, XML, XSL pages. So far ERDC, ARL, ASC, CEA, CEN, NAVO, CE and SIP pages have been tested and they are accessible for OKC portal. Problems for previous pages included not using syntactically correct HTML or JavaScript languages. When there are cgi pages (such as used by NAVO) ,this can be shown by using apache at the background.

Our general procedure is summarized in the list below. These are explained in detail in the following section. This procedure is currently performed manually by editing the indicated files, but we will automate and add user interfaces. As stated above, our first goal is to simply test support for different types of content.

1. Review process of web pages from any problems
2. Write stemlinks.xml for menu
3. Define the area okcproperties.xml
4. Register the portlet in okc-portlets.xreg
5. Add the new page to the okcmenu.jsp
6. Customize

2.8.1. Steps for adding web sites to the OKC

1. Review the pages which will be added to the OKC portal. If there are problems or incorrectness with the pages, they should be fixed. The problems might occur since they are not well formed. Let's say if you are using JavaScript in your page and you forget to write semicolon, this might result a problem in OKC Portal. This is also true for HTML pages if you forget to close a tag. This may not be a problem for current browser since they fix these kinds of errors, but for OKC Portal malformed markup languages are not acceptable. When using an html portlet there is no need to use <head> tag. But when using OKCWebPagePortlet there is no such regular web pages can be used as long as they are well formed markup languages.
2. Write or edit the stemlinks.xml file in order to display menu items. When there is a stemlinks.xml the portal engine will recognize, read and show it on the OKC menu part. An example for stemlinks.xml file is below

```
<?xml version="1.0" encoding="UTF-8"?>
  <section xmlns="http://grids.ucs.indiana.edu/okc/schema/stemlinks/v1"
xmlns:cg="http://grids.ucs.indiana.edu/okc/schema/cg/ver/1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://grids.ucs.indiana.edu/okc/schema/stemlinks/v1
  http://grids.ucs.indiana.edu/schemas/stemlinks.xsd" baseStem="/okc/rawpages/SIP/"
version="0.1">
  <title>Signal and Image Processing</title>
```

```

<description>Signal and Image Processing</description>
<metadata>
    <cg:screen view="portal" link="relative"/>
    <cg:contentscreen mainframe="body"/>
</metadata>
<item stem="."/>
    <title>SIP Home</title>
    <link>index.html</link>
</item>
<item stem="sip/">
    <title>SIP Forum 1999</title>
    <link>index.html</link>
</item>
<item stem="sip2000/">
    <title>SIP Forum 2000</title>
    <link>index.html</link>
</item>
<item stem="sip2001/">
    <title>SIP Forum 2001</title>
    <link>index.html</link>
</item>
<item stem="."/>
    <title>HPC Related Documents</title>
    <link>tools.html</link>
</item>
</section>

```

In these stem links the base directory structure is given and there are two main tags title and link in the item. The link of the page can be given in link and title can be written in between title tags.

3. If the menu wants to shown it has to be defined in one of our areas. The area declaration is done in okcproperties file which also another xml file. The directory structure for it is \$JETSPEEDHOME\WEB-INF\OKC\okcproperties.xml

A sample entry is as follows

```

<?xml version="1.0" encoding="UTF-8"?>
  <configuration xmlns="http://grids.ucs.indiana.edu/schema/okc/config/v1" version="0.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://grids.ucs.indiana.edu/schema/okc/config/v1
http://grids.ucs.indiana.edu/schemas/okcconfiguration-v1.xsd">
  <!-- contenthost>http://localhost</contenthost -->
  <!-- contenthost>http://montblanc.ucs.indiana.edu</contenthost -->
  <contenthost>http://judi.ucs.indiana.edu</contenthost>
  <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
  <!-- mediaserver>http://montblanc.ucs.indiana.edu/mediaserver/fs</mediaserver -->
  <mediaserver>http://judi.ucs.indiana.edu/mediaserver/fs</mediaserver>
  <stemlinks>stemlinks.xml</stemlinks>
  <newbrowser>
    <extention type="ppt">Microsoft Power Point Files</extention>
    <extention type="doc">Microsoft Word documents</extention>
    <extention type="pdf">Adove PDF files</extention>
    <extention type="css">Cascade Style Sheets</extention>
    <extention type="ps">Adobe PostScript Files</extention>
    <extention type="gif">GIF files</extention>
    <extention type="jpg">JPEG files</extention>
    <extention type="jpeg">JPEG files</extention>
    <extention type="mpg">MPEG files</extention>
    <extention type="mpeg">MPEG files</extention>
    <extention type="avi">AVI files</extention>
    <extention type="swf">Flash files</extention>
    <extention type="png">Image files</extention>
    <extention type="ram">Real Media files</extention>
    <extention type="rm">Real Media files</extention>
  </newbrowser>
  <areas>
    <area>
      <name>tests</name>
      <description/>

```



```

        <stem>/okc/rawpages/testcases</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/testcases</mapto>
    </area>
</areas>

```

4. The next step is to define our new portlet. Since everything is represented as portlets in jetspeed to be able to see our pages they should define as portlets. For our case the file is okc-portlets.xreg and it is located in \$JETSPPEEDHOME\WEB-INF\conf\ okc-portlets.xreg

```

<?xml version="1.0" encoding="UTF-8"?>
<registry>
  <portlet-entry name="Welcome" hidden="false" type="ref"
    parent="HTML" application="false">
    <meta-info>
      <title>Welcome</title>
    </meta-info>
    <parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
    <url>/welcome.html</url>
  </portlet-entry>
  <portlet-entry name="Help" hidden="false" type="ref" parent="HTML" application="false">
    <meta-info>
      <title>Help</title>
      <description>OKC Help Page </description>
    </meta-info>
    <parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
    <url>/BugReport.html</url>
  </portlet-entry>
  <portlet-entry name="BloomingtonWeather" hidden="false" type="ref"
    parent="CustomizerVelocity" application="false">
    <meta-info>
      <title>Bloomington</title>
      <description>Weather conditions in Bloomington,IN in a portlet</description>
    </meta-info>
    <parameter name="template" value="weather" hidden="false"/>
    <parameter name="action" value="okc.WeatherAction" hidden="false"/>
  </portlet-entry>

```

```

    <parameter name="weather_city" value="Bloomington" hidden="false"/>
    <parameter name="weather_state" value="IN" hidden="false"/>
    <parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
    <media-type ref="html"/>
</portlet-entry>
</portlet-entry>
<portlet-entry name="PET1_ERDC" hidden="false" type="ref"
  parent="OKCWebPagePortlet" application="false">
  <meta-info>
    <title>PET1/ERDC</title>
    <description>PET1/ERDC Site</description>
  </meta-info>
  <parameter name="url_variable" value="peterdc_url" hidden="true"/>
  <parameter name="home_url" value="/rawpages/pet/pet_bd.htm" hidden="false"/>
  <parameter name="title" value="PET1/ERDC" hidden="false"/>
  <url>/rawpages/pet/pet_bd.htm</url>
</portlet-entry>
</registry>

```

- There is a jsp portlet called okcmenu.jsp, located in \$JETSPEEDHOME\WEB-INF\templates\jsp\portlets\html. In order to show the new web site it has to be added to vector inside the okcmenu.jsp file

```

<%@
taglib uri="/WEB-INF/templates/jsp/tld/template.tld" prefix="jetspeed" %><%@
page import="org.gxos.debug.Log, commgrids.stemplinks.schema.*" %><%@
page import="commgrids.stemplinks.schema.*,commgrids.okc.config.*" %><%@
page import="commgrids.okc.*,commgrids.okc.portal.util.Pane" %><%@
page import="java.util.*,java.net.URL,java.io.*,javax.naming.*" %><%@
page import="org.apache.jetspeed.services.resources.JetspeedResources" %><%@
page import="org.apache.jetspeed.util.URILookup" %><%@
page import="org.apache.jetspeed.services.rundata.*"
%><%!
  static {

```

```

    OKC.initLogger();
}
%><%
    boolean error = false;
    JetspeedRunData rundata = (JetspeedRunData)request.getAttribute("rundata");
    Configuration configuration = OKC.getConfiguration(rundata);
    Hashtable areas = OKC.getAreas(rundata);
    String homePath = URILookup.getURI(URILookup.TYPE_HOME,URILookup.SUBTYPE_NONE,
rundata);
    String paneTitle = rundata.getParameters().getString("pane0");

    Vector paths = (Vector)(rundata.getSession().getAttribute("paths_vector"));
    Vector panes = (Vector)(rundata.getSession().getAttribute("panes_vector"));
    if(panes == null) {
        panes = new Vector();
        panes.add("ERDC");
        panes.add("ARL");

        paths = new Vector();
        paths.add(new Pane((String)panes.get(0),"/rawpages/pet/pet_bd.htm","peterdc_url"));
        paths.add(new Pane((String)panes.get(1),"/rawpages/ARL/PET/pet.html","petarl_url"));

        rundata.getSession().setAttribute("panes_vector",panes);
        rundata.getSession().setAttribute("paths_vector",paths);
    }

```

Please note that here, be sure to add in the same order when writing your panes.

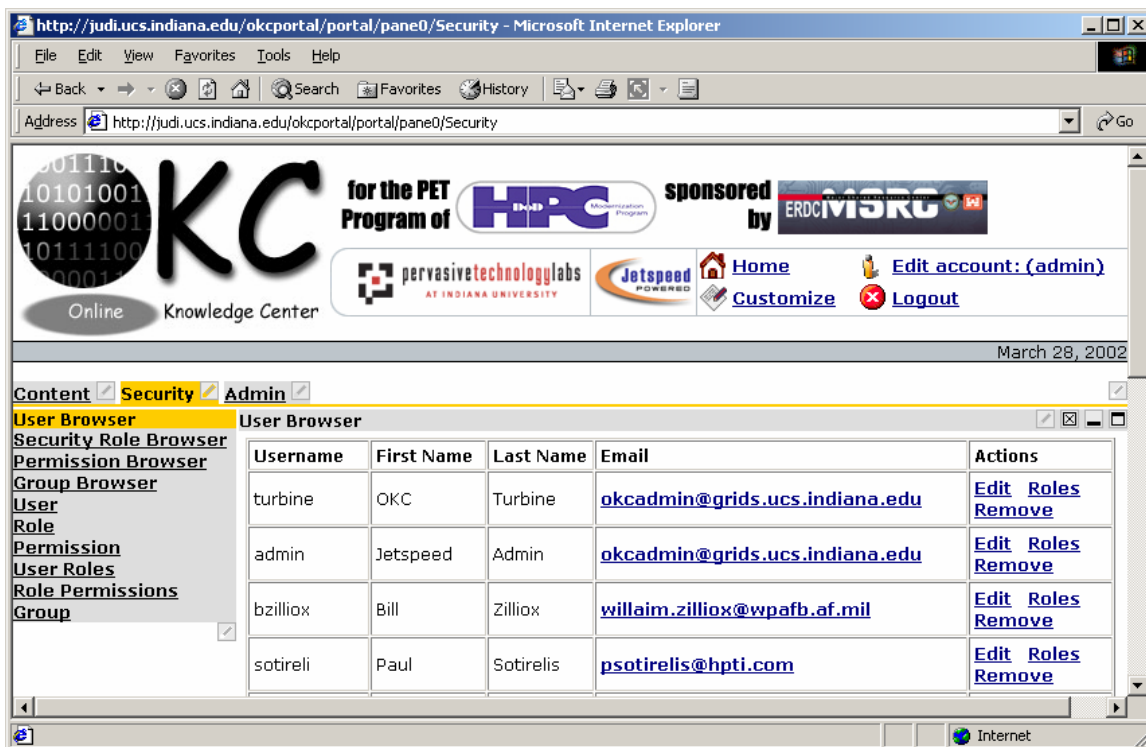
6. The last step is to update the new pages for the user to be able to see. We currently do this by editing the base PSML file and overwriting each registered user's individual copy of this file. Actually each user can update his or her own portal by own self. But there should be a page which tells people when new pages are available. In the future a help page for how to customize the pages will be also necessary. Since we are in development process there are more updated web sites so each time we modified all psml files for each user. For registered users a notification portlet for changes would be enough since some users do not want to change their portlet.
7. Stop and restart the OKC's Tomcat server.

2.9.OKC Administrator

For OKC administrator OKC portal provides environment to manage security, user and group roles and to watch the current services provided by OKC portal. There might be more than one admin if desired. When an admin enters to the portal he/she will come up a web page seen in figure 7. This is explained in more detailed below.

2.9.1. User Browser

Admin can see the current users in OKC portal and can add more users to the portal. He/She can delete the user, edit their name, password and other information and can easily see the last login date of the user.



The screenshot shows the OKC administrator interface. At the top, there is a header with the OKC logo and navigation links: Home, Customize, Logout, and Edit account: (admin). Below the header is a date indicator: March 28, 2002. The main content area is divided into a left sidebar and a main table. The sidebar contains a menu with 'Security' selected. The main table is titled 'User Browser' and contains the following data:

Username	First Name	Last Name	Email	Actions
turbine	OKC	Turbine	okcadmin@grids.ucs.indiana.edu	Edit Roles Remove
admin	Jetspeed	Admin	okcadmin@grids.ucs.indiana.edu	Edit Roles Remove
bzilliox	Bill	Zilliox	willaim.zilliox@wpafb.af.mil	Edit Roles Remove
sotireli	Paul	Sotirelis	psotirelis@hpti.com	Edit Roles Remove

Figure 7 The OKC administrator page can be used to add, remove, and modify user accounts.

2.9.2. Security Role Browser

OKC Portal security has a role based structure. Admin can assign roles to a specific user or a group and define their roles. Roles are defined for a certain permissions and admin restrict some of them or give permission for that specific users. Security Role Browser screen can be found in Figure 8 and permission names are given below.

- close
- customize
- detach
- info
- maximize
- minimize
- personalize
- view

So for the OKC Portal admins can define the roles of a user or a group by using security role browser.

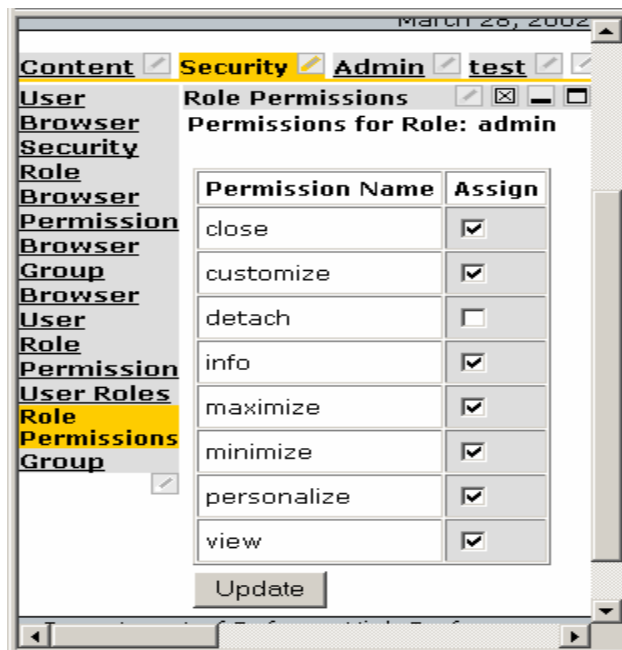


Figure 8 Administrators can edit user permissions through the illustrated interface.

2.9.3. Permission Browser

For the permission browser admin can remove some permissions such as close, view, customize and others.

1. Group Browser: By using group browser, admin can see the current groups and add more groups if desired.
2. User Role: In user role a new user can be added to the system.
3. Role: By using role browser a new role name can be added and by using security role browser specific roles can be assigned to the role.
4. Permission Name: Permission name is given here
5. User Roles: User roles are defined here.
6. Role Permissions: Role permissions for a specific user is given.
7. Group: A new group is defined here.

In Online Knowledge Center, these tools can be used to give specific roles to a certain user or by defining groups for each focus areas in OKC the security roles for OKC can be handled. Let's say we do not want CE users to see the pages for other areas we can define a group called CE and give certain permission roles to that specific group.

Administrators can also audit and check the OKC portal for the following properties. These are related with portlets and portal title, color, controller browser types, etc.:

Global

Portlet browser

Controller browser

Control browser

Media Types browser

Skin browser

Java Runtime

Daemons

Jetspeed Content

BadURLManager

2.10. OKC Users

New OKC user environments are created from the default user, turbine. When a new users signs in and enter the portal, he or she sees exactly the same thing with the turbine user does, because when a user signs in the OKC portal automaticaly copies the PSML file from the turbine user. But there is also option for us not to copy the same structure from the turbine user by configuration files.

An OKC user depending on its roles as explained in OKC administrator part can customize, minimize, personalize, etc., his or her views. If a user has permission he or she can customize the portal as he/she wants. But keep in mind administrators can restrict this if wanted.

Here is an explanation of the customize process and how a user can add a portlet:

1. Sign in to the portal
2. Click the customize link on the top
3. In customize page (Figure 9) click any layout other than pane and Add Portlet link appear
4. User can choose any of the available portlets from the portlets page (Figure 10)
5. After choosing the portlets just click the add button and for the next page just click the apply button.
6. The last step to choose the layout of the page whether single row, pane or something else from the layout menu. Also color of the portlets can be chosen from the customizer page.

It is also for a OKC user to update his or her password or other information through edit page.

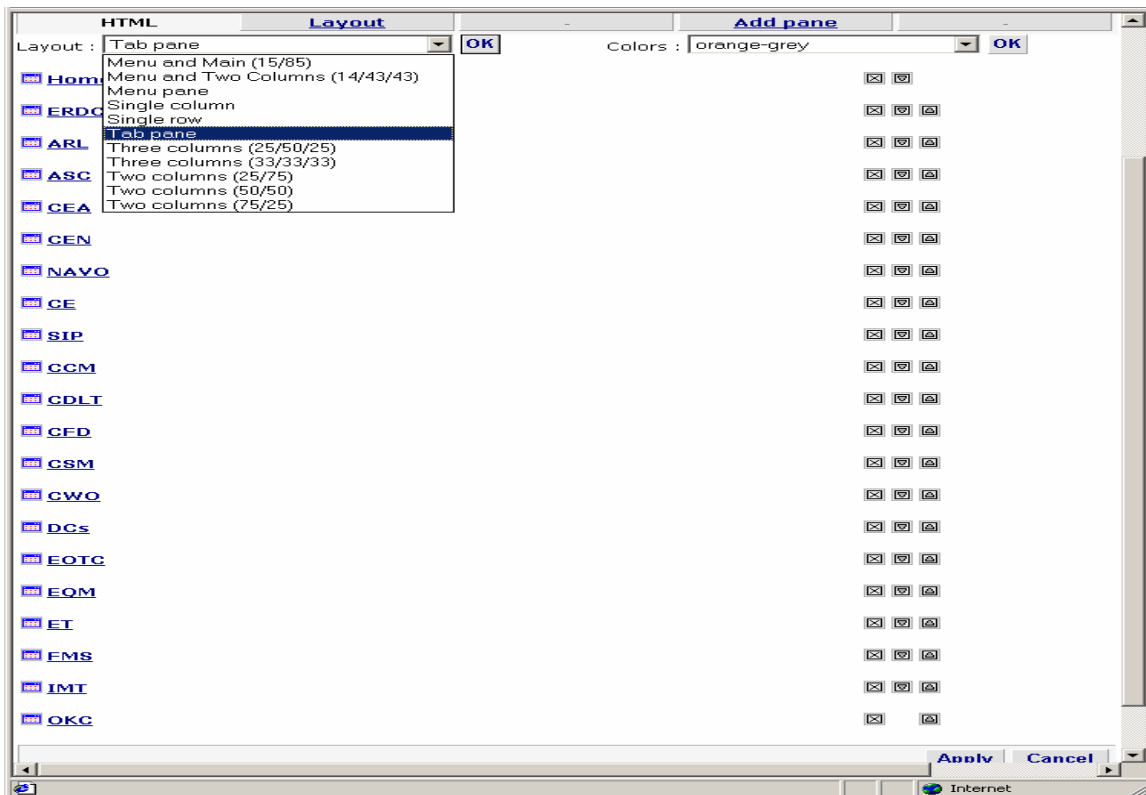


Figure 9 OKC allows users to customize their portal interface.

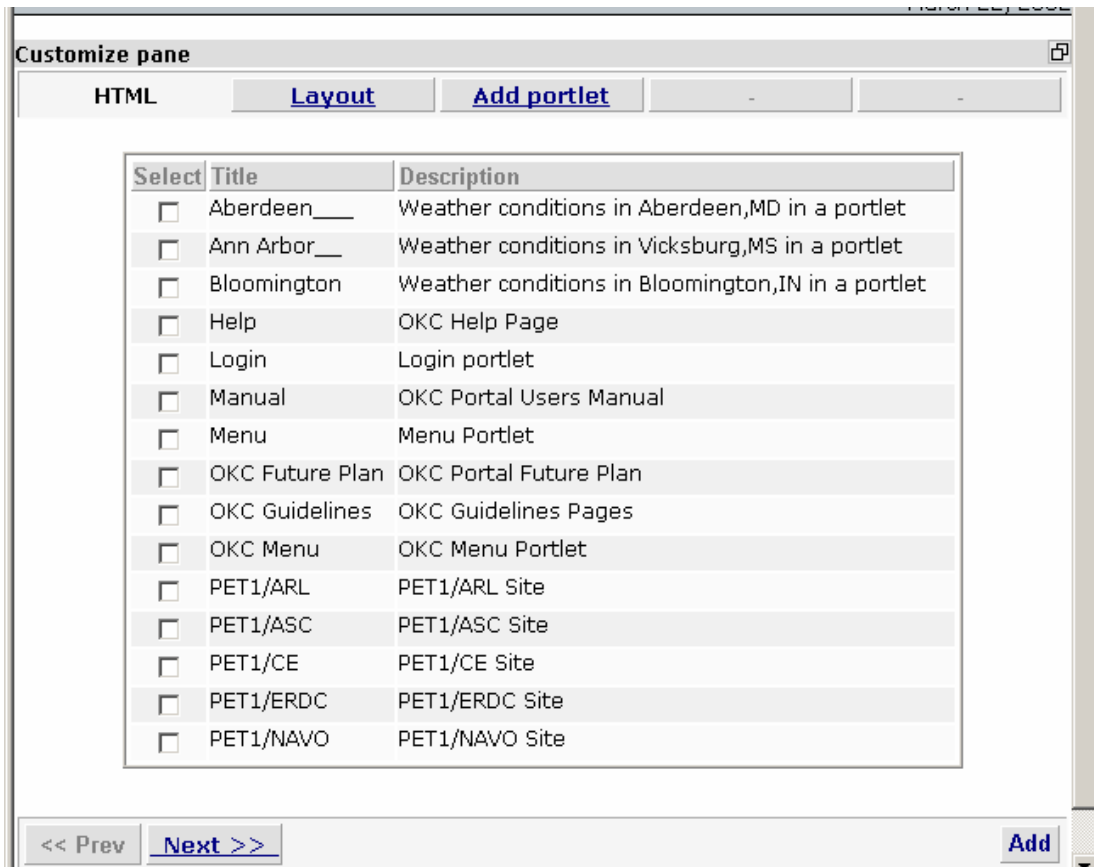


Figure 10 Users can choose the portlets that they wish to display

2.11. Netscape Navigator 4.x Problems

For Netscape Navigator 4.x users there is a problem to see the Jetspeed. This is because of the way of Netscape 4.X versions deal with nested tables. So far it is not possible to see OKC portal with Netscape 4.x. When the size of nested tables exceed more than ten NN does not render it correctly. Also, large nested tables can cause problems. We are currently working on velocity files in order to minimize the tables.

When NN 4.X loads a Web with a table layout structure, it may suddenly just "give up" and display nothing. This is because of two things:

1. The most common is that a `<td>`, `</td>`, `<tr>` or `</tr>` tag is out of place or missing. This can easily occur in tables with unequal rows or columns. IE is much more tolerant of incorrect table syntax than NN.
2. The second problem could be that you have multiple nested tables (tables within tables) and Netscape just gives up rendering them after awhile.

Simplying the Jetspeed template may reduce the number of tables needed to create the portal layout. Jetspeed.vm creates a table to that table that contains the portlet title bar and a table that contains the content. Nesting the tables in this way greatest flexibility. It might be also possible to replace tables with some other HTML. Also by placing the icons before the title and minimizing the number of columns displayed, you may be able to remove the tables altogether.

This simplification may reduce the useable functionality, that is nested tabbed panes may not display correctly.

2.12. OKC Database

Jetspeed stores information about users in a database. Stored information includes

Username

Password

User's first name

User's last Name

e-mail address

Last login time

The default database for Jetspeed is Hypersonic SQL. To change the database connection, it is needed to update the database connection settings in TurbineResources.properties file. The war file installation automatically configures Hypersonic SQL is the default database. In our case it is changed to Oracle for and the step to follow to switch to Oracle is given below. The links to switch to Oracle is given in resources section as a link (number 3).

Jetspeed should work with any JDBC 2.0 complaint driver such as.

DB2

Hypersonic SQL

MySQL

Oracle

Postgres

Sybase

2.13. Lessons Learned

As explained before there are seven main steps for adding a web page to the OKC. Plain HTML pages, XML or JavaScript pages can be added and display on portal without any problem as long as they are syntactically correct. If there are errors on the original page the OKC portal would not display them properly. The issue here is IE, NN or any other browsers will fix the errors and display them without any error notification. In order to check if your site is well formed or not you can test it from IE by choosing Internet Options from Tools and choose not to debug from Advanced features. The most common mistake when using JavaScript is forgotten to add semicolon (;). This type of HTML syntax errors can be validated by W3 web site. <http://validator.w3.org/check/referer>

In the OKC portal SIP, CE pages are plain HTML pages and they have been added to the portal easily. For ASC pages JavaScript has been used and there have been some syntax problems. For example, the semicolon has been forgotten in some lines and they have been corrected. For CEA and CEN pages there is a css file for style sheet and this has overridden by the portal's default css file this also has been fixed by defining a separate css file for CEN.

NAVO pages are actually cgi forms which some of them are generated from Perl script. Tomcat had problems to handle this situation so we installed Apache and use it for NAVO pages to display at the background.

Netscape Navigator 4 versions have problem when rendering nested tables. There are display problems when nested tables reach to ten or more. Sometimes for three nested table can cause display problem in NN 4.X when the size of each table is big. The next effort will be to solve this problem and come up with a solution.

2.14. Resources and Useful Links

1. Apache Jakarta project Jetspeed web site
<http://jakarta.apache.org/jetspeed/site/index.html>
2. One of the Developers' web site
<http://www.bluesunrise.com/jetspeed-docs/>
3. Oracle database connection
<http://www.lionelfarr.com/jetspeed/oracleHowTo.htm>
4. HP Trial map site
<http://mml.hpl.hp.com:9090/jetspeed/trailmap/index.html>
5. Release builds at <http://jakarta.apache.org/builds/jakarta-jetspeed/release/>
6. Nightly builds at <http://jakarta.apache.org/builds/jakarta-jetspeed/nightly/>
7. Jetspeed users archives
<http://www.mail-archive.com/jetspeed-user%40jakarta.apache.org/>

2.15. Appendix

ECS: The Element Construction Set is a Java API for generating elements for various markup languages it directly supports HTML 4.0 and XML.

PSML: Stands for Portal Structure Markup Language

RSS: Rich Site Summary (RSS) is an XML-based format developed by Netscape to drive channels for Netscape Netcenter. The goal of RSS is to propagate change on a site, including your Jetspeed site, to increase traffic. Using an RSS file, all you need to do is change one file, and all the external sites your channels are registered with will automatically reflect those changes. More information can be found in the <http://my.netscape.com/publish/formats/rss-spec-0.91.html>

Turbine: Turbine is a servlet based framework that allows experienced Java developers to quickly build secure web applications. Parts of Turbine can also be used independently of the web portion of Turbine as well.

Velocity: Velocity is a Java-based template engine. It permits anyone to use the simple yet powerful template language to reference objects defined in Java code. It is syntactically alternative to JSP.

2.16. Necessary Files and Scripts

2.16.1. *okcproperties.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://grids.ucs.indiana.edu/schema/okc/config/v1" version="0.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://grids.ucs.indiana.edu/schema/okc/config/v1
http://grids.ucs.indiana.edu/schemas/okcconfiguration-v1.xsd">
  <!-- contenthost>http://localhost</contenthost -->
  <!-- contenthost>http://montblanc.ucs.indiana.edu</contenthost -->
  <contenthost>http://judi.ucs.indiana.edu</contenthost>
  <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
  <!-- mediaserver>http://montblanc.ucs.indiana.edu/mediaserver/fs</mediaserver -->
  <mediaserver>http://judi.ucs.indiana.edu/mediaserver/fs</mediaserver>
  <stemlinks>stemlinks.xml</stemlinks>
  <newbrowser>
    <extention type="ppt">Microsoft Power Point Files</extention>
    <extention type="doc">Microsoft Word documents</extention>
```

```

<extention type="pdf">Adove PDF files</extention>
<extention type="css">Cascade Style Sheets</extention>
<extention type="ps">Adobe PostScript Files</extention>
<extention type="gif">GIF files</extention>
<extention type="jpg">JPEG files</extention>
<extention type="jpeg">JPEG files</extention>
<extention type="mpg">MPEG files</extention>
<extention type="mpeg">MPEG files</extention>
<extention type="avi">AVI files</extention>
<extention type="swf">Flash files</extention>
<extention type="png">Image files</extention>
<extention type="ram">Real Media files</extention>
<extention type="rm">Real Media files</extention>
</newbrowser>
<areas>
  <area>
    <name>tests</name>
    <description/>
    <stem>/okc/rawpages/testcases</stem>
    <mapto>/var/tomcat/webapps/ROOT/rawpages/testcases</mapto>
  </area>
  <area>
    <name>pet</name>
    <description/>
    <!-- contenthost>http://localhost</contenthost -->
    <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
    <stem>/okc/rawpages/pet</stem>
    <mapto>/var/tomcat/webapps/ROOT/rawpages/pet</mapto>
    <!-- portal panename="PET1"/ -->
    <!-- stemlinks>stemlinks.xml</stemlinks -->
  </area>
  <area>
    <name>training</name>
    <description/>

```

```

        <!-- contenthost>http://localhost</contenthost -->
        <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
        <stem>/okc/rawpages/training</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/training</mapto>
        <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <area>
        <name>arl</name>
        <description/>
        <!-- contenthost>http://localhost</contenthost -->
        <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
        <stem>/okc/rawpages/ARL/PET</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/ARL/PET</mapto>
        <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <area>
        <name>asc</name>
        <description/>
        <!-- contenthost>http://localhost</contenthost -->
        <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
        <stem>/okc/rawpages/ASC</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/ASC</mapto>
        <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <area>
        <name>navo</name>
        <description/>
        <contenthost>http://judi.ucs.indiana.edu:1080</contenthost>
        <mediaserver>http://judi.ucs.indiana.edu:1080</mediaserver>
        <stem>/okc/pet</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/navo/pet</mapto>
        <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <area>

```

```

        <name>navo-cgi</name>
        <description/>
        <contenthost>http://judi.ucs.indiana.edu:1080</contenthost>
        <mediaserver>http://judi.ucs.indiana.edu:1080</mediaserver>
        <stem>/okc/cgi-bin/pet</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/navo/pet</mapto>
        <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <area>
        <name>asc-techareas</name>
        <description/>
        <stem>/okc/rawpages/ASC/Techareas</stem>
        <mapto>/home/okcadmin/webcontent/ASC/Techareas</mapto>
    </area>
    <area>
        <name>gateway</name>
        <description/>
        <!-- contenthost>http://localhost</contenthost -->
        <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
        <stem>/okc/rawpages/gateway</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/gateway</mapto>
        <!-- portal panename="PET2"/ -->
        <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <!-- Modified area starting on 01/16/2002 by Necati -->
    <!-- <area>
        <name>asc-cea</name>
        <description/>
        <stem>/okc/rawpages/ASC/CEA</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/ASC/CEA</mapto>
    </area> -->
    <area>
        <name>asc-cen</name>
        <description/>

```

```

        <!-- contenthost>http://localhost</contenthost -->
        <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
        <stem>/okc/rawpages/ASC/CEN</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/ASC/CEN</mapto>
        <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <area>
        <name>asc-cfd</name>
        <description/>
        <!-- contenthost>http://localhost</contenthost -->
        <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
        <stem>/okc/rawpages/ASC/CFD</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/ASC/CFD</mapto>
        <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <area>
        <name>asc-ic</name>
        <description/>
        <!-- contenthost>http://localhost</contenthost -->
        <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
        <stem>/okc/rawpages/ASC/IC</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/ASC/IC</mapto>
        <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <area>
        <name>asc-ptes</name>
        <description/>
        <!-- contenthost>http://localhost</contenthost -->
        <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
        <stem>/okc/rawpages/ASC/PTES</stem>
        <mapto>/var/tomcat/webapps/ROOT/rawpages/ASC/PTES</mapto>
        <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <area>

```

```

    <name>asc-sv</name>
    <description/>
    <!-- contenthost>http://localhost</contenthost -->
    <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
    <stem>/okc/rawpages/ASC/SV</stem>
    <mapto>/var/tomcat/webapps/ROOT/rawpages/ASC/SV</mapto>
    <!-- stemlinks>stemlinks.xml</stemlinks -->
</area>
<!-- Modified area ending on 01/16/2002 by Necati -->
<area>
    <name>pet2-cea</name>
    <description>Computational Electromagnetics and Acoustics</description>
    <!-- contenthost>http://localhost</contenthost -->
    <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
    <stem>/okc/rawpages/complete/CEA</stem>
    <mapto>/var/tomcat/webapps/ROOT/rawpages/complete/CEA</mapto>
    <!-- stemlinks>stemlinks.xml</stemlinks -->
</area>
<area>
    <name>pet2-cen</name>
    <description>Computational Electronics and Nanoelectronics</description>
    <!-- contenthost>http://localhost</contenthost -->
    <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
    <stem>/okc/rawpages/complete/CEN</stem>
    <mapto>/var/tomcat/webapps/ROOT/rawpages/complete/CEN</mapto>
    <!-- stemlinks>stemlinks.xml</stemlinks -->
</area>
<area>
    <name>documentation</name>
    <description>OKC Documentation</description>
    <!-- contenthost>http://localhost</contenthost -->
    <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
    <stem>/okc/okcportal/okc</stem>
    <mapto>/var/tomcat/webapps/okcportal/okc</mapto>

```



```

        <!-- stemlinks>stemlinks.xml</stemlinks -->
</area>
<area>
    <name>errormessages</name>
    <description>Computational Electronics and Nanoelectronics</description>
    <!-- contenthost>http://localhost</contenthost -->
    <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
    <stem>/okc/rawpages/errors</stem>
    <mapto>/var/tomcat/webapps/ROOT/rawpages/errors</mapto>
    <!-- stemlinks>stemlinks.xml</stemlinks -->
</area>

<area>
    <name>sip</name>
    <description>DescSIP</description>
    <!-- contenthost>http://localhost</contenthost -->
    <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
    <stem>/okc/rawpages/SIP</stem>
    <mapto>/var/tomcat/webapps/ROOT/rawpages/SIP</mapto>
    <!-- stemlinks>stemlinks.xml</stemlinks -->
</area>

<area>
    <name>ce</name>
    <description>DescriCE</description>
    <!-- contenthost>http://localhost</contenthost -->
    <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
    <stem>/okc/rawpages/CE</stem>
    <mapto>/var/tomcat/webapps/ROOT/rawpages/CE</mapto>
    <!-- stemlinks>stemlinks.xml</stemlinks -->
</area>

<!-- Modified area start point on 01/29/2002 MNA-->

```

```
<area>
  <name>asc-lecture1a</name>
  <description/>
  <!-- contenthost>http://localhost</contenthost -->
  <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
  <stem>/okc/rawpages/ASC/CCM/training/courses/ACESII/lecture1a</stem>
```

```
<mapto>/var/tomcat/webapps/ROOT/rawpages/ASC/CCM/training/courses/ACESII/lecture1a<
/mapto>
```

```
  <!-- stemlinks>stemlinks.xml</stemlinks -->
</area>
```

```
<area>
  <name>asc-lecture2</name>
  <description/>
  <!-- contenthost>http://localhost</contenthost -->
  <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
  <stem>/okc/rawpages/ASC/CCM/training/courses/ACESII/lecture2</stem>
```

```
<mapto>/var/tomcat/webapps/ROOT/rawpages/ASC/CCM/training/courses/ACESII/lecture2</
mapto>
```

```
  <!-- stemlinks>stemlinks.xml</stemlinks -->
</area>
```

```
<area>
  <name>asc-lecture4</name>
  <description/>
  <!-- contenthost>http://localhost</contenthost -->
  <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
  <stem>/okc/rawpages/ASC/CCM/training/courses/ACESII/lecture4-
```

```
frame</stem>
```

```
<mapto>/var/tomcat/webapps/ROOT/rawpages/ASC/CCM/training/courses/ACESII/lecture4-
frame</mapto>
```

```
  <!-- stemlinks>stemlinks.xml</stemlinks -->
</area>
```

```

    <area>
      <name>asc-lecture5</name>
      <description/>
      <!-- contenthost>http://localhost</contenthost -->
      <!-- mediaserver>http://localhost/mediaserver/fs</mediaserver -->
      <stem>/okc/rawpages/ASC/CCM/training/courses/ACESII/lecture5-
frame</stem>

    <mapto>/var/tomcat/webapps/ROOT/rawpages/ASC/CCM/training/courses/ACESII/lecture5-
frame</mapto>

      <!-- stemlinks>stemlinks.xml</stemlinks -->
    </area>
    <!-- Modified area end point on 01/29/2002 MNA-->
  </areas>
  <errors>
    <!-- message>Page is out of area boundaries!</message -->
    <url>/okc/rawpages/errors/notavailable.html</url>
  </errors>
</configuration>

```

2.16.2. *okc-portlets.xreg*

```

<?xml version="1.0" encoding="UTF-8"?>
<registry>
  <portlet-entry name="Welcome" hidden="false" type="ref"
    parent="HTML" application="false">
    <meta-info>
      <title>Welcome</title>
    </meta-info>
    <parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
    <url>/welcome.html</url>
  </portlet-entry>
  <portlet-entry name="Help" hidden="false" type="ref" parent="HTML" application="false">
    <meta-info>
      <title>Help</title>

```

```

    <description>OKC Help Page </description>
</meta-info>
<parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
<url>/BugReport.html</url>
</portlet-entry>
<portlet-entry name="BloomingtonWeather" hidden="false" type="ref"
parent="CustomizerVelocity" application="false">
<meta-info>
    <title>Bloomington</title>
    <description>Weather conditions in Bloomington,IN in a portlet</description>
</meta-info>
<parameter name="template" value="weather" hidden="false"/>
<parameter name="action" value="okc.WeatherAction" hidden="false"/>
<parameter name="weather_city" value="Bloomington" hidden="false"/>
<parameter name="weather_state" value="IN" hidden="false"/>
<parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
<media-type ref="html"/>
</portlet-entry>
<portlet-entry name="VicksburgWeather" hidden="false" type="ref"
parent="CustomizerVelocity" application="false">
<meta-info>
    <title>Vicksburg</title>
    <description>Weather conditions in Vicksburg,MS in a portlet</description>
</meta-info>
<parameter name="template" value="weather" hidden="false"/>
<parameter name="action" value="okc.WeatherAction" hidden="false"/>
<parameter name="weather_city" value="Vicksburg" hidden="false"/>
<parameter name="weather_state" value="MS" hidden="false"/>
<parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
<media-type ref="html"/>
</portlet-entry>
<portlet-entry name="AberdeenWeather" hidden="false" type="ref"
parent="CustomizerVelocity" application="false">
<meta-info>

```

```

    <title>Aberdeen__</title>
    <description>Weather conditions in Aberdeen,MD in a portlet</description>
</meta-info>
<parameter name="template" value="weather" hidden="false"/>
<parameter name="action" value="okc.WeatherAction" hidden="false"/>
<parameter name="weather_city" value="Aberdeen" hidden="false"/>
<parameter name="weather_state" value="MD" hidden="false"/>
<parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
<media-type ref="html"/>
</portlet-entry>
<portlet-entry name="AnnArborWeather" hidden="false" type="ref"
parent="CustomizerVelocity" application="false">
  <meta-info>
    <title>Ann Arbor__</title>
    <description>Weather conditions in Vicksburg,MS in a portlet</description>
  </meta-info>
  <parameter name="template" value="weather" hidden="false"/>
  <parameter name="action" value="okc.WeatherAction" hidden="false"/>
  <parameter name="weather_city" value="Ann_Arbor" hidden="false"/>
  <parameter name="weather_state" value="MI" hidden="false"/>
  <parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
  <media-type ref="html"/>
</portlet-entry>
<portlet-entry name="PET1_ERDC" hidden="false" type="ref"
parent="OKCWebPagePortlet" application="false">
  <meta-info>
    <title>PET1/ERDC</title>
    <description>PET1/ERDC Site</description>
  </meta-info>
  <parameter name="url_variable" value="peterdc_url" hidden="true"/>
  <parameter name="home_url" value="/rawpages/pet/pet_bd.htm" hidden="false"/>
  <parameter name="title" value="PET1/ERDC" hidden="false"/>
  <url>/rawpages/pet/pet_bd.htm</url>
</portlet-entry>

```

```

<portlet-entry name="PET1_ARL" hidden="false" type="ref"
  parent="OKCWebPagePortlet" application="false">
  <meta-info>
    <title>PET1/ARL</title>
    <description>PET1/ARL Site</description>
  </meta-info>
  <parameter name="url_variable" value="petarl_url" hidden="true"/>
  <parameter name="home_url" value="/rawpages/ARL/PET/pet.html" hidden="false"/>
  <parameter name="title" value="PET1/ARL" hidden="false"/>
  <url>/rawpages/ARL/PET/pet.html</url>
</portlet-entry>
<portlet-entry name="PET1_ASC" hidden="false" type="ref"
  parent="OKCWebPagePortlet" application="false">
  <meta-info>
    <title>PET1/ASC</title>
    <description>PET1/ASC Site</description>
  </meta-info>
  <parameter name="url_variable" value="petasc_url" hidden="true"/>
  <parameter name="home_url" value="/rawpages/ASC/home.html" hidden="false"/>
  <parameter name="title" value="PET1/ASC" hidden="false"/>
  <url>/rawpages/ASC/home.html</url>
</portlet-entry>
<portlet-entry name="PET1_NAVO" hidden="false" type="ref"
  parent="OKCWebPagePortlet" application="false">
  <meta-info>
    <title>PET1/NAVO</title>
    <description>PET1/NAVO Site</description>
  </meta-info>
  <parameter name="url_variable" value="petnavo_url" hidden="true"/>
  <parameter name="home_url" value="/pet/pet.cgi" hidden="false"/>
  <parameter name="title" value="PET1/NAVO" hidden="false"/>
  <url>/pet/pet.cgi</url>
</portlet-entry>
<portlet-entry name="PET2_CEN" hidden="false" type="ref"

```

```

parent="OKCWebPagePortlet" application="false">
<meta-info>
  <title>PET2/CEN</title>
  <description>PET2/CEN Site</description>
</meta-info>
<parameter name="url_variable" value="pet2cen_url" hidden="true"/>
<parameter name="home_url"
  value="/rawpages/complete/CEN/index1.html" hidden="false"/>
<parameter name="title" value="PET2/CEN" hidden="false"/>
<url>/rawpages/complete/CEN/index1.html</url>
</portlet-entry>
<portlet-entry name="PET2_CEA" hidden="false" type="ref"
parent="OKCWebPagePortlet" application="false">
<meta-info>
  <title>PET2/CEA</title>
  <description>PET2/CEA Site</description>
</meta-info>
<parameter name="url_variable" value="pet2cea_url" hidden="true"/>
<parameter name="home_url"
  value="/rawpages/complete/CEA/index1.html" hidden="false"/>
<parameter name="title" value="PET2/CEA" hidden="false"/>
<url>/rawpages/complete/CEA/index1.html</url>
</portlet-entry>
<portlet-entry name="PET1_CE" hidden="false" type="ref"
parent="OKCWebPagePortlet" application="false">
<meta-info>
  <title>PET1/CE</title>
  <description>PET1/CE Site</description>
</meta-info>
<parameter name="url_variable" value="pet2ce_url" hidden="true"/>
<parameter name="home_url" value="/rawpages/CE/CE.htm" hidden="false"/>
<parameter name="title" value="PET2/CE" hidden="false"/>
<url>/rawpages/CE/CE.htm</url>
</portlet-entry>

```

```

<portlet-entry name="PET1_SIP" hidden="false" type="ref"
  parent="OKCWebPagePortlet" application="false">
  <meta-info>
    <title>PET1/SIP</title>
    <description>PET1/SIP Site</description>
  </meta-info>
  <parameter name="url_variable" value="pet1sip_url" hidden="true"/>
  <parameter name="home_url" value="/rawpages/SIP/index.html" hidden="false"/>
  <parameter name="title" value="PET1/SIP" hidden="false"/>
  <url>/rawpages/SIP/index.html</url>
</portlet-entry>
<portlet-entry name="OKCMenu" hidden="false" type="ref"
  parent="GridsJSP" application="false">
  <meta-info>
    <title>OKC Menu</title>
    <description>OKC Menu Portlet</description>
  </meta-info>
  <parameter name="template" value="okcmenu.jsp" hidden="false"/>
  <parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
  <media-type ref="html"/>
</portlet-entry>
<portlet-entry name="JSPMenu" hidden="false" type="ref"
  parent="GridsJSP" application="false">
  <meta-info>
    <title>Menu</title>
    <description>Menu Portlet</description>
  </meta-info>
  <parameter name="template" value="jspmenu.jsp" hidden="false"/>
  <parameter name="_enableactions" value="false" type="boolean" hidden="true"/>
  <media-type ref="html"/>
</portlet-entry>
<portlet-entry name="OKCGuidelines" hidden="false" type="ref"
  parent="GridsWebPagePortlet" application="false">
  <meta-info>

```



```

    <title>OKC Guidelines</title>
    <description>OKC Guidelines Pages</description>
</meta-info>
<parameter name="new_url" value="/okcportal/okc/guidelines.html" hidden="true"/>
<parameter name="url_variable" value="okcdoc_url" hidden="true"/>
<url>/okcportal/okc/guidelines.html</url>
</portlet-entry>
<portlet-entry name="Manual" hidden="false" type="ref"
parent="GridsWebPagePortlet" application="false">
<meta-info>
    <title>Manual</title>
    <description>OKC Portal Users Manual</description>
</meta-info>
<parameter name="new_url" value="/okcportal/okc/manual.html" hidden="true"/>
<parameter name="url_variable" value="okcman_url" hidden="true"/>
<url>/okcportal/okc/manual.html</url>
</portlet-entry>
<portlet-entry name="OKCStatus" hidden="false" type="ref"
parent="GridsWebPagePortlet" application="false">
<meta-info>
    <title>OKC Future Plan</title>
    <description>OKC Portal Future Plan</description>
</meta-info>
<parameter name="new_url" value="/okcportal/okc/status.html" hidden="true"/>
<parameter name="url_variable" value="okcplan_url" hidden="true"/>
<url>/okcportal/okc/status.html</url>
</portlet-entry>
<portlet-entry name="UnderCons" hidden="false" type="ref"
parent="OKCWebPagePortlet" application="false">
<meta-info>
    <title>UC</title>
    <description>Under Construction Site</description>
</meta-info>
<parameter name="url_variable" value="petuc_url" hidden="true"/>

```

```

    <parameter name="home_url"
      value="/rawpages/errors/construction.html" hidden="false"/>
    <parameter name="title" value="UC" hidden="false"/>
    <url>/rawpages/errors/construction.html</url>
  </portlet-entry>
  <portlet-entry name="Review" hidden="false" type="ref"
    parent="OKCWebPagePortlet" application="false">
    <meta-info>
      <title>Review</title>
      <description>Under Review Site</description>
    </meta-info>
    <parameter name="url_variable" value="petrv_url" hidden="true"/>
    <parameter name="home_url" value="/rawpages/errors/review.html" hidden="false"/>
    <parameter name="title" value="URW" hidden="false"/>
    <url>/rawpages/errors/review.html</url>
  </portlet-entry>
</registry>

```

2.16.3. okcmenu.jsp

```

<%@
taglib uri="/WEB-INF/templates/jsp/tld/template.tld" prefix="jetspeed" %><%@
page import="org.gxos.debug.Log, commgrids.stemlinks.schema.*" %><%@
page import="commgrids.stemlinks.schema.*,commgrids.okc.config.*" %><%@
page import="commgrids.okc.*,commgrids.okc.portal.util.Pane" %><%@
page import="java.util.*,java.net.URL,java.io.*,javax.naming.*" %><%@
page import="org.apache.jetspeed.services.resources.JetspeedResources" %><%@
page import="org.apache.jetspeed.util.URILookup" %><%@
page import="org.apache.jetspeed.services.rundata.*"
%><%!
    static {
        OKC.initLogger();
    }
%><%

boolean error = false;
JetspeedRunData rundata = (JetspeedRunData)request.getAttribute("rundata");

```

```

Configuration configuration = OKC.getConfiguration(rundata);
Hashtable areas = OKC.getAreas(rundata);
String homePath = URILookup.getURI(URILookup.TYPE_HOME,URILookup.SUBTYPE_NONE,
rundata);
String paneTitle = rundata.getParameters().getString("pane0");

Vector paths = (Vector)(rundata.getSession().getAttribute("paths_vector"));
Vector panes = (Vector)(rundata.getSession().getAttribute("panes_vector"));
if(panes == null) {
    panes = new Vector();
    panes.add("ERDC");
    panes.add("ARL");
    panes.add("ASC");
    panes.add("NAVO");
    panes.add("CEA");
    panes.add("CEN");
    panes.add("CE");
    panes.add("SIP");

    paths = new Vector();
    paths.add(new Pane((String)panes.get(0),"/rawpages/pet/pet_bd.htm","peterdc_url"));
    paths.add(new Pane((String)panes.get(1),"/rawpages/ARL/PET/pet.html","petarl_url"));
    paths.add(new Pane((String)panes.get(2),"/rawpages/ASC/home.html","petasc_url"));
    paths.add(new Pane((String)panes.get(3),"/pet","petnavo_url"));
    paths.add(new Pane((String)panes.get(4),"/rawpages/complete/CEA/index1.html","pet2cea_url"));
    paths.add(new Pane((String)panes.get(5),"/rawpages/complete/CEN/index1.html","pet2cen_url"));
    paths.add(new Pane((String)panes.get(6),"/rawpages/CE/CE.htm","pet2ce_url"));
    paths.add(new Pane((String)panes.get(7),"/rawpages/SIP/index.html","pet1sip_url"));

    rundata.getSession().setAttribute("panes_vector",panes);
    rundata.getSession().setAttribute("paths_vector",paths);
}

```

```
}
```

```
String menu = (String)(rundata.getSession().getAttribute("okc_menu_items"));
if(menu == null) {
    StringBuffer sb = new StringBuffer(200);
    // construct the main menu
    sb.append("&nbsp;\n");
    for(int i=0; i<paths.size(); i++) {
        Pane p = (Pane)paths.get(i);
        sb.append("<DT><A HREF=\"").append(homePath).append("/pane0/").
            append(p.name).append("?").append(p.variable).append("=").
            append(p.url).append("&okcurl=").append(p.variable).
            append("\ ">").append(p.name).append(" Home</a>\n");
    }
    menu = sb.toString();
    rundata.getSession().setAttribute("okc_menu_items",menu);
}
```

```
OKC.getLogger().debug("[ "+rundata.getUserFromSession().getUserName()+" ] from "+
    rundata.getRemoteHost()+" "+rundata.getRemoteAddr());
```

```
//URL okcreceiver = new URL(new URL(homePath), "../..../okcreceiver");
```

```
//URL metadata = new URL(new URL(homePath), "../..../metadata");
```

```
%><%=menu%><%
```

```
/*OKC.getLogger().debug("----- rundata parameters");
```

```
for(int i=0; i<rundata.getParameters().getKeys().length; i++) {
```

```
    String name=(String)rundata.getParameters().getKeys()[i];
```

```
    OKC.getLogger().debug( name+" --> "+rundata.getParameters().get(name));
```

```
}
```

```
    OKC.getLogger().debug("----- session parameters");
```

```
for(Enumeration e = rundata.getSession().getAttributeNames(); e.hasMoreElements(); ) {
```

```
    String name=(String)e.nextElement();
```

```

    OKC.getLogger().debug( name+" --> "+rundata.getSession().getAttribute(name).toString() );
}/*
System.out.println("-----");
for(Enumeration e = rundata.getServletContext().getAttributeNames(); e.hasMoreElements(); ) {
    String name=(String)e.nextElement();
    System.out.println( name+" --> "+rundata.getServletContext().getAttribute(name).toString() );
}
    OKC.getLogger().debug( "----- request parameters");
for(Enumeration e = request.getAttributeNames(); e.hasMoreElements(); ) {
    String name=(String)e.nextElement();
    OKC.getLogger().debug( name+" --> "+request.getAttribute(name).toString() );
}*/

// first find out our navigation variable
String urlVar = request.getParameter("okcurl");
int pos=-1;
if(urlVar == null) {
    //urlVar = (String)rundata.getSession().getAttribute("okcurl");
    urlVar = (String)rundata.getSession().getAttribute(paneTitle);
    if(urlVar == null) {
        pos = panes.indexOf(paneTitle);
        if(pos>=0)
            urlVar = ((Pane)(paths.get(pos))).variable;
        else
            urlVar = "okc_url";
    }
}

// find the page URL
String okc_url = request.getParameter(urlVar);
if( okc_url == null ) {
    okc_url=(String)rundata.getSession().getAttribute(urlVar);
    if(okc_url == null && pos>=0)
        okc_url = ((Pane)(paths.get(pos))).url;
}

```

```

if(okc_url != null) {
    //rundata.getSession().setAttribute("okcurl",urlVar);
    if(paneTitle != null)
        rundata.getSession().setAttribute(paneTitle,urlVar);

    rundata.getSession().setAttribute(urlVar,okc_url);
    String contenthost = null;

    Area area = AreaHandler.findArea( okc_url );
    if(area != null) {
        Section section = AreaHandler.getStems(area);
        contenthost = area.getContenthost();
        if(section != null) {
            Name base = new CompositeName(section.getBaseStem());
            if(base.size()<=0)
                OKC.getLogger().error("ERROR: Stem file has problems. baseStem attribute is invalid.");
            else {
                try {
                    // expected something like /okc/pet/...
                    // remove first okc
                    if(base.size()>2)
                        base.remove(1); // okc

%><p><b><%=section.getTitle()%></b></p>
<table cellpadding="0" cellspacing="0" border="0">
<%
                for(int i=0; i<section.getItemCount(); i++) {
                    // constructing the sub menu items
                    StringBuffer url = new StringBuffer( homePath );
                    URL uri = new URL(contenthost);
                    uri = new URL(uri, base.toString());
                    Item item = section.getItem(i);
                    if(item.getStem() != null) {
                        uri = new URL(uri, item.getStem());

```

```

        if(item.getLink() != null)
            uri = new URL(uri, item.getLink());
        String uri2 = uri.toString();
        uri2 = uri2.substring(contenthost.length());
        url.append("?").append(urlVar).append("=").
            append(uri2).append("&okcurl=").append(urlVar);
        //System.out.println("URL:"+url.toString());

%> <tr>
    <td width="10" valign="top" align="left">
        "
            width="7" height="10" border="0">
    </td>
    <td>
        <a href="<%=url.toString()%>"><%=item.getTitle()%></a>
    </td>
</tr>
<%
        }
    } // for
%></table><%
        } catch(Exception e) {
            OKC.getLogger().error("ERROR in construction sub menu", e);
        } // try-catch
    } // else
} // if section!=null
} //if area != null
} //if okc!= null
%>

```

2.16.4. *default.jsp*

```

<%@ taglib uri="/WEB-INF/templates/jsp/tld/template.tld" prefix='jetspeed' %>

<% String screenJsp = (String)request.getAttribute("screenJsp"); %>

```

```

<html>

<head>
<base href="<jetspeed:uriLookup type="BaseURL" />">
<link href="css/default.css" type="text/css" rel="stylesheet">
</head>

<body bgcolor="#ffffff">

<table cellspacing="0" width="100%" border="1" cellpadding="0">
<tr>
<td>
">
</td>
<td>
<jetspeed:navigation defaultTemplate="top_default.jsp" loggedInTemplate="top_loggedIn.jsp" />
</td>
</tr>
</table>

<table cellspacing="0" width="100%" cellpadding="0" border="0">
<tr>
<td valign="top" bgcolor="#ffffff">

<jetspeed:navigation defaultTemplate="left.jsp" />

<!-- Enable for Debugging ("Included servlet Error: 500")
Including:<%= screenJsp %> <p>
--%>
<jsp:include page="<%= screenJsp %>" flush="true" />

</td>

```



```
</tr>
</table>

<table cellspacing="0" width="100%" cellpadding="0" border="0">
  <tr>
    <td valign="bottom" bgcolor="#ffffff">
      <jetspeed:navigation defaultTemplate="html/bottom.jsp" />
    </td>
  </tr>
</table>
</body>

</html>
```

3. Jetspeed Development for OKC

3.1. Introduction

OKC Web Portal is based on Jetspeed, a Web portal from Apache Open Source Group, developed under the Jakarta project. Jetspeed provides user authentication and profiling screen layout management and configuration, HTML integration from different sources, and other various portal services.

The Jetspeed's architecture is simple and extendible. New features can be added or current ones can be modified easily. It is portable in a sense that it would work in any platform where a JSP engine is provided. A Java API for portlet development comes with the package, and it is simple and well documented.

OKC portal development team has considered Jetspeed as a candidate to implement distributed content management portal. However, further research and development was needed because of the weaknesses and bugs found in Jetspeed design and implementation to accomplish the team's goal. Here in this section we try to summarize the problems and the solutions we have developed and offered to the Jetspeed open source community.

3.2. Navigation Problem

Jetspeed serves as a thin Web interface. This means that Jetspeed retrieves XML and HTML content from different sources and present such content as a personalized portal page. On this page, besides Jetspeed's menu and navigation links, all the links point the sources of content. Any click on links inside the content takes users to the original page where the content is downloaded. For example, the portal page might contain a list of top news fetched from a news agency. A click on this link will take users to the news page on this agency's Web site. Similarly, the entire HTML page of another site can be presented as a portal page; however, any click on the links at this page will behave as if it was on the page's original location, and the portal's view will be replaced by a non-portal page.

OKC Portal presents HTML content from different sources while it keeps users navigating through the content within the portal environment. This is important because we want to keep usage statistics, have the control on users' actions, and provide users ability to customize their working environments, usage simplifications, simplicity and integrity in design.

3.3.Human Errors in HTML

XML and HTML content fetched from different sources must be processed first before forming the final portal page. The raw content as downloaded is not presented directly. During this process, XML content is transformed into HTML by XSL transformers. This process is performed on the portal side; hence the portal developers have the control of HTML output. However, when the HTML content is prepared by other parties, there is not much control on the HTML structure or syntax. Such content is prone to human errors. This lack of control enforces the portal developers to parse and verify pages downloaded from external sites. OKC Portal eliminates such structural or syntax errors in HTML to prevent unexpected behaviors from browsers.

3.4.HTML Integration Problems

All the HTML pages have their own elements and characteristics according to the context they are written for. The title information, metadata information, scripts, colors, font faces, styles, images, Java applets, ActiveX controls and many other technologies are widely used.

3.4.1. *Scripts*

Forming a new portal page from different HTML sources requires sacrifice of features which do not work with others quite well, or scripts which do not fit into the portal environment. For example, pages with JavaScript may not act as expected when integrated with the portal page. After the HTML integration, the URLs of pages, for example, change to the portal's URL and any absolute references to images or other resources within scripts cannot be detected during HTML processing mentioned in the previous section. If an image is referenced as `image_array[n].src = "file_name"` which assumes the image name is relative to the current page's location, when such a page is used to form the portal page, relativity is completely lost and scripts must be parsed so that the image URL can be rewritten as the absolute URL, i.e. `image_array[n].src = "http://content_host/path/file_name,"` which results overhead in portal computing.

OKC Portal resolves JavaScript image problem by providing a small script library which is available for all the portal pages. JavaScript writers can include this library functions in their applications. The library is written so that it does alter program behavior if used outside the portal.

3.4.2. Headers

Another integration problem results because of having multiple HTML headers from different sources for the content to be presented on a single HTML page. Jetspeed places different HTML content within TABLE elements when forming the portal page. If HTML headers are to be preserved, then HTML headers and META tags appear within the TABLE elements which result illegal HTML structure. Some Web browsers do not complain and interpret such header information as if it belongs to the page's header section, while others ignore wrongly placed tags. OKC Portal keeps title headers, moves style sheet links to the top level header section, and removes other header values because when there are more than one HTML page shown on the portal, there rises a conflict between metadata information provided by different HTML sources.

3.4.3. Frames

Jetspeed forms the portal page as a single window without frames. Any HTML page based on frames invalidates HTML structure if source code with frames is placed within a TABLE element after forming the portal page. Direct accesses from content pages to frame pages should not be used during content development. This requirement does not prevent developers to use frames. OKC Portal does not support pages with frames within the portal page. Any page with frames appears as a blank portlet due to the ill-structured HTML on the final portal page.

3.4.4. Restricting Hosts

As we solved the navigation problem, we further had to improve such capability so that navigation can be dedicated to a certain content host. This has allowed us to follow some content development guidelines due to the restrictions of rendering HTML in accordance with other pieces of screen display of the portal.

3.5. Solving Navigation Problem

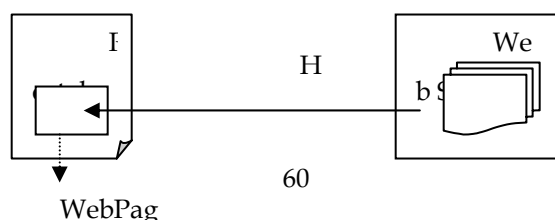
Jetspeed integrates HTML content from different sources into a one-page display. One type of these sources can be Web servers or similar services. HTML is fetched from the serving host and placed into its proper location during the construction of the portal display. Jetspeed has a portlet component which performs such connections and parses HTML for later display. This portlet is called WebPagePortlet, which is basically an implementation of AbstractPortlet interface from the Jetspeed Portal API, acting like a Web client without session support.

```
<portlet-entry name="JavaWeb" hidden="false" type="ref"
  parent="WebPagePortlet" application="false">
  <meta-info>
    <title>JavaSoft</title>
    <description>JavaSoft</description>
  </meta-info>
  <parameter name="dont_remove_applet" value="yes" hidden="false"/>
  <url>http://java.sun.com/</url>
</portlet-entry>
```

Figure 11: WebPagePortlet registry entry.

WebPagePortlet is initialized with a URL address [Figure 11], which is used to connect and fetch the HTML content. [Figure 12] This URL is provided in the portlet registry file inside Jetspeed configuration directory. Further, this portlet parses through the HTML content and rewrites all the HTML links to its absolute form so that whenever users click a link on that page, they are forwarded to the linked page.

WebPagePortlet does not allow navigation in its current version. Whenever a link on this portlet is clicked, the browser's view is replaced with the new page whose URL is rewritten as an absolute URL during HTML processing. However, to provide navigation, after a link is chosen, the portlet's content which presents the remote HTML content must be reprinted from the new page's HTML content.



**Figure 12: WebPagePortlet with absolute URLs.
Any click spawns a new browser window.**

OKC development team has developed a new version of WebPagePortlet, which we have called OKCWebPagePortlet. In this new version, first, the navigation problem has been solved. The following steps were followed for the solution.

From Jetspeed specifications, we know that

- Jetspeed portlet instances are run by the same code, but with different data.
- Configuration information must be provided for each instance.
- Also, design schemes of portlets resemble Java servlet architectures.

```
<portlet-entry name="PET1_ERDC" hidden="false" type="ref"
  parent="OKCWebPagePortlet" application="false">
  <meta-info>
    <title>PET1/ERDC</title>
    <description>PET1/ERDC Site</description>
  </meta-info>
  <parameter name="url_variable" value="peterdc_url" hidden="true"/>
  <url>/rawpages/pet/pet_bd.htm</url>
</portlet-entry>
```

Figure 13: OKCWebPagePortlet registry entry.

First, we have assigned a unique identification, shown in <parameter> tag in [Figure 13] to distinguish between the portlets. We need this ID because when there are more than one portlet working at the same time, during the navigation, it is highly likely that these portlets will be referring different HTML pages. In case of reconstruction the portal page with current URLs, they have to reprint whatever the page they are left with. Jetspeed developer community is working on some new features for portlets, one of which is to assign a unique ID to each portlet at run-time. We had to implement this solution in

advance. When such features are implemented in production releases, we may consider switching to new features.

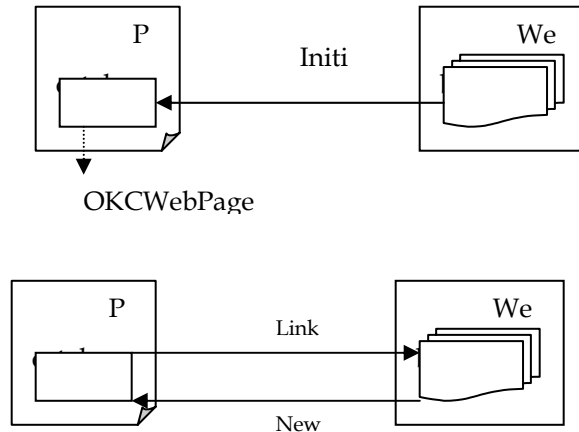


Figure 14: Navigation with OKCWebPagePortlet.

Second, we have rewritten HTML links fetched from the remote content so that instead of spawning a new browser window with an absolute URL address, we now redirect this absolute URL as a parameter to our modified WebPagePortlet.

<p>A link on a page at <code>http://remote.host/parent_directory</code>:</p> <pre>link info</pre>
<p>After WebPagePortlet processes the link:</p> <pre>link info</pre>
<p>After OKCWebPagePortlet processes the link:</p> <pre>link info</pre>

Figure 15: How links change after being processed.

Jetspeed runs on Turbine which provides user authentication and session handling. It is basically a top level servlet which handles connections and calls appropriate services to reply user requests. Jetspeed is a set of Turbine services. Hence, any URL parameter, i.e. “...portal?unique_id=xxx&...,” sent to

Jetspeed portal is distributed to every portlet currently running in a special Turbine object which is called “rundata.” This object is a storage for servlet request and response objects, user profiler, page layouts and customization information.

Jetspeed portlets, such as OKCWebPagePortlet, extract the servlet request object from the rundata and process URL parameters. After rewriting the URL with the portlet unique IDs, whenever a user clicks on a link, the new URL for the next step is obtained from the URL parameter, and the portlet whose unique ID matches to the URL parameter uses the URL value to fetch the next HTML page. As users continue to click on the links within the same portlet, navigation is provided and the problem is solved.

However, if there are more than one portlet used within a portal session, then the OKCWebPagePortlets whose IDs do not match with the URL parameter cannot retrieve any URL to construct their views. To solve this problem, whenever a user clicks a link, the URL obtained from the URL parameter is stored in Jetspeed’s session object. This session object is also the same as the servlets’ session object and available in the rundata. When a user clicks another link within a different portlet later, OKCWebPagePortlet first checks if the URL parameter matches its unique ID. If not, then it checks if any URL value is stored in the session previously. Finally, if there is no URL stored in the session, the initial URL stored in the registry file is used as in the case of WebPagePortlet.

3.6. Restricted Hosts

As we solved the navigation problem, we also had to prevent HTML content from different sources displayed within the portal because such content would not be following our guidelines and free from errors. Web pages with frames and scripts may behave unexpectedly.

To restrict content hosts, we have developed area concept where each area is mapped to a unique host and URI. Any URI access from such hosts is verified against an area configuration file. Accesses to other hosts are displayed in a new browser window. Only those local accesses are displayed inside the portal display.

Area configuration file is defined in XML and provides a list of areas, each area consists of a content host, a media server, a URI, a local directory if there is any, and a stem links file. Stem links are in control of content authors. These files reside along with the content and must be placed in a location where the index page is placed. The stem links file format is in XML and a derivation of RSS schema. It first contains a title and a description of content, and a list of items, each item having a title, a description and a link where the link is the

relative to the initial directory. When these content pages are navigated, stem links files are used to construct the submenus which appear next to the content pages.

4. Distributed Content Management

4.1. Design Considerations and OKC Requirements

By using Jetspeed, the OKC separates web content display from interface control. Page control is managed by a central web server, and most page content (except for OKC specific material) comes from one or more content servers. During the initial phase of evaluation, the IU OKC team has manually added web material provided by other groups to the OKC to verify Jetspeed's ability to display content created by a number of different authoring techniques. For production use, this manual procedure has to be replaced by a distributed content management system that allows authors to edit and upload their own material. For this, we have chosen Slide from Apache-Jakarta team as a prototype for the OKC content management system. As will be explained, Slide can be used both for the OKC content web servers and for content editing and version control.

4.2. Slide Overview

Slide is a Java-based file management system. Remote Slide clients interact with the Slide web server by using the Web-DAV protocol, an IETF standard extension of HTTP. Thus Slide is primarily suitable for web site file management, although the project goals are more general and multipurpose.

For the OKC, we are testing the use of Slide as both the content server and the editing system. Privileged users can log directly into Slide, download files and edit them, then upload again and have the revised page immediately displayed. Slide's role is to enforce access control rights for specific files and to lock files while they are being edited to prevent incompatible edits and lost updates. Thus for example, one or more members of the SIP team could be responsible for updating the SIP web pages. The person in charge of the SIP pages can log into Slide and access the SIP pages for changes. If more than one person maintains the SIP web pages, they can be checked out and locked sequentially. Finally, Slide access controls can be used to ensure that only the privileged SIP user has write access to the SIP pages, and likewise the SIP user does not have write access to other CTA pages.

Slide is developed as part of the Apache-Jakarta project, the Apache group's umbrella for open source Java-based web programming projects and uses the Tomcat web server for remote file management.

4.2.1. *Slide Features*

Slide consists of client and server programming interfaces and applications that implement the major features of the Web-DAV protocol, described below. We wish to emphasize the following major features of Slide:

1. **Namespaces:** Describe a hierarchical collection of files. This is analogous to (and for us, maps to) a Unix directory structure or Windows file system.
2. **Roles:** Particular nodes in a Namespace can be associated with a user or group. Each role has a defined set of actions (read, write, and manage) that can be assigned to it. So for example a particular Namespace can be read-restricted for most users, but a particular user or group can be given write access as well. For the OKC, this maps to a file directory structure having read/write privileges for remote users.
3. **Principals:** A particular role can be associated with a single user, or a role can be associated with a group, with a particular set of users extending and inheriting from this group role. Thus User A and User B can be members of a particular group and inherit all of the permissions and restrictions of that group.

4.2.2. *Slide Views*

Web pages are served up by Slide in one of three different views, each associated with three different type of user. The Slide server runs three different ports, one for each view. The views are

1. **Client View:** this is the publicly available view of the web material, requiring no previous authentication. The Client View can be used as a normal web server.
2. **Editor View:** this view is accessible only to privileged users and requires a password login. Users with editing privilege can modify web pages in directories on the Slide server to which they have write privilege.
3. **Administrator View:** the Slide superuser can manage editor accounts and start and stop the Slide web server.

The OKC is thus being prototyped to use Slide's client view as the content server and its editor view for web page management.

4.3. Web-DAV

Web-DAV, short for Web-Based Distributed Authoring and Versioning, is a set of IETF standard extensions to the HTTP protocol that have been designed to support distributed authoring of web content. As

such, Web-DAV defines a metadata model for describing remote documents (both individually and collectively) and a set of mechanisms for interacting with these documents. The metadata types are name/value pairs and include page authors, creation and modification times, page versions, and so forth. Groups of web pages can be associated to form a collection. A typical example of a collection would be a directory of related files, such as all the files associated with the a particular CTA.

The associated Web-DAV file management mechanisms include

1. Metadata Property Querying: client applications can find out the authors, creation dates,
2. Locking: files with more than one author can be locked to prevent multiple edits. The file can only be checked out again by another author after it has been checked in. Shared locks are used for access control and define who has the right to modify a particular file.
3. Management Commands: Web-DAV includes commands for copying, moving, creating and deleting files.

Web-DAV is a wire protocol only, and as such does not define how the clients and servers at either end of the communication are implemented. It is thus (like HTTP itself) programming-language independent and relies on the endpoints properly creating and consuming Web-DAV commands. Slide is one such implementation. Further, Slide clients and server pieces should interoperate with other Web-DAV implementations.

4.4. OKC Slide Features

4.4.1. JSP Slide Client

As mentioned above, Slide comes with client and server application programming interfaces (APIs), as well as specific implementations of those APIs. Client-side implementations include a command line shell interface to Slide functions (similar to a Unix shell environment), Java client applications built with swing-based Slide GUIs, and a web-folder interface compatible with Internet Explorer. However, we prefer JSP-based browser interfaces since they are simpler to learn and use than command line interfaces, are platform independent, and assume only that the user has a browser and does not need to download a “fat” client application to manage his or her web content.

Current path: /slide

File Name	Size	Locked	Last Modified	Command
 Up				Refresh
 actions			Mon Apr 01 11:17:11 EST 2002	view copy move delete rename lock unlock properties
 files			Mon Apr 01 11:17:11 EST 2002	view copy move delete rename lock unlock properties
 ptiu			Mon Apr 01 11:17:11 EST 2002	view copy move delete rename lock unlock properties
 users			Mon Apr 01 11:17:11 EST 2002	view copy move delete rename lock unlock properties

New Directory	Upload File	Log Out
-------------------------------	-----------------------------	-------------------------

Figure 16 Slide's top level node contains the indicated namespaces.

We thus have implemented the Slide shell as a Java class with a JSP interface. A screen shot of this interface is given in Figure 16 and Figure 17. Users log in through their browsers and navigate the remote Slide directory tree. Figure 16 is a JSP display that shows the default Slide namespaces. Clicking these links allows a user to move to down the file hierarchy to their user directory, where they have permission to upload files. As shown in Figure 17, the user mpierce can view a list of his files in (a Word document) and subdirectories. Through the JSP client, the files can be viewed, copied, moved, etc. Files can be locked or unlocked while they are being modified to prevent others from checking out the files. They can modify the remote web file structure (adding, deleting, and renaming files and directories, for example) and can upload multiple files from their directories to the remote file system. All of these links construct Web-DAV commands and are sent to the Slide server for execution.

Current path: /slide/users/mpierce/

File Name	Size	Locked	Last Modified	Command
 Up				Refresh
 test			Tue Mar 26 16:37:00 EST 2002	view copy move delete rename lock unlock properties
 mp2mp.doc	22528		Tue Mar 26 16:32:03 EST 2002	view copy move delete rename lock unlock properties

[New Directory](#) [Upload File](#) [Log Out](#)

Figure 17 A Slide user can execute the indicated commands in his or her directory.

Remote files can be uploaded to the remote file system through another JSP interface, shown in Figure 18. This interface implements HTTP uploading and allows users to select one or more files.

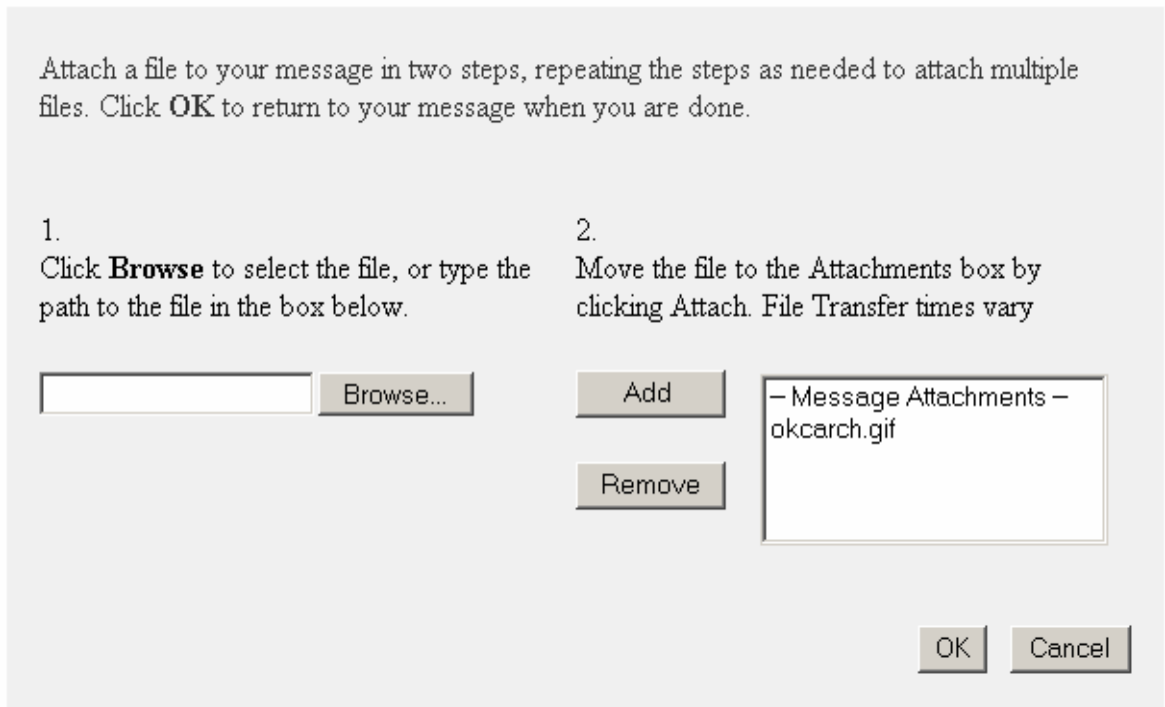


Figure 18 Files can be uploaded to a user's Slide area.

4.4.2. Oracle Support

Slide comes with the database Hypersonic SQL, a free, file system based database. Slide stores namespace structures (but not the files themselves), revision histories, locks, labels, and file properties in the database. We wanted to test Slide with Oracle. The steps are as follows:

- 1) Edit slide.def to use the Oracle JDBC driver.
- 2) Created database tables as described at Slide web site.

This worked without any problems. However, we have noticed performance is sometimes slow. We do not currently know the source of this problem, but it may result from development work using our Oracle database and so will not be a problem for production use. We will investigate this problem.

4.4.3. Future Work

We do not yet support version control. Slide automatically maintains a version history of every file uploaded. We are still learning how to work with this so that users can download, manage, and delete earlier versions of their files.

We will improve the JSP user interface to support multiple (checkbox) selection options for delete, lock, and unlock commands. This will allow a user to, for instance, delete several files at once.

There is a bug in Slide when adding new namespaces. The default namespace, slide, contains subspaces (ObjectNodes in Slide terminology) called files, users, and actions. We tried adding additional namespaces such as OKC with their own subspaces files, etc. These were create correctly, but the Slide administrator only works right now with the default namespace, slide. We will fix this in the future.

Slide editor authentication is currently a simple password-based system. Since this provides access to areas of the remote server's file system, we will investigate adding support for Kerberos and SecurID authentication in the near future.

5. OKC Page Wizards

5.1. What is a Wizard?

In the context of the OKC project a wizard creates the user interfaces for parts like newsgroups and the training-registration process. It is responsible for providing forms for users, obtaining user inputs, creating XML data from these inputs and publishing these data to the JMS server or storing in a server.

5.2. Steps for creating a Wizard

When developing a new wizard for a particular OKC feature, we must take the following steps. These are illustrated in Figure 19.

1. An XML Schema which has all the necessary elements regarding with the project is created.
2. Using Castor Source Generator [CASTOR], Java classes which correspond to the elements of the schema are created.
3. An HTML form which has form elements to represent the schema elements is created using JSP technology.
4. Inputs are obtained from the user and using Castor XML binding methods an XML output is generated.
5. Generated XML string is published to the JMS Server by *News Group Message Generator*.

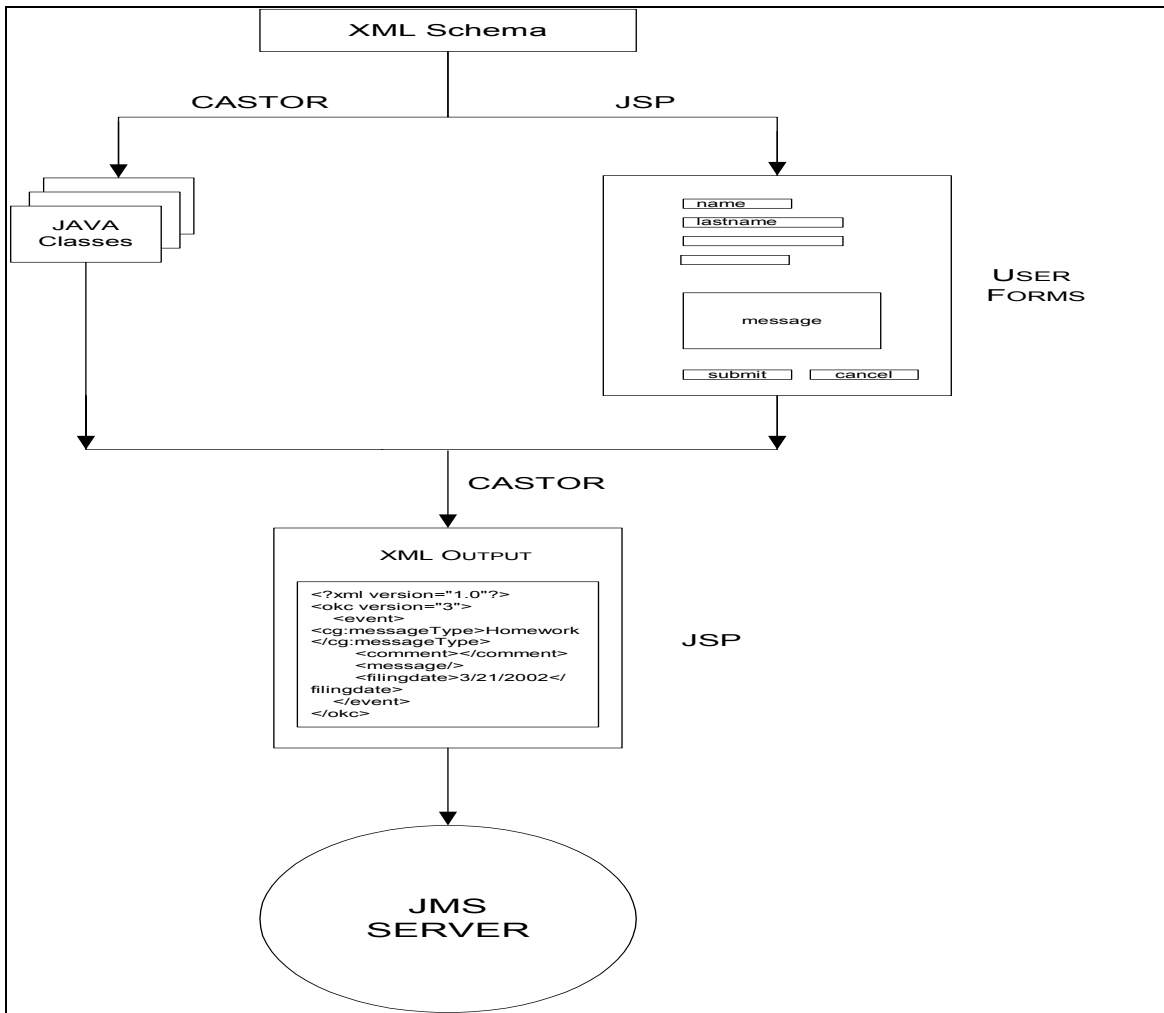


Figure 19: OKC page wizards map XML schema into both Java classes and JSP web forms.

5.3. Wizards developed for OKC

5.3.1. MetadataWizard

This wizard is an implementation of the SCORM [SCORM] specifications. We are integrating this wizard with SLIDE for content management.

While adding new content to the repository users are expected to provide some basic information about the new content. The system itself will get the basic information about the new content and produce metadata in XML format in case the user does not provide any info. These metadata can be used in various places - like searching-. An XML Schema for MetadataWizard was created based on the SCORM Content Metadata Generator.

5.3.2. NewsWizard

This wizard provides interfaces for composing newsgroup messages to the OKC newsgroup. We have designed an OKC message schema and created the html forms according to this schema. Further explanation about this wizard is provided in the next section.

5.3.3. SchemaWizard

Since there are expected to be many schema versions which we need to adapt, we have designed our system so that new forms based new versions are easily generated. While developing a wizard the next step after designing the schema is to create HTML forms according to the related XML schema. This is a tedious job to do for complex schemas and each time the version changes.

To automate the creating of the HTML forms using JSP we are working on the development of a SchemaWizard which takes a valid XML Schema as input and creates the necessary JSP pages and form fields to correspond to the elements of the schema.

5.4. Technologies used to create wizards

5.4.1. XML Schema

The first step is creating an XML Schema according to the needs. In this schema all the elements and attributes are defined. Thus any relationships between these elements are formalized. Creating a valid XML document at the end of the process is also guaranteed by using a Schema at the beginning.

Using XML Schema provides us a great flexibility of reusing the same global element by creating a reference to it. We have used this feature in our schemas to avoid redundancy. Also XML Schema is used for the validation purposes where it is necessary.

XML Schema provides a common ground for different developers of the OKC Project.

5.4.2. OKC Schema

Newsgroup architecture is one of the main parts of the OKC project. Our News Wizard provides users appropriate forms for composing newsgroup messages. To create a newsgroup message form firstly an XML Schema is created. The elements of this schema are thought to be most common and usable elements for all different news groups. Although the current version of the newsgroup architecture supports single event messages a typical OKC message might have multiple events, concerning different topics and each event might have different destinations.

The OKC Newsgroup schema is illustrated in Figure X.

5.4.3. OKC Schema Elements and Short Descriptions

event: Main element of the OKC message, contains other sub elements, can be multiple.

definition: Any definition regarding with this event. This element has two attributes: ref and uri. The value given in ref attribute is associated with the URI given in uri attribute, and can be used within the OKC message anywhere like a variable, uri="\$me/ideas/\$today/1". The body of this element can be used as comment area.

comment: Any comment information regarding this event.

sender: Source of this event. Sender can be a user, an application or any other type of event source.

distribution: Destination of the message. A newsgroup or an individual.

organization:

update: This element defines job to perform. Has two attributes mode and uri. mode is an enumeration of create | update | remove. uri is the URI of the object that is affected by the process defined in mode.

keywords: Any keyword regarding with the message or this event. Keywords are separated by comma or white space. Also a user can create keywords using special <keyword> tag inside the message body.

subject: topic of the message

message: Body of the message compiled by user is put into this element. To keep the original format of the message new lines are checked and explicitly added into the message as breaks during XML creation since Castor does not preserve the new line characters.

Keywords can be added using special tags inside the message body so that user can express some words that way.

Example: We have 16 <keyword> newsgroups </keyword> for this project.

filingdate: Creation date of this event. This date is automatically generated and displayed in the form. Format is MM/DD/YYYY.

attachments: This element might have multiple *attachment* elements since an event can have multiple attachments. uri attribute is used to save the attached file's name.

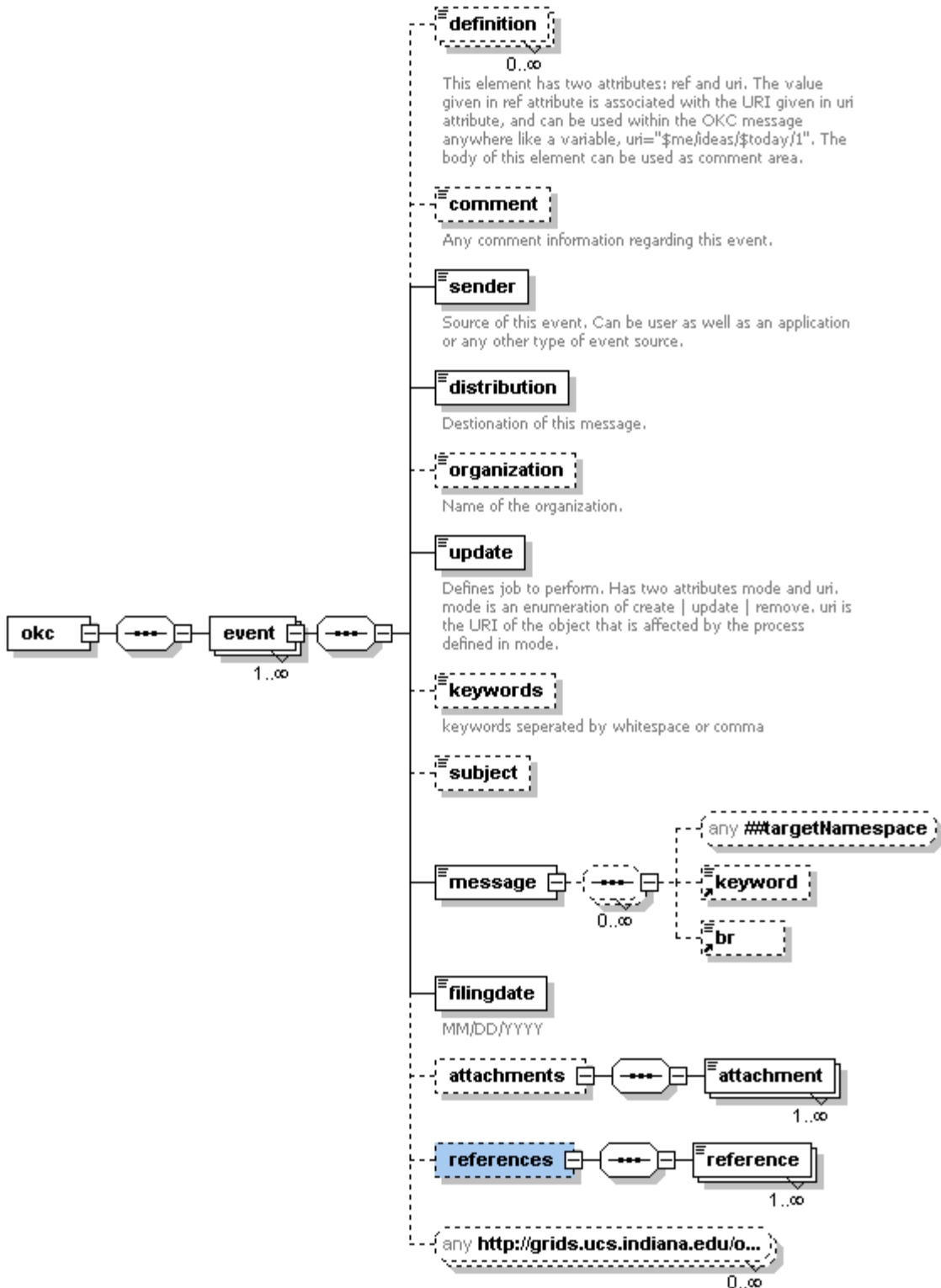


Figure 20 The OKC newsgroup schema.

5.5. XML Data Binding with CASTOR

5.5.1. *Data Binding*

Data binding is the process of mapping the components of a given data format, such as SQL tables or an XML schema, into a specific representation for a given programming language that depicts the intended meaning of the data format (such as objects, for example). Data binding allows programmers to work naturally in the native code of the programming language, while at the same time preserving the meaning of the original data. It allows us to model the logical structure of the data without imposing the actual specific structure imposed by the format in which the data is stored.

In most cases the content of an XML document, though stored in XML as simply character data, represents a number of different data types such as strings, integers, real numbers, dates, and encoded binary data to name a few. These different data types are usually grouped together in some logical hierarchy to represent some special meaning for the domain in which the XML data is intended. Ideally, interacting with the XML content as a data model represented as objects, data structures, and primitive types native to the programming language we are using, and in a manner which closely reflects the meaning of that data, would make programming with XML more natural, much less tedious, and improve code readability and maintainability.

This means that instead of dealing with such things as parse trees, event models, or record sets, we interact with objects, integers, floats, arrays, and other programming data types and structures. To summarize, data binding gives us a way to:

- 1 - Represent data and structure in the natural format of the programming language we decide to program in.
- 2 - Represent data in a way that depicts the intended meaning.
- 3 - Allows us to be agnostic about how the data is actually stored.

5.5.2. *How Data Binding Works with XML*

XML data binding simply refers to the mapping of structural XML components, such as elements and attributes, into a programmatic data model that preserves the logical hierarchy of the components, exposes the actual meaning of the data, and represents the components in the native format of the programming language. In Java, our data model would be represented as an Object Model.

An object model in Java is simply a set of classes and primitive types that are typically grouped into a logical hierarchy to model or represent real-world or conceptual objects. An object model could be as simple as consisting of only one class, or very complex consisting of hundreds of classes.

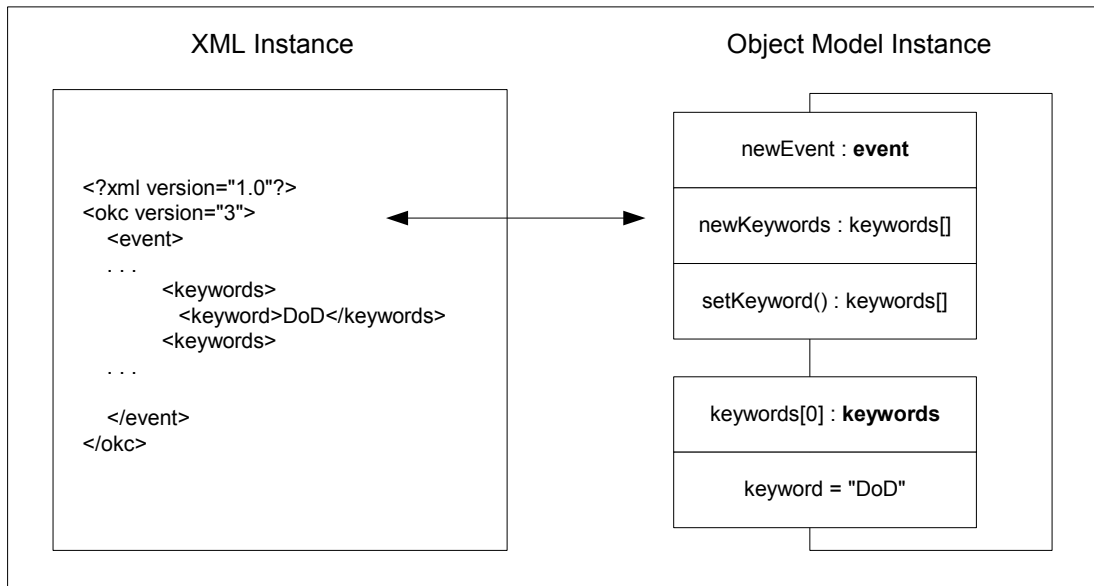


Figure 21 The XML data binding process.

5.5.3. Simple Data Binding Concepts

To bind our XML instances to a Java object model we can use a data-binding framework. A databinding framework is an application (or set of applications) that allows data binding solutions to be developed more easily – these frameworks typically come with a number of features, such as source code generation and automatic data binding.

XML data binding consists of three primary concepts – the *specification of XML to object model bindings*, *marshalling*, and *unmarshalling*.

Marshalling: Converting an object model instance into an XML instance.

Unmarshalling: Converting an XML instance to an object model.

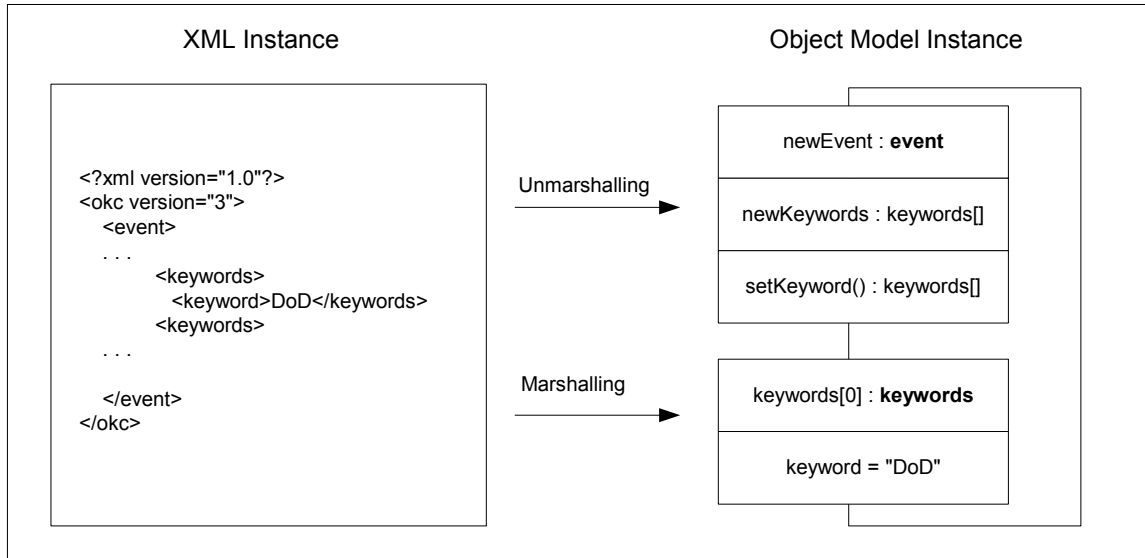


Figure 22 Marshalling and unmarshalling between XML and Java.

5.5.4. Data Objects

To manage the XML instances we created and manipulate the data we need to convert these instances into a more manageable model that is native to the programming language that we are using. This allows for a more natural way of working with the data and makes our code more readable and maintainable. In our case we'd like to convert the XML model into a Java object model for manipulation and then back into the XML representation when we are finished. In most cases, however, the objects in our model simply do nothing more than hold the data that was once stored as XML, and preserve the logical structure of the XML document. Since these objects have no complex programming or business logic, and just contain typed data, we call them Data Objects.

Typically our data objects will also be Java Beans. A Java Bean is simply a Java class that adheres to the Java Bean design pattern. This simply means that we follow some basic guidelines when writing our classes so that information can be obtained about our data objects. In most cases, simply following the method naming conventions of the design pattern for Java Beans is sufficient for tools to obtain information about the fields (also called properties) of our classes by examining the method signatures of a given class. This examination process is called Introspection (as defined by the Java Beans Specification). The main guideline for the design pattern is that all publicly accessible fields have proper getter and setter access methods. For a given field, a getter method is quite simply a method that returns the value of that field, while a setter method is one that allows us to set the value of the field.

The Java Beans design pattern indicates that getter methods should begin with "get", followed by the field name, with the first letter of the field name capitalized. The setter methods follow the same pattern, but begin with "set". For example, if we had a field called Name, the getter method should be called getName and the setter method setName.

5.5.5. *CASTOR vs. DOM and SAX*

The W3C DOM is tree-based while SAX is event-based, but both APIs are structure centric in that they deal purely with the structural representation of an XML document. See Refs [DOM] and [SAX] for more information about these. The basic format of an XML document is a hierarchical structure comprised mainly of elements, attributes and character data. There is nothing actually wrong with these APIs if they are used for their intended purpose, manipulating XML documents in way that represents the generic structure of the XML document.

Most applications that need to access the contents of an XML document, however, will not need to know whether data was represented as an attribute or an element, but simply that the data is structured and has a certain meaning. APIs such as the DOM and SAX, while providing meaningful representations of the data, can only provide extremely generic representations that make accessing our data long winded, which for high-level XML manipulation adds complexity to our applications that is unnecessary. Programmers must essentially walk the structure of the XML in order to access the data. This data also often needs to be converted into its proper format before it can be manipulated, which can be a very tedious process. This is necessary because the data returned using these APIs is simply a set of characters, or a string. If the contents of an element should represent some sort of date or time instance, then the programmer must take the character data and convert it to a date representation in the programming language. In Java this would most likely be an instance of java.util.Date (the Java date function). If the element itself represents a set of other elements, we would need to create the appropriate object representation and continue converting sub-elements.

The main problem for us with DOM and SAX is that these APIs are very tedious for a large scale XML based project like OKC. It is not logical to write code for hundreds of elements. So we use another framework which allows us to specify how an XML schema should map into an object model and automatically handle the data binding for us.

This framework is called CASTOR which is an open source data-binding framework for Java that supports binding XML as well as relational databases, and LDAP.

Castor is an open source data binding framework for Java. It's basically the shortest path between Java objects, XML documents, SQL tables and LDAP directories. Castor provides Java to XML binding and Java to SQL/LDAP persistence.

5.5.6. Creating data object model with CASTOR

To create a data object model of our XML Schema Castor provides a source code generator which can automatically generate a Java object model to represent a given XML schema. The source code generator not only creates the Java object model, but also all necessary XML binding information needed by the marshalling framework to properly marshal, unmarshal, and validate instances of the given schema. A source code generator is a valuable time saving tool when it comes to writing XML enabled applications. The source code generator is invoked from the command line giving the name of the schema file and output directory name as parameters.

```
>java org.exolab.castor.builder.SourceGenerator -i okc-v3.xsd -package okc.messages
```

The screenshot shows a web-based "News Message Composer" interface. At the top, there is a "Generate Message" button. Below it, several form fields are visible: "Sender" (Galip Aydin < gaydin@indiana.edu >), "Organization" (Indiana University), "Distribution" (Community Grids Newsgroups), "Category" (General), "Comment" (sample message), "Keywords" (java, xml, castor), "Filing Date" (3/22/2002), and "Subject" (Data Binding with Castor). There is also an "Add/Remove Attachments" button. At the bottom, a "Message" field contains the following text:

```
<keyword> Data binding </keyword> gives us a way to:  
1 - Represent data and structure in the natural format of the  
programming language we decide to program in.  
2 - Represent data in a way that depicts the intended meaning.  
3 - Allows us to be agnostic about how the data is actually stored.
```

5.6. JavaServer Pages

Once we have our schema and data object model which represents the elements in the schema we need to provide an interface for users to enter their data. We use JSP to produce web pages on demand. These pages consist of some tables which are created according to the schema elements. Each form element in the HTML form corresponds to an element in the XML Schema. Figure X is a snapshot of the newsgroup message composer which is built according to the okc-v3 schema. Using the java classes created by Castor, an OKC object is created, this objects has a main element called event. Each field in the okc schema is a sub element of this event object. The data entered into the form fields are assigned to the corresponding java objects when user hits the Generate Message button. The second step here is to show the composed message to the user in XML format. This is done by calling Castor's marshal method.

```
newswizard.NewsBean.marshalMessage(okcmessage);
```

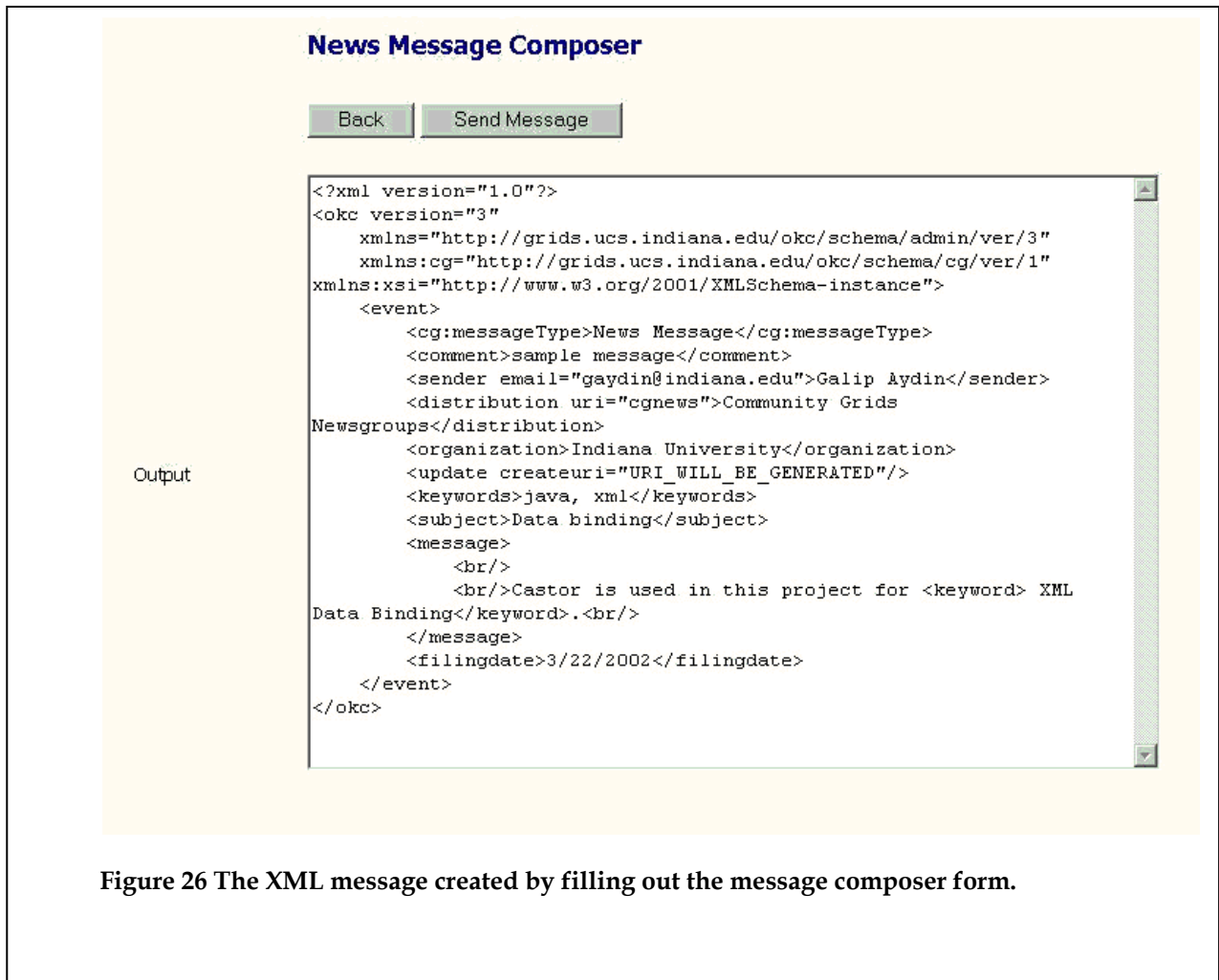



Figure 26 The XML message created by filling out the message composer form.

5.7. Publishing the message

The last step here is to publish this message to the JMS Server so that the message can be distributed to the related topic. A message publisher is developed for this purpose by members of our team. We use Java Messaging Service for publishing the messages.

```
NewsgroupMessageGenerator ngmg = new NewsgroupMessageGenerator(distribution);
```

Created XML message is passed as an argument to the method `NewsgroupMessageGenerator`, this method creates a connection to the JMS server and publishes the message.

```
ngmg.setMessage(msg);
.
.
ngmg.createNewsgroupHeaderMessage();
```

```
ngmg.constructBody();  
ngmg.publish();
```

6. Email Handler and Distributor

6.1. Introduction

In this section we will discuss two bridge programs that deliver the messages in email server to JMS server and the messages in JMS server to email server. These programs are very important in over all architecture of the Online Knowledge Center system because they play the communication channel roles between different messaging systems like email messaging system and JMS messages. We are using XML format messaging here for several reasons. First, XML is becoming universal language for internet based message passing applications. Second, data transferring by XML encapsulation makes very easy to access any data in a huge document or to change it.

In the following sections we will discuss email handler first then email distributor.

6.2. Email Handler

Email handler is a bridge program that connects the Email server and JMS server and it is kind of a middle level server. Email Handler just sits in the cross-section SOAP-JMS and SMTP. There we used all three technologies in implementation of Email Handler server. Basically, Email Handler listens to email server, if there is a new message which should be in XML format and in our schema it reads that message and convert it into soap attachment object. After this step it publishes new reformed message to JMS server.

Figure X illustrates the interaction between Email Server, Email Handler and Email Distributor. The end user can only use his/her email account to send or receive a message. As it shown in the figure, Email Handler connects the Email Server and JMS server. It reads messages from the Email Server and publishes it to the JMS server. We are using publishing method instead of point-to-point because there are several listeners in the systems which listen to JMS topics for new messages. Email Distributor is the communication channel between JMS Server and users. It gets the messages from the JMS server and delivers it to qualified users.

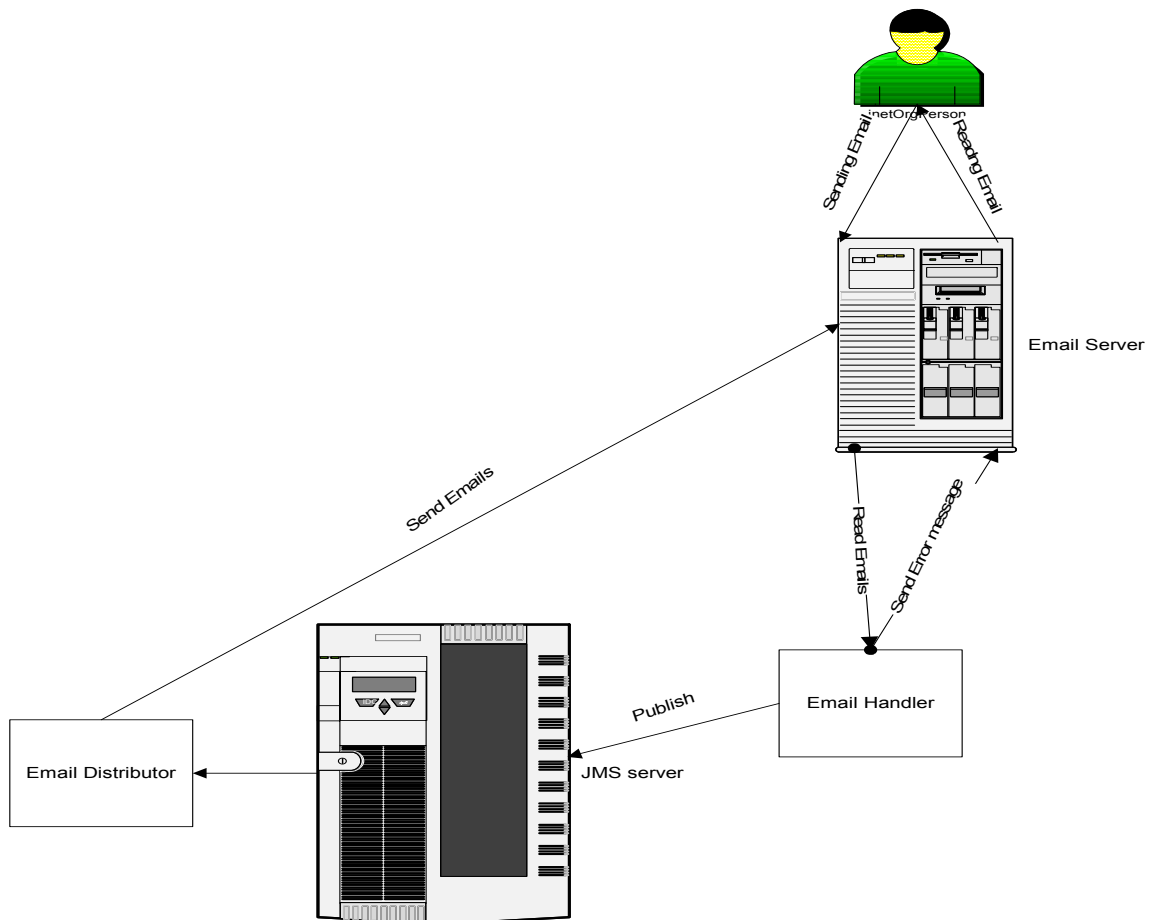


Figure 27 Email Handler and Email Distributor interaction with JMS and Email servers.

Before describing the Email Handler in detail, let's overview those technologies.

6.2.1. JMS

JMS (Java Message Service) is an application program interface (API) from Sun Microsystems that supports the formal communication known as messaging between computers in a network. Sun's JMS provides a common interface to standard messaging protocols and also to special messaging services in support of Java programs. Sun advocates the use of the Java Message Service for anyone developing Java applications, which can be run from any major operating system platform.

The messages involved exchange crucial data between computers - rather than between users - and contain information such as event notification and service requests. Messaging is often used to coordinate programs in dissimilar systems or written in different programming languages.

6.2.2. SOAP

SOAP (Simple Object Access Protocol, see Ref [SOAP]) is a simple, lightweight protocol for structured and strong-type information exchange in a decentralized, distributed environment. The protocol is based on XML and consists of three parts:

1. An envelope that describes the contents of the message and how to use it
2. A set of rules for serializing data exchanged between applications
3. A procedure to represent remote procedure calls, that is, the way in which queries and the resulting responses to the procedure are represented

Similar to object distribution models (e.g., CORBA and DCOM), SOAP can call methods, services, components, and objects on remote servers. However, unlike these protocols, which use binary formats for the calls, SOAP uses text format (Unicode), with the help of XML, to structure the nature of the exchanges.

SOAP messages must be carried by over the wire by a transport protocol and are particularly well-suited to the HTTP protocol. SOAP's HTTP binding defines a reduced set of parameters that are specified in the HTTP header, making it easier to pass through proxies and firewalls.

6.2.3. SMTP

SMTP (Simple Mail Transfer Protocol) is a TCP/IP protocol used in sending and receiving e-mail. However, since it's limited in its ability to queue messages at the receiving end, it's usually used with one of two other protocols, POP3 or Internet Message Access Protocol, which let the user save messages in a server mailbox and download them periodically from the server. In other words, users typically use a program that uses SMTP for sending e-mail and either POP3 or IMAP for receiving messages that have been received for them at their local server. Most mail programs such as Eudora let you specify both an SMTP server and a POP server. On UNIX-based systems, sendmail is the most widely-used SMTP server for e-mail. A commercial package, Sendmail, includes a POP3 server and also comes in a version for Windows NT.

SMTP usually is implemented to operate over Transmission Control Protocol port 25. The details of SMTP are in Request for Comments 821 of the Internet Engineering Task Force (IETF). An alternative to SMTP that is widely used in Europe is X.400.

6.2.4. JAVA MAIL API

The Java Mail API (see Ref [JAVAMAIL]), one of the latest standard extension APIs from Sun, should equally benefit client- and server-side application developers; with its platform- and protocol-independent

mail/messaging solution. With help of the Java Mail API programmers can develop application which reads email from the email server and sends email using SMTP servers.

6.2.5. *Now actually what does Email Handler do?*

Email Handler connects any type of email server which protocol can be POP3, IMAP or SMTP and listens to it in a pre-set time interval. Right now we are using 5 minutes to check if there is a new message in the inbox or not. We are using the Java Mail API to listen the email server. When the new mail comes to inbox, our Email Handler reads it and starts to process it. First it checks if the message is a valid XML document or not. We construct three XML documents after processing the incoming message. These are:

1. GXOS message
2. Incoming message
3. Incoming message header

If it is valid XML document then it parses the message to get the necessary information that will be used when the GXOS package is constructed. GXOS message is used to integrate these messages to GXOS system. Please see the appendix for the sample of GXOS message.

Incoming message is the actual user message and it is in XML document format. Its XML schema is predefined by the system administrator and user must follow that schema in order to communicate the system successfully. In other words, their message will not be published into JMS server. You can see the sample user incoming message in appendix.

Incoming message header is the email header information. That information is not an XML format when it is read from the Email Server. Email Handler converts that header information which includes the mail delivery route information into XML document format. By saving email header information, we do not lose any information from the original email. It can be used for security purpose in the future, too. Please refer to appendix for the sample email header XML document.

There is a question now: How does Email Handler deliver the messages in to right topics:

We get the destination topic of JMS from incoming message. There is a field as

`<distribution uri="cgreports">Community Grids Project Reports</distribution>` which tells us where to publish this message.

If the incoming message incoming message is not in XML format or it not a valid XML document then our Email Handler sends an error message to the user.

6.2.6. Security

Before publishing message to JMS server Email handler connects the database and check if the current user has a right to publish message into that topic. If the user does not have publishing right into that topic, an error message is generated and sent it to the user. If the user has publishing right and everything goes well, then we create the SOAP attachment message and publish it JMS. The following XML document is our SOAP attachment message format just before publishing the JMS. Encrypting the message in all the sending process is one of our future plans in Email Handler enchantment.

6.3.Email Distributor:

Email Distributor is responsible for delivering messages to users who have rights to read that topics. It connects the JMS server and listens to it if there is new message or not. If there is a new message then it connects to database and get the email list of all users who has read access to that topic. Then it sends that message to all qualified users.

Since Email Handler and Distributor are very similar to each other we do not repeat similarities here. If we consider the all system as a two way communication highway: Email handler is one way and Distributor is other way.

It has the same security system as in Email Handler as we mentioned before. Email Handler and Distributor both have attachment capability. They can handle messages with attachments.

6.4. Appendix

6.4.1. GXOS Message Sample:

```
<?xml version="1.0"?>
<OKCMessage xmlns="http://grids.ucs.indiana.edu/schemas/okc/message/1"
xmlns:gxos="http://aspen.csit.fsu.edu/project/gxos">
  <gxos:NodeName>12345</gxos:NodeName>
  <gxos:NodeType>Event</gxos:NodeType>
  <gxos:NodeRooting>Leaf</gxos:NodeRooting>
  <gxos:NodeVersionNumber>0.3</gxos:NodeVersionNumber>
  <gxos:NodeStartTime>
    <gxos:Time>1005786509068</gxos:Time>
    <gxos:TimeSyntax>GMTMillisecs</gxos:TimeSyntax>
    <gxos:ReferenceAction>begin</gxos:ReferenceAction>
  </gxos:NodeStartTime>
  <gxos:EventDestination>
    <gxos:Destination>
      <gxos:DestinationType>Topic</gxos:DestinationType>
    </gxos:Destination>
    <gxos:DestinationforJMS>
      <gxos:DestinationTopic>
        Community Grids Research Group</gxos:DestinationTopic>
      </gxos:DestinationforJMS>
    </gxos:EventDestination>
  <gxos:SourceDevice>
    <gxos:InternalAddress>gxos://okctest/devices/colima</gxos:InternalAddress>
  </gxos:SourceDevice>
  <gxos:Priority>50</gxos:Priority>
  <gxos:EventUserName>mailhandler</gxos:EventUserName>
  <gxos:EventUserObject>
    <gxos:InternalAddress>gxos://okctest/users/mailhandler</gxos:InternalAddress>
  </gxos:EventUserObject>
  <gxos:GMSMessageID>
    <gxos:GMSSequenceNumber>12345</gxos:GMSSequenceNumber>
    <gxos:GMSStream>
      <gxos:InternalAddress>gxos://okctest/events/mailhandler
        </gxos:InternalAddress>
    </gxos:GMSStream>
  </gxos:GMSMessageID>
  <Sender email="alikapla@indiana.edu">Ali Kaplan</Sender>
  <Subject>&lt;okc subject="this is about okc"&gt;&lt;/Subject>
  <Attachments>
    <Attachment href="cid:attachments/okc_body"
      contenttype="text/xml">OKC User Message Body</Attachment>
    <Attachment href="cid:attachments/mailheaders"
      contenttype="text/xml">Original Mail Headers</Attachment>
  </Attachments>
</OKCMessage>
```

6.4.2. User Message (in XML Document) Sample:

```
<?xml version="1.0"?>
<okc version="3" xmlns="http://grids.ucs.indiana.edu/okc/schema/admin/ver/3">
  <event>
    <category main="facility"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="java:commgridsv1.xsd.schema.Category"/>
    <messageType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="java:commgridsv1.xsd.schema.MessageType">newsmesssage</messageType>
    <comment></comment>
    <sender email="alikapla@indiana.edu"> Ali Kaplan</sender>
    <distribution uri="cgreports">Community Grids Project Reports</distribution>
    <update createuri="gxos://okc/newsgroups/cgreports/78/$next"/>
    <keywords></keywords>
    <subject>Re:Fw: and test and test</subject>
    <message>tesing the system in the morning<br/>
    </message>
    <filingdate>1/16/2002</filingdate>
    <references>
      <reference uri="gxos://okc/newsgroups/cgreports/78"/>
    </references>
  </event>
</okc>
```

6.4.3. User Message Header (in XML Document) Sample:

```
<mailheaders>
  <header name="Received">
    by colima.ucs.indiana.edu
    mbox alikapla) Sat Oct 13 12:13:44 2001)
  </header>
  <header name="X-From_">
    owner-cgstudents-l@LISTSERV.INDIANA.EDU
    Fri Oct 12 21:13:25 2001
  </header>
  <header name="Return-Path">
    owner-cgstudents-l@LISTSERV.INDIANA.EDU
  </header>
  <header name="MIME-Version">
    1.0
```



```

    </header>
    <header name="Content-Type">
        text/plain; charset="iso-8859-1"
    </header>
    <header name="Date">
        Fri, 12 Oct 2001 20:00:25 -0500
    </header>
    ...
</mailheaders>

```

6.4.4. All Messages in SOAP Attachment Message Format Sample:

```

MIME-Version: 1.0
Content-Type: multipart/related;
boundary="-----=_Part_0_7469984.1005786325213"

```

```

-----=_Part_0_7469984.1005786325213
Content-Type: text/xml
Content-Transfer-Encoding: 8bit
Content-ID: <1651189.1005786325213.apache-soap.montblanc>
Content-Length: 296

```

```

<?xml version="1.0"?>
<OKCMessage xmlns="http://grids.ucs.indiana.edu/schemas/okc/message/1"
xmlns:gxos="http://aspen.csit.fsu.edu/project/gxos">
  <gxos:NodeName>12345</gxos:NodeName>
  <gxos:NodeType>Event</gxos:NodeType>
  <gxos:NodeRooting>Leaf</gxos:NodeRooting>
  <gxos:NodeVersionNumber>0.3</gxos:NodeVersionNumber>
  <gxos:NodeStartTime>
    <gxos:Time>1005786509068</gxos:Time>
    <gxos:TimeSyntax>GMTMillisecs</gxos:TimeSyntax>
    <gxos:ReferenceAction>begin</gxos:ReferenceAction>
  </gxos:NodeStartTime>
  <gxos:EventDestination>
    <gxos:Destination>
      <gxos:DestinationType>Topic</gxos:DestinationType>
    </gxos:Destination>
    <gxos:DestinationforJMS>
      <gxos:DestinationTopic>
        Community Grids Research Group
      </gxos:DestinationTopic>
    </gxos:DestinationforJMS>
  </gxos:EventDestination>
  <gxos:SourceDevice>
    <gxos:InternalAddress>gxos://okctest/devices/colima</gxos:InternalAddress>
  </gxos:SourceDevice>
  <gxos:Priority>50</gxos:Priority>
  <gxos:EventUserName>mailhandler</gxos:EventUserName>
  <gxos:EventUserObject>
    <gxos:InternalAddress>gxos://okctest/users/mailhandler</gxos:InternalAddress>

```

```

</gxos:EventUserObject>
<gxos:GMSMessageID>
  <gxos:GMSSequenceNumber>12345</gxos:GMSSequenceNumber>
  <gxos:GMSStream>
    <gxos:InternalAddress>gxos://okctest/events/mailhandler
      </gxos:InternalAddress>
  </gxos:GMSStream>
</gxos:GMSMessageID>
<Sender email="alikapla@indiana.edu">Ali Kaplan</Sender>
<Subject>&lt;okc subject="&quot;this is about okc&quot;"/&gt;</Subject>
<Attachments>
  <Attachment href="cid:attachments/okc_body"
    contenttype="text/xml">OKC User Message Body</Attachment>
  <Attachment href="cid:attachments/mailheaders"
    contenttype="text/xml">Original Mail Headers</Attachment>
</Attachments>
</OKCMessage>

```

```

-----=_Part_0_7469984.1005786325213
Content-Type: text/xml
Content-ID: cid:attachment/okc_body

```

```

<?xml version="1.0"?>
<okc version="3" xmlns="http://grids.ucs.indiana.edu/okc/schema/admin/ver/3">
  <event>
    <category main="facility"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="java:commgridsv1.xsd.schema.Category"/>
    <messageType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="java:commgridsv1.xsd.schema.MessageType">newsmessage</messageType>
    <comment></comment>
    <sender email="alikapla@indiana.edu"> Ali Kaplan</sender>
    <distribution uri="cgreports">Community Grids Project Reports</distribution>
    <update createuri="gxos://okc/newsgroups/cgreports/78/$next"/>
    <keywords></keywords>
    <subject>Re:Fw: and test and test</subject>
    <message>tesing the system in the morning<br/>
  </message>
  <filingdate>1/16/2002</filingdate>
  <references>
    <reference uri="gxos://okc/newsgroups/cgreports/78"/>

```

```

        </references>
    </event>
</okc>
-----=_Part_0_7469984.1005786325213
Content-Type: text/xml
Content-ID: cid:attachment/mailheaders

<mailheaders>
    <header name="Received">
        by colima.ucs.indiana.edu
        mbox alikapla) Sat Oct 13 12:13:44 2001)
    </header>
    <header name="X-From_">
        owner-cgstudents-l@LISTSERV.INDIANA.EDU
        Fri Oct 12 21:13:25 2001
    </header>
    <header name="Return-Path">
        owner-cgstudents-l@LISTSERV.INDIANA.EDU
    </header>
    <header name="MIME-Version">
        1.0
    </header>
    <header name="Content-Type">
        text/plain; charset="iso-8859-1"
    </header>
    <header name="Date">
        Fri, 12 Oct 2001 20:00:25 -0500
    </header>
    ...
</mailheaders>
-----=_Part_0_7469984.1005786325213--

```

7. NEWSRECORDER AND NEWSFEEDER

7.1.INTRODUCTION

This program includes two modules, newsfeeder and newsrecorder. These are responsible for taking postings to news groups, archiving them in a database, and then retrieving the messages from the database at the request of the News Group Reader, described in Section 7. Newsrecorder receives the messages and stores them. Newsfeeder is invoked by a request and produces xml file for a message. The diagrams below are the module diagram of newsrecorder and newsfeeder. This section is divided into the following parts:

1. A breakdown of the XML message received from the JMS server by the Newsrecorder.

2. A description of the Newsrecorder class, which breaks down the XML message (a newsgroup posting, possibly with attachments).
3. A description of the Newsfeeder, which queries the data base for the messages and reconstructs them.

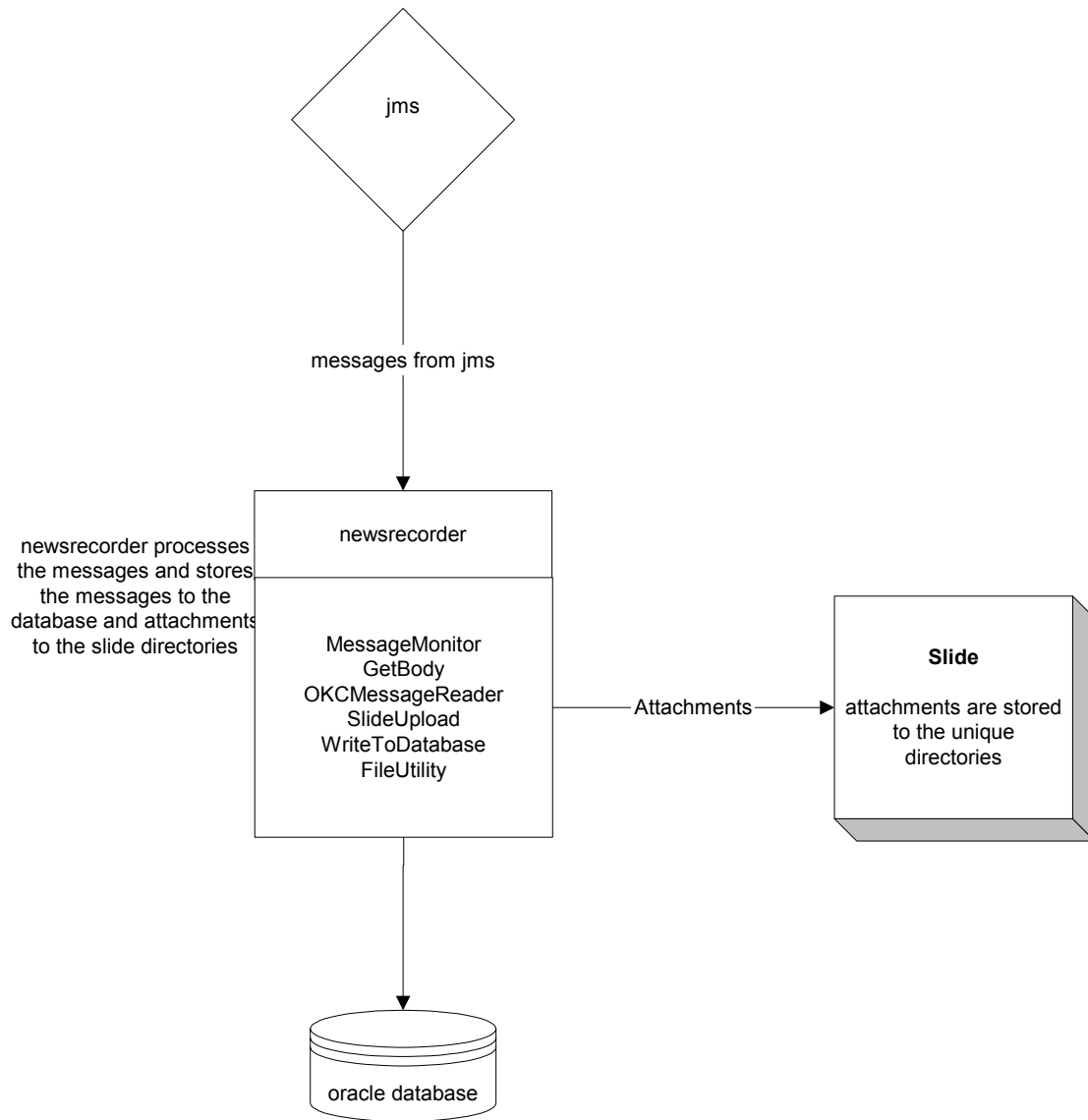


Figure 28 The NewsRecorder module diagram.

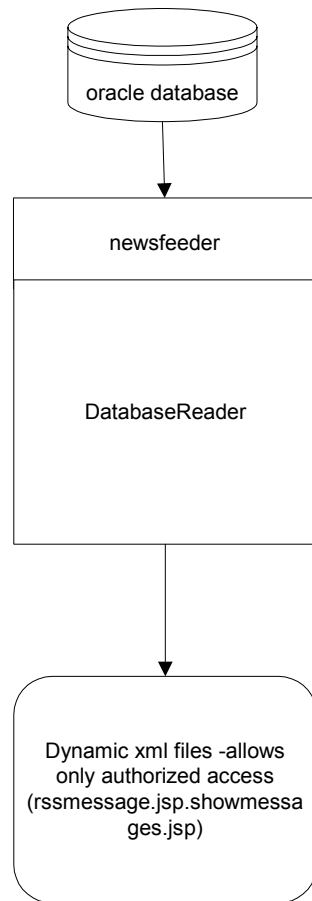


Figure 29 NewsFeeder diagram.

7.2. THE MESSAGE

The OKC Message object is developed to be used within Grid Messaging System (GMS) and it complies with the GXOS EventObject specifications. It extends EventObject with additional elements.

7.2.1. *Message Structure*

In this version, the MIME message structure is adopted. MIME (for Multipurpose Internet Mail Exchange) is a standard Internet protocol for describing messages with multiple parts, non-ASCII text messages, and so forth. An OKC message with its attachments is serialized as a single JMS message body and posted as is. The following chart illustrates the message structure for MailHandler messages.

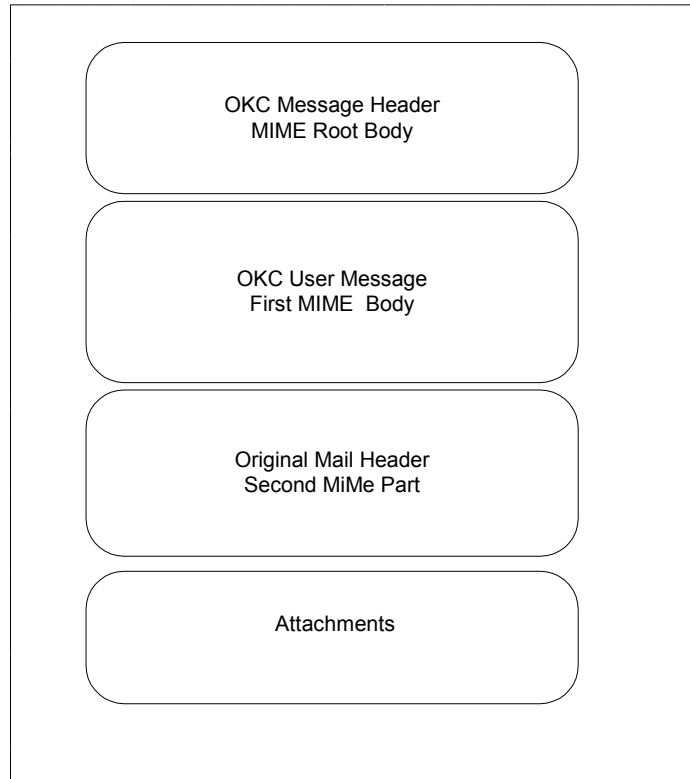


Figure 30 Components of an OKC newsgroup posting.

1. **OKC Message Header** The message header contains all the necessary information for a GMS message. It inherits EventObject which is derived from TreeObject.

2. **OKC User Message** The User Message has xml format and mainly includes the message from user, message id,URI and reference uri, which is used for replied messages.

3. **Original Mail Header** This part of the message consists the mail header.

4. **Attachments** There can be more than one attachment. Each attachment comes in a MIME part. After third MIME Part, we assume the MIME parts are attachment.

7.2.2. *Sample Messages*

MIME-Version: 1.0

```
Content-Type: multipart/related;
boundary="-----_Part_0_7469984.1005786325213"
```

```
-----_Part_0_7469984.1005786325213
Content-Type: text/xml
Content-Transfer-Encoding: 8bit
Content-ID: <1651189.1005786325213.apache-soap.montblanc>
```

Content-Length: 296

```
<?xml version="1.0"?>
<OKCMessage xmlns="http://grids.ucs.indiana.edu/schemas/okc/message/1"
xmlns:gxos="http://aspen.csit.fsu.edu/project/gxos">
  <gxos:NodeName>12345</gxos:NodeName>
  <gxos:NodeType>Event</gxos:NodeType>
  <gxos:NodeRooting>Leaf</gxos:NodeRooting>
  <gxos:NodeVersionNumber>0.3</gxos:NodeVersionNumber>
  <gxos:NodeStartTime>
    <gxos:Time>1005786509068</gxos:Time>
    <gxos:TimeSyntax>GMTMillisecs</gxos:TimeSyntax>
    <gxos:ReferenceAction>begin</gxos:ReferenceAction>
  </gxos:NodeStartTime>
  <gxos:EventDestination>
    <gxos:Destination>
      <gxos:DestinationType>Topic</gxos:DestinationType>
    </gxos:Destination>
    <gxos:DestinationforJMS>
      <gxos:DestinationTopic>Community Grids Research
Group</gxos:DestinationTopic>
    </gxos:DestinationforJMS>
  </gxos:EventDestination>
  <gxos:SourceDevice>
    <gxos:InternalAddress>gxos://okctest/devices/colima</gxos:InternalAddress>
  </gxos:SourceDevice>
  <gxos:Priority>50</gxos:Priority>
  <gxos:EventUserName>mailhandler</gxos:EventUserName>
  <gxos:EventUserObject>
    <gxos:InternalAddress>gxos://okctest/users/mailhandler</gxos:InternalAddress>
  </gxos:EventUserObject>
  <gxos:GMSMessageID>
    <gxos:GMSSequenceNumber>12345</gxos:GMSSequenceNumber>
    <gxos:GMSStream>
      <gxos:InternalAddress>gxos://okctest/events/mailhandler</gxos:InternalAddress>
    </gxos:GMSStream>
  </gxos:GMSMessageID>
  <Sender email="ozgur@csit.fsu.edu">Ozgur Balsoy</Sender>
  <Subject>&lt;okc subject="this is about okc";&gt;</Subject>
  <Attachments>
    <Attachment href="cid:attachments/okc_body"
      contenttype="text/xml">OKC User Message Body</Attachment>
    <Attachment href="cid:attachments/mailheaders"
      contenttype="text/xml">Original Mail Headers</Attachment>
  </Attachments>
</OKCMessage>
```

-----=_Part_0_7469984.1005786325213

Content-Type: text/xml

Content-ID: cid:attachment/okc_body

```
<?xml version="1.0" encoding="UTF-8"?>
<okc xmlns="http://grids.ucs.indiana.edu/okc/schema/admin/ver/1"
xmlns:cg="http://grids.ucs.indiana.edu/okc/schema/cg/ver/1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://grids.ucs.indiana.edu/okc/schema/admin/ver/1
```

```

http://grids.ucs.indiana.edu/schemas/okc-v1.xsd
http://grids.ucs.indiana.edu/okc/schema/cg/ver/1
http://grids.ucs.indiana.edu/schemas/commgrids-v1.xsd">
  <comment>OKC message schema developed</comment>
  <sender>Ozgur Balsoy</sender>
  <distribution>Community Grids Research Group</distribution>
  <organization>Community Grids Laboratory,Indiana University</organization>
  <update mode="create" uri="gxos://okctest/users/balsoy/12november2001/1"/>
  <keywords>okc community grids mail handler message schema</keywords>
  <message whitespace="preserve" >

```

In this weekly meeting with Ali Kaplan and Ahmet Topcu, we have published the OKC message schema version 1. The schema is inherited from <keyword>GXOS</keyword> Event Object with additional elements such as Sender, Subject, and Attachments. The schema is available online at

<http://grids.ucs.indiana.edu/schemas/mailhandler/index.html>. Its namespace is <http://grids.ucs.indiana.edu/okc/message/1>.

```

  </message>
  <filingdate>11/12/2001</filingdate>
  <cg:category main="general" />
  <cg:category main="facility" sub="okc" />
  <cg:category main="facility" sub="other" other="mailhandler" />
  <cg:category main="research" sub="okc" />
  <cg:messagetype>Weeklyreport</cg:messagetype>
</okc>

```

```

-----=_Part_0_7469984.1005786325213
Content-Type: text/xml
Content-ID: cid:attachment/mailheaders

```

```

<mailheaders>
  <header name="Received">by mailer.csit.fsu.edu (mbox ozgur)
  (with Cubic Circle's cucipop (v1.31 1998/05/13) Sat Oct 13 12:13:44
  2001)</header>
  <header name="X-From_">owner-cgstudents-1@LISTSERV.INDIANA.EDU  Fri Oct 12
  21:13:25 2001</header>
  <header name="Return-Path"><owner-cgstudents-
  1@LISTSERV.INDIANA.EDU></header>
  <header name="MIME-Version">1.0</header>
  <header name="Content-Type">text/plain; charset="iso-8859-1"</header>
  <header name="Date">Fri, 12 Oct 2001 20:00:25 -0500</header>
  ...
</mailheader>
-----=_Part_0_7469984.1005786325213-

```

7.3.NEWSRECORDER

Newsrecorder subscribes to the JMS server and listens for all messages sent from JMS posters. When a message is received, it is processed and recorded to the database. The attachment, if there is any, is stored to the unique directory where it can be looked up and retrieved later. We are currently using Slide for this.

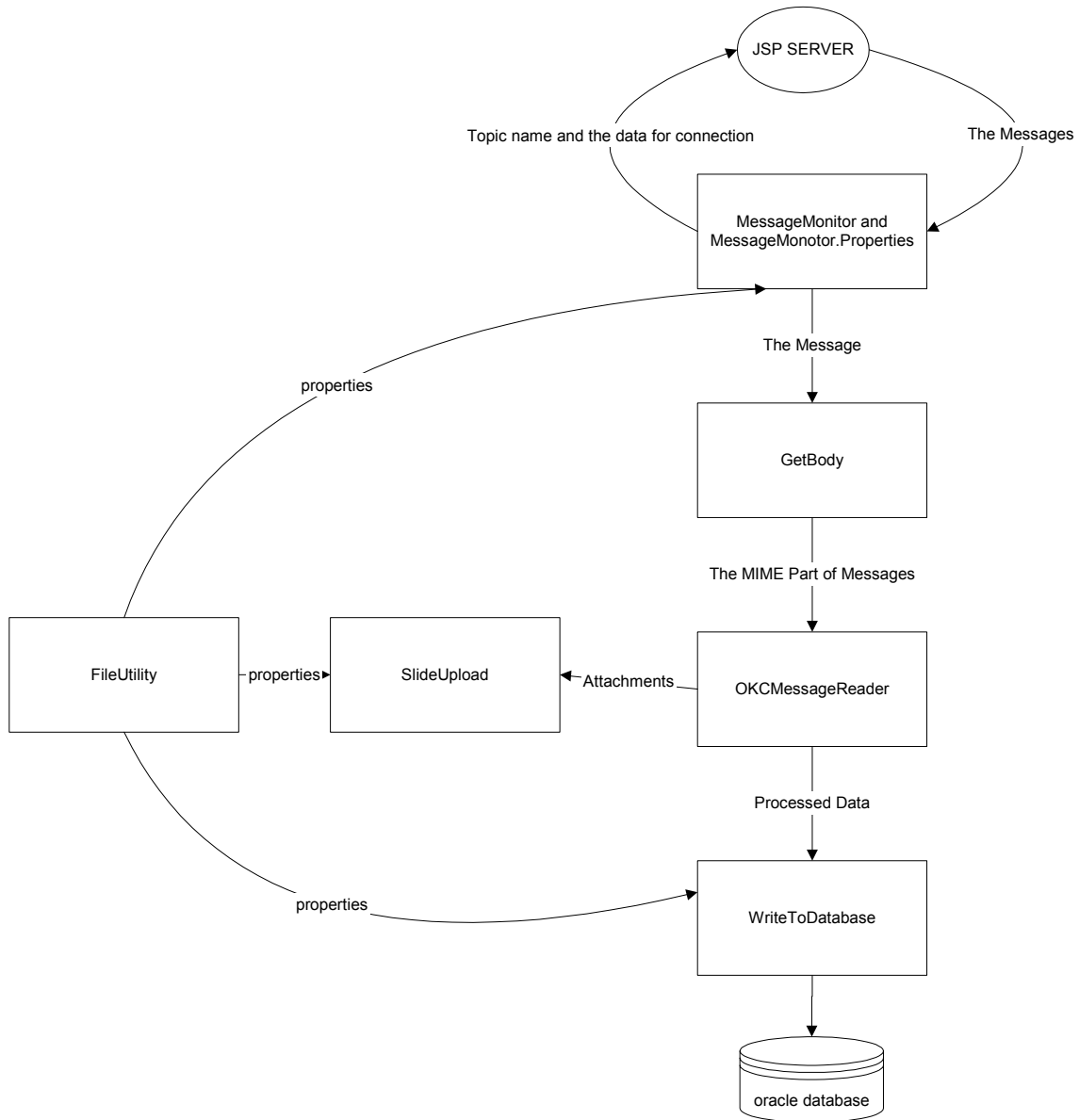


Figure 31 Data flow chart for NewsRecorder.

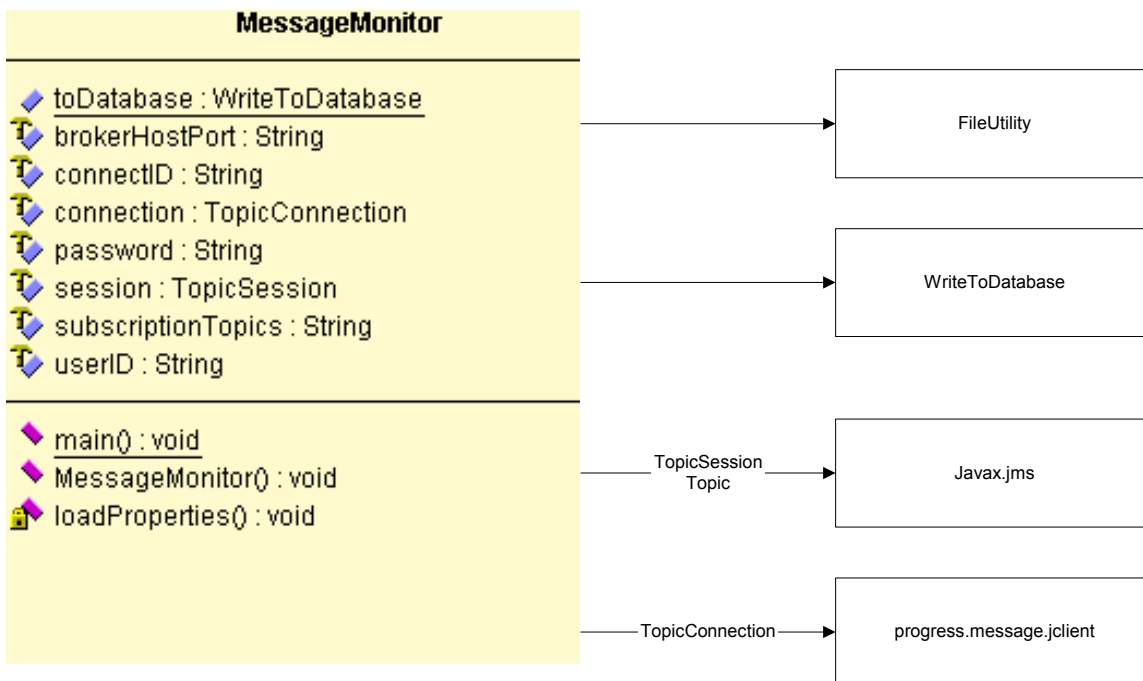
7.3.1. Class Structures

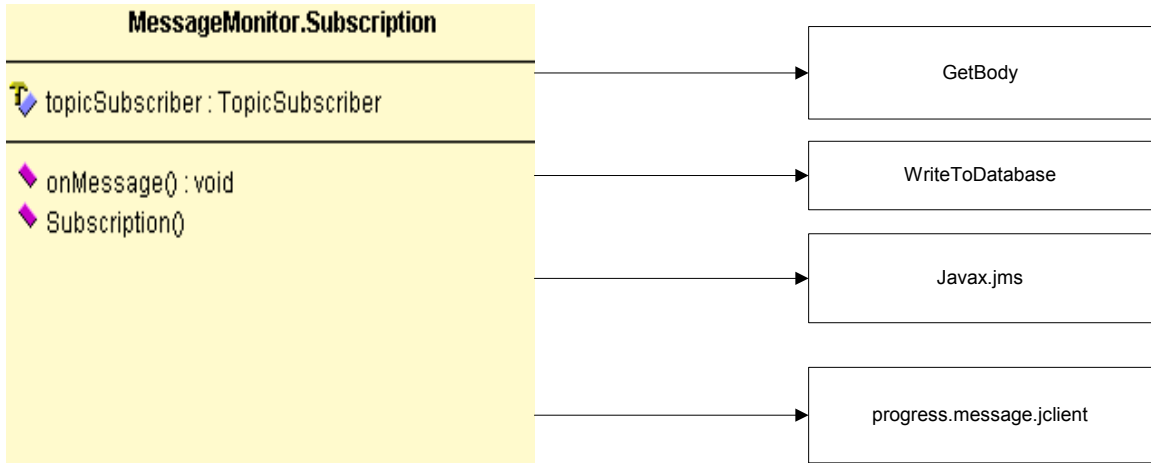
There are six classes in this module. The program is started by calling the main function in MessageMonitor. After the running this program, it starts to subscribe the JMS Server by getting the required information from the properties file. The topic names which will be subscribed are loaded from the topic table, in database. After the subscription to the topics, MessageMonitor.Subscription starts to listen the topics and

receives the messages according to the topic name. The received message is sent to the Getbody class to get the MIME Parts.

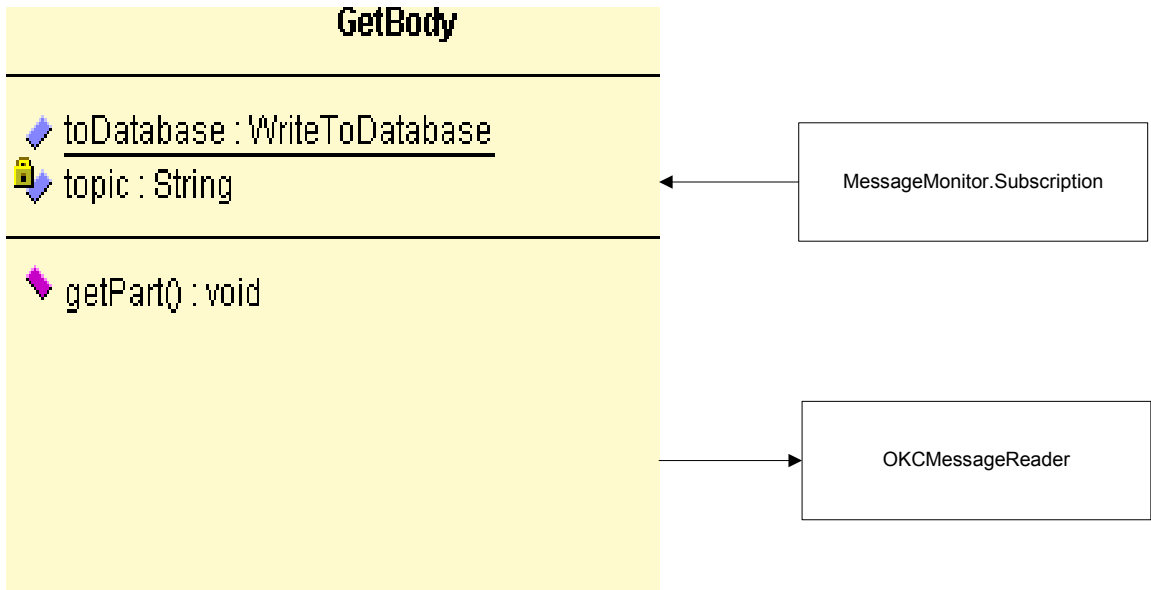
Topic table description:

Name	Null?	Type
TOPICID	NOT NULL	VARCHAR2(128)
TOPICNAME		VARCHAR2(256)





GetBody receives the message as a byte array and gets the parts from it. The message consists of four parts, OKC Message Header, OKC User Message, Original Mail Header and Attachments. After getting these parts, they are sent to the OKCMessageReader .



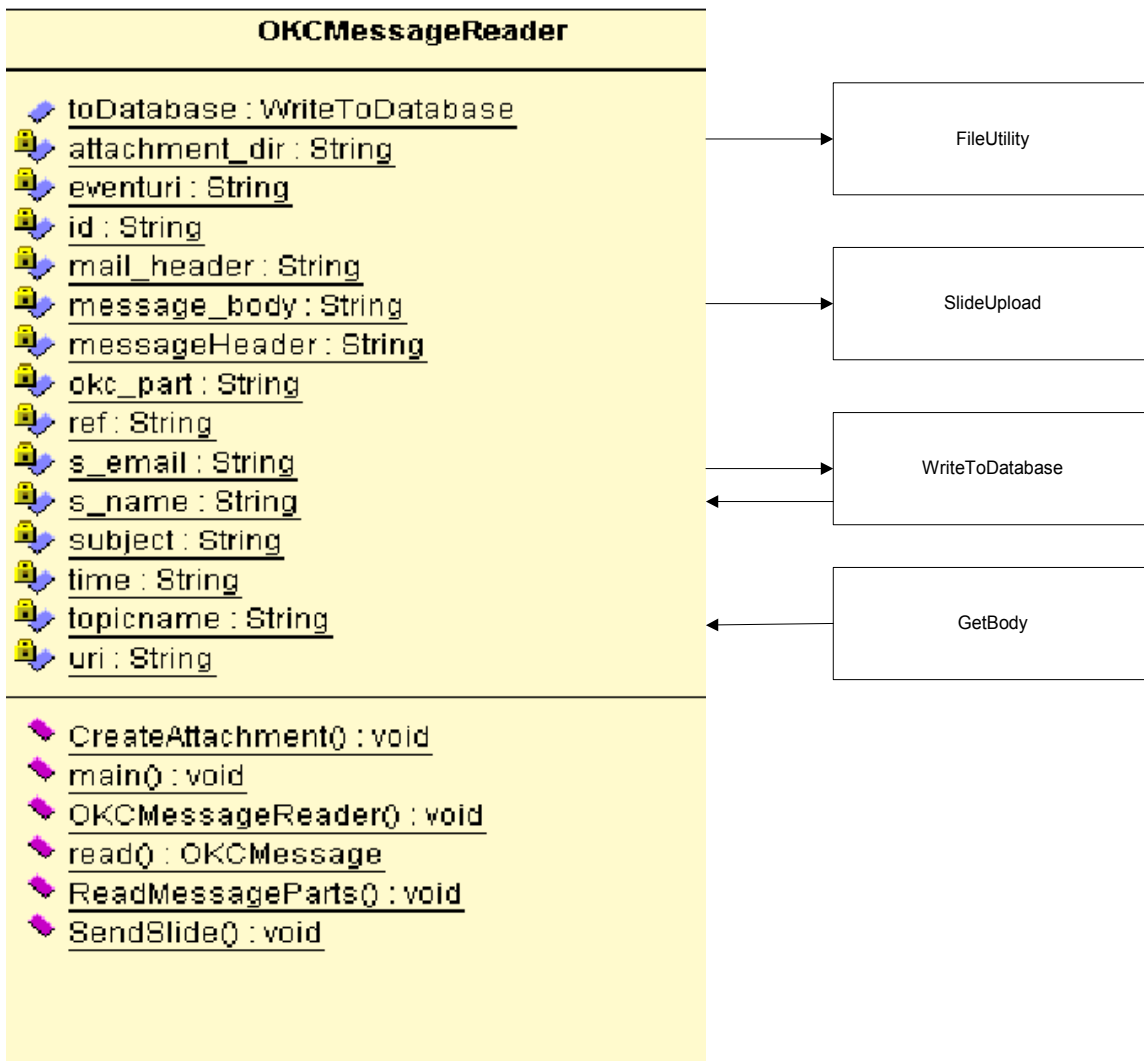
The first MIME Part is OKC Message Header. It includes required information to store the data to the database. The data is gotten by making it unmarshal .These data are sender name, sender email, subject , eventuri ,time ,topicname.

The second MIME Part , OKC User Message consists of the original user message. It is also made unmarshal to get uri, unique id of the messages, and reference uri. The reference uri is used in replying the messages.

The third MIME Part is the Original Mail header.

These three parts, OKC Message Header , OKC User Message,Original Message header, and derived data, uri , referenceuri , topic , sender , email ,subject, date , eventuri, are sent to WriteToDatabase to store them.

After Third MIME Part, the coming parts to the OKCMessageReader are attachments. File name and unique directory, unique id of the message , are sent to the SlideUpload to upload the file.



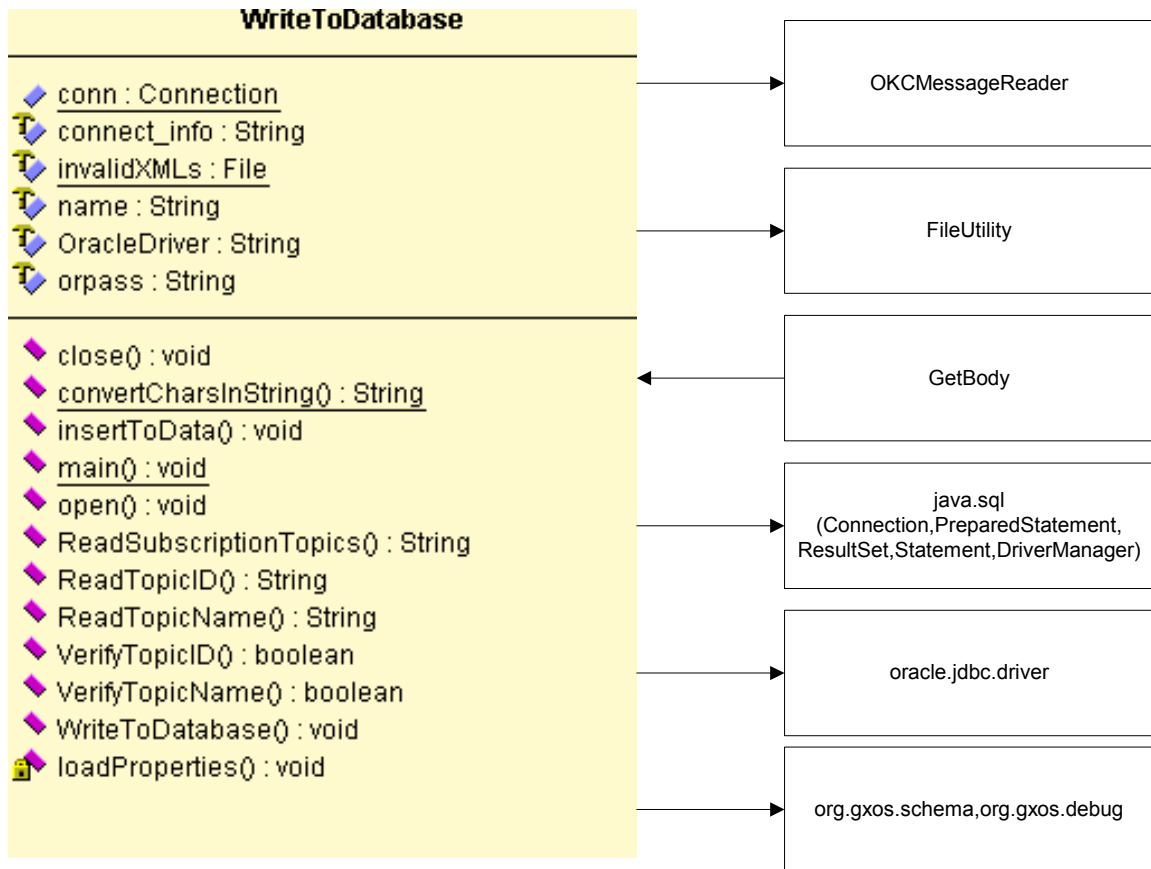
After getting the required data from the OKCMessageReader, they are stored to the database by using JDBC. We use two main database tables to store a message.

Newsgroup table description:

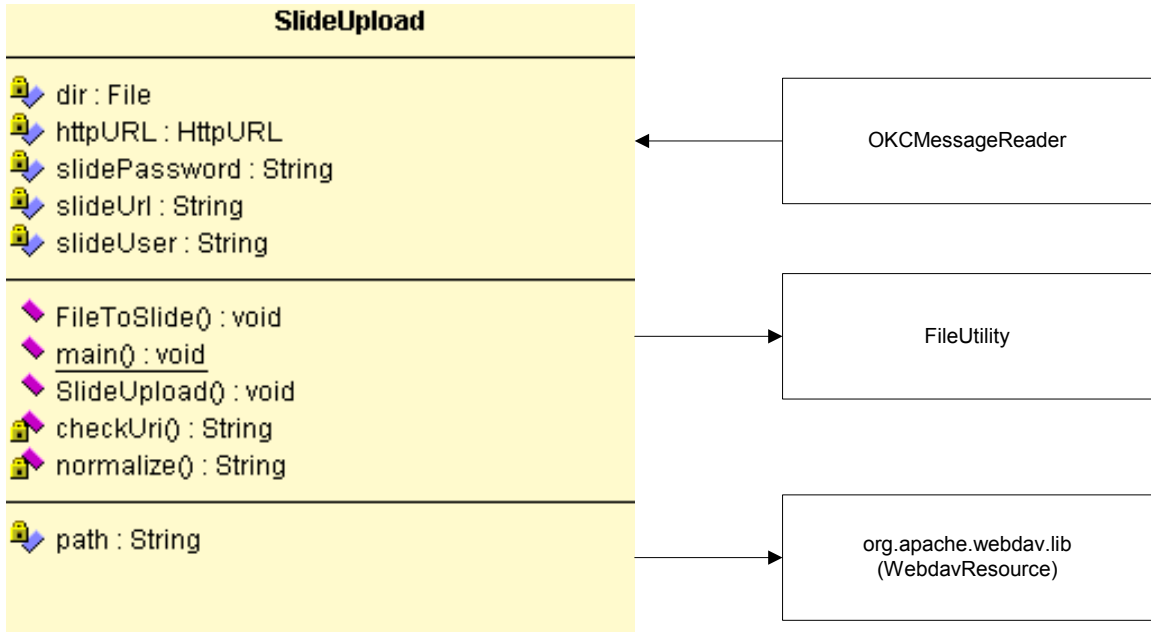
Name	Null?	Type
URI	NOT NULL	VARCHAR2(256)
TOPICID	NOT NULL	VARCHAR2(16)
SENDER	NOT NULL	VARCHAR2(64)
SUBJECT	NOT NULL	VARCHAR2(256)
FILLINGDATE	NOT NULL	VARCHAR2(16)
EVENTURI	NOT NULL	VARCHAR2(256)
MESSAGE		SYS.XMLTYPE
HEADERS		SYS.XMLTYPE
REFERENCEURI		VARCHAR2(256)
EMAIL		VARCHAR2(256)

Gxosobjects table description:

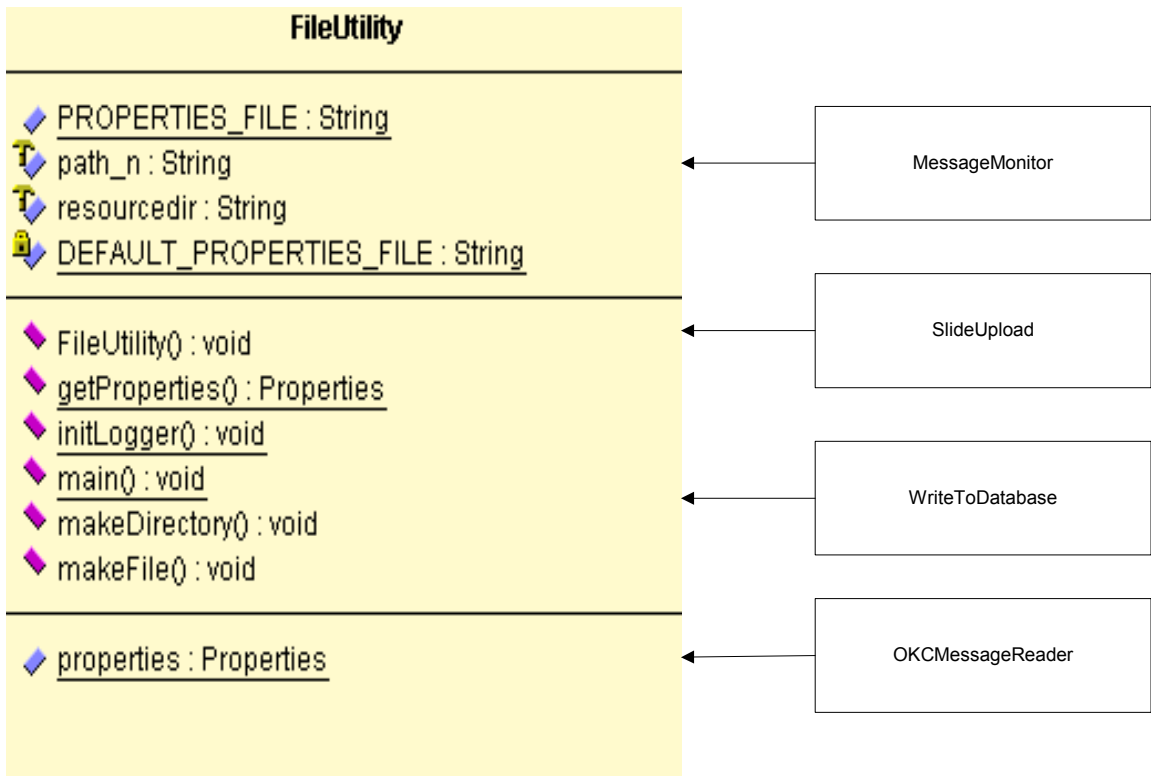
Name	Null?	Type
URI	NOT NULL	VARCHAR2(256)
OBJECT		SYS.XMLTYPE



Slide's SlideUpload class is use to store the attachments to the unique directory. This directory is message id of the messages,uri. The main advantage of the using Slide is that the attachments can be saved to the other servers as well as to the same machine. Beside this, we can see the attachments by using a browser, if we are authorized.



The required data from the properties file is gotten by using the FileUtility class. It can be also used to create file and its path directories.



7.4.NEWSFEEDER

The NewsFeeder is responsible for retrieving and reconstructing messages from the database. The news feeder is invoked by a request from the NewsReader (described in Section 7). The NewsReader controls the visual display of the data provided to it by the NewsFeeder.

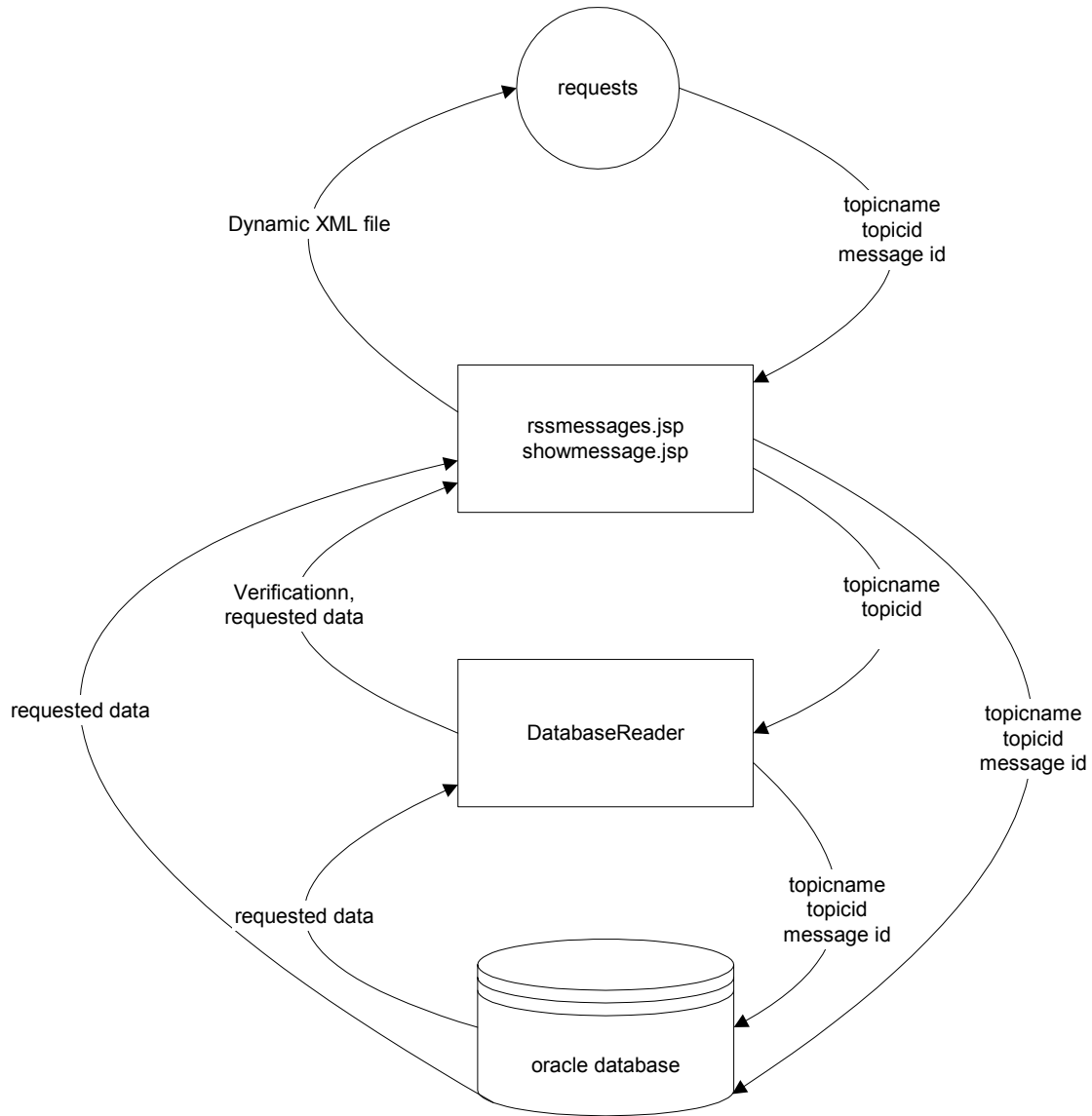


Figure 32 Data flow chart for NewsFeeder.

7.4.1. *rssmessage.jsp*

This JSP file creates a dynamic RSS xml file. RSS (for Rich Site Summary) is a standard XML dialect for displaying link channels. For us, it displays a set of links corresponding to message postings. It takes a topic name and opens database, searches it and closes it with DatabaseReader help. The file includes the required information of all messages in a topic to get them. The requester takes this dynamically created xml files and derived required information to call the showmessage.jsp to get the message body. An example of rss file is below.

```
<?xml version="1.0" ?>
- <rss version="0.91" xmlns:cg="http://grids.ucs.indiana.edu/okc/schema/cg/ver/1">
- <channel>
  <title>Community Grids Lab General Topics</title>
- <image>
  <title>ptllogo</title>
  <url>http://www.communitygrids.iu.edu/img/smallLOGO.gif</url>
  <link>http://www.communitygrids.iu.edu</link>
  <description>Pervasive Technology Labs Logo</description>
  </image>
- <item>
  <title>test ab</title>
  <link>showmessage.jsp?topicname=cgnews&id=gxos://ptliu/newsgroups/cgnews/1</link>
  <description>Fri Mar 22 13:20:48 EST 2002</description>
  <messageuri>gxos://ptliu/newsgroups/cgnews/1</messageuri>
  <sender email="byildiz@grids.ucs.indiana.edu">Beytullah Yildiz</sender>
  </item>
- <item>
  <title>Re:test ab</title>
  <link>showmessage.jsp?topicname=cgnews&id=gxos://ptliu/newsgroups/cgnews/1/1</link>
  <description>Fri Mar 22 14:45:19 EST 2002</description>
  <messageuri>gxos://ptliu/newsgroups/cgnews/1/1</messageuri>
  <sender email="byildiz@grids.ucs.indiana.edu">byildiz</sender>
  <referenceuri>gxos://ptliu/newsgroups/cgnews/1</referenceuri>
  </item>
  </channel>
</rss>
```

7.4.2. *showmessage.jsp*

Showmessage.gsp is also dynamically created xml file. It shows the message body. It takes topic name and message id, URI, as an argument searches the database and gives the file to the requester. The file, which is below, is the first message of the above rss file. It can be seen easily that

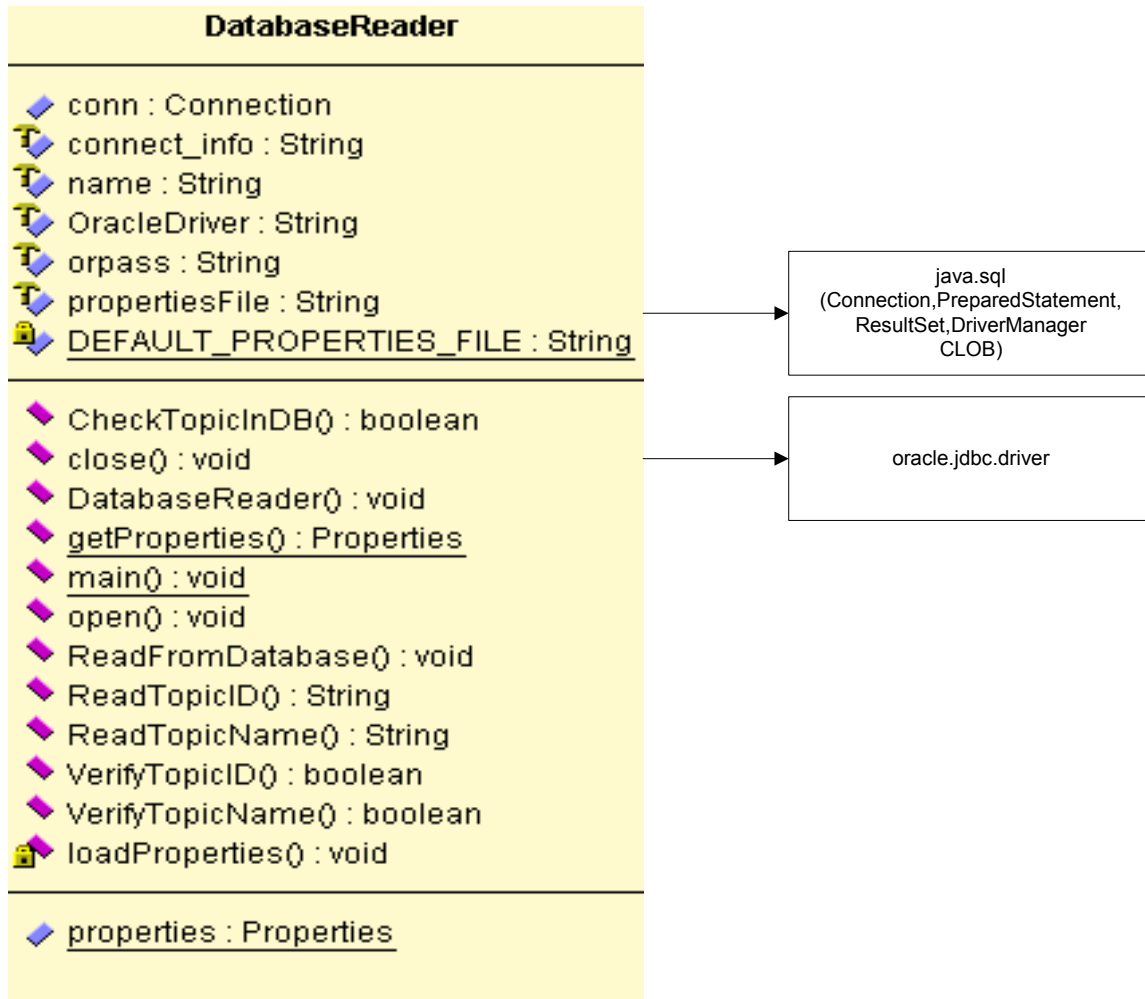
gxos://ptliu/newsgroups/cgnews/1 is the message id which is used to get the message.

```
<?xml version="1.0" ?>
- <okc version="3" xmlns="http://grids.ucs.indiana.edu/okc/schema/admin/ver/3"
  xmlns:cg="http://grids.ucs.indiana.edu/okc/schema/cg/ver/1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <event>
  <cg:messageType>News Message</cg:messageType>
  <comment />
  <sender email="byildiz@grids.ucs.indiana.edu">Beytullah Yildiz</sender>
  <distribution uri="cgnews">Community Grids Newsgroups</distribution>
  <organization>hi</organization>
  <update createuri="gxos://ptliu/newsgroups/cgnews/1" />
  <keywords />
  <subject>test ab</subject>
- <message>
  test ab
  <br />
  </message>
  <filingdate>3/22/2002</filingdate>
  </event>
  </okc>
```

7.4.3. *DatabaseReader.java*

The connection to the database in this module is made by this class. Topic name and topic id search and verification can be made by using ReadTopicID ReadTopicName, VerifyTopicName and VerifyTopicID

methods. These are important because we can get messages according to both topic id (okcnews) and topicname (OKC Newsgroups). The program should internally work according to topic id. However, we must support news group postings both through our JSP interface wizard (described in Section 4) as well as messages posted through a regular email client (such as Outlook), as described in Section 5. Thus, we cannot assume that the user who sends the mail by modifying the xml files by hand knows the topic id. Therefore, the program should work for both of them.



7.5. Security Note: Authorized Access

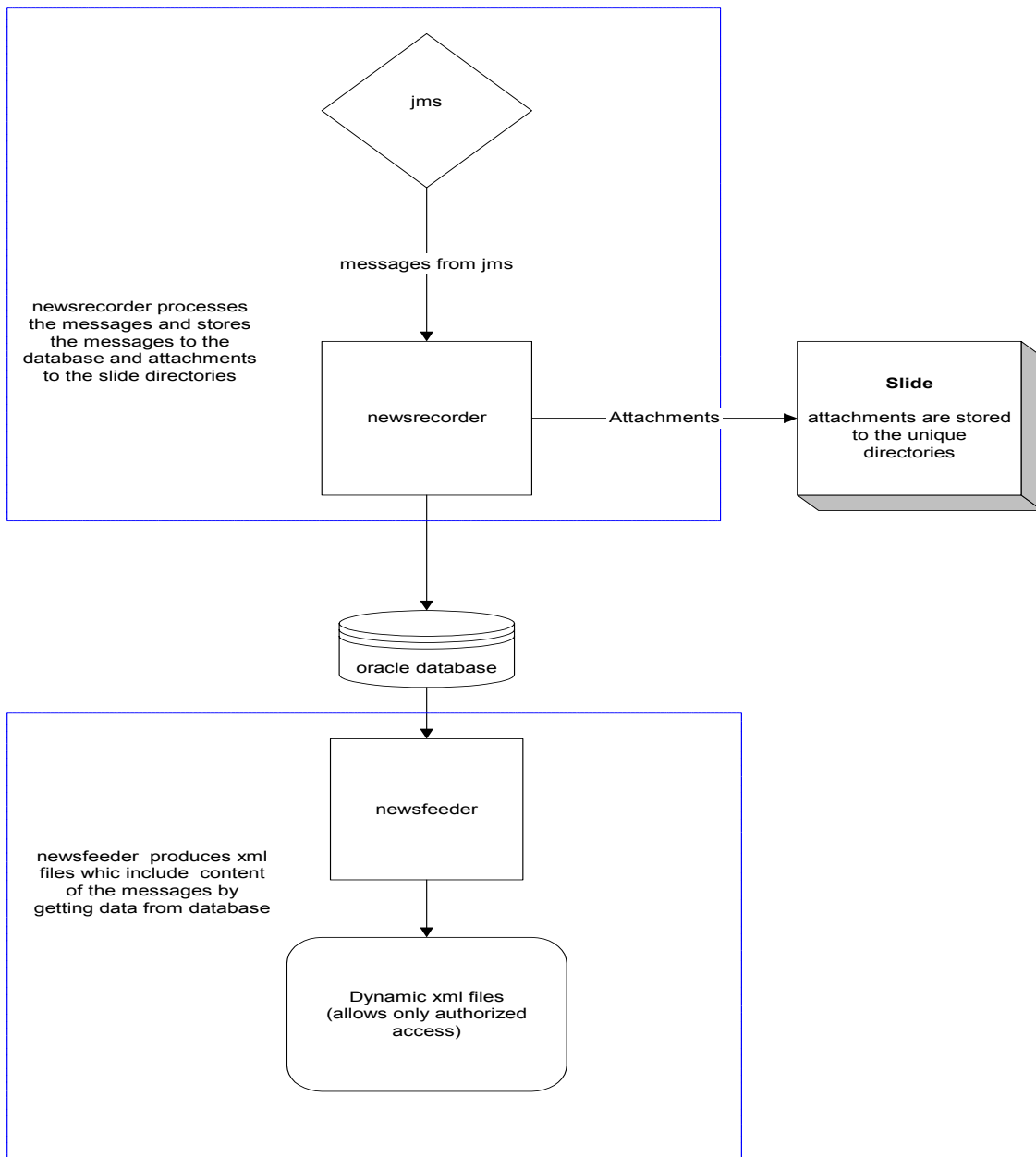
The JSP files rssmessage.jsp and showmessage.jsp are normally accessed directly only by other JSP files. However, there is nothing to prevent a user that knows the URL of showmessage.jsp from directly accessing it. Thus a person could read messages posted to newsgroups without first getting permission from the newsgroup

administrator (described in Section 7). Therefore, these two files check to see if the coming request comes from an authorized source.

7.6.CONCLUSION

In this part of OKC, there are two modules. One of them, newsrecorder, takes the messages from the jms server, which is subscribed to its topics when the program is started. Newsrecorder processes and stores the received messages to the database and uploads the attachments to a unique directory by using slide.

The second module is newsfeeder. It is invoked by a request. It searches the database and returns the asked data in xml format. JDBS, JMS,JSP and Slide are used to make this program. The figure below shows the combined newsrecorder and newsfeeder system.



8. Newsgroup Reader, Request, and Administration

8.1.Introduction

Newsgroup reader, request and administrator are the main structure of the newsgroup. In this structure, we have a security model for system. Users come to a JSP interface that allows them to sign up to newsgroup. The user can request several optional permission levels: to read messages, to post messages, and to become admin for particular newsgroups. This request is confirmed by the administrator before the user can access the newsgroup.

Besides permissions, the system gives the user the option to get the copy of the newsgroup message to his or her regular email account. Users can also optionally received attachments posted to newsgroups, and can receive their own newsgroup postings by email. These options give the user a flexible control mechanism to use newsgroups.

The administrator for a newsgroup has always the option to confirm user requests, modify user access, and add new users to a newsgroup. By having multiple administrators, the system doesn't depend on one administrator to control system. Each newsgroup may have multiple administrators to use systems. However, all the users must register to Jetspeed in order to use the newsgroups.

This system has dynamic structure to show multiple newsgroup lists on the page, if the user has access to use that newsgroups, user have the option to select desired newsgroup from the list. According to user's rights, the messages will be shown on the group. The JSP page is used for this structure. All the user rights and request are stored in the Oracle Database, and Java Script is used for user-friendliness.

In this section, request and administrator information are stored in the database. User requests for selected newsgroup.

8.2.Newsgroup Structure

The following summarizes the permissions that can be requested when registering for a newsgroup. These are confirmed or denied by the administrator for that group.

Permissions

rA, read access: You want to read messages using the NewsReader.

wA, write access: You want to send messages to this newsgroup.

aA, admin access: You want to administer this newsgroup.

Configuration

eN, email notification: You want to receive other people's messages.

aN, attachment notification: You want to receive attachments messages might have.

ccN, CC notification: You want to receive your message as well.

Steps for using the newsgroup structure figure shown below:

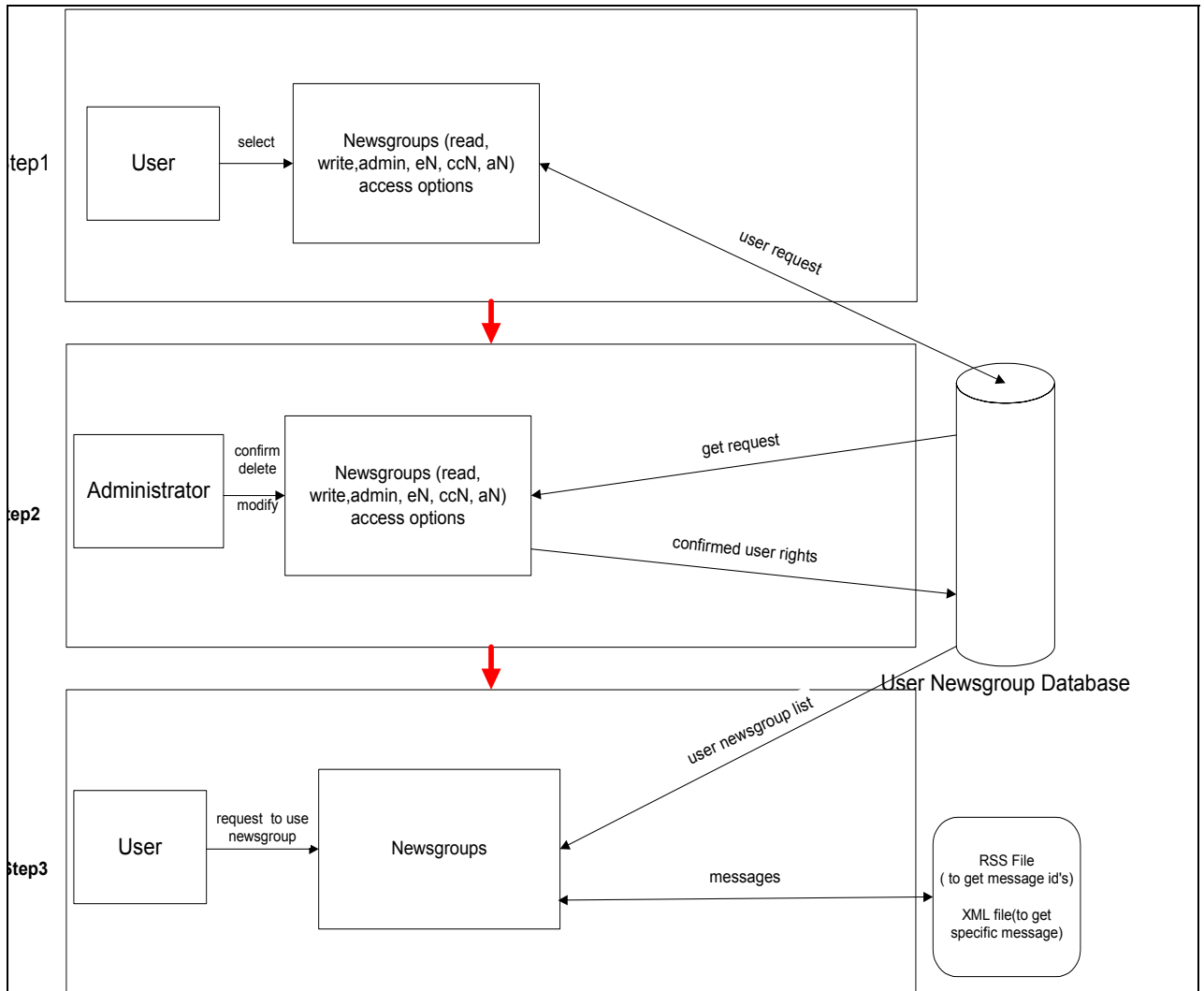


Figure 33 Steps for registering for a newsgroup.

8.2.1. Step 1: Request

User makes request for any newsgroup which is open to the users. In order to make a request you need to enter email address in the email sections because the system has the structure for using email for attachment notify, cc Notify and email Notify.

There are four sections for request section.

1. **Current rights (unchanged):** for newsgroups. For this section, users only see their confirmed rights for newsgroups.
2. **Pending request:** User's pending will be shown in this section.
3. **Unselected newsgroup:** for that user hasn't selected that newsgroup yet.
4. **Current rights:** User can modify their access in this newsgroup section.

The first part cannot be modified or deleted by user, but for the other section, user can have a request or modify the newsgroups.

Newsgroup current right section: User can see their current registered newsgroup list. In this newsgroup list, they can observe their rights for newsgroups without changing their rights. When the user makes a request for newsgroups, their request is kept in the database. It will be stored in the request table until their request is confirmed by the administrator. After the requests are confirmed by the administrator, users can automatically use the confirmed newsgroups.

In order to distinguish the newsgroups, checkbox name is unique for each newsgroup. For example, "group_cgnews" is different than the "group_cgreports". "group_" string added to the beginning of the each newsgroup name. The difference is shown below:

```
<td width="10%" height="34"><b>cgnews</b></td>
<td width="5%" height="34"><input type="checkbox" name="group_cgnews" value="read" checked
disabled></td>

<td width="10%" height="34"><b>cgreports</b></td>
<td width="5%" height="34"><input type="checkbox" name="group_cgreports" value="read"
checked disabled></td>
```

User has to check the Select checkbox for requested newsgroup. Otherwise, the request will not be accepted.

8.2.2. Step2: Confirmation

The current system has an option for selection mechanism for newsgroup. We define roles for newsgroup. Admin role, user role, and top admin role defined in the system. This allows us to define other roles, when we need to define and use. This structure allows dynamic structure in the newsgroup.

8.2.3. Step3: News reader

OKC newsgroup displayer uses RSS URI to construct the e-mail/newsgroup hierarchy and to get the body of messages. The NewsFeeder (described in Section 6) constructs the RSS file at the request of the NewsReader. The NewsReader is then responsible for formatting the display of the news group postings. XSL is used to extract data from XML file in order to show the required messages to the users. OKC newsgroup displayer checks the user's access rights by using the database. User access rights allow users to read from and write into newsgroup topics. Figure shown below explain the scenario.

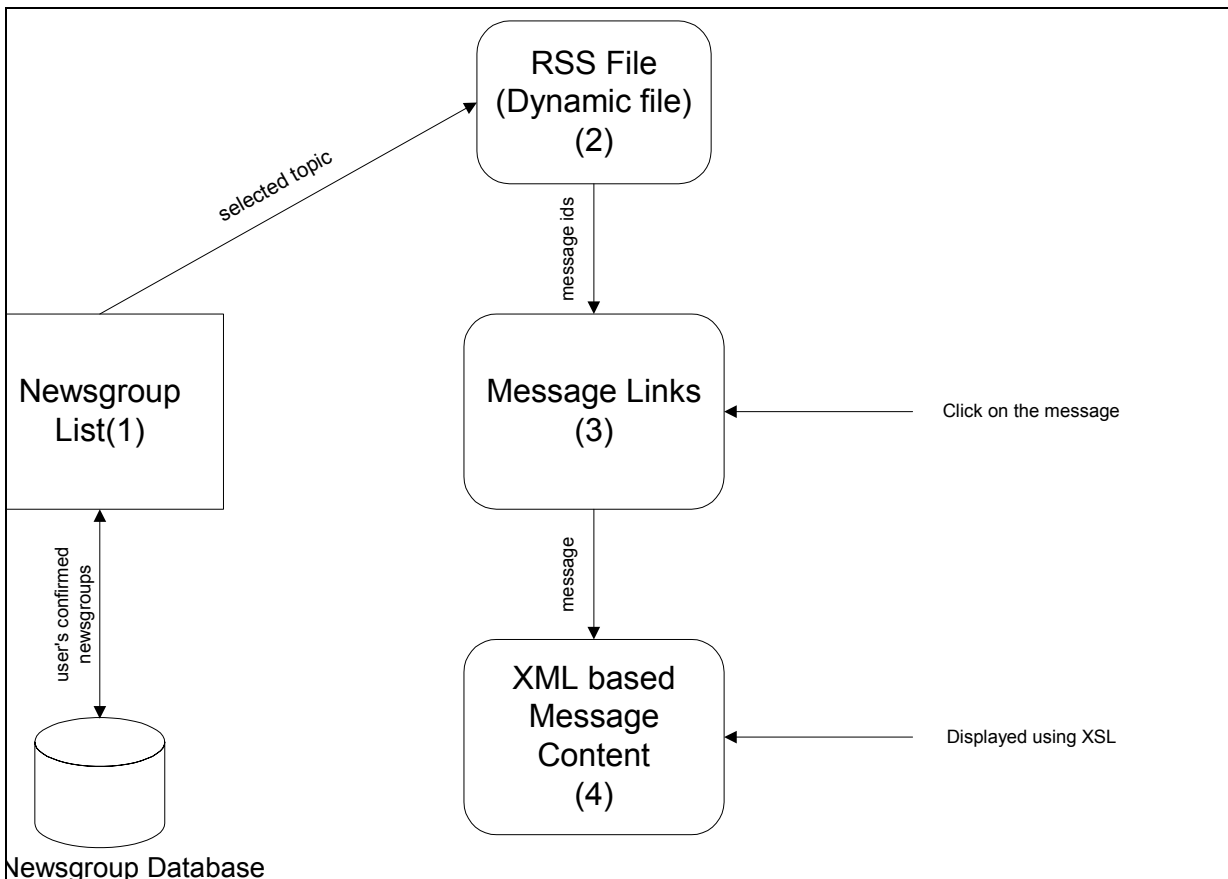


Figure 34 Recovering and displaying messages for a newsgroup.

In the newsreader, when the user selects the newsreader, user's current access newsgroup will be listed on the newsgroup list section. The newsgroup list will shown, when user click one of the links on the newsgroup list. A user can access the messages if he/she has read access of the newsgroup. Here, we secure our newsgroup according to the user's rights. If they have access to read only, user can read the message. Users only post messages if they have write access to newsgroup. These rights are requested by user first and then are confirmed by administrator.

The posted messages keep in the RSS file. According to user's selection of the newsgroup, the message list will extracted from RSS file. Dom Parser is used to parse the XML object in newsreader part. Here is the example for RSS file. Messages will be threaded according to the **messageuri**. Each message has unique message id. If the message is replied to one of the message, the reference id will be shown in the RSS file. Using the messageuri and referenceuri, threaded message structure is established in the message lists. RSS file shown below:

```
<? xml version="1.0" ?>
<rss version="0.91" xmlns:cg="http://grids.ucs.indiana.edu/okc/schema/cg/ver/1">
<channel>
  <title>Portal Related Projects</title>
<image>
  <title>ptllogo</title>
  <url>http://www.communitygrids.iu.edu/img/smallLOGO.gif</url>
  <link>http://www.communitygrids.iu.edu</link>
  <description>Pervasive Technology Labs Logo</description> </image>

<item>
  <title>test</title>
  <link>showmessage.jsp?topicname=portal&id=gxos://ptliu/newsgroups/portal/1</link>
  <description>Fri Mar 08 18:39:31 EST 2002</description>
  <messageuri>gxos://ptliu/newsgroups/portal/1</messageuri>
  <sender email="maysan@indiana.edu">Necati AYSAN</sender>
</item>
<item>
  <title>(no subject)</title>
```

```
<link>showmessage.jsp?topicname=portal&id=gxos://ptliu/newsgroups/portal/2</link>
<description>Fri Mar 22 13:08:41 EST 2002</description>
<messageuri>gxos://ptliu/newsgroups/portal/1/messageuri>
<sender email="atopcu@indiana.edu">Ahmet Topcu</sender>
<referenceuri> gxos://ptliu/newsgroups/portal/1/1</referenceuri></item>

</item>
</channel>
</rss>
```

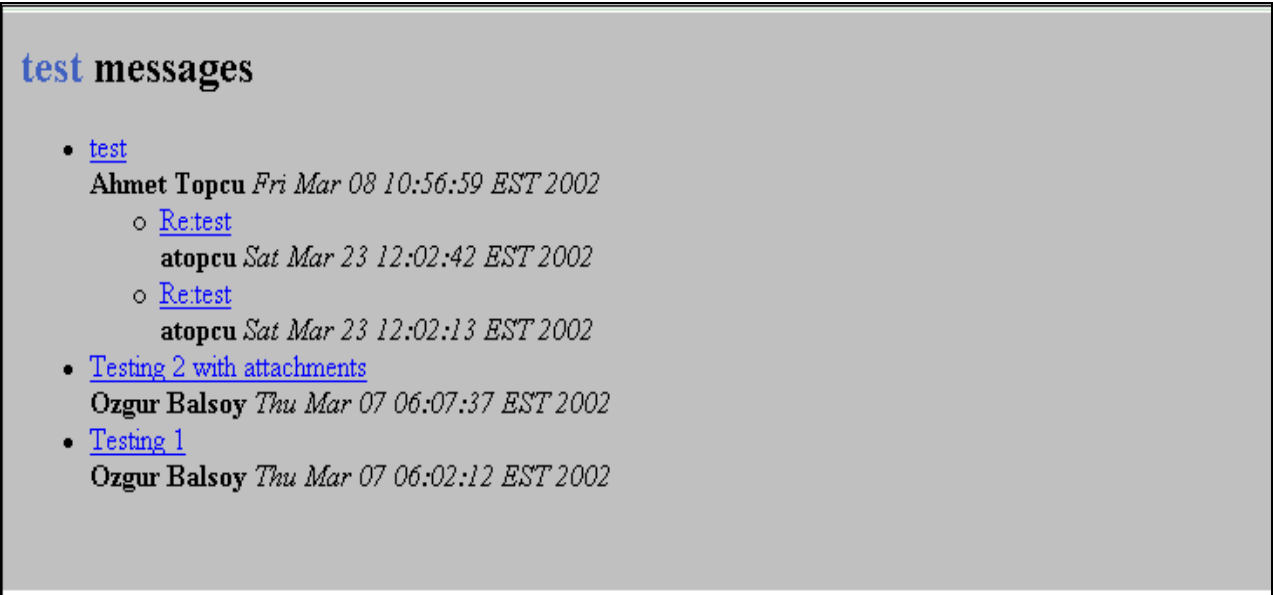


Figure 35 A sample newsgroup listing display generated from RSS with XSL.

8.2.3.1 XSL

This specification defines the Extensible Stylesheet Language (XSL). See Ref [XSL] for more information. XSL is a language for expressing stylesheets. In our model, all the messages have an URI. The message is in the XML form. In order to show the message content, it's transformed into the XSL form. By using the XSL, the message source content can be styled, laid out, and paginated onto desired view. When the user selects one of the messages from the message list, message displayed as shown below:

REPLY NEW POST	
Sender	Ahmet Topcu <atopcu@indiana.edu>
Comment	testing
Distribution	Test Messages
Organization	pervasive tech labs
Update	gxos://ptliu/newsgroups/test/4
Keywords	XSL
Message	<pre> hi, this is test regards </pre>
Filingdate	3/23/2002
Message Type	News Message

Figure 36 A newsgroup posting entry obtained by clicking a link in the thread display.

8.3. Newsgroup User Interface

We have developed a set of browser interfaces for newsgroup users and administrators to manage their use of the system. To summarize, users can modify their settings for subscribed groups, see which requests have been submitted but not yet accepted by the administrator, and subscribe to new newsgroups.

8.3.1. *User Interface*

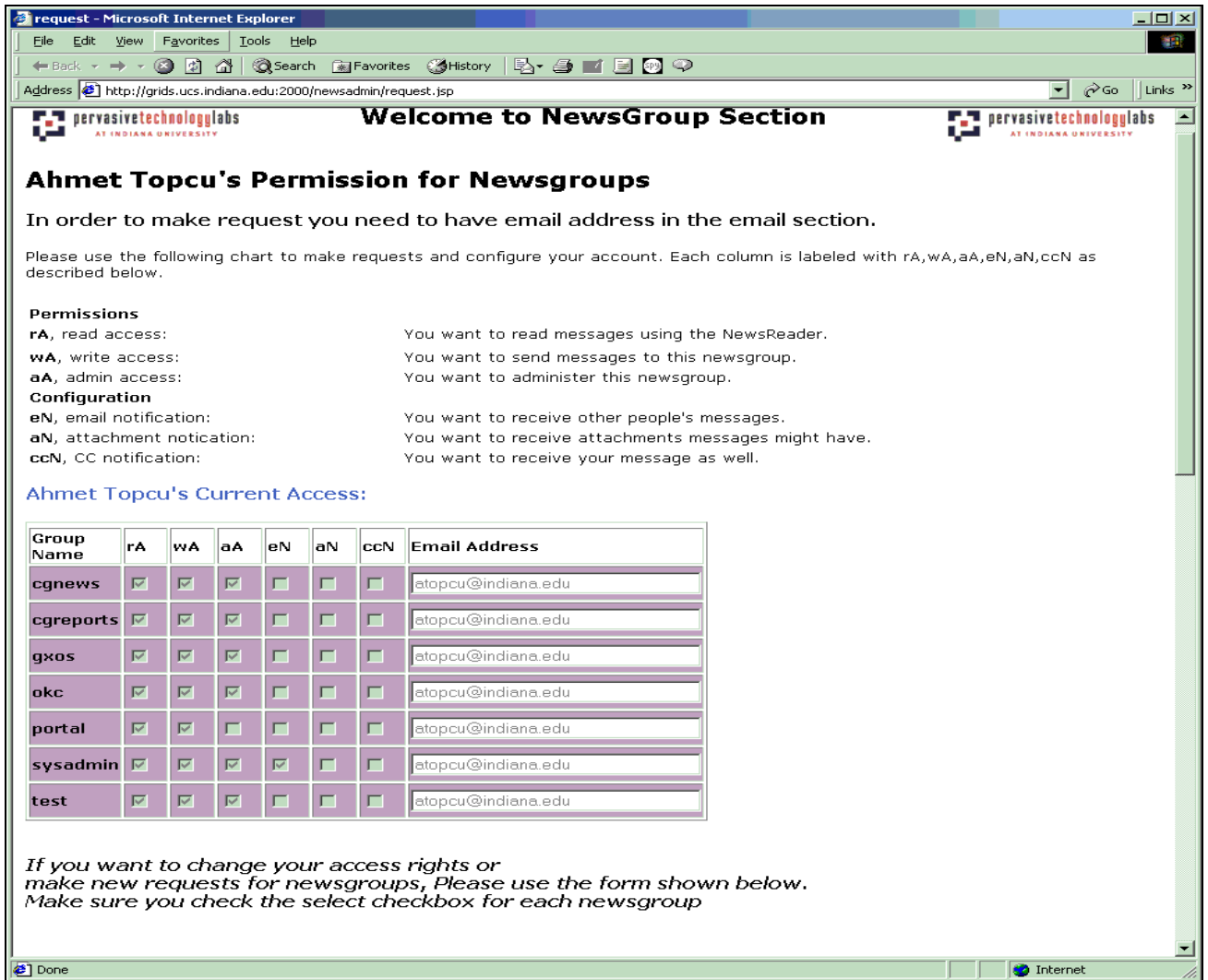


Figure 37 The user interface for requesting changes in newsgroup permissions and settings.

Request Interface: This page has four parts. Figure above shows the user's current rights for newsgroup. User gets the current rights for the newsgroups.

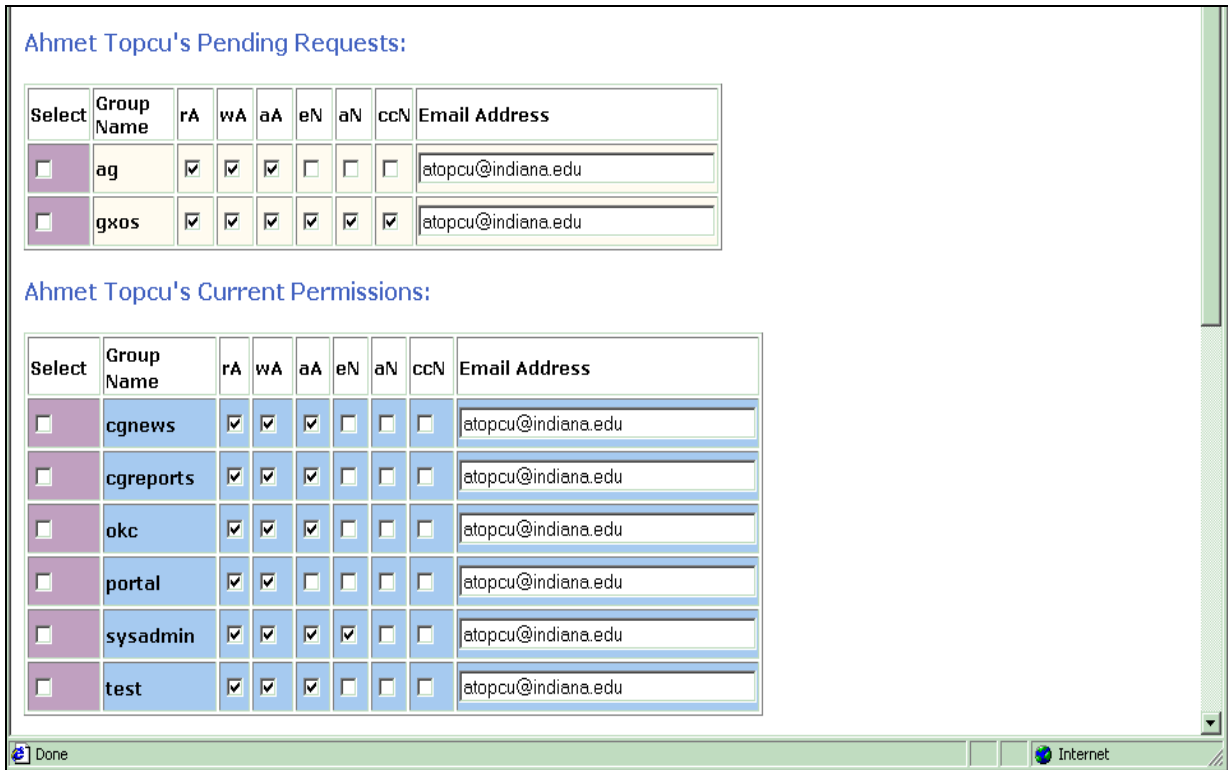


Figure 38 A display of pending requests not yet approved by an administrator.

Pending Request: In the first part of the figure shown above, user's pending request displayed. These newsgroups are not confirmed by administrators. They are waiting for confirmation by administrator.

Current Permission: Second part of the figure gives user an option to change current permission of the newsgroup. Actually, it repeats the disabled current newsgroup shown in the previous figure. But, it doesn't repeat the same newsgroups exist in the pending newsgroup.

Ahmet Topcu's Unselected Newsgroups:

Select	Group Name	rA	wA	aA	eN	aN	ccN	Email Address
<input type="checkbox"/>	database	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	gateway	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	ges	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	globus	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	hpjava	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	hwtest	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	jms	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	jxta	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	kerberos	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	multimedia	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	pda	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	svg	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	uddi	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu
<input type="checkbox"/>	wSDL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	atopcu@indiana.edu

Submit Selections

Done Internet

Figure 39 A list of unselected newsgroups.

Unselected Newsgroup: If the user doesn't request some of the newsgroup, these are listed on the unselected newsgroup section. Email address is automatically displayed in the newsgroups email field. It is same as the JetSpeed registered email address. But, users have option to change email address. In order to make request, use have to select the selection checkbox for each newsgroup. Otherwise, request cannot be accepted.

8.3.2. Administration

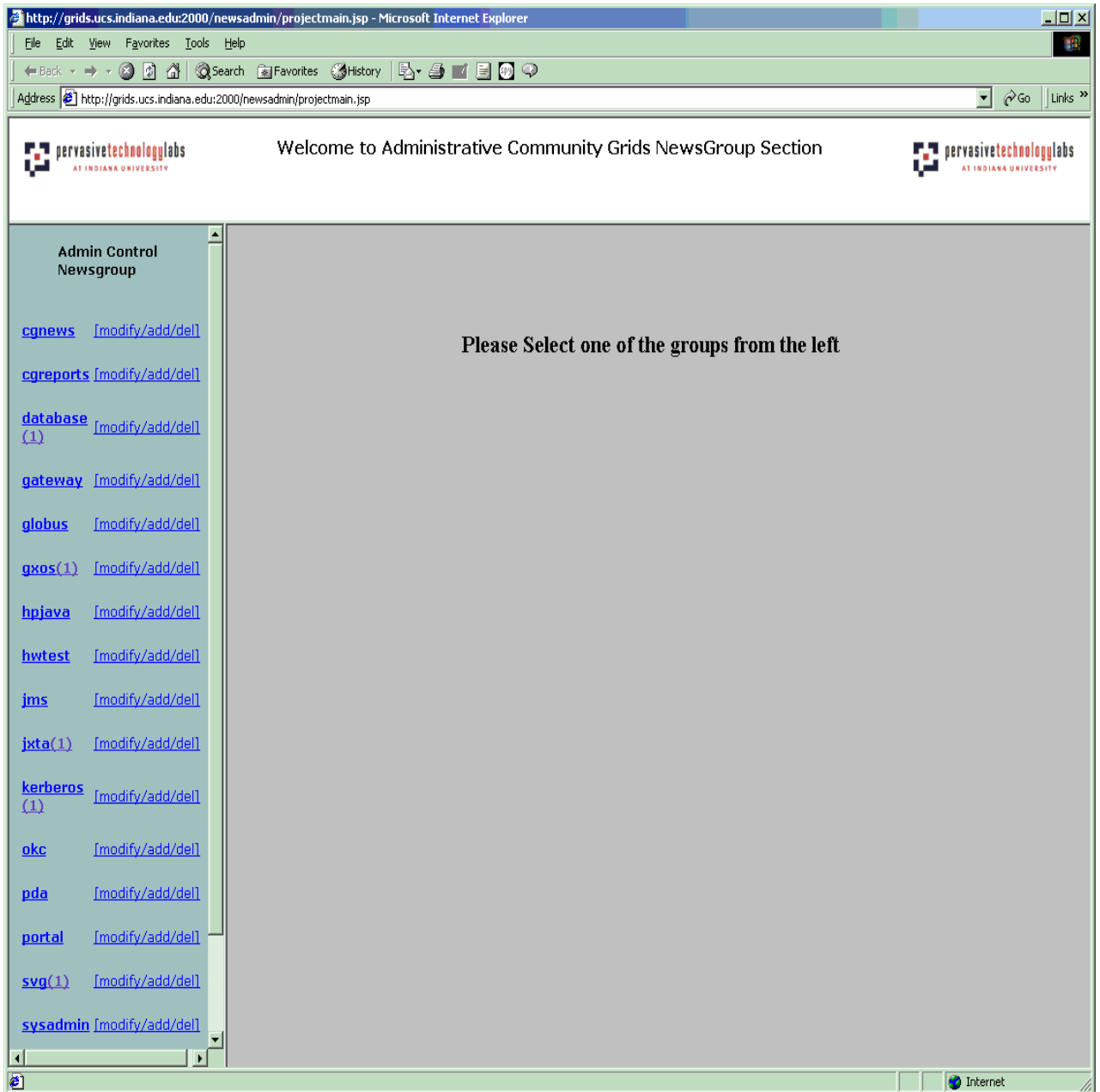


Figure 40 The entry page for administrating newsgroups.

Each newsgroup has an administrator. Administrator can manage more than one group. Newsgroup administrators are themselves managed by a superadministrator, who decides who becomes administrator for

newsgroup. In our current authorization model, the administrator of a newsgroup can also assign administrator privileges for that newsgroup to other users.

When the administrator selects the administrator section in the newsgroup, all the newsgroup which user has administrator rights for newsgroups will be displayed on the list. Administrator can see whether there is any request for particular newsgroup on the list. If the user clicks the link of the newsgroup, which has pending request, they can see the current rights and pending request for newsgroup. The following page has shown in the figure below.

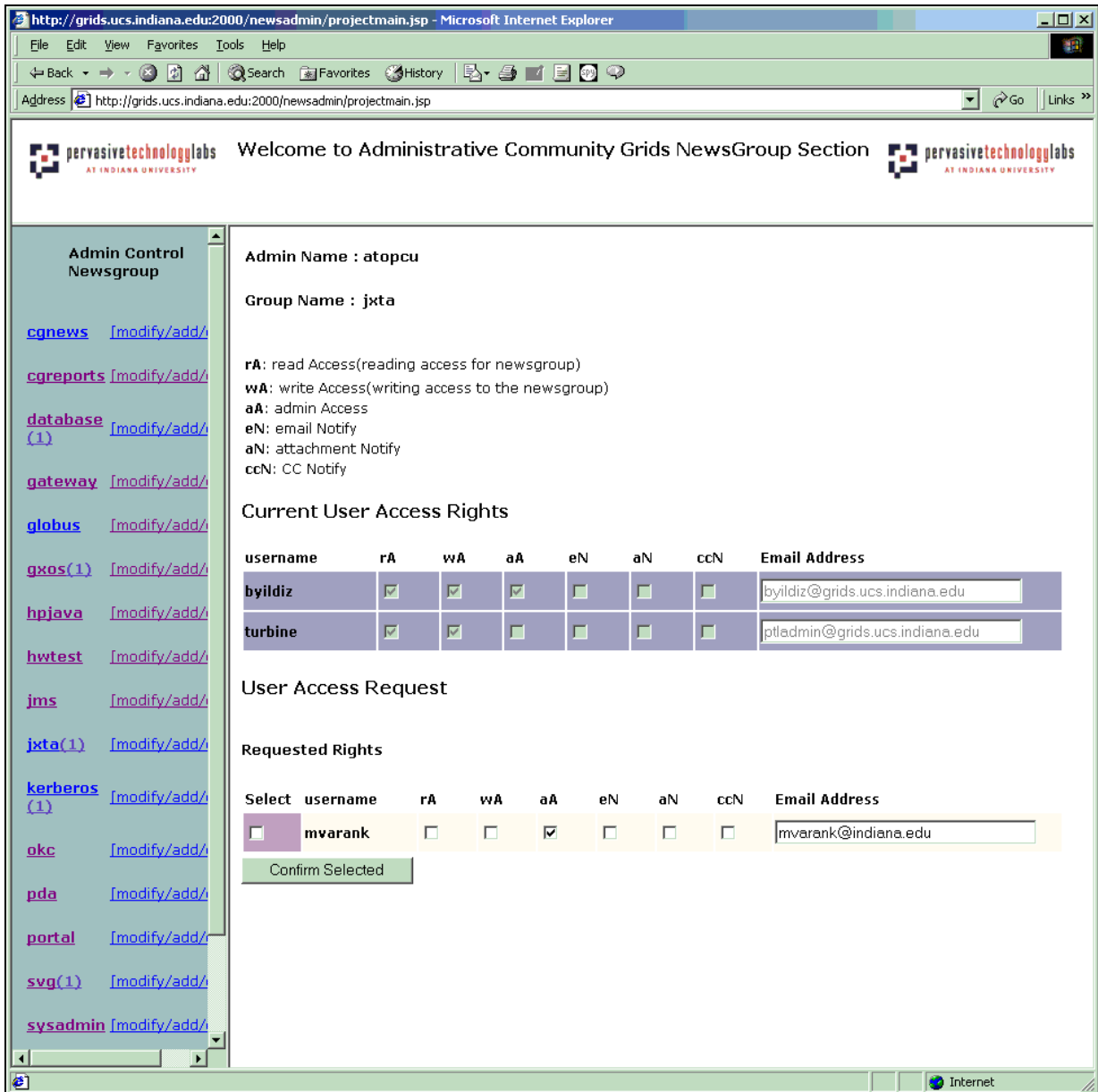


Figure 41 A listing for the JXTA newsgroup shows registered users and new users requesting access.

User clicks the jxta newsgroup and admin can see that which users use the jxta newsgroup, and pending request for the newsgroup. User can confirm the newsgroup request. They can also change the requested rights.

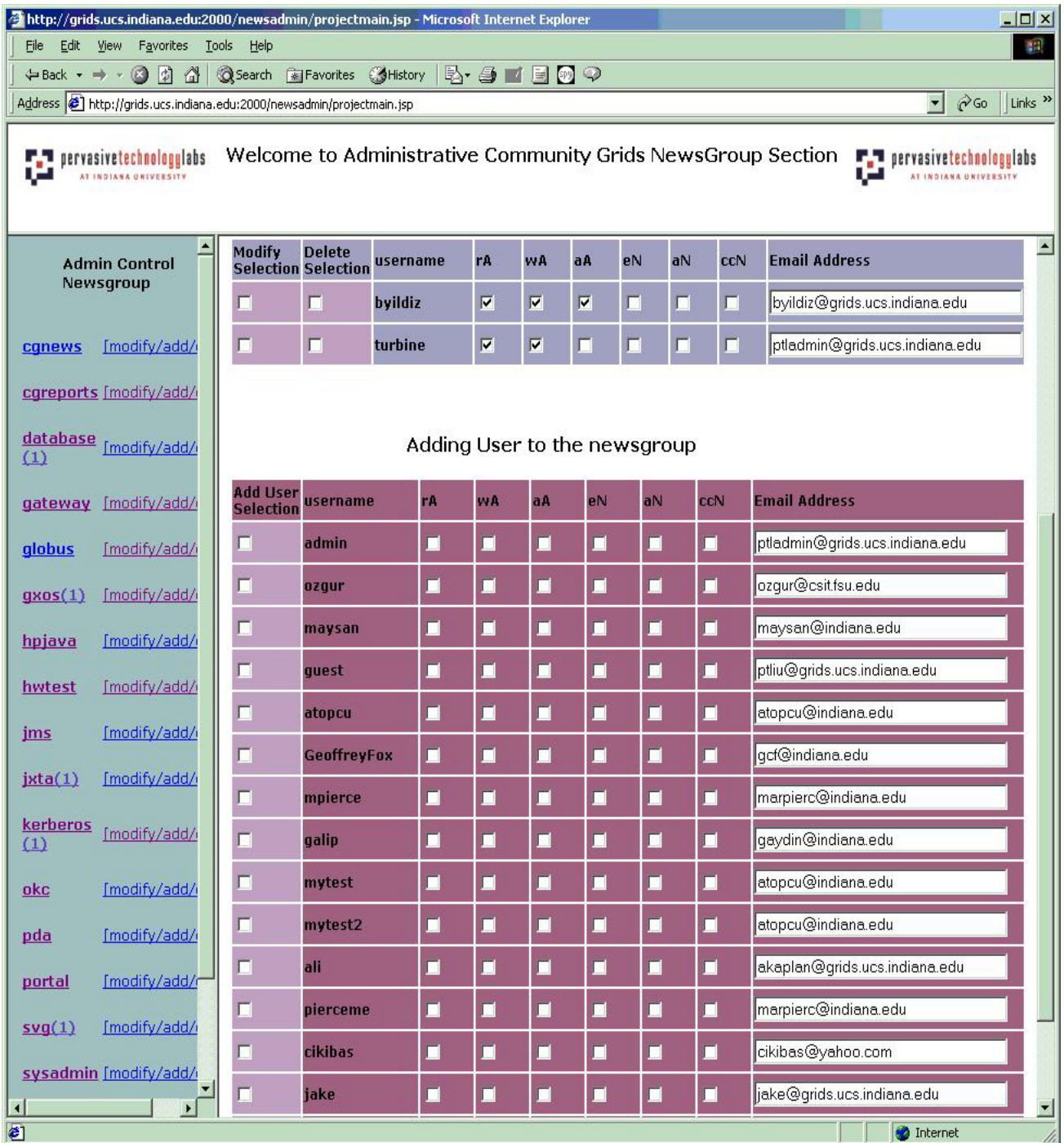


Figure 42 Administrators can add any registered user to any newsgroup.

If the administrator clicks on the “modify/add/del” link for particular newsgroup, he or she can modify or delete user rights. Also they have an option to add user to newsgroup without user request. This mechanism allows us to add many users to the newsgroup easily.

8.3.3. Newsgroup

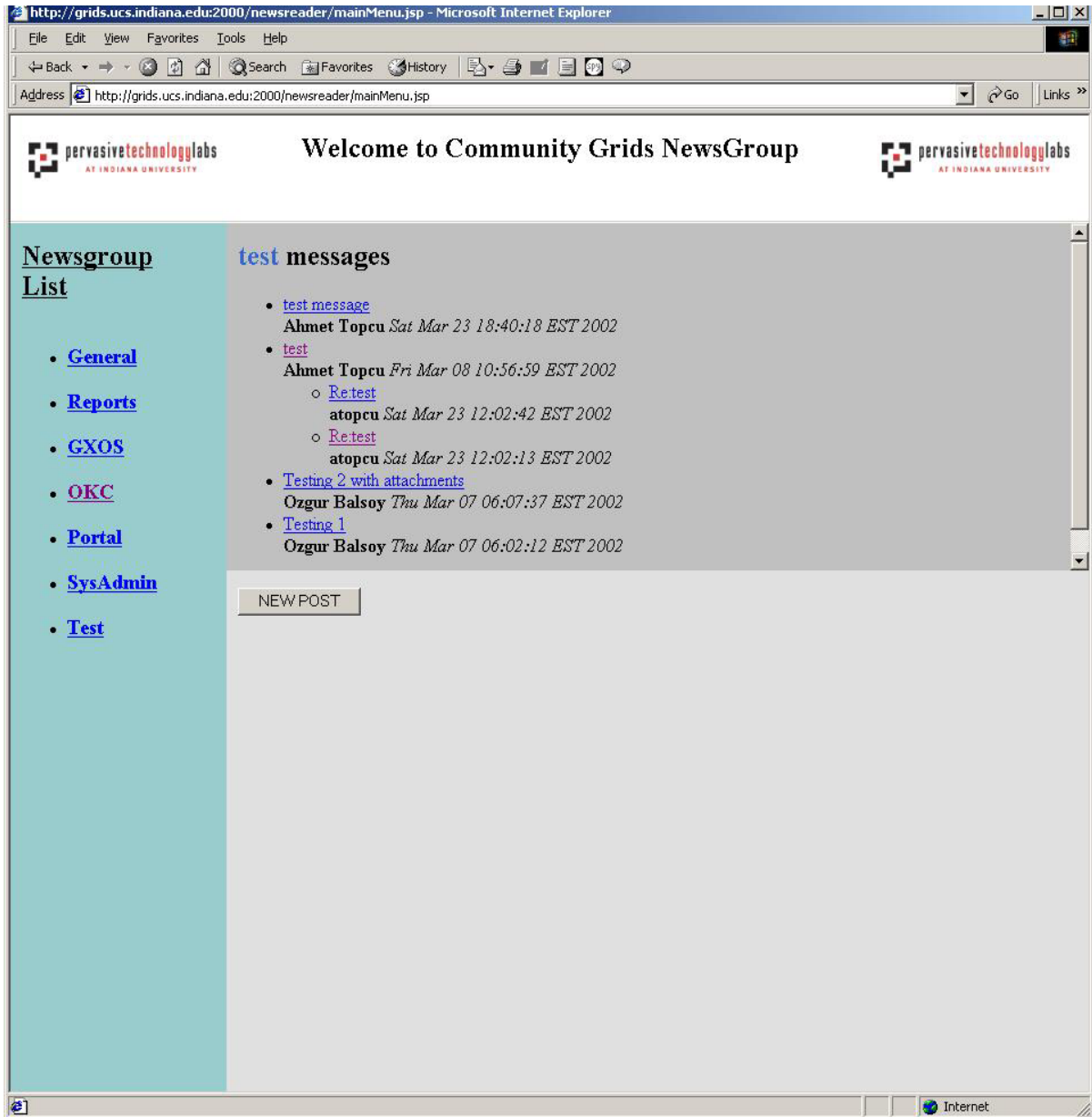


Figure 43 A screen shot of the newsgroup reader.

When the user want to send or read the messages, the newsgroup main page will be displayed on the screen. Newsgroups are listed according to user's access of the confirmed newsgroup by administrator on the left of the main page. When the user select one of the group, all the messages will be displayed by topic thread. If the user has post (write) message access for newsgroup, new post event will be active. If user only has read access, they don't have post message rights to the newsgroup.

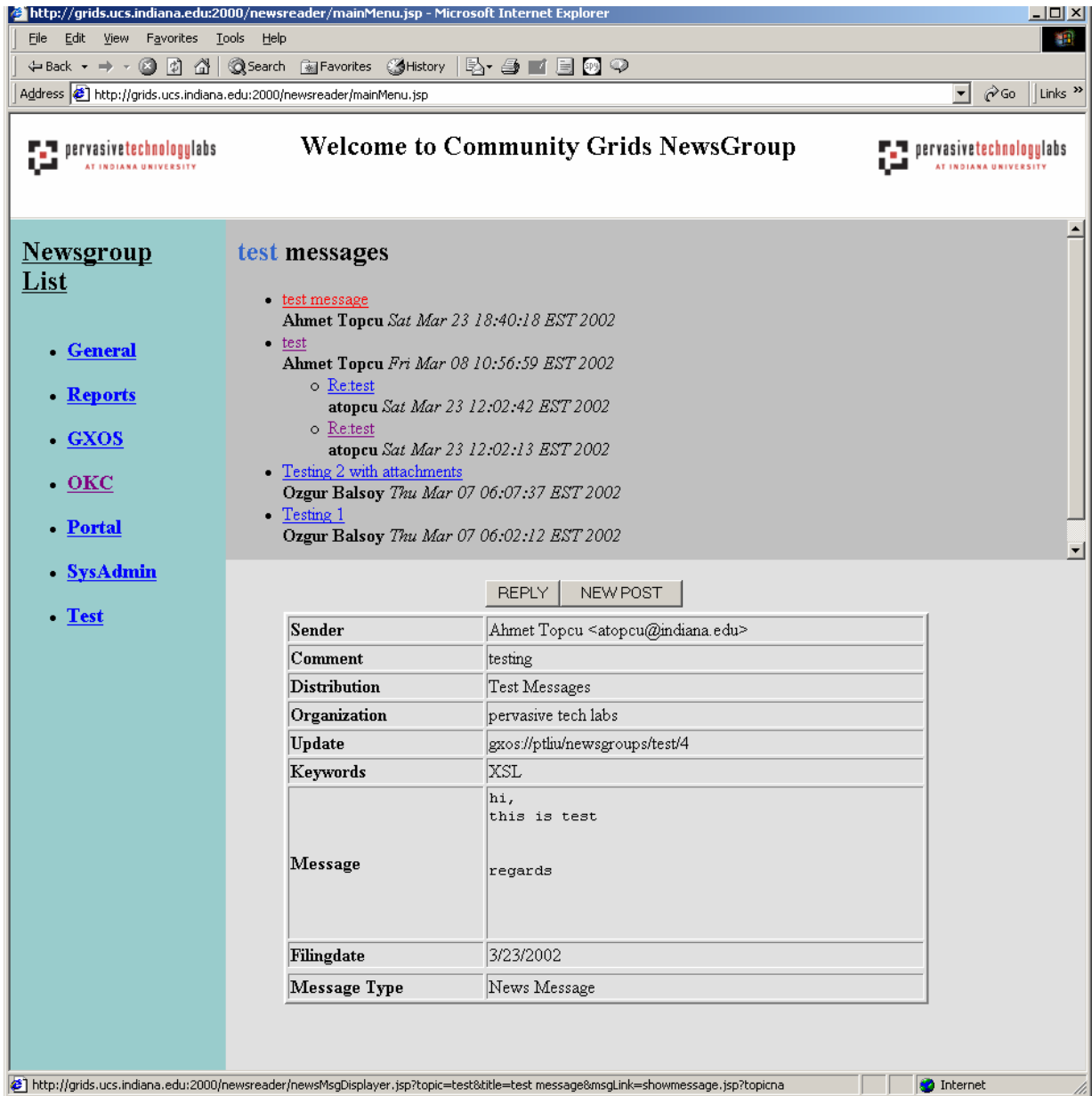


Figure 44 Same as above, but after a message has been selected.

The figure above shows the message content when the user clicks the message link on the upper part. The user can see the message content and other kind of information on the page. This message content is transformed by XSL to display. This structure allows us to change the shown message element easily. By using the reply button, the user can reply to any messages in the newsgroup. When they want to post new messages, they click the new post button.

8.4. Conclusion

One can apply reporting and/or workflow functions to the events customized to each topic. The News Group events have a straightforward display report. Note that the use of a Schema to support News Group allows "key words" and other value added features compared to plain text (as opposed to XML enhanced) email versions OKC Update request events will be passed to OKC administrator for approval. We understand there are alternative and simpler ways of doing News Groups, but we believe systematic use of the XML-based approach will be generally applicable to a number of related page creation features.

We define roles for newsgroup. Admin role, user role, and top admin role are defined in the system. This allows us to define other roles when we need to define and use. This structure allows dynamic structure in the newsgroup. This structure allows us to have secure model for newsgroup.

We will have email notification options for newsgroup, when user's rights are confirmed or modified by the user. User also can get the newsgroup message, when they want to get the copy of the message. But, user should have read access for this particular newsgroup.

We will have message chunk navigation system. It has of some size like 40 (user can choose). One displays Sender Time Subject and/or some selectable XPATH value (latter would be obvious if XPATH value sorted on).

This module consists of one java class and two jsp files. DatabaseReader connects to the database and gets the requested data and sends them to the jsp files. The required information to connect the database is taken from the properties file. This module is invoked by a request. The request is received by the jsp files.

9. Search Capabilities

9.1. Overview

As described in previous sections, the OKC is designed to use a centralized, customizable user interface server that manages distributed content from other web servers. Searching the wide range of material that will

be stored in the OKC is an example of an unstructured search. Fortunately for us, a number of technologies already address the problem of unstructured searches over both binary and ASCII documents. Both Google and Oracle provide search capabilities over all common document formats, including plain text, postscript, PDF, and Microsoft Word. The actual choice of search technologies depends on deployment choices. Google provides a powerful, free search feature for local web sites. Thus it is ideal for searching the content servers *if they are directly publicly accessible*. If content servers are not public, Oracle provides a similarly powerful document search over private web material, although this would potentially require that Oracle be installed on multiple content servers.

The OKC also will contain structured documents in XML formats. A number of XML-based searchable databases exist, and we highlight Oracle's capabilities for this. In the process of our survey, we learned however, that no XML-based search engines can search linked external documents. So for example, no current product that we are aware of can search over both newsgroup postings (structured) and their attachments (unstructured). We are thus developing a prototype system for doing this.

9.2. Public Web Search with Google

We need a powerful search engine for public content of the OKC Web Portal. This search facility should provide search mechanism for both OKC Web-portal (public side, of course) and other web pages through OKC. There are many commercial and open-source search solutions which provide this service. We conducted an intensive survey on this. We believe that Google's University Search (see Refs [GOOGLE1, GOOGLE2]) provides the most efficient search option for our needs with no cost. This extensive service is provided only to educational institutions. Any accredited university or educational institution, or any organization at an accredited educational institution is eligible for Google's university search option. As long as OKC Content servers are located under university related domains, we will be able to use this service which we will explain in detail. We will explain the steps to setup Google into a web portal. We will also discuss the possible search customization by using Google Search for different content servers.

9.2.1. Google Search

Google provides an extensive search with no cost. It runs on a unique combination of advanced hardware and software. The search speed depends on the efficiency of our Google search algorithm and the thousands of low cost PC's that they networked together to create a fast search engine.

The heart of Google software is PageRank which is a system for ranking web pages. PageRank relies on the nature of the web by using its link structure as an indicator of an individual page's value. Basically, Google interprets a link from page A to page B as a vote, by page A, for page B. However, Google looks at more than

the total volume of votes, or links a page receives; it also analyzes the page that casts the vote. High-quality, important, sites receive a higher PageRank, which Google remembers each time it conducts a search. Important pages mean only if they do match your query. So, Google combines PageRank with sophisticated text-matching techniques to find pages that are both important and relevant to your search.

Unlimited queries, deep crawls, high level customization, traffic reports, no advertising are some of the features that Google is providing. Google indexes web-pages via distributed crawlers. Google's index, which is comprised of more than 2 billion URLs, is the first of its kind and represents the most comprehensive collection of the web pages on the Internet. This Index size, provided by Google, is good determination key of the quality results. It has also an effect on the likelihood of a relevant result being returned. In addition to site search facility, Google's Web Search also enables your site visitors to search the Internet.

9.2.2. Google Search Features:

- Google supports searches over 12 different file types in addition to standard web formatted documents in HTML.
- Google supports searches in multiple languages.
- Google offers the user the ability to "View as HTML", allowing users to examine the contents of these file formats even if the corresponding application is not installed. The "View as HTML" option also allows users to avoid viruses which are sometimes carried in certain file formats.
- If you prefer to see a particular set of results without file types (for example, PDF links), you can simply type `-filetype:[extension]` (for example, `-filetype:pdf`) within the search box along with your search term(s).
- Google takes a snapshot of each page examined as it crawls the web and caches these as a back-up in case the original page is unavailable. If you click on the "Cached" link, you will see the web page as it looked when we indexed it.
- GoogleScout technology automatically scouts the web for pages that are related to this result, when you click on the "Similar Pages" link for a search result.
- You can query `link:siteURL` and It will show you all the pages that point to that URL.
- You can restrict your search to a specific site by using the word "site" followed by a colon.
- Google also maintains a site for Webmasters giving them several options for curtailing or turning away search crawlers. Google provides each Customer a Client Name. All search queries sent to the Google by or on behalf of the Customer must contain the Client Name and must originate from the

Site. Web site administrators can add to their pages a simple "robots.txt" file that will turn the crawling bots away.

9.2.3. How to Setup Google Search:

It is fairly easy to register with Google and have them index your domain. First step would be following the "Search solutions" link in Google site. After choosing university search option you will be asked to register your domain. Next step would be filling the required form for registration. After entering the university domain to be searched, you can immediately see how many pages are already indexed in your domain. To use this customized University Search, you must create a search form on your web portal. After you register your domain properly, Google provides the html code which you need to plug into your web-site as a starting point. This code allows users to do both site search and web search. After your customizations have been submitted, it can take up to two hours for them to be pushed out to their servers. We opened an account for our OKC development site. Here is the HTML Code which is provided by Google for web portal domain search.

```
<!-- Search Google -->
<center>
<FORM method=GET action=http://www.google.com/u/okc>
<TABLE bgcolor=#FFFFFF cellspacing=0 border=0><tr valign=middle><td>
<A HREF=http://www.google.com/>
<IMG SRC="http://www.google.com/logos/Logo_40wht.gif" border=0
ALT=Google></A>
</td>
<td>
<INPUT TYPE=text name=q size=31 maxlength=255 value="">
<INPUT type=submit name=sa VALUE="Google Search">
```

9.2.4. Discussion on Google Search for Private Content:

Google does not provide secure content crawling within university search option. This means that with Google option we can't search behind the authentication. That is, we cannot search web sites that are password protected. At this moment we need to also mention about other Google Search options which provide also secure content search in addition to extensive search features. These options require monthly fees. For instance Google ask ~\$2000 for Silver/Gold Search option. We don't think it is feasible to go for expensive secure content search options while we have other options like Oracle Search with no cost. Google university search option is a good solution for searching content servers which are open to public. However, with this option

OKC server itself is not searchable because it is password protected. Google Search can't index any other files rather than welcome/login page.

9.2.5. Possible approaches to use Google Search for OKC Content Servers

We should be able to search efficiently any OKC Content Server with Google Search, if these content servers are kept public. To provide this service we might follow following approaches. Firstly, we might have an individual site search for each content server. Secondly, we might have a site search option which would lead the user search all public content servers.

9.2.6. First approach:

We will need to register each content server domain with Google individually. We will use a separate portlet on Jetspeed for Google Search and we will provide the site search options for different content servers. With this approach user will be able to do specific queries on specific content server. This approach would possibly reduce the amount of search results and save some time for user.



Figure 45 Sample Google site search engine interface.

9.2.7. Second approach

We can also search through all Content Servers in a combined way and gather the results coming from different content servers. In this approach, we will also have to register server domains with Google. Whenever user does a query we can apply this query to different server domains and post the results all together. This approach would provide extensive combined site search. You may see the illustration of this approach below.

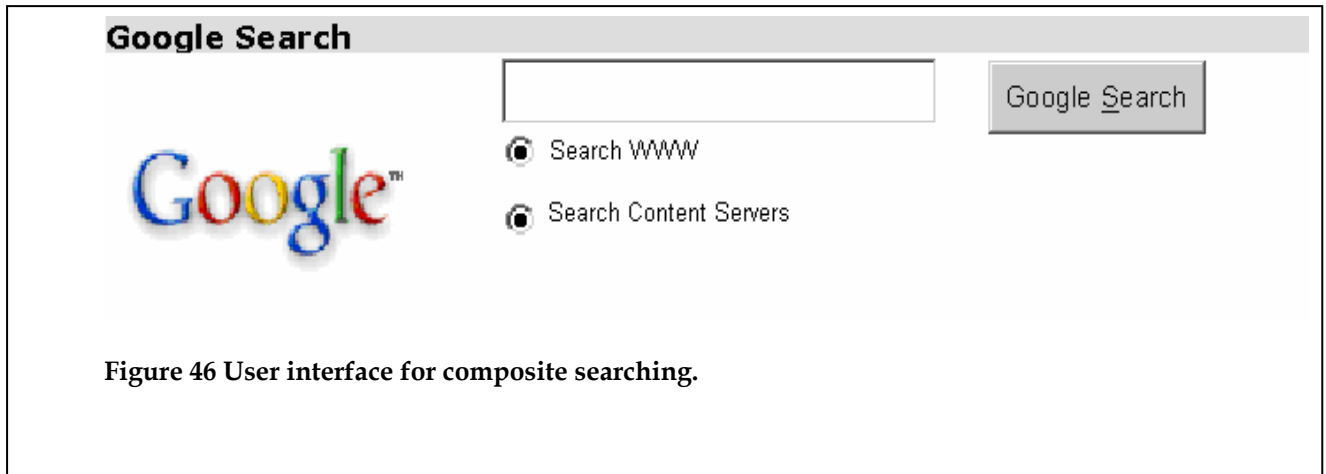


Figure 46 User interface for composite searching.

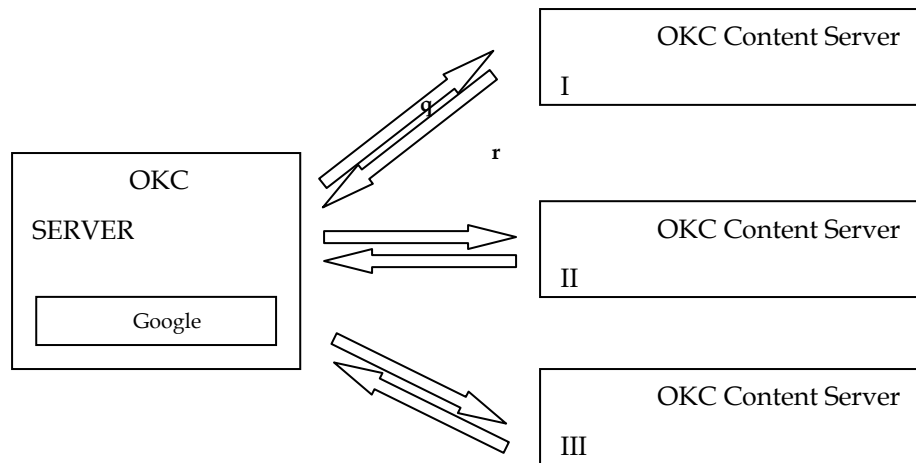


Figure 47 Illustration of combined Google Search

With this approach we intent to provide an intensive search on all content servers. To do this we can modify the provided HTML code and send the queries to all registered content servers. We can also post the results coming from distributed content servers by using JSP technology.

9.2.8. Conclusion

OKC should provide a search facility which has extensive search ability for the content which will be kept public. This search facility should provide both web search and site search options. We believe that Google provides us the most efficient search mechanism among other options on the market. That's why we can use Google in different ways to search public content within the OKC.

9.3. Oracle Search Capabilities

9.3.1. Introduction

The contents of the OKC Web pages and other documents including their meta-data should be retrieved through Web portal inquiries. To perform the search mechanism, we need to collect the contents of the OKC resources to an indexed system and we can extract the user-required information using queries over the indexed system. Without such indexes, we cannot be guaranteed to attain the wanted information within the reasonable time. Since we use XML - the standard exchange form on the Web - for meta-data, the proper technologies are also naturally required to query on the XML resources. Some resources, e.g. the Newsgroup data, include external documents on their XML documents; therefore, another hybrid search mechanism for XML documents and external files should be supplied.

There are a lot of tools for the text search and the extraction of elements and attributes from the XML documents repositories. However, the Oracle database system, a leading commercial DBMS, provides the convenient tools what we require. The XML documents can be stored in the Oracle database tables with XMLType, an object type. In the XMLType data, the elements and attributes can be obtained through the queries with the path expressions. Those queries are also able to be executed in the Java programs embedded on the Web. For the text search, we can leverage Oracle Text, which has a capability of the text indexing and query. Of course, the Application Program Interface (API) for Oracle Text can be used in the Java programs. The hybrid search obtains from a Join operation of XMLType and Oracle Text queries. The Oracle Ultra Search is another useful tool for collecting, indexing, and searching for the Web resources.

9.3.2. Oracle XMLType

The XMLType is a new system-defined object type in the *Oracle 9i* database system. XMLType can be used as columns in tables and views of Oracle database system. XMLType has built-in member functions that offer a powerful mechanism to create, extract and index XML data. Users can also generate XML documents as XMLType instances dynamically using the SQL functions and PL/SQL packages. For example, function *Extract* extracts a specific element or attribute of XML documents stored as XMLType instances. Some frequently accessed elements or attributes are stored in separate columns and indexed for the improvement of the performance.

9.3.3. Oracle Text

The Oracle Text is a tool that makes it possible to build text query applications and document classification applications. Oracle Text provides indexing, word and theme searching, and viewing capabilities for text. There are two types of application with Oracle Text: a text query application and a document

classification application. The purpose of a text query application is to enable users to find text that contains one or more search terms. The text is usually a collection of documents. The application using Oracle Text can index and search common document formats such as HTML, XML, plain text, or Microsoft Word. For example, an application with a browser interface might enable users to query a company website consisting of HTML files, returning those files that match a query. To build a text query application, a CONTEXT or CTXCAT index and query the index with CONTAINS or CATSEARCH can be used. A document classification application is one that classifies an incoming stream of documents based on its content. They are also known as document routing or filtering applications. For example, an online news agency might need to classify its incoming stream of articles as they arrive into categories such as politics, crime, and sports. The CTXRULE index type can be used for building classification applications. However, only plain text, XML, and HTML documents can be applicable to this index type.

To query, the SELECT statement is used. In the case of context index, the CONTAINS operator includes in the query. The context index of Oracle Text can be created with columns of type VARCHAR2, Character Large Object (CLOB), Binary Large Object (BLOB), CHAR, BFILE (Binary File), and XMLType. The supported documents formats can be divided into binary files and text files. The binary files need to be filtering before indexing.

9.3.4. UltraSearch

The Oracle Ultra Search provides search across multiple resources – Oracle databases, Internet Message Access Protocol (IMAP) mail servers, HTML documents on the Web servers, and files on the local host. The Oracle Ultra Search enables a portal search across the content assets of a company, an organization, or an institute. It is an Oracle Text based application, built using the same public interfaces that are available to users of Oracle Text, but enhanced with considerable expertise in aggregating information for indexing, translating queries for the best quality search, and optimizing operations for scalability. The Crawler in the Oracle Ultra Search is a Java process to browse the target resources. The fetched documents are indexed with using Oracle Text. The Crawler is activated by the Oracle server according to a set schedule. When the Crawler encounters embedded, non-HTML documents during the crawling, it uses the Oracle Text filters to automatically detect the document type and to filter and index the documents. A Java API for querying indexed data is also included and the returned value against API can be an HTML fragment. A Java Server Page (JSP) web application is supported for the administrations.

9.4. Hybrid Search Prototype

We made a test prototype of the hybrid search for the XML documents attached with external files. An XML document can have some link tags, which designate some external documents. The external documents

may be Microsoft Word files, Microsoft Power Point files, PDF documents, or Post Script files. The point for such hybrid documents search is that we want to find not only the external documents including some target key words but also the XML data satisfying with some tag predicates. For example, the XML instance in Figure 1 shows a meta-data of a paper and the paper is stored in an external file. When we search the documents which have contents of "XML", we may also want that one of authors of those documents should be "Jungkee Kim."

```
<?xml version="1.0"?>
<paper>
  <title> A Search mechanism of the OKC system</title>
  <authors>
    <author>
      <name> Jungkee Kim </name>
      <affiliation> Florida State University </affiliation>
    </author>
    <author>
      <name> Marlon Pierce </name>
      <affiliation> Grids Communication Laboratory
</affiliation>
    </author>
    <author>
      <name> Mehmet Aktas </name>
```

Figure 48. An example of an XML instance that represents a paper

Actually, we cannot know specific information of the paper document without an XML instance that represents meta-data with a title, authors, affiliations, the source, and the published year. The benefits of the meta-data are not only the particular meta-information but also the performance improvement with narrowing the group to be searched for the context index. However, the large amount of XML documents may drop the performance for extracting particular nodes from XMLType columns in the Oracle database tables. In that case, we can consider ordinary indexed column types mapping to the nodes of the XML instances. In the test

prototype, an XML instance has a single external document to simply the architecture. So, we made an XMLType column type table for the XML instances, a BFILE large object type column table for the document, and a relationship table described the relationship between the XML and the file. The Entity-Relationship diagram for the test prototype is in Figure 49.

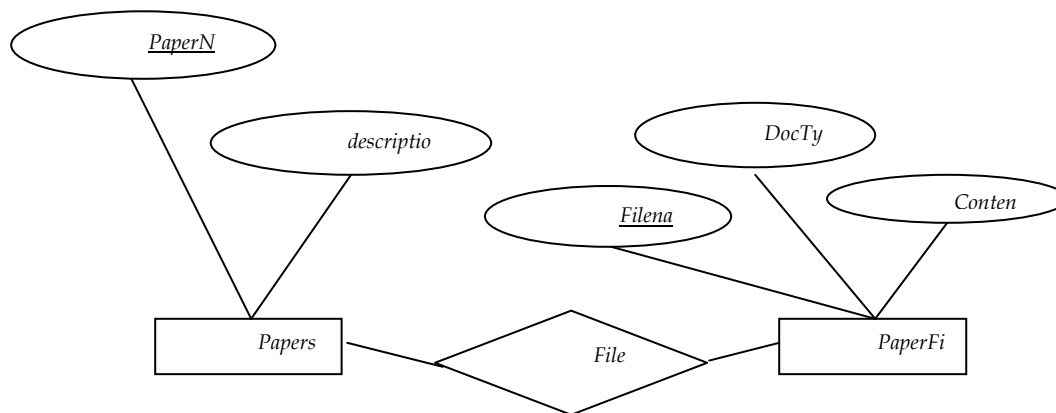


Figure 49. E-R diagram of the test prototype.

The *descriptions* attribute of the *Papers* entity set represents the XML instances with XMLType object type and the *PaperND* (Paper Name and Directory) is a key field with a unique value of Name and Directory in the *Papers* table. The *Contents* attribute of the *PaperFiles* entity set has the BFILE large object type and the actual file will be stored in the designated subdirectory in the outside of the database. The *DocType* column has a filtering option: *BINARY* will be filtered and *TEXT* is not necessary to filter when indexed. The additional update, insert, or delete will change the status of the index and the new information will be reflected into the index using *synchronize* package included in the Oracle 9i database system. The *FileLocator* table shows the relationship between two data tables: *Papers* and *PaperFiles*.

To catch the user inquiry for the particular information and key words for the contents, a Web page is designed as in Figure 3. The inquiry parameters of the Web page will be sent to a middle-tier tool, a Java Servlet. The Java Servlet makes a SQL query from the parameters passing from the user search Web page, gets the result set querying through JDBC connection from the database tables, and spreads them to the user back. The Figure 4 shows the architecture of the test prototype.

Paper Search

Please fill one or more search categories and type in keywords.

You should fill at least one field. (Those fields are from XML instances.) [Sample](#)

Title: (A word)

Author's Name: (A name - first name or last name)

Author's Affiliation: (An affiliation name)

Source: (A conference name or URL)

Year: (A 4 digit number)

You should fill in Keywords. (This field is for the Oracle text.)

Keywords: (A OR B, A AND B)

Figure 50. A User Web Page for the Test Prototype

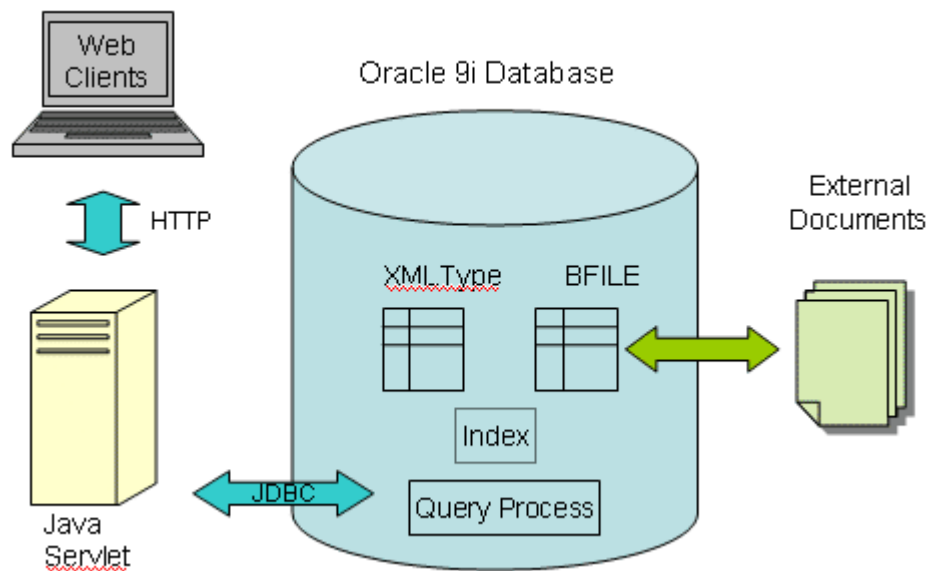


Figure 51. The Architecture of the Test Prototype

10. OKC Development Team

Geoffrey Fox, Indiana University, is the OKC's principal investigator and project leader. Student developers and their projects are as follows:

1. Ozgur Balsoy: OKC/Jetspeed development and testing, overall project development.
2. Mehmet Aktas: search capabilities.
3. Galip Aydin: Newsgroups wizard interfaces.
4. Necati Aysan: Legacy PET web page integration
5. Cevat Ikibas: Slide testing and integration.
6. Ali Kaplan: Newsgroup email subsystem development
7. Jungkee Kim: search capabilities and databases
8. Ahmet Topcu: newsgroup user interfaces
9. Beytullah Yildiz: Newsgroup middleware

Marlon Pierce is the project manager.

11. Selected Reference

- [APACHE] The Apache Jakarta Project home page: <http://jakarta.apache.org>
- [CASTOR] Castor home page: <http://castor.exolab.org>
- [DAV] WebDav Resources: <http://www.webdav.org>
- [DOM] Document Object Model: <http://www.w3c.org/DOM>
- [ECS] Element Construction Set: <http://jakarta.apache.org/ecs/>
- [GOOGLE1] Google search engine: <http://www.google.com/about.html>
- [GOOGLE2] Google university search: <http://services.google.com/googleuniv/login>
- [JAVAMAIL] Javamail API 1.2: <http://java.sun.com/products/javamail>
- [JETSPEED] Jetspeed Overview: <http://jakarta.apache.org/jetspeed/site/index.html>
- [JMS] Java Message Service API 1.0.2: <http://java.sun.com/products/jms>
- [JSP] JavaServer Pages Technology: <http://java.sun.com/products/jsp>
- [JSR] Java Specification Request 168, Portlet Specification: <http://jcp.org/jsr/detail/168.jsp>
- [RSS] RDF Site Summary Specification: <http://groups.yahoo.com/group/rss-dev/files/specification.html>
- [SAX] SAX Project website: <http://www.saxproject.org>
- [SCHEMA] XML Schema: <http://www.w3c.org/XML/Schema>
- [SCORM] Sharable Content Object Reference Model: <http://www.adlnet.org>
- [SLIDE] The Jakarta Slide Project: <http://jakarta.apache.org/slide>
- [SOAP] Simple Object Access Protocol (SOAP) 1.1: <http://www.w3c.org/TR/SOAP>

[TOMCAT] Jakarta Tomcat: <http://jakarta.apache.org/tomcat/index.html>

[TURBINE] Jakarta Turbine: <http://jakarta.apache.org/turbine>

[W3C] World Wide Web Consortium: <http://www.w3c.org>

[XML] Extensible Markup Language (XML): <http://www.w3c.org/XML>