DEVELOPMENT OF JSON AND AJAX ENABLED, DATABASE DRIVEN, USER INTERFACES FOR SCIENCE APPLICATION PORTALS

A Thesis
Presented to the
Faculty of
San Diego State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Hetang Gautam Shah
Summer 2012
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Hetang Gautam Shah:

Development of JSON and AJAX Enabled, Database Driven, User Interfaces for

Science Application Portals

Mary Thomas, Chair
Department of Computer Science

Roger Whitney
Department of Computer Science

Jose Castillo
Department of Mathematics and Statistics

May 24, 2012
Approval Date
Copyright © 2012
by
Hetang Gautam Shah
All Rights Reserved
DEDICATION

To my mom and dad for their unconditional love and support. Everything, I owe to my mother and father.
ABSTRACT OF THE THESIS

Development of JSON and AJAX Enabled, Database Driven, User Interfaces for Science Application Portals
by
Hetang Gautam Shah
Master of Science in Computer Science
San Diego State University, 2012

Science application portals are used to provide familiar World Wide Web (WWW) based access to the general user community. This allows them to run complex science simulations while encapsulating the details of those computing resources that are required. The SDSU Cyberinfrastructure Web Application Framework (CyberWeb) simplifies the utilization of heterogeneous computational environments required by high-performance computing applications. CyberWeb has three core components: (1) an advanced user interface based on the Pylons Web 2.0 WSGI application framework that uses relational databases, XML, JavaScript, AJAX, Google Gadgets, social networks, and security; (2) Administration web pages for configuring CyberWeb installations, applications, users, remote resources and services; and (3) a job distribution WWW service framework (JODIS) for task execution and management.

The objective of this research project is to develop the following Python based components for the CyberWeb project:
1. Develop a CSS, JSON and Ajax enabled architecture for the user interface that will minimize server calls and give better performance to enhance the user experience.
2. Design an advanced administration module, which provides a generalized interface to the database, with access control based on user roles in order to customize the data used by the CyberWeb system.
3. Provide a “plug-n-play” configuration of the remote resources and their services.
4. Integrate these tools into a CyberWeb Demonstration Portal.
5. Contribute to Publications and Presentations.

With these tools, CyberWeb applications can use an administration module to give controlled access to users and that will allow real-time configuration of users, distributed resources and services, and accounts.
**TABLE OF CONTENTS**

<table>
<thead>
<tr>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT ............................................................................................................................... v</td>
</tr>
<tr>
<td>LIST OF FIGURES ................................................................................................................. ix</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS ...................................................................................................... xi</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS ........................................................................................................ xii</td>
</tr>
<tr>
<td>CHAPTER</td>
</tr>
<tr>
<td>1 INTRODUCTION ............................................................................................................... 1</td>
</tr>
<tr>
<td>1.1 Motivation for Advanced Science Portals ................................................................. 1</td>
</tr>
<tr>
<td>1.1.1 What is a Portal? ......................................................................................................... 2</td>
</tr>
<tr>
<td>1.1.2 The Need for Science Portals ................................................................................... 2</td>
</tr>
<tr>
<td>1.1.3 The Importance of an Advanced User Interface .................................................... 3</td>
</tr>
<tr>
<td>1.2 The CyberWeb Solution ............................................................................................. 3</td>
</tr>
<tr>
<td>1.3 My Research Goals ..................................................................................................... 4</td>
</tr>
<tr>
<td>1.3.1 Create an Advanced User Interface Design ......................................................... 4</td>
</tr>
<tr>
<td>1.3.2 Develop Improvements to the Database ................................................................. 4</td>
</tr>
<tr>
<td>1.3.3 Deploy Dynamic Configuration and Use of Resources and Services .................... 5</td>
</tr>
<tr>
<td>1.3.4 Demo Portal ............................................................................................................. 8</td>
</tr>
<tr>
<td>1.3.5 Contribute to Publications and Presentations ....................................................... 8</td>
</tr>
<tr>
<td>2 BACKGROUND .............................................................................................................. 10</td>
</tr>
<tr>
<td>2.1 Science Portals ........................................................................................................... 10</td>
</tr>
<tr>
<td>2.2 Advanced User Interfaces/Web 2.0 ............................................................................. 11</td>
</tr>
<tr>
<td>3 CYBERWEB OVERVIEW .............................................................................................. 13</td>
</tr>
<tr>
<td>3.1 CyberWeb Goals ........................................................................................................ 13</td>
</tr>
<tr>
<td>3.2 The CyberWeb Advantage and Difference ............................................................... 14</td>
</tr>
<tr>
<td>3.3 Architecture Overview ............................................................................................. 15</td>
</tr>
<tr>
<td>3.3.1 Pylons Web Application Framework ................................................................... 16</td>
</tr>
<tr>
<td>3.3.2 Model and Database Services ............................................................................... 17</td>
</tr>
</tbody>
</table>
7 FUTURE WORK ................................................................. 54
REFERENCES ............................................................................. 55
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Account status after configuration</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>Admin flowchart for configuring user account on a resource</td>
<td>6</td>
</tr>
<tr>
<td>1.3</td>
<td>Account status after configuration</td>
<td>7</td>
</tr>
<tr>
<td>1.4</td>
<td>CyberWeb demo portal application example</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td>CyberWeb goals</td>
<td>14</td>
</tr>
<tr>
<td>3.2</td>
<td>CyberWeb architecture</td>
<td>16</td>
</tr>
<tr>
<td>3.3</td>
<td>Pylons architecture</td>
<td>20</td>
</tr>
<tr>
<td>3.4</td>
<td>Shows sample application pylon egg structure vs. CyberWeb egg structure</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>Sample routing code and sample controller code</td>
<td>22</td>
</tr>
<tr>
<td>3.6</td>
<td>Sample mako templates</td>
<td>23</td>
</tr>
<tr>
<td>3.7</td>
<td>AJAX architecture showing request and response flow for web applications</td>
<td>25</td>
</tr>
<tr>
<td>3.8</td>
<td>Object notations in JSON string</td>
<td>27</td>
</tr>
<tr>
<td>3.9</td>
<td>Array notations in JSON string</td>
<td>27</td>
</tr>
<tr>
<td>3.10</td>
<td>Value notations in JSON string</td>
<td>27</td>
</tr>
<tr>
<td>4.1</td>
<td>Class diagram of the admin module</td>
<td>28</td>
</tr>
<tr>
<td>4.2</td>
<td>Implementation diagram of the admin module showing the presentation tier, application tier, and persistent tier</td>
<td>29</td>
</tr>
<tr>
<td>4.3</td>
<td>Account add operation sequence diagram</td>
<td>30</td>
</tr>
<tr>
<td>4.4</td>
<td>CyberWeb database UML diagram</td>
<td>31</td>
</tr>
<tr>
<td>5.1</td>
<td>Architecture of menu rendering</td>
<td>40</td>
</tr>
<tr>
<td>5.2</td>
<td>Showing recursive menu rendering</td>
<td>41</td>
</tr>
<tr>
<td>5.3</td>
<td>User interface for admin user definition</td>
<td>42</td>
</tr>
<tr>
<td>5.4</td>
<td>Grouping of the multiple definitions on a single page using tabs</td>
<td>43</td>
</tr>
<tr>
<td>5.5</td>
<td>User interface for adding new user</td>
<td>44</td>
</tr>
<tr>
<td>5.6</td>
<td>Modifying user record</td>
<td>45</td>
</tr>
<tr>
<td>5.7</td>
<td>List of currently available resources</td>
<td>46</td>
</tr>
</tbody>
</table>
Figure 5.8. Configuring new accounts with the help of administration interface.................47
Figure 5.9. Specifying credential for the account.................................................................49
Figure 5.10. The new accounts is configured and ready for use...........................................50
Figure 5.11. Newly added accounts are available for use by all resources and services........51
Figure 5.12. CyberWeb application example........................................................................52
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>SDSU</td>
<td>San Diego State University</td>
</tr>
<tr>
<td>WSGI</td>
<td>Web Server Gateway Interface</td>
</tr>
<tr>
<td>DDWAP</td>
<td>Dynamic Database With Administrator web Pages</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>CCSM</td>
<td>Community Climate System Model</td>
</tr>
<tr>
<td>OGCE</td>
<td>Open Grid Computing Environment</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GSI</td>
<td>Grid Credential Authentication</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>JS</td>
<td>Java Script</td>
</tr>
<tr>
<td>ORM</td>
<td>Object Relational Model</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Professor Mary Thomas for giving me the opportunity to work on her project and for providing me with constant support and motivation. I would also like to thank Mr. Carny Cheng for initial work on database, models and Jodis that made it easy for my work on CyberWeb. Finally, I would like to thank Dr. Roger Whitney and Dr. Jose Castillo for serving on my thesis committee and my sincere appreciation to CyberWeb team for building a wonderful portal and giving me an opportunity to contribute to the CyberWeb project. (place this section after the text and before the reference list).
CHAPTER 1

INTRODUCTION

Advances in the technologies, resources and services used for parallel, grid, and distributed computing have often presented science researchers with the challenge of migrating or creating a model or application that is compatible with increasingly complex computing systems. In order to generate and manage data, modern science applications require a huge number of computations, massive data and archival resources, and high-speed networks. Often, the model or application is legacy code that is stable, but cannot be ported easily to the high-end systems that are now available. Cyberinfrastructure (CI) integrates hardware and software for computing, data management and information retrieval, and visualization and analysis by using interoperable software/middleware and services that are based on World Wide Web (WWW) and Internet technologies [1]. Because much of today’s research requires computing power beyond that of modern desktop computers, scientists need to push their data on to the remote resources for storage, computation and backup. This is the function of cyberinfrastructure. CI utilizes high performance computing software and advanced Web and Internet technologies.

The SDSU Cyberinfrastructure Web Application Framework (CyberWeb) provides a platform for scientists to develop scientific applications that utilize remote computing resources and services [2]. It is an essential tool for allowing data to be shared with applications under a secure environment. This environment is specified by an administrator who gives access roles and controls on a resource or on services. Hence, scientists can easily run their applications or models on a resource.

1.1 MOTIVATION FOR ADVANCED SCIENCE PORTALS

Scientific web portals are seen as the way to see a progress to improve the use of computing infrastructure and high-performance computing facilities. Different types of web portals exists: Personal portals, News portals, Government web portals, Cultural portals, Corporate web portals, Stock portals, Search portals, Tender’s portals, Hosted web portals,
and Domain-specific portals [3]. But, none of the above portals is providing the infrastructure for high performance computation. Hence, a Science portal resembles the technical infrastructure of compute-job submission systems.

1.1.1 What is a Portal?

In the context of the WWW the term portal is used to describe a number of different types of websites. Simply, a portal collates “a variety of useful information into a single, ‘one-stop’ web-page, helping the user to avoid being overwhelmed by ‘infoglut’ or feeling lost on the web” [4].

In the case of CyberWeb, we think of the science portal as the first place to go for a scientist’s on-line interactions with advanced, high-end applications and services. The science portal gathers information and displays it in a user-centric way to help scientists to visualize and understand their data using graphs, tables or images [4].

1.1.2 The Need for Science Portals

Because scientists often must deal with huge amounts of computations and the resultant data, their computations and data are best managed in the ‘Cloud,’ so data can be accessible at any time and from any place. Note that from the perspective of the CyberWeb project, Cloud computing refers to the delivery of computing and storage capacity as a service to a heterogeneous community. Processing and transfer of these computations and data requires a very long time. The scientists use these applications every day and they need a platform that helps them visualize their data in a more meaningful and logical way. Often these applications are interconnected in a variety of ways, so a platform is needed where they can configure and run their application using cyberinfrastructure.

So, what do these challenges have to do with a portal? Looney and Lyman suggested a need to reconsider the learning environment and the mechanism through which research is conducted. Daniel (1996) has commented that, “re-engineering the learning environment will not occur without the development of a technology infrastructure” [4].

The development of a science portal might be seen as one part of this infrastructure, the part that joins the dots between administration, social organization and research communities.
1.1.3 The Importance of an Advanced User Interface

In the world of the science portal, education portal and other productivity tools, the ability to save time is always one of the most important features. A user does not want to spend hours learning to use a portal, which is supposed to be helping and streamlining his or her research work. Linus Torvalds said, “in many cases, the user interface to a program is the most important part for any product: whether the programs works correctly or not seems to be secondary” [5].

Therefore, to be successful, the software must have a sophisticated but simple user interface which lets one do what needs to be done, intuitively, without even really thinking about it [5].

A good user interface usually consists of the following: [6]

- Positioning the elements in places the user expects them
- Good use of color-coding
- Speed and fast page loading
- Activity indicators
- Help and tool tips
- Proper conveyance of error conditions

1.2 THE CYBERWEB SOLUTION

There are many problems and challenges for modern science applications. The CyberWeb Framework provides a user-friendly interface and a platform for running an application using Cyberinfrastructure and Clouds. For building portals, CyberWeb has various modules and access controls. The CyberWeb framework runs on the top of three core components. These are:

1. Pylons, a Web 2.0 WSGI application framework that uses relational databases, XML, JavaScript, AJAX, Google Gadgets, social networks, and security;
2. A dynamic database, with administrator web pages, for configuring CyberWeb installations, applications, users, remote resources and services;
3. The job distribution WWW service framework (JODIS) for task execution and management.
4. Provides a platform to run a demo task and testing of the CyberWeb functionality.
My task was to create a dynamic database, within the administrator web pages for configuring CyberWeb installations, applications, users, remote resources and services. This is described in detail in chapter 3. Other core functionality is being developed by other team members of the CyberWeb developer group composed of Carny Cheng, Smita More and Professor Mary Thomas.

1.3 MY RESEARCH GOALS

When I began my thesis work, the CyberWeb project was in its early prototype phase. What was needed at that time was a more flexible and scalable administration module (Admin module) to seamlessly integrate with the database. The administrator should be able, at run-time, to configure resources and services for running applications. Also the Admin module needs to provide a user interface for the database. This interface needs to be easy to use for both the administrator and the typical user. This is the main objective for building an Admin module and my goal was to provide it with a rich and generic user interface. This requirement led to the following list of tasks that composed the main body of my thesis work.

1.3.1 Create an Advanced User Interface Design

The interface that has been developed – dynamic database with administrator web pages (DDWAP) - follows the principles outlined above and as shown in Figure 1.1. It provides step-by-step information for configuration of the database records. It uses the modern AJAX (see section 4.1.2), JavaScript (section 4.1.4) and JSON (section 4.1.3) computer languages to provide ease of interaction with web pages. This improves the performance of web page loading. The administration interface uses icons and color-coding to indicate different operations. AJAX makes page loading faster because it allows for asynchronous loading of the pages [5].

CyberWeb also has dynamic configuration of menus and their submenus. Menus are configured in the JSON file. This file is read at the start of the application and CyberWeb creates the menu object, which is used to render the menu at run-time.

1.3.2 Develop Improvements to the Database

The Admin module offers many key features to CyberWeb. It provides a good user interface for the database. The user interface provides for addition to, updating of, and group
deletion of records. This improves the database operation and optimization based on the mandatory values. The Admin module has validation logic to check for duplicate records and mandatory values and performs type checking.

1.3.3 Deploy Dynamic Configuration and Use of Resources and Services

The Admin module has the capability to configure resources and use them at runtime. This is important because a scientist may have a new server or cloud to run his application but the system might not be enabled to make this resource available until the application server is restarted. This is a common problem in science portals, where a typical user needs to wait until the administrator confirms changes or restarts the application server. A unique feature in CyberWeb is that a scientist with the right credential information can configure the server and it is available for his or her immediate use. Figures 1.2 [7] and 1.3 illustrate the dynamic configuration of resources.
Figure 1.2. Admin flowchart for configuring user account on a resource. 
Figure 1.3. Account status after configuration. The highlighted box in the column on the right shows the current state of the account.
Figure 1.2 shows an example of the workflow used to configure and activate accounts for CyberWeb users (CWusers) on a remote compute resource. If the resource exists and is active (configured), the CWuser enters resource data into a form, and selects a preferred authentication mechanism. CyberWeb securely manages accounts (sensitive data is encrypted and stored outside WWW space, and no passwords are stored), maps CyberWeb user accounts to the users accounts on remote resources, and tracks preferred authentication schemes (e.g. PKI/SSH, GSISSH) [7].

1.3.4 Demo Portal

The demo portal has several features typical of science portal including: user, account management; job management; resource configuration; access control management. My task is to build a demo portal, which demonstrates this usability and features to the users. This demo portal is seen in Figure 1.4.

1.3.5 Contribute to Publications and Presentations

Recently published paper has been accepted for publication as part of the International Conference on Parallel and Distributed Processing Techniques and Applications and on which I am a co-author [7]. Other than publish paper, I contributed in poster that has been presented at the 8th Annual Applied Computational Science and Engineering Student Support (ACSESS) meeting.
SSH Cmd

Cyberweb Execution Services.

- Test pre-built demo jobs in this section.
- Authentication Required for some jobs

Figure 1.4. CyberWeb demo portal application example. Image created by author.
CHAPTER 2

BACKGROUND

The domains of science portals and research concentration govern CyberWeb. In this section we will present a short background on the science portals and the hypothesis that helped drive the development of CyberWeb. We will then provide a historical reference for the evolution of the advanced user interfaces that were developed and led up to the CyberWeb framework. After a brief overview of the CyberWeb framework, we will discuss one other science portal framework that had a great influence on CyberWeb.

Many projects exist in the computer world that are similar to CyberWeb. They are designed to facilitate the development of the services needed to execute programs on the WWW and cyberinfrastructure. NEWTS, NCLab, Teragrid are projects closely related to CyberWeb. They also provide a rich GUI on top of the application programming. But, CyberWeb addresses various issues with those projects. The major issue of those applications is a highly complicated system. All these systems do not guide users through simple steps; they simply open the tutorial area of the system. The user needs to spend hours of time learning the system and its interactions. This issue motivated the development of the CyberWeb DDAWP to do simple tasks and to require minimal effort in learning the system.

2.1 SCIENCE PORTALS

Portal generally synonymous with the term gateway is a major starting site for users visits an interconnected or internal site for an application or service. There are general portals and specialized or niche portals. A number of large access providers offer portals to the WWW for their own users. Most portals have adopted the style of content categories with a text-intensive, faster loading page those visitors will find easy to use which will encourage them to return. A science portal provides the heterogeneous environments [8] required by high-performance computation applications [4].
These efforts include projects such as the Java based Open Grid Computing Environments (OGCE) project [9] the Perl based GridPort Toolkit [10], and the Python/Django-based Clarens extensible toolkit [11].

There are many portals and gateways that are closely related to CyberWeb portal. These portals are designed for specific task and goals. These portals are:

- NERSC Web Toolkit (NEWT) [12]
- Networked Computing Lab (NCLab) [13]
- Extreme Science and Engineering Discovery Environment (XSEDE) TeraGrid gateways [14]
  - Asteroseismic Modeling Portal
  - Community Climate System Model (CCSM) TeraGrid Gateway
  - XSEDE user portal …

Section 3.2 explains the advantage of CyberWeb over these portals as well as it explains the difference between them. CyberWeb is an ongoing project that is part of the Open Grid Computing Environment (OGCE) project. The OGCE is developing a Java based framework for providing different computation possibilities for the scientist. Similarly, CyberWeb uses Python to proved generic framework for similar kinds of functionality. CyberWeb provides a “plug-n-play” approach for remote resource configuration and its services. This allows scientists to run jobs on the remote resource - a capability that is not present in any other portal applications.

This makes CyberWeb unique in the realm of scientific portals in that it configures resources and makes them available quickly. CyberWeb provides common authentication and security mechanisms, global namespace and file systems, remote job submission and monitoring, file transfer services, advanced support services, and a user portal designed to help researchers work as efficiently and effectively as possible. The section below explains the advanced user interface implementation for CyberWeb and how it differs from that of other portals.

### 2.2 Advanced User Interfaces/Web 2.0

Almost all of the rich web application that we currently see on the web; they do not just follow the traditional approach to the retrieval and storage of data. Web 2.0 websites allow users to do more than just retrieve information. By increasing what was already
possible in "Web 1.0", they provide the user with a better user-interface and more software
and storage facilities - all through their browser. Users can provide the data that is on a Web
2.0 sites and then, via Web 2.0, access and exercise some control over that data. In her
article, "Fragmented Future", DiNucci writes (as quoted in Wikipedia):

> The Web we know now, which loads into a browser window in essentially static
screenfuls, is only an embryo of the Web to come. The first glimmerings of Web
2.0 are beginning to appear, and we are just starting to see how that embryo might
develop. The Web will be understood not as screenfuls of text and graphics but as
a transport mechanism, the ether through which interactivity happens. It will
appear on your computer screen, on your TV set, your car dashboard, your cell
phone, hand-held game machines maybe even your microwave oven. [15]

This states the important role the WWW plays in our life. Every day millions of users
use the WWW for many hours and preforming various tasks. This generates a great need for
web developers to makes a user’s life easy by providing a meaningful interactive experience.
The Web 2.0 provides advanced techniques to improve user interactions with the WWW.
This facilitates the development of web pages built with a rich user interface using advanced
techniques. Section 3 covers the advanced technology used in CyberWeb [6].

We will see how these techniques and technologies are used in CyberWeb and other
portals. Chapter 3.4 covers the core technology used to build advanced user interfaces. This
technology is common web technology used for building web pages but the way it is
optimized in CyberWeb differs from that of other portals. CyberWeb also provides the
necessary interface elements that helps and guides user to use CyberWeb portal, such as:

- Page loading is faster from traditional approach, because of the lazy loading of the
  pages with the help of AJAX.
- It guides the user to do required tasks with the help of notes and tool tips.
- The page alerts the user to errors and provides options for fixing them.
- Operations are indicated on the page with icons instead of text [6].
CHAPTER 3

CYBERWEB OVERVIEW

CyberWeb is a new implementation of a science web application framework. A CyberWeb web application framework is a software framework that is designed to support the development of research applications, dynamic remote resources configuration, and Job management services. It is written in the Python programming language. CyberWeb simplifies the utilization of the heterogeneous computational environment required by high-performance computing applications. A key objective of the CyberWeb project is to be able to produce a “plug-n-play” approach for resource configuration and services that imposes minimal impact on the resources used (no specialized software needed). CyberWeb is also a part of the ongoing National Science Foundation project and Open Grid Computing Environments (OGCE) projects. It is evolved from the GridPort Toolkit Project.

In this section, we will describe the development of the CyberWeb system. We describe the alterations of GridPort Toolkit Project that were carried out to provide the necessary representation in the CyberWeb framework.

3.1 CYBERWEB GOALS

The key goal of CyberWeb project is to provide a bridge between generalized users and high-end resources, emerging technologies, and cyberinfrastructure. It is being developed to simplify the use of these assets by using common/familiar WWW and emerging technologies including python, pylons, AJAX, javascript, jQuery, gnuplot and gnuploy.py library [7]. This means it will provide the science portal developer with both the basic and advanced features needed to helps develop Web enabled applications that will allow scientists and community access portal for expert and non-expert users to better understand the science involved. CyberWeb needs to provide the heterogeneous environment [8] required to run these applications. Figure 3.1 shows the workflow of a CyberWeb Application. Here the term, Client can be a web browser, or remote Command applications, or portal or any JODIS service. Client can ask the CyberWeb about the job status or can
submit jobs to remote resource. Client can be an application programmer that can develop application and configure with the administration interface to run inside CyberWeb.

### 3.2 The CyberWeb Advantage and Difference

CyberWeb is a collection of REST based services that allow scientists, programmers, staff, and the public to write web applications for High - Performance Computing (HPC). REST is important to Web applications frameworks. It provides various key goals to the system by following REST architecture. Those goals are:

- Scalability of component interactions
- Generality of interfaces
- Independent deployment of components
- Intermediary components to reduce latency, enforce security and encapsulate legacy systems

Because of these goals CyberWeb is following the REST protocol. There are many other frameworks available on the market, which are similar to CyberWeb. So what makes CyberWeb different and unique? CyberWeb is unique because it is designed to be used by a research project, rather than a commercial product. In a sense, CyberWeb is designed to help
scientists in their research and make launching their applications easy. Other products (mentioned in section 2.1.1) in the market are commercial products and were built to support a business and make profit. They were not designed for researchers in a specific area of concentration. Another main difference in CyberWeb is the “plug-n-play” approach for acquiring a resource and using it. Few, if any, frameworks provide instant use of resources after its configuration. This makes CyberWeb unique and, we think, better than other frameworks for carrying out scientific research. Being able to dynamically add and configure resources and services makes it easy for the project administrator to manage them. Other features also make CyberWeb different. CyberWeb avoids the following challenges in a science portal by using a rich user interface and “plug-n-play” approach:

- Old way = SSH + command line + batch system
- People now expect web interfaces for everything
- Usability - scientific computing should be as easy and secure as online-banking
  - Avoid generic options/tools not applicable to your discipline
  - Avoid dealing with backend, middleware, UNIX CLI etc.

### 3.3 Architecture Overview

In this section we cover the basic architecture of the CyberWeb Framework. Relevant descriptions of the components can be found in Chapter 4 (database admin module), Chapter 5 (the user interface), and in a recently published paper [7].

Figure 3.2 [7] shows the 3-tire architecture of the CyberWeb. This architecture is found in many portal applications [7]. The architecture represents the human interaction with the middleware hosting which is done by the Pylons Web Server. The front-end client can be a web browser of either desktop or mobile device, applications or any of the JODIS services. The last layer (back-end layer) includes the local services of remote resources, web services, other applications, and remote grid, computing and data services. The middleware layer is where most of the main tasks of CyberWeb are executed. The Pylons Web server has the capability to configure remote resources, internal services, JODIS services and routing of user requests to concerned applications. Those remote resources are a HPCI Clusters or different workstations or Cloud servers. Pylon Web Application Framework uses Web 2.0 services at its core [7].
3.3.1 Pylons Web Application Framework

Pylons is an open source web application framework written in Python. It uses Web Server Gateway Interface (WSGI) standard to promote reusability and to separate functionality into distinct modules. It is strongly influenced by Ruby on Rails: two of its main components, Routes and Web Helpers, are Python reimplementation of Rails features [16]. Chapter 4 of this thesis explains the Pylons architecture in more detail.

Overall, Pylons presents a highly adaptable alternative to more "all in one" Python web frameworks like Django or Turbo Gears. At its core, Pylons intends to be simple glue that can be applied to various existing or future application components that would otherwise find it difficult to interact [16]. The best use cases for Pylons include:

1. Porting existing applications to a new language or framework during a product development shift, server architecture upgrade, or new hires.
2. Fleshing out a more lightweight prototype that was used to test a proof-of-concept but lacks the polish of a background framework to piece some of the meatier future components together
3. Highly complicated model and database architectures that require low-level access to the database, which can be facilitated by powerful ORM frameworks, like SQLAlchemy or can even be custom built and plugged directly into Pylons [16].

3.3.2 Model and Database Services

The database service is the core of CyberWeb. The entire configuration, user session and user information is stored inside the database. The CyberWeb administration module provides the interface for the database. Hence, CyberWeb is built using a “plug-n-play” approach i.e. once the configuration or changes are committed to the framework then it is readily available to use by the components. This configuration is achieved after the initial setup of the database. The database also provides the information after its initial setup, such information’s are: active resources (available resources), operation status of the service, user information, session information, and user preferences [7].

The other major component of the database is the schema design. Schema design affects database usability and performance in many areas, so it is very important to make initial investment in time and research to design a database that is efficient and meets the needs of the CyberWeb-hosted applications. Consequently, the schema design was based on an evaluation of schema used by multiple projects [7] (TeraGrid [17]; Open Grid Forum [18]; and W3C/IETF standards). The requirements of the schema include:

- Needs to be simple and lightweight; especially for non-experts.
- Easily modified and extended
- Must support the information required to create cyberinfrastructure-based applications.

The schema design needs to flexible, so that the application developer can build his or her own application by extending the CyberWeb framework [7]. The database design has four key components: SQLAlchemy [19], SQLite, Object Relational Mapping (ORM) with the help of pylon modules, and loading of the initial data with the help of flat file such as JSON. SQLAlchemy is a Pylons component that is a wrapper to a large number of standard databases, and one of reasons that Pylons was chosen for the foundation software for CyberWeb.
3.3.3 Database Administration Interface

CyberWeb has very rich and generic interface for database. Normally, the database for an application is managed using command line tools of the SQLite (or MySQL) by a system administrator. However, these tools are complicated and hard to learn for non-expert users. Hence, there arises the need for a user-friendly interface for the database. This interface helps application developer to add, delete, modify, configure and test: resources, the services that will run on them, and the users or groups to control access. The other major important feature, the “plug-n-play” mode, is achieved using the admin interface. This interface supports dynamic configuration of remote resources and services. This interface also guides the user in performing operations like add, edit or delete on any entity. The module is developed using JQuery, AJAX, and Pylons modules [7]. The database administration interface is a key aspect of the research addressed in this report and is explained in more detail in Chapter 5 of this document.

3.3.4 Job Distribution Service (JODIS)

CyberWeb offers another service called Job Distribution Service (JODIS). This is a heart of the CyberWeb, where all jobs are created and Submitted to concerned resource. The JODIS project is the MS Thesis work of another graduate student, Carny Cheng. A brief overview of JODIS is given below, and more details about JODIS can be found in CSRC report [20] and PDPTA publication [7].

The need of JODIS arises to perform distributed job across several SDSU campus compute clusters that are running on a different resources and each are controller by different system administrator [7]. The main responsibility of the JODIS is to distribute application work workloads across the different heterogeneous computing systems. Based on the research paper [7], JODIS is designed based on the master/worker design pattern to schedule tasks or jobs. This pattern gives performance gains by utilizing parallelization. CyberWeb is tightly coupled with the JODIS framework and JODIS acts as the heart of the system [21].

JODIS provides various application-programming interface (API) to integrate application into CyberWeb. Those APIs provides much needed functionality to the CyberWeb like:
1. Job Management: provides information about jobs, including what kind of job is running, the number of tasks, when it was run and the completion time. This provides the information about the application is running state.

2. Job Queue API: provides an interface to queuing system for Job Submission and its execution. JODIS Queue System API abstracts the different syntax and command-line parameter of the each application and provides the generic queuing application regardless of different environment.

3. File Transfer API: provides secure copy between one resource and another, regardless of the protocol being used for the connection. Using this API, CyberWeb applications can transfer data on the fly as needed to achieve visualization, archiving or sharing with other developer.

4. JODIS API: provides the ability to execute raw commands on a remote resource to mimic the command interface for the user. Many users like to work with command rather than clicking on a user interface. This API provides such functionality to CyberWeb. This gives the developer the ultimate flexibility when developing applications using CyberWeb and JODIS.

### 3.4 CORE TECHNOLOGIES OF CYBERWEB

The CyberWeb project uses various advanced user interface technologies that are described in the following sections. The Pylon web framework uses Model, View, and Controller (MVC). Model holds the business data and rules, and the Controllers provide the link between the model and the user. The View is the visual element such as web page, XML data, or the other dynamic data.

#### 3.4.1 Pylons Web Framework

Pylons, as a general Python web framework is similar to other frameworks, including Django and TurboGears. That being said, however, Pylons has some very interesting core "do-it-yourself” components and philosophies that can make it quite appealing to developers looking to integrate a framework around existing applications or even for porting applications built in one framework / language to the other [22].

Figure 3.3 [21] shows the basic architecture of the Pylons Web Framework. In this framework, a client selects a new task or service or initiates a request. This request is an HTTP request for the Pylons Web Framework. The request is captured by the Pylons router script and mapped to a corresponding action (controller) script. The controller script selects the correct module, executes the block of code (this may include local or external calls to other applications) and makes a connection to the grid to run it. At the end of the block, the
controller calls a script to render the HTML (template) based on data passed to the template script. The output (JSON, XML or HTML) is returned back along the script chain to the router, which serves the HTML data to the right URL.

3.4.1.1 FRAMEWORK STRUCTURE

All Pylons applications are encapsulated in easily distributable Pylons project packages called eggs. Each egg includes the breakdown shown in Figure 3.4. Pylon egg can be easily distributed to others by registering with the third-party Python CheeseShop [22].

3.4.1.2 PASTE AND WSGI

The primary difference between Pylons and other frameworks is its dependence on and internal structuring based on the popular WSGI (Web Server Gateway Interface) middleware. There are numerous applications that WSGI includes as Django, CherryPy, Google App Engine, Trac, Pyramid [23], but most of these are Content Management Systems.
Figure 3.4. Shows sample application pylon egg structure vs. CyberWeb egg structure.

(CMS). Pylons was chosen because it is not a CMS and it is a simpler framework. Paste is built on the emerging Python WSGI server interface and, ideally, all Pylons applications are able to interact with each other with minimal need for documentation between two development teams [22, 24].

Using Paste, Pylons can interact with URL requests in more detailed and powerful ways because actual Python code is facilitating the server-client interaction. This deep integration with the emerging WSGI gives Pylons the ability to process more detailed HTTP requests from the client's application, allowing developers to pass higher level context information within the dispatched URL, such as callback functions, extra information, arbitrary Python objects and other developer-defined elements that get evaluated at the actual HTTP interface level [22, 24].

3.4.1.3 MODELS AND ORM'S

As with any database driven application framework that attempts to abstract the application data from the document presentation layer, Pylons attempts to track all necessary data objects in one pre-defined model component. The central models file allows developers to build the data and objects that will be manipulated throughout the application from one easily accessible location. In Pylons, this model file will vary depending on the particular
object-relational mapping (ORM) that has been chosen (SQLAlchemy, SQLObject, or simply DB-API); however, current versions of Pylons default to SQLAlchemy, which allows for 3 levels of SQL database interaction:

1. The (ORM) allows interaction with the database using chosen object classes specific to the domain of the application as opposed to writing direct SQL code.
2. The SQL expression language allows more experienced database engineers to write code directly modifying the database in its native dialect.
3. Low-level executive methods accept literal SQL strings for modifying specific table or column entries and types [22].

### 3.4.1.4 Controllers

About the only component of Pylons that cannot be substituted for by another user defined library or component is the controller definitions. As expected under typical MVC architecture, specific URL’s call upon controllers to process input data, format results, and render the output to pre-defined templates for display. The controllers are called based upon a pre-set routing method defined in the Pylons file: routing.py [22]. An example of a controller code (right) and its routing code (left) is shown in Figure 3.5.

```python
def make_map():
    """Create, configure and return the routes Mapper"""
    map = Mapper(directory=config['pylons.paths'][controllers],
                 always_scan=config['debug'],
                 map.minimization=False
    map.connect('/:controller/{action}')
    map.connect('/:controller/{action}/{id}')
    return map

import logging
from helloworld.lib.base import *
log = logging.getLogger(__name__)

class HelloController(BaseController):
    def index(self):
        # Return a rendered template
        # return render('/some/template.mako')
        # or, Return a response
        return 'Hello World!
```

**Figure 3.5. Sample routing code and sample controller code.**
3.4.1.5 TEMPLATES AND PRESENTING INFORMATION

Perhaps one of the strongest advantages of Pylons over other Python frameworks is the ability to import a preferred template markup language. Templates allows a web designer or developer to build the visible components of an application that are independent of the application logic and data objects (because of the abstraction of the web framework paradigm) and can use the markup language most comfortable to the designer. Pylons has support for all kinds of templating languages such as Mako, Genshi, Jinja, Kid, Cheetah - all powered by the templating API Buffet. By default, Pylons uses the Mako template language. An example is shown in Figure 3.6. This example shows several features of mako, including: inheritance of style and data; use of embedded python code; use of standard HTML; access to global data objects.

```html
<%inherit file="base.html"/>
<%
  rows = [[v for v in range(0,10)] for row in range(0,10)]
%>
<table>
  % for row in rows:
    $ {makercow(row)}
  % endfor
</table>
<%def name="makercow(row)">
<tr>
  % for name in row:
  <td>$ {name}</td>
  % endfor
</tr>
</%def>
```

Figure 3.6. Sample mako templates.
3.4.1.6 WebHelpers and AJAX

Pylons borrows directly from Ruby on Rails with regard to some of its built in AJAX tools, including directly porting over some of the most common Rails functions such as: start_form, number_to_phone, and link_to_function that has provided some of the more useful AJAX interactions in the most common Rails applications. Pylons WebHelper libraries also build upon the JavaScript libraries Prototype and Scriptaculous [22].

3.4.1.7 Default Library Packages

While Pylons has default library packages for templating (Mako), ORM modeling (SQLAlchemy), and request dispatching (Routes), its creators intentionally built the framework such that it is easy to plug in other libraries, whether they are other popular alternatives or even user-developed or internal packages. All in all, the creators of Pylons intended developers to pick and choose what parts of their framework they would like to use and those that best meet the needs of their user's projects [22].

3.4.1.8 WSGI Interaction

Although it is still a fairly new initiative, the Python Web Sever Gateway Interface project intends to create more dynamic methods for server and client browsers to interact while also creating a standard interface language by which all WSGI Python applications can communicate. While the benefits of more rich client-side input data provides more interactive applications, the WSGI standard hopes to provide a way for all Python web applications to easily communicate and leverage each-other's resources. Ideally, Pylons would allow for powerful future collaboration efforts with other teams, companies, and application domains [22].

3.4.2 AJAX

AJAX is an acronym for Asynchronous JavaScript and XML. AJAX is a group of interrelated web development techniques used on the client-side to create asynchronous web applications. With AJAX, web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page. Data is usually retrieved using the XMLHttpRequest object. Despite the name,
the use of XML is not needed (JSON is often used instead), and the requests do not need to be asynchronous [25].

AJAX is not a single technology, but a group of technologies. The Document Object Module is accessed with JavaScript to dynamically display, and to allow the user to interact with, the information presented. JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads. HTML and CSS can be used in combination to mark up and style information.

Figure 3.7 shows the architecture of AJAX. Here all XMLHttpRequest request is initiated from browser to server is displayed. The server response is getting into JavaScript to render user interface dynamically.

![AJAX Architecture](image)

**Figure 3.7. AJAX architecture showing request and response flow for web applications.**

Before AJAX, most web sites were based on complete HTML pages; each user action required that the page be re-loaded from the server (or a new page loaded). This process is inefficient, as reflected by the user experience: all page content disappears then reappears, etc. Each time a page is re-loaded due to a partial change, all of the content must be re-sent instead of only the changed information. This can place an additional load on the server and use excessive bandwidth [25].
3.4.3 JavaScript

JavaScript is a popular client side scripting language widely supported in most web browsers. The primary use of JavaScript is to write functions that are embedded in or included from HTML pages and interact with the Document Object Module of the page [5]. JavaScript is used in this application for performing client side validation of the forms. Because these validations are done on the client side it can respond quickly, thus making the application look more responsive and dynamic. This also helps in making sure that the data will be accepted before the form is submitted to the server - thereby reducing the number of unwanted page requests between the web browser and the server. HTML by itself does not have any functionality and is used only for display purposes. JavaScript is embedded in HTML pages for adding functionalities like input form validation [26].

3.4.4 JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write and easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999 [15]. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language [27].

JSON is built on two structures:

- A collection of name and value pairs. In various languages, this is realized as an object, record, structure, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all-modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures [27].

In JSON, they take on these forms:

- An object is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by: (colon) and the name/value pairs are separated by, (comma) [15]. See Figure 3.8 [27].
- An array is an ordered collection of values. An array begins with [ (left bracket) and ends with ] (right bracket). Values are separated by, (comma) [15]. See Figure 3.9 [27].

- A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested [15]. See Figure 3.10 [27].
CHAPTER 4

DESIGN OF THE DATABASE ADMINISTRATION MODULE

The design of the administration (Admin) module is based on a very rich user interface. The user interface guides a user in performing operations - such as add, edit or delete - on any entity. The section below shows the basic architecture, implementation and flow diagrams of the administration module.

4.1 ARCHITECTURE OF THE ADMINISTRATION MODULE

The Admin module was developed using JQuery, AJAX, JavaScript, and Pylons. Figure 4.1 shows the class diagram for the Admin module.

![Class Diagram of the Admin Module](image)

Figure 4.1. Class diagram of the admin module.

This class diagram represents the architecture of the Admin module for the server end. The Admin module follows the MVC architecture. Hence, the client is a view in the Admin module and the newAdmin.py file is a controller. A controller accepts a request and...
forwards it to the appropriate module. Here, model classes define the database operations and are responsible for the persistence of the data. Figure 4.2 shows the Admin module implementation graphically.

![Implementation diagram of the admin module showing the presentation tier, application tier, and persistent tier.](image)

The Admin module is divided in three standard tiers for its implementation. Figure 4.2 shows the implementation of the Admin module and its divide the implementation in three tier called Presentation, Application and Persistent tier. This tier is explained in the section below.

### 4.1.1 Presentation Tier

The presentation tier creates a advanced GUI and more usability for users. In the Admin module, JavaScript, CSS and images are used to create the GUI. The presentation layer renders the application data in a way that is meaningful to the users. This tier follows the good user interface standards as mention in section 1.1.3. In the Admin module, all of the requests are AJAX controlled. This results in a much better user experience for page loading and dynamic content display.

### 4.1.2 Application Tier

This tier controls the request forwarding and processing of application data. It accepts a request from the presentation tier. After accepting a request, it generates the output and
returns it to the presentation tier for rendering. The Admin module uses AJAX calls to send a request to the application tier. The corresponding response is always a JSON string of the required data. The application tier handles any error situations and conveys a message to the presentation tier. The presentation tier converts error codes to text that the user can understand.

### 4.1.3 Persistence Tier

The persistence tier is responsible for moving stored data to the database. The persistence tier uses SQLAlchemy modules to store the data in the database. Figure 4.3 shows the sequence of operations used to carry out an add operation.

![Account add operation sequence diagram](image)

**Figure 4.3. Account add operation sequence diagram.**

Referring to the sequence diagram, the user starts the operation by clicking Save, Delete or Update. Here Save is an AJAX request that is invoked by the user and is passed to the application tier. The application tier’s task is to create an account object and pass it to the persistence layer to store in the database and then pass a success result back to the presentation layer. Figure 4.4 shows the UML diagram of the database inside CyberWeb. This expresses the database complexity and flexibility for any application developer.
Figure 4.4. CyberWeb database UML diagram.
4.2 DATABASE DESIGN

The Database design is used to describe the different areas of needed system. The database can also be thought of as the logical design of the data structure needed to store information of data associated with that system. CyberWeb utilizes two database structures: a relational database to contain tables and views; and an object database that represents entities and the relationship. CyberWeb uses object relationship model (ORM) to manage the data stored in the relational database. Pylons uses the python based SQLAlchemy object relationship mapper API to map the objects in the ORM to the data in the relational database. A key feature of SQLAlchemy is that it has API’s to a variety of different relational databases for e.g. MySQL, SQLite. Figure 4.4 shows the relational database structure of the CyberWeb database. For example, the accounts information is stored in Account table; resource information is stored in Resource table; and service information is stored in Service table.

4.3 HOW TO ADD DEFINITIONS IN THE ADMINISTRATION INTERFACE

This section describes the Admin module framework. The user interface design and design pattern used to build administration modules are based on composition pattern, factory pattern and MVC framework. It is very easy to define a new entity in the Admin module framework with a few steps.

There are three basic steps for doing this.

1. Create and write a Mako file.
2. Create and write a database .py file.
3. Create and write a JS file.

These steps are described in more detail below.

4.3.1 Create the Mako File

The Mako file displays the HTML structure of the page. This provides the HTML page layout and server-end logic to be incorporated into pages. This allows pages to be displayed dynamically. From within the Admin module the following steps are used to create a Mako file:
1. Inherit newAdmin.mako file
2. Import the JS file in headtags
   `<%def name="headtags()">`
   `<script type="text/javascript" src="<<Your JS name>>.js"></script>`
   `</%def>`
3. Update the main data. There are few steps to do this:
   a. Define any tabs on the page by:
      `<div id="menu">
       <ul id="menuList">
        <li id="<<Unique Tab1 Id>>" class="selected" onclick="switchTabs(this);">"<<Tab Display Name>></li>
        <li id="<<Unique Tab2 Id>>" onclick="switchTabs(this);">"<<Tab Display Name>>
       </ul>
      </div>`
   Note: - if no tabs are to be added then just add the blank menu div.
   b. After adding Tabs add each Tabs content div.
      `<div id="<<Tab Name>>" class="classTab">
       <h2 class="header"> <<Tab Header>></h2>
       <div id="<<Error Console Id>>" class="errorStyle"></div>
       <div id="<<Loading Activity Pane Id>>">
        <div id="opertaionDiv" class="operation">
         <a href="#" class="addNew menuLink"><img src="/images/icon_add-plus.gif" width="34" height="34"/>Add New</a>&nbsp;<a href="#" class="delete"><img src="/images/delete_icon.png" width="34" height="34"/>Delete</a>
        </div>
        <div id="<<Content Id>>" class="searchContent">
4.3.2 Create the Database .py File

The database.py file helps to manage data transfer to the database, with models. To write a database file just a few steps are needed:

1. Inherit DatabaseOperations as a parent class
2. Implement the following method definition in python class

```python
def add(self, parameters):
    # Need to implement add method"
def update(self, parameters):
    # Need to implement add method"
def delete(self, parameters):
    # "Need to implement add method"
def view(self, parameters):
    # "Need to implement add method"
```

3. This Admin module is a JSON driven module. Hence, it is necessary to return a JSON string to the front-end container from the database file. From the web container, this method gets calls as an AJAX request.

4. One must also modify the new Admin.py file. In this file, make an entry in the dictionary called “database_dict”. One must specify the key and value as a database .py object. For example:

   'account': database.account.AccountOperation()
4.3.3 Create the JavaScript File

JavaScript (JS) is critical to the function of the Admin module. A large part of the Admin module display and data manipulation routines is written in JavaScript and uses the JQuery library. In the Admin module we use JQuery and its plugin to produce a better and more dynamic look and feel. To write a JS file, these steps must be taken:

1. Create an adminData object and initialize it with default values. In the following example, columnList is an array that holds the display name of each column to be displayed. In the data object, dataKeyNames is the array of key/variable name.

   ```javascript
   var protocolAdminObj = new adminData();
   protocolAdminObj.columnList = new Array('protocol Name','Description','is Active');
   protocolAdminObj.dataKeyNames = new Array('name','description','active');
   protocolAdminObj.divName = 'protocolSearchContent'; //Specify the div name to load the content dynamically
   protocolAdminObj.errorConsole = 'errorConsoleProtocol'; //Specify error handling div to display messages, warning or errors from server
   protocolAdminObj.activityPane = 'activity_pane_protocol'; //This gives the loading notification to user
   protocolAdminObj.checkBoxName = 'check_protocol_'; //Specify the checkbox name
   protocolAdminObj.tableName = 'protocolTable'; //Specify the table Name of the grid
   protocolAdminObj.tableRowName = 'protocolRow_'; //Name will be appended to each row of the table
   protocolAdminObj.typeData = 'protocol'; //Specify the key name which needs to pass to server for request routing
   
   Note: - If there multiple tabs then create multiple adminData objects.
   ```

2. Initialize the init function, which is called after loading of the page.

   ```javascript
   function init() {
       tabOperationObj.init(); //Initializing tabs
       protocolAdminObj.setData(protocolData); //Setting data function to admin object
   }
   ```
protocolAdminObj.setParseResponse(parseProtocolResponse); //setting ajax response function to handle data manipulation
protocolAdminObj.hideConsole();

protocolAdminObj.getData('/newadmin/forwardRequest','method=view&type=protocol'); //loading first tab data

3. Create a data function to hold each row value as an individual object.

```javascript
function protocolData(listData) {
    this.id = listData['id']; //Id is a primary key of the record. This is mandatory for each entity
    //Specify Other entity value
}
```

4. Implement the response handler to handle AJAX responses. This function needs to forward this response to adminData’s parseResponse method.

```javascript
function parseProtocolResponse(data) {
    //If you want to manage or update data of the server
    protocolAdminObj.parseResponse(data);
}
```

5. Implement the click event handler for adding, modifying and deleting a record from the display as well as from the database. Below is an example of this handler:

```javascript
//handler for adding record
$('.'+addNew+').live('click', function(event) {
    var newData = {
        id: "new_" + protocolAdminObj.newRowId,
        name: "",
        description: "",
        active: 'True'
    }
    var dataObj = new protocolData(newData);
    protocolAdminObj.addOperation(dataObj, makeEditableRow, saveRow);
});
```
// handler for deleting record

$('.delete').live('click', function(event) {
    protocolAdminObj.deleteOperation(protocolAdminObj);
});

// handler for editing record

$('.greyRow,.blueRow').live('click', function(event) {
    var input = $('input', event.target);
    if (event.target.tagName.toUpperCase() === "INPUT" || input.length > 0) {
        // link exist in the item which is clicked
        return true;
    } else {
        protocolAdminObj.makeEditable(this, makeEditableRow, saveRow);
    }
});

6. After adding the handler, override the two methods called makeEditableRow and saveRow. These two methods are responsible for edit and save functionality.

/* This method getting called for each column. */

function makeEditableRow(index, tdObj, dataObj) {
}

/* This method getting called for save button click. */

function saveRow(rowId, rowObj) {
    var paramData = {
        // Need to create a JSON object of parameters to pass to server
        type: protocolAdminObj.typeData,
        method: method,
        parameter: JSON.stringify({
            // Create a request parameter JSON object list
        })
    };
}
$.ajax({ url: '/newadmin/forwardRequest', data: paramData, success:
    function(data) {
        if(method == 'add') {
            rowId = data['dataId'];
            $(rowObj).attr('id', protocolAdminObj.tableRowName + rowId);
            var checkBox = $('#'+ $(rowObj).attr('id') + " input:checkbox")[0];
            $(checkBox).attr("id", protocolAdminObj.checkBoxName + rowId);
            $(checkBox).val(rowId);

            var newData = {
                id: rowId,
                name: protocolName,
                description: protocolDescription,
                active:active
            }

            dataObj = new protocolData(newData);

            protocolAdminObj.dataList[rowId] = dataObj;
        } elseif(method == 'save') {
        }

        protocolAdminObj.convertToNonEditable(rowObj, dataObj);
    }, error: function(xhr, textStatus) {
        throw "Error occured while saving data. Please try again later.";
    }, type: "POST", dataType:"json"});
CHAPTER 5

APPLICATION EXAMPLES

In this chapter, we show examples of how the database admin module is used within CyberWeb. The database is used by all CyberWeb graphical user interface. The examples included in this section demonstrate several different ways that the database admin interface is used.

5.1 DYNAMIC MENU RENDERING

Figure 5.1 shows the design and architecture of dynamic menu loading inside CyberWeb. All menus and their representation are specified in a JSON file. This file is read at the start of the application server and creates the menu object. The apps_global.py file does this task. Each layout Mako files needs to call apps_global for menu rendering based on the current URL and its action.

This architecture provides various benefits to the CyberWeb framework.

1. It makes CyberWeb menus dynamic so they can be added to the system at any point in time.
2. It provides flexibility and scalability.
3. It reduces hard coding and replication of the menu code.
4. Readability of the menu structure is achieved and
5. No code changes are required for adding or deleting menu items.
6. Menus are rendered recursively, with minimal coding effort. Now, Figure 5.2 shows an example of this recursive menu rendering. In the Authentication header menu shows three-sub menus on left side of page (CyberWeb password, PKI Credentials, and GSI Credentials). After selecting PKI Credentials, the menu is expanded to three submenus (CredInfo, AddResource, and Create PublicPrivatekey) shows below.

5.2 ADMIN DATABASE USER INTERFACE (ADUI)

The graphical user interface (GUI) for this application is not very complicated and is very user friendly and well organized. The user interfaces were built using HTML - with some aspects of Cascading Style Sheets (CSS). The options available on the screen for the user depend on the access level of the logged in user.
Figure 5.1. Architecture of menu rendering.
Figure 5.2. Showing recursive menu rendering.

Figure 5.3 shows the rich and generic user interface for the administration module. This explains the 2-column layout usage, where the left column lists all the entity and the right column displays the database records with tabular form.

The Administration interface uses tab to group similar entities to help organize the database into logical collections. Figure 5.4 shows a group consisting of Users, Groups entities and “User Group” into a single page. “User Group” provides the user association to group information management capability.

Figure 5.5 shows the addition of a record to database with the help of administration interface. It is very easy and convenient for user to do such operation on the database. Figure 5.6 shows the modification of the database record with the help of administration interface. If the user clicks on any row of the record, it will convert into an editable form with its value.

5.3 USING THE ADMIN API: DYNAMICALLY CONFIGURING AND USING RESOURCES AND SERVICES

This section demonstrates the “plug-n-play” feature of the CyberWeb. Figure 5.7 shows the list of available resources to use. If the user wants to add new resource account to CyberWeb, he/she selects the configure account link. Figure 5.8 shows the account configuration page after clicking the configuration link.
<table>
<thead>
<tr>
<th>User Name</th>
<th>Name</th>
<th>Email</th>
<th>Institution</th>
<th>In Active</th>
<th>Available From</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>admin</td>
<td><a href="mailto:admin@somewhere.com">admin@somewhere.com</a></td>
<td>cyberweb</td>
<td>True</td>
<td>2012-05-04 10:54:40:305B11</td>
</tr>
<tr>
<td>cyberweb</td>
<td>cyberweb</td>
<td><a href="mailto:cyberweb@somewhere.com">cyberweb@somewhere.com</a></td>
<td>cyberweb</td>
<td>True</td>
<td>2012-05-04 10:54:40:305B11</td>
</tr>
<tr>
<td>cnguest</td>
<td>cnguest</td>
<td><a href="mailto:cnguest@somewhere.com">cnguest@somewhere.com</a></td>
<td>cyberweb</td>
<td>True</td>
<td>2012-05-04 10:54:40:305B11</td>
</tr>
<tr>
<td>mary</td>
<td>Mary Thomas</td>
<td><a href="mailto:mary@science.sdsu.edu">mary@science.sdsu.edu</a></td>
<td>SDSU</td>
<td>True</td>
<td>2012-05-04 10:54:40:305B11</td>
</tr>
<tr>
<td>cary</td>
<td>Cary Cheng</td>
<td><a href="mailto:cary.cheng@gmail.com">cary.cheng@gmail.com</a></td>
<td>SDSU</td>
<td>True</td>
<td>2012-05-04 10:54:40:305B11</td>
</tr>
<tr>
<td>smits</td>
<td>Smits More</td>
<td><a href="mailto:smits.smiling@gmail.com">smits.smiling@gmail.com</a></td>
<td>SDSU</td>
<td>True</td>
<td>2012-05-04 10:54:40:305B11</td>
</tr>
<tr>
<td>hetang</td>
<td>Hetang Shah</td>
<td><a href="mailto:hetang.shah@gmail.com">hetang.shah@gmail.com</a></td>
<td>SDSU</td>
<td>True</td>
<td>2012-05-04 10:54:40:305B11</td>
</tr>
</tbody>
</table>

Figure 5.3. User interface for admin user definition. Image created by author.
Figure 5.4. Grouping of the multiple definitions on a single page using tabs. Image created by author.
Figure 5.5. User interface for adding new user. Image created by author.
<table>
<thead>
<tr>
<th>User Name</th>
<th>Name</th>
<th>Email</th>
<th>Institution</th>
<th>Is Active</th>
<th>Available From</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>admin admin</td>
<td><a href="mailto:admin@somewhere.com">admin@somewhere.com</a></td>
<td>cyberweb</td>
<td>True</td>
<td>2012-05-04</td>
</tr>
<tr>
<td>cyberweb</td>
<td>cyberweb cyberweb</td>
<td><a href="mailto:cyberweb@somewhere.com">cyberweb@somewhere.com</a></td>
<td>cyberweb</td>
<td>True</td>
<td>2012-05-04</td>
</tr>
<tr>
<td>guest</td>
<td>guest guest</td>
<td><a href="mailto:guest@somewhere.com">guest@somewhere.com</a></td>
<td>cyberweb</td>
<td>True</td>
<td>2012-05-04</td>
</tr>
<tr>
<td>mary</td>
<td>Mary Thomas</td>
<td><a href="mailto:mary@science.sdsu.edu">mary@science.sdsu.edu</a></td>
<td>SDSU</td>
<td>True</td>
<td>2012-05-04</td>
</tr>
<tr>
<td>carly</td>
<td>Carly Chang</td>
<td><a href="mailto:carly.chang@gmail.com">carly.chang@gmail.com</a></td>
<td>SDSU</td>
<td>True</td>
<td>2012-05-04</td>
</tr>
<tr>
<td>smith</td>
<td>Smith More</td>
<td><a href="mailto:smith.smith@gmail.com">smith.smith@gmail.com</a></td>
<td>SDSU</td>
<td>True</td>
<td>2012-05-04</td>
</tr>
<tr>
<td>hetang</td>
<td>Hetang Shah</td>
<td><a href="mailto:hetang.shah@gmail.com">hetang.shah@gmail.com</a></td>
<td>SDSU</td>
<td>True</td>
<td>2012-05-04</td>
</tr>
</tbody>
</table>

Figure 5.6. Modifying user record. Image created by author.
Figure 5.7. List of currently available resources. Image created by author.
Figure 5.8. Configuring new accounts with the help of administration interface. Image created by author.
Figure 5.8 shows the account information page, which lists current account. To add a new account to CyberWeb, the user clicks the Add New link; A new record is added with the blank values displayed on the screen. The user specifies the required values for the new account and then click save.

The PKI credential pop-up window appears if the user clicks on Configure PKI/SSH Credential link or hit save then Administration interface will ask for the account credential i.e. username and password for the remote resource. This is shown in Figure 5.9.

Figure 5.10 show that an account has been configured successfully. This can be verified in the available list of resources as can be seen in Figure 5.11.

5.4 APPLICATION: SCIENCE PORTALS

The demo portal has several features typical of science portal including: user, account management; job management; resource configuration; access control management. Example of these capabilities can be seen in the Ocean Model Simulation Portal shown below.

The Unified Curvilinear Ocean Model (UCOAM) is a high-resolution ocean model, which has been shown to have greater accuracy than similar models and to achieve results more efficiently [28]. CyberWeb is being used to develop a computational environment for the UCOAM project, which includes a community access portal for expert and non-expert users [29]. Figure 5.12 [28] shows a prototype of the UCOAM portal, which use many of the CyberWeb components, including: account management; job management; data file management; and visualization of results, and the database admin module (See Chapter 4 of this paper).
Figure 5.9. Specifying credential for the account. Image created by author.
Figure 5.10. The new accounts is configured and ready for use. Image created by author.
Figure 5.11. Newly added accounts are available for use by all resources and services. Image created by author.
Figure 5.12. CyberWeb application example. Source: M. P. THOMAS AND J. E. CASTILLO, Proceedings of the 12th International Conference on Computational Science and its Applications, Salvador de Bahia, Brazil, 2012, ICCSA.
CHAPTER 6

CONCLUSION

The goal of this research was to develop an advanced user interface for the CyberWeb Framework, to use this interface to develop an advanced database administration module, and to integrate this interface to develop the capability to dynamically configure resources and services. The database Admin GUI is designed to provide step-by-step user guides that allow them to dynamically view, manage, and change the configuration of remote resources and services. The outcome is that this system has been shown to reduce the overhead of manually configuring remote resources, and performing availability checks. Users can configure new resources, services, or accounts on these resources and have them become immediately available for use. The result of this work has been made available, in the form of the CyberWeb Demo Portal. This portal will be available for download with the intent that users can install this portal and launch a local high performance-computing portal. The demo portal supports all the basic operations of science portals, and simplifies the installation steps needed by the CyberWeb administrator. Via the pylons and database interfaces, new features can be added to the CyberWeb application as needed. With these features and capabilities, we believe that the CyberWeb Framework will help scientists in developing their research applications, and decrease the difficulties scientists face when using high performance computational resources and services.
CHAPTER 7

FUTURE WORK

There are many extensive plans for CyberWeb, including the capability to integrate a Cloud resource. This will enhance CyberWeb capabilities. Future plans including a Job Builder module, which be used to automatically schedule jobs or interrupt jobs based on a given set of priorities. Currently, this is a highly requested for the next revision of CyberWeb additional include:

- Development of user interface for application deployment.
- Peer to Peer transfer of job data with the help of a file browser
- Auto deployment CyberWeb installation with the help of Pylons eggs.
- A unit test system for the modules of CyberWeb.
REFERENCES


