SPATIAL DATA MINING WEB PORTAL

Jared M. Moore

A thesis submitted in partial fulfullment of
the requirements for the degree of
Master of Science

Department of Computer Science

Central Michigan University
Mount Pleasant, Michigan
February, 2011

Accepted by the Faculty of the College of Graduate Studies,

Central Michigan University, in partial fulfullment of

the requirements for the master's degree

Thesis Committee:

	Committee Chair
	Faculty Member
	Faculty Member

Date: February 24, 2011

	Dean College of Graduate Studies

Date: 3/18/11

Committee:

Patrick Kinnicutt, Ph.D., Committee Chair

Michael Stinson, Ph.D.

Paul Albee, Ph.D.

ACKNOWLEDGEMENTS

I would like to express my utmost gratitude towards my Thesis Committee: Dr. Patrick Kinnicutt, Dr. Michael Stinson, and Dr. Paul Albee. My committee has been a tremendous help in conducting my research and developing this document. I would especially like to thank Dr. Kinnicutt and the Body Scanning Laboratory for their support in producing this work as well as the opportunity provided by the laboratory for the research. Finally, I wish to acknowledge the support of Central Michigan University in producing this work.

ABSTRACT

SPATIAL DATA MINING WEB PORTAL

by Jared M. Moore

Web portals are an essential component of a research laboratory necessary to present information to external interested parties. This thesis implements a web portal for the purposes of querying the Body Scanning Laboratory's database of over eight hundred subjects, enabling users easy access to the data based on specified constraints. Users can get access to thermal images, raw thermal data, non-personally identifiable demographic information, measurement information and basic statistics on the selected group of subjects queried. Constructing a web portal requires the use of secure programming practices and a knowledge of web development to implement a user-friendly interface that delivers information clearly and concisely. It must also maintain information security and avoid common mistakes that can lead to vulnerabilities affecting the function of the portal. In this thesis, the technologies used in web portal development to ensure secure, intuitive access to the data are discussed in the context of developing this web portal. It was found that by using modern technologies such as AJAX, JavaScript and JFreeChart, a web portal can be created with a desktop-like feel which is easy to use, is secure, and provides flexibility for users.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Presentation and dissemination of information is a crucial part of generating interest and raising awareness of a research group in today's information centric world. The Body Scanning Laboratory focuses on maintaining a database of subjects with information obtained through thermal scans of the human torso as well as various spatial measurements from each subject. By using innovative technologies such as three-dimensional body scanners, infrared imaging systems and environmental chambers, the research group has been able to compile a database of thermal scans of the human torso coupled with three-dimensional scans and demographic data. In order to raise awareness of the group's work , the Functional Apparel Design Web Portal was established in order to create a data repository for the results collected in the lab. The web portal seeks to create a bridge between the information gathered in the laboratory and researchers and organizations based all over the world.

In order to provide a rich user experience, the web portal provides interactive functionality to allow a user to access data in the database and work with the data both on the website as well as offline through targeted downloads of specific sections of the database. A number of design goals were proposed in order to create an interactive portal that allows outside users the ability to gain crucial insight into the data while also being restricted from editing any of the data in the database. The web portal's main functionality is to provide access to the information gathered at the Body Scanning Laboratory in a way that is simple and intuitive. The portal also provides access to the database of thermal images collected from the subjects to provide the user with a visual representation of the subject being investigated to aid in visualizing the subject.

CHAPTER II

PRIOR DEVELOPMENT & OBJECTIVES

Previous work on the web portal was conducted by Jack Slingerland, a former graduate assistant. The following two sections detail the prior development completed on the web portal prior to the work completed during the work detailed in this thesis as well as the objectives relating to the web portal. The objectives detailed are meant as a general guideline to why the work undertaken but do not necessarily represent all the goals of the web portal.

Prior Development

Prior development of the web portal consisted mainly of layout and associated design characteristics. The basic navigational structure of the website was established during this development and form the core of the current navigational strategy. The query engine implemented the basic features but had not been tested for errors and correctness. The back-end database had been created and housed both the data from the body scanning laboratory as well as the user profile and login information. The ability to download query data and images was implemented; however, the implementation had to be altered as information in the database had changed and required revision to the existing code. The original file hierarchy of the website mainly consisted of a primary folder that held most of the files in it with a few sub-folders for things such as: images, thermal images, a few external packages and some of the web pages in the pages folder. The primary data storage for the data collected during the process of body scanning are two Microsoft Access databases housed in the laboratory. The first database contains personally identifiable information not used for the normal research activities, while the second database contains non-identifiable information that is then used to conduct the various research projects associated with the laboratory. These databases are local to the laboratory and do not have the capabilities to

share information readily with outside parties. In the current setup of the laboratory, they are still in use as primary data storage for the body scans which are then transferred to the PostgreSQL database used by the web portal.

Objectives

The main focus of the web portal is to provide an interface that aids in the presentation and dissemination of information collected by the body scanning laboratory to an audience of academic and industry professionals. In order to create a web portal that meets this focus, a number of objectives have been proposed: create a portal that is user friendly and intuitive, deliver information in an efficient and concise manner, and provide users with the ability to access detailed information for use on their own machines. The main focus of this thesis is the presentation and delivery of the information collected at the body scanning laboratory. Core issues facing the presentation and delivery of information pertain to database design, efficient and concise delivery of information through the web portal and information security from presentation to delivery.

In order to ensure that the web portal is user friendly and intuitive, the interface of the web portal will be periodically reviewed by the professors that work in the body scanning laboratory and suggestions will be reviewed and implemented where necessary. The overall interface of the web portal will be a typical web layout consisting of a menu bar, side navigation bar, and a main body area for the current page being viewed. Each page will attempt to convey it's message in as straightforward manner as possible and provide tooltips where applicable. Most pages will be directly accessible from any page on the site in order to reduce the time required to navigate from one page to another. Another aspect of improving usability will be the implementation of AJAX in places where it is beneficial to increasing the user experience by reducing page loads and providing a more seamless experience for the user.

CHAPTER III

BACKGROUND

A variety of different programming languages and web technologies have been used in designing and implementing the web portal. Through the use of PHP, SQL, JavaScript, Java applets and AJAX, a robust web application has been developed to meet the needs of the research group. The following sections describe a few of those technologies in detail as well as some of the concerns associated with each one.

PHP

PHP is one of the more popular web scripting languages for constructing websites on the Internet today. First conceived in 1995 by Rasmus Lerdorf, PHP has undergone a vast transformation from a set of Perl scripts into a full implementation in the C programming language [26]. As of July 2007, PHP was estimated to have been used on over twenty million separate domains and over one million separate IP addresses [25]. Its widespread popularity can be attributed to it's open source license and the ability of users to be able to contribute to the development of PHP to make it more robust and offer more features as development continues [27].

Dynamic Page Generation

Modern day websites rely heavily on dynamically generated pages to accomplish a variety of tasks such as: customizing a user's browsing experience to their preferences, increasing sales through targeted advertising, establishing repeat users through custom presentation catered to the user's needs, and refreshing the content of the site without having to rewrite the entire source code of the page. In order to satisfy the requirement of a modern day web scripting language to offer dynamic page generation, PHP offers website designers language features that allow for dynamic generation of web pages based on preferences

or settings related to individual users and guests. Highly customized pages can be generated as requested by using conditional formatting, in conjunction with the preferences and settings for an individual user. With conditional formatting, one script can deliver a variety of different pages depending on the conditions specified by a user allowing for both flexibility in programming as well as improving the content of the website. By using these features, multiple HTML files can be condensed into a single PHP script reducing the code repetition that would be required to account for minor differences on variations of a single page.

Database Integration

In conjunction with dynamic page generation, databases have allowed websites to increase their usability and scope by providing a more robust system to deliver content as well as provide storage for user information in a structured and easy to implement format for web designers. PHP integrates with many of the currently available database systems through various vendor specific extensions [28]. The vendor specific extensions allow for each database system to be used to it's full potential by allowing unique database features to be coded into the PHP scripts while harnessing features that the database designers intended. These extensions do sacrifice code portability between database systems, as the code may contain database language features not supported by other vendors. This may lead to the need for significant rewrites should the database used in an application be changed.

The issue of code portability between database systems in PHP code was addressed by the PEAR database abstraction layer. It was designed to provide a bridge between PHP code and specific database systems to prevent code from becoming locked into a specific database vendor's system. Currently in it's second form, commonly referred to as the MDB2 extension, the database abstraction layer sacrifices some of the vendor specific

features in favor of providing an implementation that allows designers to change between database systems with relatively minimal code editing [24]. PHP does not enforce the use of the MDB2 extension, instead choosing to leave it up to the application designer to decide what approach to utilize allowing greater freedom of choice when designing a web application.

Security Concerns

Security is a top concern for any web application. Sensitive data stored in databases can be vulnerable to a variety of attacks due the relatively high profile of a website. Unlike a desktop application that does not present itself openly to the world, a web application is exposed to more risk from unauthorized access. This increased risk, coupled with the growing preference for online applications, makes security an ever growing concern. There are a variety of common attacks found in the web programming domain such as XSS scripting, also known as cross-site scripting, and SQL injection attacks being the most common attacks targeted at all web applications. According to the Open Web Application Security Project, OWASP, SQL injection ranks as number one on their list of the top ten web risks in 2010 closely followed by cross-site scripting [16]. PHP is also commonly vulnerable to an attack specifically targeted at the global variables specific to the language.

SQL Injection

Databases are an integral part of most web applications and require SQL statements from the web application in order to submit and retrieve data. In a traditional web application, the SQL statements will be generated according to predefined code by the developer and user input will be placed at specific points in the code in order to generate a correct SQL statement. A properly coded web application will sanitize input from the user so that it will not be treated as executable SQL code. It is possible that due to programmer er-

ror, language design, or simply a lack of features to sanitize the input, a vector of attack is possible through the input of a user. This class of attacks are commonly referred to as SQL injection domain attacks. Using this vector, malicious users attempt to enter SQL commands in the input that will allow unauthorized access to the data that can result in a loss of data or a loss of control, both of which are undesirable from a security standpoint.

It is possible to mitigate the risk of SQL injection through the use of careful programming techniques and application of recently developed testing procedures. Initially, work on SQL injection mitigation focused on static methods for detecting SQL injection attacks [11]. Another approach to handling SQL injection attacks involves real time analysis of SQL queries using an automated system that checks for certain flags or format to SQL queries, and allows only the approved queries to be executed[12]. Automated tools have also been recently developed in order to test possible SQL injection attack vectors through methods that allow for a large variety of attack styles to be tested programatically with the results being reported to the programmer [8][6]. By having a knowledge of possible attack vectors and techniques and the careful use of tools to correct vulnerabilities, a web designer can minimize the risk of an SQL injection attack.

Cross-site Scripting

Cross-site scripting attacks, often referred to as XSS attacks, involve an outside intruder attempting to inject scripts into a website's source code in an attempt to get the malicious code executed by another user when they access the infected page. XSS attacks are primarily successful when a web application does not properly filter user input from text boxes. These unfiltered input boxes provide a vector for malicious users to attempt to insert scripts into an input box which are then executed by the server when the input is submitted. The server will treat the code snippets from the input box as actual code to execute instead of just simple text delivered by a user. A variety of methods have been proposed to counter

XSS attacks in recent years, including the development of input scanners that utilize string analysis to filter out characters and character sequences that are known to be used in scripts. Further refinement of detection techniques has led to the addition of scanners that check for untrusted scripts in the generated code of a website as proposed by Gary Wassermann and Zhendong Su from the University of California at Davis [30]. While these scanners are not perfect, they can be effective at greatly reducing the vulnerability to an XSS attack by both helping to locate areas of potential vulnerability while also actively filtering out attempted attacks.

PostgreSQL

Databases are an integral part of almost any web application. Whether it be the need to store user information or code snippets to be loaded on demand, a database provides many essential services required to make a web application function in a timely, efficient manner. PostgreSQL is a database system, composed mainly of C code, designed for both UNIX and Microsoft Windows based systems and released under the PostgreSQL license, which is an open source license similar to the BSD license [18]. Users of the database are freely allowed to distribute the code for the database in their own packages as long as they relinquish the right to hold PostgreSQL's creators liable and display the copyright for PostgreSQL in any software that is distributed [20]. The license even allows for users to modify the source code of the database engine itself as long as the user does not hold the creators responsible for the modifications [20]. Keeping with PostgreSQL's open source goals, the project is maintained by a community of developers throughout the world and anyone is free to contribute to the project [19]. A 2006 study of the contributors of PostgreSQL found that most of the code is written by a small team of core developers of under thirty members. Further investigation found that patches had been submitted by a vast number of members and only integrated into the code by the core team [10].

CHAPTER IV

LITERATURE REVIEW

Dealing with large amounts of data and a potentially vast audience of interested members have led to portals as a common interface between laboratory data and members. Web portals are used in a variety of applications and domains ranging from data delivery tasks, such as the one provided by the F.A.D.D. web portal, up to complex data analysis portals at one of the nation's largest research laboratories, Oakridge National Labratory [29]. Common concerns when developing a web portal of any kind involve: scalability of the portal to handle the demands of the expected user base [29], information security [4, 22, 14, 21, 15, 13, 11, 12], development of interfaces that are intuitive as well as attractive in order to keep users active in the portal [3], and finally different ways of storing data for retrieval other than traditional databases [17, 7]. The following paragraphs further explore the different areas of web portal development concerns organized by each topic.

Scalability Issues

Differing levels of traffic create scalability issues for web portals as the servers must deal with the load associated with access from many users. For a web portal to be successful, the data should be available at will and an inadequate server can result in costly downtime. Downtime can damage the reputation of a portal as well as potentially reduce the user base if there is a competing portal available. Oakridge National Laboratory performs a large amount of research in the neutron scattering field which both requires highly scalable data storage design as well as a web portal to meet the needs of researchers around the world wishing to gain access to the data. In order to meet this tremendous demand, the designers of the portal utilized a Java based programming approach to create a portal that seamlessly integrated with the dataset as well as open-source visualization tools to provide information to the researchers in an efficient, and secure manner [29]. In order to meet

the demand of the portal requirements, the designers utilized a multi-tiered architecture to efficiently divide the tasks of information processing and delivery. Although most web portals will not deal with the large amounts of data present in the neutron project, correct data storage and access practices can greatly increase system stability and efficiency.

Security Issues

Information is the key value of most web portals which makes the security of information is a top concern for any web portal. Prior research into information storage not only relates to the scalability issues facing large databases but also the issues of how to present the data and how much to make freely available. A study conducted by a team of researchers at the University of Illinois at Urbana-Champaign investigated "deep-web" sources; information that is stored in databases available on the Internet. The information contained in these databases are only available from behind query interfaces that keep information from web crawlers that attempt to index the information available on the Internet. The study found that there were thousands of these databases available on the Internet and estimated that the search engines may eventually be able to access this information but only after advancing current practices drastically in order to increase the efficiency at which web crawlers can gain access to the information [4]. Continuing research into web database security involves techniques to identify user interactions with web databases, as often users are grouped into anonymous database connection groups for efficiency of accessing the database making tracking of attacks and malicious users difficult if not an impossible task. A study conducted by a research team at the Open University, Raanana, Israel, developed a solution to this problem by creating an identifying key, not accessible to a user, that identifies what user performed what operations at a given time. Because the user is not able to tamper or know the key, a malicious user cannot spoof another user and perform operations on the database without being identified. By adding in such an iden-

tifier the database system is able to offer more fine-grained access methods to data and prevent a user from gaining access to information they are not authorized to view[22].

Another area of research into security of websites involves investigating the vulnerabilities and security patches required to mitigate risks from SQL Injection, Cross-site scripting and URL injection attacks. As noted earlier, SQL Injection is one of the more common vulnerabilities [16] and correcting these vulnerabilities is important to both the success and security of any web application. The solutions proposed to mitigate SQL Injection attacks vary in their approach to solving the problem with possible avenues of correction ranging from new code libraries that attempt to remove coding mistakes from creating vulnerabilities [14] to various detection methods involving automated detection [21, 15]. SQL Injection vulnerabilities often arise from improper handling of user input in text fields and other places where users have the ability to enter text [13]. Implementing code practices and strategies to reduce or eliminate the possibility of vulnerabilities appears to be an easy solution, but can be difficult in actual practice. A 2007 study by Nicolas Juillerat at the University of Fribourg proposed implementing a code library to handle SQL Injection attack vectors as well as some limited cross-site scripting attack mitigation and detection by performing interactions with the database through predefined methods in the library that verify statements to identify malicious ones. Overall results of the study were promising with implementations in C# and Java performing well considering the relative youth of the library. Limitations were present, mostly due to the language constructs and unfinished design at the time of publication [14].

There are a variety of different ways of detecting injection attacks with some of the more recent methods involving static detection of SQL injection attacks[11], comparing the SQL string to a known format for a query[15], evaluating the syntax of an SQL query to detect anomalies[12] as well as by applying a statistical analysis of traffic to a database and detecting queries that do not fit a known profile [21]. Two prevalent styles of detecting

SQL injection attacks involve analyzing the query syntax for anomalies or comparing the query to a known traffic pattern and then making a decision based on the differences from the norm. By evaluating the query syntax, certain differences such as the placement of keywords and operators can give away whether a statement should be permitted or not and can help to reject potentially harmful queries[12]. By examining traffic patterns to a database, it has been found that often, the same queries are run continually as a web interface usually provides a few main forms of interaction with the database and filtering out queries that do not match the established traffic patterns can eliminate a significant portion of harmful queries [21]. These two styles are not mutually exclusive and have often been implemented together in different research papers in order to provide a more comprehensive filtering solution [11, 21].

Development Issues

Developing a query engine for any application usually involves a large amount of custom design and code, but insight into other approaches can save time and effort that could be better used in other areas of design. When developing a query interface for a user, a variety of conditions must be accounted for to develop the system in a way that is user friendly and allows a high degree of customizability. Past research into query engines has led to methods of adapting data from a relational data model that is found in most relational databases and transforming it into structured XML data that could then be queried through the use of custom XML queries that lack the restrictions imposed in a traditional RDBMS model [17]. Although not every query can benefit from transforming the database into an XML data format, the different ways of approaching query design and development can provide inspiration for new and more efficient ways of designing queries.

The semantic web, a concept currently being discussed in research circles and originally proposed by Tim Berners-Lee [2] has created a new focus for web portals using

semantic web concepts such as machine understanding and interpretation to deliver a new form of web portal. With the existing number of web portals already developed, and the likelihood that these portals can still deliver relevant information when paired with newer semantic web portals, a framework has been suggested to facilitate information gathering in semantic web applications information gathering interface that can deal with data in a variety of formats [7]. The interface attempts to work with all information, whether it be in conventional formats found in normal web portals such as: database information, XML files or simple text files, or data that is already in a semantic web style format. This framework has been implemented in a few different projects already and appears to present a good design framework for future projects [7].

CHAPTER V

DATA PRESENTATION

The central focus of the web portal is the presentation of data to the outside world. Consequently, the main focus when designing the web portal is how to provide outside users access to the data in a manner that is clear and concise while also preventing the users from having direct access to the database. The overall goal of the web portal is to present the information obtained at the Body Scanning Laboratory in a clear, robust manner and yet keep it simple for new users to understand and use. Due to the dual nature of the data being both statistical and visual, a multi-faceted design was chosen to try and blend the best of both types of data into an intuitive package.

The demographic information provides basic statistics of each subject such as: height, weight, gender, age, BMI and a variety of other categories available as needed. In order to present this data on a subject by subject basis as well as a more abstract form that presents groupings of subjects, the data presentation was broken down into two major categories. The first method of presentation involves individual subject records which is handled by the query engine. Users are able to work with structured queries and can specify constraints on the dataset based on a predefined categories with customizable ranges. The second form of presentation deals with abstracting the subjects into defined groups based on up to three grouping attributes that are also predefined. This type of presentation is handled by the data analysis engine.

The second form of data stored in the database is divided up into thermal scan images and raw data. Each set of images is associated with a subject in the database and it was decided that the thermal images should be presented as an added feature in the query engine results. Each line of the results in the query engine relate to a specific subject with a set of images. If a user clicks on the row, a window is brought up using AJAX with the subject's images and a thermal scale in the window along with demographic information.

The thermal images can also be downloaded in JPEG format for later viewing or use in reports.

Query Engine

Presenting information on a subject by subject basis was one of the key information presentation goals of the web portal. The query engine and associated web pages were designed to accomplish this task. The basic premise of the query engine is to allow a user to select a predefined set of constraints on the data and then execute a query with the specified constraints to get a result set of subjects. This result set could then be searched through to gain some type of insight on a subject by subject basis. By using a combination of predefined constraint choices and user-defined ranges, the query is able to be flexible, but also prevent users from gaining complete control and knowledge of the dataset. It also allows the user to view the thermal images for the resulting subjects and provides the ability to download the measurements of each subject, thermal images of a subject and the thermal properties of each subject. These results are delivered in a comma-separated value format which can be opened in traditional spreadsheet applications such as Microsoft Excel or similar products. In order to accomplish the goals of the query page, a variety of programming techniques and technologies were required to build the multiple components of the engine and resulting page.

Constraint Generation

First and foremost, PHP and PostgreSQL are used to construct the constraints options in order to build a predefined query with user-defined values. The code in Figure 1 illustrates the use of PHP to predefine the query while the constraints are dynamically generated by a separate PHP function in Figure 2 that takes in a new constraint for each one that is selected and appends the information to the current list of constraints for the query.

```
$this->query = "
            SELECT d.id, age, weight, height, exercise, occupation,
                   gender, bmi, inseam_left_in, waist_girth_in,
                   bust_chest_girth_in, mid_neck_girth_in
            FROM  demographicinfo d, measurements m
            WHERE   d.id = m.id".
$this->queryConstraints.$constraintValues[21]." ".$orderBy;
```

Figure 1. PHP Predefined Query and Dynamically Generated Constraints

```
private function procOps($type, $cat, $upper, $lower,
    $currentConstraints) {
  $newConstraint = " AND ";
  switch($this->getOp($type)) {
    case 1: //">"
      $newConstraint = $newConstraint."($cat > $upper)";
    break;
    case 2: //"<"
      $newConstraint = $newConstraint."($cat < $upper)";
    break;
    case 3: //">="
      $newConstraint = $newConstraint."($cat >= $upper)";
    break;
    case 4: //"<="
      $newConstraint = $newConstraint."($cat <= $upper)";
    break;
    case 5: //"to (inclusive)"
      $newConstraint = $newConstraint."($cat >= $lower)
                AND ($cat <= $upper)";
    break;
    case 6: //"to (exclusive)"
      $newConstraint = $newConstraint."($cat > $lower)
                AND ($cat < $upper)";
    break;
  }
  return $newConstraint$;
}
```

Figure 2. PHP Processing Function for Individual Constraint

By using a predefined query with user-defined constraint values, the dynamic portion of the query is the only portion of the query that can be directly impacted by the users choices and is a potential avenue for SQL Injection attacks. The combination of the user

input and predefined query limit the chances for a successful attack. It also makes managing access to the data easier to prevent users from gaining complete access to the data. As a further step in preventing malicious attacks, JavaScript is used to execute functions that validate the input for correctness and reject incorrect user input data. As the five possible parameters are either numbers or a drop-down selection box, the validation function rejects information that is not a valid integer or decimal depending on the constraint.

Once the user has chosen a desired set of constraints and filled in the constraint values, the query is executed and the resulting set of subjects is returned to the user. The result set is placed into a tabular format consisting of ten subjects per page with proper buttons to move through the pages one at a time or to jump to a given page number. The result table contains the bulk of the functionality in the query page with additional features other than listing the subjects such as: modal windows for each subject's thermal images, sorting by different columns using AJAX, ability to download a .csv file of the subjects, as well as additional download options for measurements and thermal images.

Modal Windows

In order to increase the usability of the web portal and to increase cohesiveness between the features, the thermal images for each subject can be accessed by clicking on a subject in the result set. A modal window is then displayed with the subject's thermal images retrieved from the server on an as needed basis. The modal window displays statistical information about the subject, the subject's thermal images and a thermal scale. In order to create the seamless feel of the modal window integration, JavaScript and AJAX have been used to fetch the data from the server for placement into the modal window to create the modal window inside of the browser. An example of the window can be seen in Figure 3 illustrating the modal windows placement on the same page as the query information.

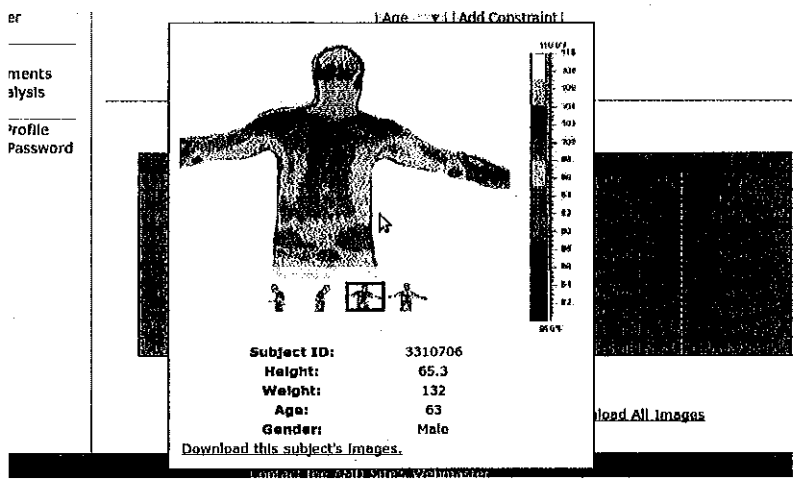

Figure 3. Modal Window for Thermal Images of a Subject.

When a user clicks on a subject, the information necessary for the modal window to be presented is retrieved from the server using an AJAX call to reduce page loading times. By retrieving the data on an as needed basis, only the images that a user wishes to see are loaded which can greatly reduce the page load time and bandwidth usage. The use of AJAX and modal windows helps to make the integration of thermal images and the query page seamless to the user experience resulting in a more desktop like feel to the query engine. It would be impractical to load every subject's thermal images on each query as the majority of the time they will not be viewed by the user and the overhead and bandwidth required to retrieve the information would be prohibitive and potentially impact the performance of the web server. Although the AJAX call is required when a user clicks on the subject to display the modal window and it does require an indeterminate amount of time to query and retrieve this information from the server, in practice, the time required to retrieve the necessary information has little or no delay and presents a seamless interface to the user without the added clutter of new windows for each subject.

Data Analysis

The data analysis engine is the second of the two engines used in the web portal. Whereas the query engine is designed to deal with the available data on a individual subject by subject basis, the data analysis engine is meant to perform more aggregate analysis tasks by dividing subjects into groupings based on user defined criteria. The central goal behind the engine is to provide the user with a means of dividing up the subjects on a level that is not just focused on the individual. Rather, it is meant to help identify trends that may occur in specific groupings of subjects. For instance, it may be possible that subjects in the twenty to twenty-five year old age range have a tendency to have a body mass index, BMI, of under twenty-four as compared to subjects in the thirty to thirty-five year old age range. By using the data analysis engine, it is possible to create these groupings and evaluate the results to attempt to spot trends.

Information provided by the data analysis wizard is related to statistical information for a target attribute that is specified during the user interaction with the wizard. The target attribute is then analyzed by returning information such as the minimum, maximum, average, standard deviation and number of subjects in each subset. Figure 4 illustrates the data analysis table with a target attribute of age and a primary grouping attribute of BMI with a secondary attribute of chest girth. Each row in the table represents a subset of the subject's in the database defined by the primary and secondary attributes.

Figure 4. An Example of the Data Analysis Table.

Engine Structure

Generating a data analysis table requires multiple user choices in order to deliver a customized analysis. This is accomplished by a wizard that spans multiple web pages with each successive page being are dynamically generated from choices made on the previous pages. The back-end supporting the multiple pages consists of one file that houses the different functions required to dynamically generate the HTML for the possible combinations and passes the generated HTML code to the front-end for display to the user. Each page in the process contributes information that is eventually used in both subsequent pages as well as in generating the final output table complete with information obtained from the database.

```
Data Analysis Constraints = Array(   [0]=Target Attribute,
                                     [2]=Primary Grouping Attribute,
                                     [3]=Number of Primary Groupings,
                                     [4]=Secondary Grouping Attribute,
                                     [5]=Number of Secondary Groupings,
                                     [6]=Tertiary Grouping Attribute,
                                     [7]=Number of Tertiary Groupings )

Data Analysis Groupings = Array(     [0]=Primary Grouping 1 Lower Bound,
                                     [2]=Primary Grouping 1 Upper Bound,
                                     [3]=Primary Grouping 2 Lower Bound,
                                     ... )
```

Figure 5. Data Analysis Information Arrays

In order to facilitate the transfer of information in an easy to use, concise manner, a set of arrays were used to hold the necessary information for the wizard. The array implementation was used due to PHP's support for sessions which allows for arrays to be ported between pages without the need for a traditional transfer of information in HTML such as POST. Two arrays are created after the first transfer of information from the client

to server and are modified as needed during the execution of the data analysis wizard. Figure 5 lists the contents of the arrays used in the wizard.

```
$v = $_SESSION['grouping'];
$v[0] = str_replace(" ", "-", $v[0]);
$v[2] = str_replace(" ", "-", $v[2]);
$v[4] = str_replace(" ", "-", $v[4]);
$v[6] = str_replace(" ", "-", $v[6]);
$v[3] = $_POST['groupby1'];
$v[5] = $_POST['groupby2'];
$v[7] = $_POST['groupby3'];
$_SESSION['grouping'] = $v;
$code = "<div align='center'>
  <form id='step3' action='./index.php?pid=9&sid=8' method='POST'
    align='center' onsubmit=\"return validateranges('".$v[2].
      "','".$v[3]."','".$v[4]."','".$v[5]."','".$v[6].
      "','".$v[7]."')\">"."<table align='center'>";
$code .= $this->attrgroupings('primary',$v[2],$v[3],0)."<tr></tr>";
if($v[4] != "None") {
  $code .= $this->attrgroupings('secondary', $v[4], $v[5], 1).
    "<tr></tr>";
  :
} else {
  $v[5] = 0;
  $v[7] = 0;
}
$code .= "</table><input type='submit' value='Next' name='third'>
  </form></div>";
$_SESSION['grouping'] = $v;
return $code;
```

Figure 6. PHP Code for Generating a Page of the Wizard

The structure of the PHP code to deal with the arrays mainly involves functions that translate the numerical and string data contained in the array into information that is used to generate the HTML code for the various pages. The various forms are generated dynamically, with each individual page being generated according to the number of groupings, types of groupings and values that a user selects as they move through the wizard. In this way, a general PHP script can be created that can generate the required HTML code necessary to deliver a highly customized data analysis result to the user. Figure 6 illustrates

the code used to generate page three of the data analysis wizard where a user would enter value ranges for each grouping that they had chosen in the previous pages. By using the session array, information from the first two pages is carried over and the new information from the previous page is entered into the array by POST actions seen in lines six through eight. The HTML code is then generated in lines ten through twenty-five and returned to the calling function which will output the HTML code to the browser. The array implementation simplifies the transfer of information and facilitates simplified code generation as seen in the calls to the attrgroupings() function in lines fifteen and seventeen. By using PHP's built in support for Session variables, the information is able to persist without the need for complicated POST variable setups or database interactions.

Downloading Result Sets

While the data analysis engine and page provide a user with a basic engine to view and manipulate the data retrieved from the database, from time to time a user may wish to view all of the generated tables at once or save the tables for later analysis off-line. Two download options have been created that allow a user to obtain the currently displayed table on the screen with associated subject information for each table as well as an overview of just the tables. The data is presented in a .xls format compatible with spreadsheet software.

Downloaded files are generated dynamically when a request from a user has been received by using the PHPExcel package [1] available under the GNU Library General Public License[9]. A set of PHP scripts are used to implement the PHPExcel library in combination with the different data formats available in the web portal. The PHPExcel package allows for formatted .xls files to be generated easily with formatting such as borders and shading being implemented in a straightforward manner as well as the ability to set document metadata properties easily through predefined methods. Figure 7 illustrates the PHPExcel package's ability to handle setting the metadata attributes of a .xls file. Set-

ting the metadata of a .xls file without these methods is cumbersome and often requires a fair amount of coding, but by harnessing the power of PHP extensions, it is possible to use predefined extensions that allow for a reduction in code development time as well as simplified, easy to understand code.

```
$objPHPExcel = new PHPExcel();
$objPHPExcel->getProperties()->setCreator("FADD - Central Michigan
    University")
        ->setLastModifiedBy("FADD - Autogen")
        ->setTitle("Data Analysis Results")
        ->setSubject("Data Analysis")
        ->setDescription("Data Analysis Results.")
        ->setKeywords("data analysis FADD")
        ->setCategory("Data Analysis Results");
$objPHPExcel->getActiveSheet()->getPageSetup()->setOrientation(
    PHPExcel_Worksheet_PageSetup::ORIENTATION_LANDSCAPE$);
```

Figure 7. PHPExcel Formatting

Figure 8 demonstrates the process of inserting values into spreadsheet cells. First, the data is fetched from the database using a combination of a predefined query combined with user specified input values, which is then used to return a row of results. Each result from the database correlates to one row in the resulting spreadsheet and, as such, is given

```
while($row = pg_fetch_array($subjects)) {
    $xlsrow++;
    $xlscol = 'A';
    for($i = 0; $i < pg_num_fields(\$subjects); $i++) {
      if($i >= 4 && $i <= 6) {
        $objPHPExcel->setActiveSheetIndex($w)
          ->setCellValue($xlscol++.$xlsrow, $row[\$i]);
      } else {
        $objPHPExcel->setActiveSheetIndex($w)
          ->setCellValue($xlscol++.$xlsrow, $row[$i]);
      }
    }
  }
  $objPHPExcel->getActiveSheet()->getStyle("A$anchorrow:L$xlsrow")->
    applyFromArray($boxstyle);
```

Figure 8. PHPExcel Insertion

a single row in the resulting file. Each value from the result is then written to the file, cell by cell for each result row. As spreadsheet files are commonly mapped by row and column, it is possible to set up an iterator that moves through each record cell by cell fairly easily. The loop in the figure moves row by row through the results in the while loop and the for loop moves the cursor cell by cell to insert each value into the file. The final line of code illustrates the process of applying a predefined style to a row in the spreadsheet by specifying the area to apply the style to. This style rule is used to define the header row of the resulting file to increase readability of the resulting file. The overall running time for the script is minimal in practice taking only a few seconds for very large datasets. It does not appear to impact the usability of the web portal to generate the file on demand and is sufficient for the purposes of the portal.

Data Analysis and Query Integration

Creating a seamless, highly cohesive web portal was a major driving force in designing the web portal. With the development of the query engine and the data analysis engine, covering the subject by subject presentation of data and a more aggregate view of the subjects divided up by more generalized factors respectively, the main goals of data presentation were achieved. Upon further examination, the individual rows presented in the data analysis engine relate to queries that can be run in the query engine and present an avenue for integrating the data analysis and query engines into a unified approach that allows users the ability to perform an aggregate search in the data analysis engine and then drill down those results by analyzing each row as an independent search in the query engine. By harnessing the power of each engine to perform different parts of a unified search, the portal is able to deliver aggregate results that may yield interesting results while also providing users with the ability to view the individual subjects that comprise a grouping in the data analysis engine for maximum information presentation.

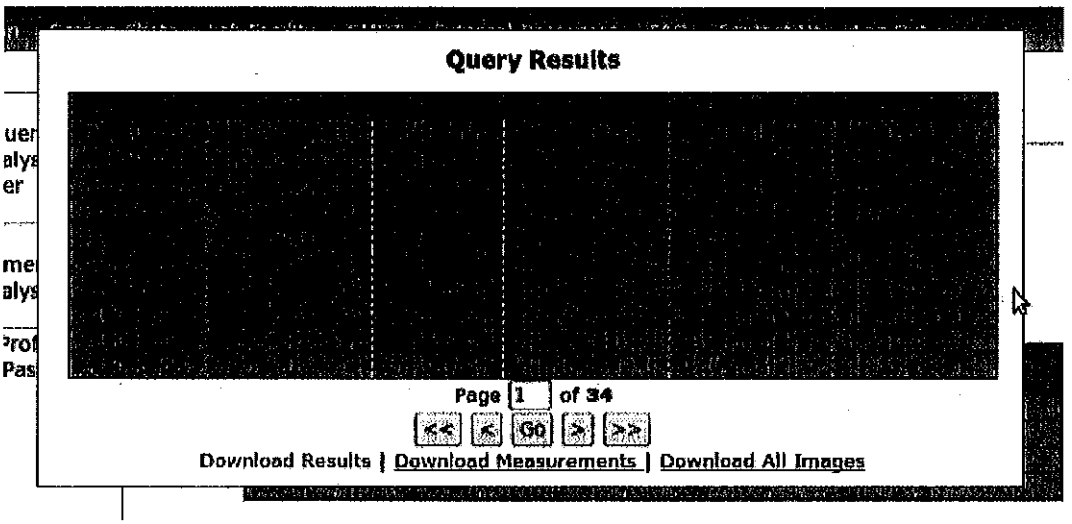

Figure 9. Query Table Integrated into the Data Analysis Engine

The integration between the data analysis engine and the query engine is handled in two main ways. The first type of integration involves displaying the individual subjects in a data analysis result row through the use of a modal window to display a query engine style results table. Figure 9 illustrates the modal window showing the result set of an individual row in the data analysis engine. The second type of integration involves copying the relevant constraints from the data analysis engine over to a query in the query engine for more in-depth analysis of the result set. By approaching the integration in a twofold manner, a user is able to decide whether they just want to quickly browse the subjects that compose a group or whether they wish to perform further analysis of an individual group by moving it to the query page.

Integration Details

Due to the shared nature of the data between the data analysis engine and the query engine, integration between the two mainly consisted of manipulating existing SQL queries and adding relevant code to the web portal in order to facilitate the proper transfer of data

between the two engines. The major changes are listed and described in the following sections while some of the more minor implementation details are left out as they are specific to the web portal and not major contributors in the overall understanding of the integration.

Query Engine Integration

Integration into the query engine required minor modifications to be made to the session array holding the query constraints as well as a few modifications and additions to the source code to allow for query tables to be generated apart from the constraints table found on the query engine page. This change allowed for the query results for a row in the data analysis table to be displayed inside of a modal window popup on the data analysis page itself. Modifications were necessary as the original designs of the two engines did not take into account the possibility of integrating the two engine's constraints. The following section details the changes necessary to make integration possible.

Figure 10 illustrates the modification to the query engine session array that allows for the integration of the query and data analysis engines. The addition of the data analysis constraints allows the extra information available to add as constraints in the data analysis engine to be incorporated into queries in the query engine. Modifications to the back-end code associated with the query engine were minimal as only a few lines of code were added to check if the query was from the data analysis engine by checking a POST HTML variable and the one element in the constraint session array.

Integration on the query engine side was completed by the addition of code to generate query engine tables apart from the constraints table and other components of the query page. An additional class was created to call only the methods necessary from the query engine class that were required to generate the query table. The resulting table is then used as a modal window on the data analysis page when a user clicks on a specific row.

```
Query Engine Constraints = Array(   [0]=Lower-Age,
                                    [1]=Operation-Age,
                                    [2]=Upper-Age,
                                    [3]=Lower-BMI,
                                    [4]=Operation-BMI,
                                    ⋮
                                    [21]=Data Analysis Constraints )
```

Figure 10. Query Engine Constraints with Integration Constraints

Data Analysis Integration

Completing the integration of the two engines on the data analysis side involved two main changes. The first required change of the engine was to provide a JavaScript method call in the HTML code for each row of the result set in the data analysis engine to provide a way of creating the query table in a modal window. The second change required creating a way to separate the extra query constraints available in the data analysis engine from the rest of the query so that these query constraints could then be appended to the constraints available in the query engine. These two changes created the ability to integrate the two engines into a unified presentation that provided a seamless interaction between the two engines to the user.

A JavaScript function and AJAX functions using the jQuery AJAX library were necessary to perform the required table construction and information presentation required to provide a modal window in which to display the query table to the user. Figure 11 shows the functions necessary to generate the query table for the modal window and to display the modal window. The function first clears anything that may be in the HTML div element containing the query table. It then makes a call to the PHP script ajaxdaqueryresults with the necessary data from the data analysis engine to build the query. The script then returns the HTML markup necessary to display the query table. This markup is then appended to

the division element. Finally, a call to the .blockUI package in jQuery displays the division as a modal window to the user, similar to what is seen in Figure 9.

```
function daredirect(primg, primlow, primhigh, secg, seclow, sechigh,
  tertg, tertlow, terthigh, gender) {
  $('#querytablediv').empty();
  $.ajax({
    type: "POST",
    url: "processing/ajaxdaqueryresults.php",
    cache: false,
    data : {primg: primg, primlow: primlow, primhigh: primhigh,
        secg: secg, seclow: seclow, sechigh: sechigh,
        tertg: tertg, tertlow: tertlow, terthigh: terthigh,
        gender: gender},
    success: function(data) {
      if (data != "") {
        $('#querytablediv').append(data);
      }
    }
  });
  $.blockUI({
    css: {
      border: '1px solid #000',
      top: '15%',
      left: '24%',
      width: '52%',
      padding: '10px',
    },
    title: 'Query Results',
    message: $('#querytablediv'),
    overlayCSS: {
      backgroundColor: '#000',
      opacity: 0.6
    }
  });
  $('.blockOverlay').attr('title','Click to unblock').click(
      unblockUI);
}
```

Figure 11. JavaScript for Data Analysis Redirect

Separating the query constraints of the data analysis engine from the actual query was necessary in order to use the added constraints in the query engine as the data analysis engine has more options for constraining a search than the query engine does. This was accomplished by evaluating the constraints based on their relevant target column and divided

according to whether or not they match up with query engine constraints. The common attributes between the two engines, such as: age, weight, height, gender and BMI, are placed into the relevant sections of the constraint array held in a PHP session variable and illustrated in Figure 10. The additional constraints not available in the query engine are added into an extra field in the session array for constraints which are then appended to a query produced by the query engine in order to generate the results from a data analysis query. Appending the additional constraints in lieu of creating more fields for each constraint was an acceptable compromise between giving the user more control and reducing the amount of constraints available for normal queries.

Integration Conclusion

Integrating the two engines was only natural and helped to create a seamless appearance between the aggregate data analysis engine and the more focused query engine rather than two distinct, separate engines. The integration allows a user to approach a data analysis process with both an expectation of aggregate results combined with the option to drill down into each grouping and get the subject-by-subject results in order to fully understand the available data. Overall integration of the two engines was accomplished with a moderate amount of new code while also maintaining the ability to use the two engines separately if desired by the user.

CHAPTER VI

WEB TECHNOLOGY INTEGRATION

Implementation of current web technologies is important in any website in order to create an engaging user experience and present information in a manner that is consistent with other current websites. As development on the web portal progressed, different web technologies were implemented to improve both the front-facing usability of the web portal as well as the back-end information design and presentation. The different technologies implemented in the web portal include: Java graphing applets, AJAX implementation of the query and data analysis engines, and a pseudo-framework of the file system in order to improve the file structure of the web portal.

AJAX Implementation

Initial design of the web portal relied on full page refreshes. As development of the portal continued, the two engines lent themselves well to implementations using AJAX, Asynchronous JavaScript and XML, to only refresh parts of a page that required a refresh to update specific components. Sections of different pages such as a query results table or the modal windows in both engines were good candidates for implementing AJAX due to the features offered by the components. AJAX has been implemented in the web portal mostly through the jQuery JavaScript library [23] due to the simplification of AJAX interactions as well as the feature set provided by the library.

AJAX calls to the server have mostly been implemented by the JavaScript event handler by performing the basic process of: identifying the element clicked, handling the event as specified by the event handler loaded with the initial page load, making the relevant AJAX calls in the code for the event, then finally loading the returned information. The code in Figure 12 illustrates the general format of a JavaScript event handler coupled with AJAX calls to get information from the server.

```
$(document).ready(function() {
  $('#rightcolumn').click(function(e) {
    var target = e ? e.target: window.event.srcElement;
    switch(target.id.toLowerCase()) {
      case 'query':
        e.preventDefault();
        if(validateQueryInput() == true) {
          var curConstr = getConstraintValues();
          if(curConstr[18] !== 0) {
            $.ajax({
              type: "POST", async: false,
              url: "processing/ajaxqueryconstraintspost.php",
              cache: false,
              data : {curConstr : curConstr.toString()},
            });
          }
          //Execute the query.
          var html = "";
          var pageNum = 1;
          if(document.getElementById("pageNum")) {
            pageNum = document.getElementById("pageNum").value;
          }
          $.ajax({
            type: "POST", async: false,
            url: "processing/ajaxquery.php", cache: false,
            data : {query: target.value,
              pageNum: pageNum
            },
            success: function(data) {
              $('#querytableshell').empty();
              $('#querytableshell').append(data);
            }
          });
          document.getElementById("savequery").style.visibility="
              visible";
        }
      break;
    }
  });
```

Figure 12. Javscript/AJAX Example Process

Query Engine AJAX

The query engine design was very well suited to the use of AJAX calls to improve the user experience as the engine uses one page with no layout changes and most operations involve modifying small components of the display. Through the use of AJAX, most

operations that the query engine provides have eliminated the need for a full page refresh, presenting the user with a seamless, almost desktop application feel when using the engine. Most notably, the operations involving modifying the table such as: changing the page, sorting columns, and adding constraints have been implemented using AJAX calls.

The bulk of the query engine functions through the use of JavaScript click events and AJAX calls in the event handlers for each. The querying operation has been converted to an event with AJAX calls and is listed in Figure 12. The general process of the querying operation involves the document ready function first detecting the source of the click which happens in lines 2 and 3 of the referenced code listing in the figure. The identity of the click is then used in a switch statement to locate the relevant block of code for handling the event which matches at line 5 of the code listing in the referenced figure. The default operation for the query button is then prevented through a call in line 6 before moving on to processing the actual query information. Validation is performed using JavaScript function calls to make sure that the information entered by the user is correct and acceptable for the query engine to proceed.

Once validation is successful, the constraint values are gathered before the first AJAX call is made in lines 13 through 21. The call is made to process the constraint information at the url specified as an attribute of the call, in this case "processing/ajax-queryconstraintspost.php". This call is specifically made with the asynchronous feature of AJAX turned off through the use of the "async: false" attribute in line 15 as the processing of the constraints must be completed before the actual query is performed as the constraints are an integral part of the query. The next AJAX call on lines 31 through 56 are for the actual query operation and output the result of the call to the browser. The call is made and then the success attribute on lines 50 through 55 handle the returned information from the server. This function takes in the HTML to output to the screen, clears the div element that holds the query table and then uses the .append() method to place the HTML into the div

element finishing the process of creating the query table by making it visible to the user while at the same time avoiding a full page refresh that would be required if AJAX and JavaScript were not used.

Other events on the query page that use the JavaScript event handler and AJAX calls are handled in a similar manner. Operations such as viewing the next page of results in the table, sorting the table, and adding constraints involve much simpler calls without the need for the extra information required in the case of the query click event. The code listed in Figure 12 lines 61 through 63 demonstrate a simple call to handle the next page click event. In lieu of using the robust .ajax() call as was necessary in the query event handler, a call to the .load() method simply loads the returned code from a call to the "processing/ajaxquery.php" script into the "querytablenest" div element. Most of the AJAX operations can be handled as concisely as this event and do not require the extra information necessary for a complete .ajax() call.

A unique problem was encountered during implementation of the AJAX and JavaScript event handlers on the query page that is not discussed much in the introductory tutorials and literature pertaining to the AJAX calls and JavaScript events. The Document Object Model[5], D.O.M., used by most modern web browsers implements a tree structure of all elements in a page with the parent elements being elements that contain a target element. AJAX calls and insertion of HTML into a previously loaded page do not automatically add new elements to the D.O.M. and can cause problems when using the elements id attribute as handlers for JavaScript events. Instead, JavaScript allows a programmer to program an event handler by using a parent element and then deciding on the target of the event from the children of the parent element. This can be seen in Figure 12 on line 2 where the parent element, "rightcolumn", is used to identify all events that happen in the query engine by then finding the name of the element triggering the event on line 3 by making a call to the srcElement property. This problem is easily rectified by implementing event handlers

in this way and also makes the design somewhat future-proof as new functionality can be added to the handler without having to check to make sure all AJAX added sections of code are not affected by a new change.

Data Analysis Engine AJAX

The data analysis engine relies less heavily on AJAX and JavaScript, when compared to the query engine, due to the nature of the process to create the analysis table as well as the feature set offered by the engine. The main function of AJAX and JavaScript in the data analysis engine is to perform the loading of the query table to present a query engine style table of the subjects in a result row from the data analysis engine. This process is similar to the process of creating a query table as discussed in the previous section and uses similar calls.

Framework

Initial design of the web portal did not include implementing the portal within the confines of a development framework, which led to a lack of a well defined file hierarchy. One reason for the lack of a well defined file hierarchy was the fluid design of the web portal as new ideas and technologies were implemented in the lab. Changes were also made to the design of the web portal which kept the overall design in a fluid state and made it hard to commit to a concrete design strategy. As the laboratory and the web portal became more defined, the feature set was better specified which allowed for a revision to the file hierarchy. In order to improve the navigability of the web portal's folder hierarchy as well as to limit the overall clutter in the main folder, a pseudo-framework was implemented to help make the web portal's folders more intuitive.

Due to the complexity and time required to re-implement the web portal in a common web framework, it was decided that slight revisions to the file hierarchy would be the

best course of action. A complete overhaul of the existing file structure of the web portal just did not make sense from a labor and organizational standpoint. The new file structure attempts to implement a pseudo-framework that organizes the different files into a usable, well organized hierarchy to aid in code revision and ease of locating the different files by their purpose. Whereas the initial design left the bulk of the code in the main folder with a few folders such as: pages, images, includes, etc, the current structure divides the files up into a more logical file structure. Folders such as: pages, images, CSS, includes, processing, applets, etc have been created to partition the different files. The division of the files into logical folders presents a more intuitive layout which makes locating files easier and indirectly makes the coding process more efficient.

The basic premise for the new file hierarchy involved dividing up the files by their basic function into a form that is similar to the popular Model-View-Controller architecture. Files concerned with the HTML presentation of the different pages were placed into the pages folder. These files contain the public facing part of the website and involves the basic code that forms the underlying structure of the pages. In a traditional model-view-controller architecture(MVC) this would be considered the view section. The back-end code that generates the dynamic content of these pages is located in the processing folder, which also holds other files related to processing dynamic content for the AJAX interaction parts of the site as well as a few other files for various other processing tasks that are required to make the site function. This folder would be analogous to a controller folder in the traditional MVC architecture. The current design does not abstract database interactions into their own folder as the original files handle database interactions and there is no plan to migrate the database to another vendor. Due to the amount of time required to move the current database interactions into another separate class, it was decided that this would be an acceptable compromise to continue developing the database interactivity in this way. Functionality is not impacted and is transparent to the user.

In addition to the main folders mentioned previously, there are also a few folders that sit outside the realm of rendering pages. As the traditional MVC folders deal with the structure and content of a page, there are a variety of helper classes and methods that are left outside the bounds of either a view or controller folder. These methods deal with various tasks in the web portal relating to: user saved queries, various scripts, files included that provide utilities, images, CSS files, and external packages that have been used to implement features in the web portal. In a traditional MVC approach, these files would lay outside of the actual MVC folders and are normally handled as accessory folders and placed somewhere inside of a framework.

When implementing the pseudo-framework, the decision was made to attempt to categorize these files by the respective tasks and divide them up accordingly. All CSS files for the website were then moved into the CSS folder. Similarly, all images have been placed in the images folder and external packages have been placed in an individual folder for each package in order to make maintenance more streamlined. Files that deal with methods called from include statements were placed in a folder named "includes" as these are called by multiple different files using a PHP includes statement. The includes files deal with header files, cookies and errors that are common to all of the different pages in the web portal. The final folder that was created to fit the new hierarchy was the scripts folder that holds the various JavaScript files relating to the functioning of JavaScript on the site.

Future improvements to the file hierarchy and pseudo-framework could involve abstracting database interactions into their own class in order to simplify the code and increase code portability. If there is not a need to change the vendor of the database, this may or may not be useful depending on the amount of database interactions that are needed in future implementations of the web portal. The folders could also be reexamined at a later time and the file hierarchy could be further refined to implement even more divisions between

tasks such as further refining the scripts folder or the processing folder. At present, the current file system is sufficient for the needs of the web portal.

Web Technology Integration Conclusion

The various web technologies and design strategies detailed in the previous sections were implemented in the web portal in order to increase usability for day to day users as well as to improve the back-end design structure in order to make future code revisions and additions more efficient from a design standpoint. The use of AJAX and JavaScript aid the user by providing a seamless, desktop style feel to the application while reducing load times when operations are performed. The improvements to the back-end code structure of the web portal have created a structure that better classifies the different files while attempting to implement a pseudo-framework to make future revision easier to implement. While a framework would be ideal for the portal, implementation of one is currently not feasible due to the time required to complete the change.

CHAPTER VII

CONCLUSION

The overall goal of the web portal is to provide an interface that aids in the presentation and dissemination of information collected by the body scanning laboratory to an audience of academic and industry professionals in order to improve current research and products dealing with the thermal properties of the human torso. In this thesis, the main point of emphasis on the development of the web portal was the presentation and delivery of information to an audience of academic and industry professionals. This aspect was addressed through the use of structured query engines and various forms of data presentation, the web portal is able to deliver the information in a concise, useful manner. Information gathered by the laboratory is successfully delivered in aggregate form by the data analysis engine and on a subject-by-subject basis by the query engine. The two engines work in tandem to allow for aggregate results to be further investigated by subject to allow a user to evaluate the data from a perspective that will aid in analyzing the data as the user sees fit. Secondary goals for the presentation and delivery of information revolve around database design, efficient and concise delivery of information through the web portal and information security from presentation to delivery.

In order to satisfy both the primary and secondary goals of the web portal, a variety of design techniques were employed to create a web portal that delivered information in a straightforward manner while also providing basic information security to prevent unauthorized users from gaining access to the information. The basic information presentation engines described earlier were placed inside of a secure area of the web portal that requires a login ID that is first verified by a member of the laboratory. Once confirmed, a user is granted access to the data and they are free to then use the various engines and data presentation aids to investigate the data set. The information available to users is presented in two forms, aggregate and individual using the two engines and methods have been provided for

a user to download the data sets for further investigation using third-part statistical analysis tools.

Coding of the web portal was completed using a variety of web programming languages such as: PHP, JavaScript, AJAX and SQL. The web pages themselves and most of the back-end code is written in PHP and all database interaction occurs using PHP and the native database calls available through the PostgreSQL library. JavaScript and AJAX were employed heavily on the query and data analysis engine pages in order to present the information in seamless, desktop application style manner that minimizes full page reloads, thus improving the user's experience. The file structure of the web portal was modified during the development in order to adhere to a more structured, logical format that should allow future system improvements to be completed in an efficient manner by reducing the time it takes to locate various files and functionalities.

Future Work

While the web portal is currently stable and satisfies the basic goals initially defined prior to the undertaking of the project, there are a few areas of improvement that could be worked on in future iterations of the portal. First and foremost, the graphing functionality currently provided could be better integrated into the query engine to allow for a more seamless transition between the two pages. The graphing functionality was implemented before the use of AJAX in the web portal and these could possibly now be integrated as modal windows on the query page itself. They could also be further refined to present the information in a more polished format with some further work on the graphical user interfaces. The overall appearance of the web portal could also be improved to bring it more in line with current web development styles but this is an expected maintenance task for any website and should be done periodically.

REFERENCES

[1] Maarten Balliauw, Mark Baker, and Erik Tilt. Phpexcel. http://phpexcel.codeplex.com/, October 2010.

[2] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[3] Vince Bruno, Audrey Tam, and James Thom. Characteristics of web applications that affect usability: a review. In *OZCHI '05: Proceedings of the 17th Australia conference on Computer-Human Interaction*, pages 1–4, Narrabundah, Australia, Australia, 2005. Computer-Human Interaction Special Interest Group (CHISIG) of Australia.

[4] Kevin Chen-Chuan Chang, Bin He, Chengkai Li, Mitesh Patel, and Zhen Zhang. Structured databases on the web: observations and implications. *SIGMOD Rec.*, 33(3):61–70, 2004.

[5] Nicholas Chase. Javascript and the document object model. IBM developerWorks: Technical Library, July 2002.

[6] Angelo Ciampa, Corrado Aaron Visaggio, and Massimiliano Di Penta. A heuristic-based approach for detecting sql-injection vulnerabilities in web applications. In *SESS '10: Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, pages 43–49, New York, NY, USA, 2010. ACM.

[7] Oscar Corcho, Angel López-Cima, and Asunción Gómez-Pérez. A platform for the development of semantic web portals. In *Proceedings of the 6th international conference on Web engineering*, ICWE '06, pages 145–152, New York, NY, USA, 2006. ACM.

[8] Bernardo Damele and Miroslav Stampar. Sqlmap: automatic sql injection and database takeover tool. World Wide Web, March 2010.

[9] Inc. Free Software Foundation. Gnu lesser general public license v3.0. http://www.gnu.org/licenses/lgpl.html, June 2007.

[10] Daniel M. German. A study of the contributors of postgresql. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 163–164, New York, NY, USA, 2006. ACM.

[11] William G. J. Halfond and Alessandro Orso. Preventing sql injection attacks using amnesia. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 795–798, New York, NY, USA, 2006. ACM.

[12] William G. J. Halfond, Alessandro Orso, and Panagiotis Manolios. Using positive tainting and syntax-aware evaluation to counter sql injection attacks. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 175–185, New York, NY, USA, 2006. ACM.

[13] Sverre H. Huseby. Common security problems in the code of dynamic web applications, June 2005.

[14] Nicolas Juillerat. Enforcing code security in database web applications using libraries and object models. In *Proceedings of the 2007 Symposium on Library-Centric Software Design*, LCSD '07, pages 31–41, New York, NY, USA, 2007. ACM.

[15] Konstantinos Kemalis and Theodores Tzouramanis. Sql-ids: a specification-based approach for sql-injection detection. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 2153–2158, New York, NY, USA, 2008. ACM.

[16] OWASP: Open Web Application Security Project. Owasp top 10 for 2010. World Wide Web, April 2010.

[17] Michalis Petropoulos, Yannis Papakonstantinou, and Vasilis Vassalos. Graphical query interfaces for semistructured data: the qursed system. *ACM Trans. Internet Technol.*, 5(2):390–438, 2005.

[18] PostgreSQL Global Development Group. License. http://www.postgresql.org/about/licence, September 2010.

[19] PostgreSQL Global Development Group. Postgresql developer faq. World Wide Web, August 2010.

[20] PostgreSQL Global Development Group. Postgresql faq. http://wiki.postgresql.org/wiki/FAQ, August 2010.

[21] Frank S. Rietta. Application layer intrusion detection for sql injection. In *Proceedings of the 44th annual Southeast regional conference*, ACM-SE 44, pages 531–536, New York, NY, USA, 2006. ACM.

[22] Alex Roichman and Ehud Gudes. Fine-grained access control to web databases. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, SACMAT '07, pages 31–40, New York, NY, USA, 2007. ACM.

[23] The jQuery Project. jquery project. http://jquery.org/, October 2010.

[24] The PHP Group. Mdb2. http://pear.php.net/package/MDB2.

[25] The PHP Group. Php: Php usage statistics. http://php.net/usage.php, July 2007.

[26] The PHP Group. Php: History of php. http://us3.php.net/manual/en/history.php.php, August 2010.

[27] The PHP Group. Php: License information. http://www.php.net/license/, August 2010.

[28] The PHP Group. Php: Php usage statistics. http://www.php.net/manual/en/refs.database.php, August 2010.

[29] Sudharshan S. Vazhkudai, James A. Kohl, and Jens Schwidder. A java-based science portal for neutron scattering experiments. In *PPPJ '07: Proceedings of the 5th international symposium on Principles and practice of programming in Java*, pages 21–30, New York, NY, USA, 2007. ACM.

[30] Gary Wassermann and Zhendong Su. Static detection of cross-site scripting vulnerabilities. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 171–180, New York, NY, USA, 2008. ACM.