Diabetes Research Vortex:
A Novel Information Management System for Type I Diabetes

by

Kia Javanmardian

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical [Computer] Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 12, 2005

Copyright 2005 Kia Javanmardian. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Kia Javanmardian

Department of Electrical Engineering and Computer Science

May 12, 2005

Certified by

Douglas A. Lauffenburger
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Theses
Table of Contents

Table of Contents................................................................................. 2
Abbreviations........................................................................................ 5
Abstract.................................................................................................. 6

Introduction............................................................................................ 7
1 Biological Background........................................................................ 9
   1.1 Type I Diabetes Background................................................... 9
   1.2 Genetics Background............................................................ 9
   1.3 Role of Genetics in Type I Diabetes....................................... 11
   1.4 Biological Motivation for DRV........................................... 12
2 System Features.................................................................................. 13
   2.1 Administrator Tasks............................................................... 14
      2.1.1 Strain Management......................................................... 15
      2.1.2 Idd Management............................................................. 17
      2.1.3 Gene Management.......................................................... 19
      2.1.4 Keyword Management.................................................. 20
      2.1.5 Literature Data Mining................................................... 20
      2.1.6 Administrator Task Summary........................................... 24
   2.2 User Tasks.................................................................................. 24
      2.2.1 Strain Search................................................................. 25
      2.2.2 Idd Search................................................................. 26
      2.2.3 Gene Search................................................................. 27
      2.2.4 Keyword Search.......................................................... 27
      2.2.5 Microarray Search....................................................... 27
      2.2.6 Uploading Documents.................................................. 27
      2.2.7 Uploading Microarray Data.......................................... 28
      2.2.8 User Task Summary..................................................... 28
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNA</td>
<td>Deoxyribose Nucleic Acid</td>
</tr>
<tr>
<td>DRV</td>
<td>Diabetes Research Vortex</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>Idd</td>
<td>Insulin Dependent Diabetes (Region)</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Page</td>
</tr>
<tr>
<td>mRNA</td>
<td>Messenger Ribose Nucleic Acid</td>
</tr>
<tr>
<td>MIAME</td>
<td>Minimum Information About Microarray Experiment</td>
</tr>
<tr>
<td>RNA</td>
<td>Ribose Nucleic Acid</td>
</tr>
<tr>
<td>T1D</td>
<td>Type One Diabetes</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Diabetes Research Vortex: 
A Novel Information Management System for Type One Diabetes 
by 
Kia Javanmardian 

Submitted to the 
Department of Electrical Engineering and Computer Science 

May 12, 2005 

In Partial Fulfillment of the Requirements for the Degree of 
Bachelor of Science in Computer [Electrical] Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science 

ABSTRACT 

Information management is becoming a necessary task for modern research laboratories. As scientific research within a particular domain progresses, the amount of data and publications in the domain increases drastically. Type one diabetes (T1D) research is no exception. The DRV (Diabetes Research Vortex) addresses the information management challenge. 

The DRV’s main objective is to provide a web accessible information management system for diabetes related publications and experimental data. The DRV was inspired by current T1D research paradigms. A user can choose to upload a publication or experimental data under one of four research categories: a strain, gene, Idd (Insulin Dependent Diabetic) genetic region, or custom keyword. 

The effectiveness of the DRV lies in the biological concepts it encapsulates. The DRV allows a user to view the system’s contents by browsing a strain, gene, Idd, or custom keyword. Furthermore, the DRV allows a user to seamlessly browse among related strains, genes, and Idd’s. The DRV links information between each of these domains and allows a user to intuitively acquire knowledge about a specific interest. 

The system can also be asked to mine publications from PubMed based the terms stored in the four aforementioned categories. Furthermore, the system complies with the MIAME standard (Minimum Information About Microarray Experiment). Therefore, the system allows a systematic way to upload and retrieve microarray data. 

The DRV presents a starting point for a T1D information management system. The concepts governing the overall design of the DRV should also be relevant and applicable to many other research fields. Therefore, the potential impact of the DRV may stretch beyond the domain of T1D.
Introduction

As Type-1 diabetes (T1D) research progresses, the publications and experimental data pool pertaining to the disease grows dramatically. Thus, the need to effectively organize the vast amount of publications and experimental data pertaining to the disease becomes a key concern of research laboratories. A knowledge management system that can effectively organize publications and experimental data will save researchers time and enable them to effectively organize and manage the information relevant to their research. The Diabetes Research Vortex (DRV) is an attempt to solve the information management challenge confronting T1D researchers. Furthermore, the DRV incorporates many ideas that may be applicable to information management systems serving other research areas.

DRV Goals

The DRV differs from a general database in the way that it is structured. The DRV uses biological concepts to effectively store and manage publications and experimental data. The biological concepts used by the DRV are directly derived from three main focal points of T1D research: Idd (Insulin Dependent Diabetes) regions, mouse strains, and genes. By leveraging the biological relationships between these different categories, the DRV is able to link different categories of publications and experiments. Thus, the DRV tackles the problem of information management by categorizing and linking different publications and experiments into one of the three aforementioned groups.

The DRV facilitates retrieval of publications indirectly related to a research interest. While a user’s specific interests may lie with an Idd, the system will enable the user to easily view information under the strain or gene category related to the Idd.

The DRV accomplishes this task through by maintaining links between different Idd’s, strains, and genes.

The DRV also allows users to upload microarray data adhering to the MIAME (Minimum Information About Microarray Experiment) standard.
experiments are quickly being utilized throughout biological research; therefore, storing microarray data became one of the DRV’s priorities.

Lastly, the DRV is able to automatically mine PubMed for publications relevant to the T1D research domain. The DRV accomplishes this by implementing a database dependent, standalone mining application that allows the user to mine for specific categories of knowledge pertinent to T1D.

Document Outline

The biological background related to the DRV is outlined in the first chapter. The first chapter highlights the biological knowledge that inspired the overall organization of the DRV. The second chapter discusses the different tasks supported by the DRV in an attempt to illustrate its usefulness and effectiveness in managing T1D publications and experiments.

Chapter 3 begins the discussion regarding the overall design of the DRV. The DRV has three major components contributing to the overall functionality of the system: the JSP’s (Java Server Pages), the Tomcat JSP engine/web server, and the MySQL relational database. The first two components are involved in the interface (front-end) portion of the DRV, while the database is responsible for the organization and maintenance of the DRV’s information.

Chapters 4, 5, and 6 provide a detailed outline of the database, Tomcat server, and the use of the Java programming language, respectively. Chapter 7 details the JSP hierarchy of the system. Chapter 8 outlines the mining application of the DRV, and Chapter 9 is a discussion regarding the MIAME standard.
Chapter 1

Biological Background

This section provides the biological background that motivates the structure of the DRV. The genetics of type one diabetes is discussed as well as current areas of research focused around the disease. In doing so, this section will give a biological basis for the design and development of the DRV.

1.1 Type I Diabetes Background

Type I diabetes is an autoimmune disease which leads to the destruction of the Beta Cells (insulin producing cells) located in the pancreas’s Islets of Langerhans. It leads to the body’s eventual inability to produce insulin. This forces affected individuals to depend on insulin injections for survival [1]. Currently, type one diabetes effects on the order of 100 million people worldwide, and the rate of incidence is increasing [1]. Everyone affected is forced to monitor their condition constantly and adhere to a strict diet and lifestyle. Neglecting to do so can result in blindness, brain damage, and death [1].

1.2 Genetics Background

In order to understand the complex nature of T1D, one must first have a basic understanding of genetics. The explanation presented in this section will be fairly brief,
but it should provide a basis for understanding the nature of the disease. As a result, this chapter will lay out the biological factors that motivated the current design of the DRV.

DNA (deoxyribose nucleic acid) is central to understanding genetics. DNA is a double stranded compound composed of distinct sequences of four different bases: adenine, guanine, cytosine, and thymine. Together, these four bases encode the different genes of an organism. DNA encodes different genes, which express proteins and regulate the different processes throughout an entire organism. An organism’s brain structure, metabolism, and overall shape are all governed by the proteins encoded by its different genes. The human genome (collection of genes) contains approximately 25,000 distinct genes and is composed of approximately three billion base pairs.

Every cell of an organism contains its entire genome. In different cell types, however, only certain genes are expressed. A cheek cell, for example, will express different genes than a lung cell. The difference in expression is due, in no small part, to the different function each of the cells has within an organism.

The way in which genes regulate different processes is quite complex. Gene expression in a particular cell type is a three step process. First, the cell must transcribe the DNA encoding a particular gene. The result of transcription is mRNA (messenger ribose nucleic acid). The mRNA is a sequence of base pairs, similar to DNA, encoding the synthesis of a specific protein. The mRNA sequence is transported to an organelle within the cell called a ribosome. A ribosome will take the mRNA sequence and construct a protein based on the the mRNA sequence (derived from the DNA sequence). Thus, gene expression is simply prescribed protein synthesis. Together, different proteins regulate he behavior of a cell, and different cells regulate the behavior of an entire organism. Figure 1.1 shows a hierarchy—DNA leads to protein expression—protein expression leads to cell behavior—cell behavior leads to the behavior of an entire organism.
Differences in DNA account for much of the differences between individuals. For example, two different individuals may have two different sequences of DNA along analogous parts of their genome. These differences account for different versions of a specific gene and can lead to different protein expression. Furthermore, different protein expression may lead to different physiological behavior. In a very simple sense, differences in DNA can lead to different protein expression and the presence or absence of a particular disease. This is thought to be the case with T1D.

1.3 The Role of Genetics in Type One Diabetes.

Type I diabetes is an inherited disease [2]. Significant T1D research centers upon regions of DNA referred to as Idd (Insulin Dependent Diabetes) regions. [3] These regions represent sequences of DNA which are thought to influence the incidence of the disease within an individual. The genes present in the Idd regions are thought to make people more or less susceptible to T1D. Currently, there are around 20 different Idd’s being investigated.

An Idd is present if a pre-specified version of DNA is present within a particular region of an organism’s genome. To say an Idd is not present does not mean an analogous sequence of DNA is not present. It merely means that a specific version of the sequence, defined by Idd 1, is not present. Practically, an Idd can be thought of as a specific version of genetic material along a given interval of an organism’s genome.
In research labs, scientists focus on different strains of mice to better understand Idd regions. A strain is defined by the Idd regions that it contains, and different Idd’s contain different genes. Therefore, different strains exhibit different rates of diabetes incidence.

1.4 Biological Motivation for the DRV

The DRV is partitioned into three domains: genes, Idd’s, and strains. Genes are the most basic components. A single gene encodes for the production of a single or multiple proteins (through alternate splicing). An Idd represents a specific version of DNA along a particular interval. Idd regions partially or entirely encode different genes. The different version of genes an organism has determines many of its physiological traits. In the case of type one diabetes, different genes can make a person more or less susceptible to disease incidence. Finally, the different combinations of Idd’s define a particular strain of mouse. Different strains, due to their differing genetic matter, display varying incidence of diabetes.

The relationship between strains, Idds, and genes is complex. There roughly twenty Idd’s meaning \(2^{20}\) possible strains of mice. Each of these Idd’s can contain the code for many different genes, adding further complexity to type one diabetes research. This leads to a potential deluge pertaining to T1D research. The DRV’s organizational approach allows it to address this challenge. By leveraging the relationship between strains, Idd’s, and genes, the DRV is able to cope with the complex nature of type one diabetes research. The DRV uses biologically motivated relationships in order to effectively organize the complex web of information pertaining to the disease. The manner in which the DRV accomplishes this goal is the focus of the document’s subsequent sections.
Chapter 2
System Features

This section focuses on the capabilities of the DRV through the set of tasks that it supports. The overall goal of the DRV is to provide researchers a way to effectively store and manage research publications and experimental data. In order to do this, the DRV must be able to store these resources in a biologically intuitive manner. The DRV accomplishes this by basing its organization on biologically motivated concepts. The concepts of strain, Idd, and gene (discussed in Chapter 1) provide the conceptual backbone for the DRV. By outlining the tasks supported by the system, this section will illustrate the overall usefulness of the DRV to researchers. The section will also highlight the DRV's biological influences.

Before delving into specific tasks, it is important to fully grasp the way in which strains, Idd’s, and genes are related. Chapter 1 outlines the relationship, but a typical user scenario further illustrates the point. When a biologist is conducting an experiment on type-one diabetes, the resolution of the primary focus may vary. The broadest scope involves interest in a particular strain of mouse. If a user is interested in a particular
strain of mouse, he should ideally be able to retrieve information relevant to that strain from our system. A user may be interested in a particular Idd rather than strain. He too should be able to go into our system and retrieve information pertinent to the Idd of interest. Finally, a user may be interested in a specific gene, and our system should be able to relay information relevant to his interest.

Beyond being able to serve information between strains, Idd’s, and genes, the DRV should capture the interrelation between the three concepts. If for example a user is interested in a particular strain, he may become interested in the Idd’s related to that strain. Also, a user interested in a particular Idd may become interested in the genes related to that Idd. The DRV should allow a user to efficiently retrieval information between different strains, Idd’s, or genes. In essence, the DRV should connect each of three domains and accommodate seamless information retrieval across them.

The ability to connect strains, Idd’s, and genes makes this system extraordinary. By leveraging biological knowledge, the system breaks out of the mold of a simple database. It uses biology to motivate its organization and interface, and presents its information in a manner that is familiar to the researcher. The tasks outlined in this section illustrate the way in which the system accomplishes its goals.

The system accommodates two major classes of tasks: administrative tasks and user tasks. The administrative tasks are aimed at creating, maintaining, and managing the relationships between the different strains, Idd’s, and genes of the system. Administrative tasks also include mining for publications relevant to the research system.

Once the structure of the system is established, the user tasks allow a user to upload information, and navigate through the system. The system also allows the user to see the relationships between different strains, Idd’s, and genes and the resources relevant to each.

2.1 Administrator Tasks

The administrator tasks allow a user to make changes to the organizational structure of the system. The admin tasks allows for the management of strains, Idd’s, genes, and key words. The admin tasks also allow a user to mine for new publications and approve newly mined publications.
2.1.1 Strain Management

Managing strains is an important feature of the DRV. The tasks involved with strain management include:

- Remove an Idd from a strain.
- Add an Idd to a strain.
- Remove the strain from the system.
- Add a new strain to the system.

In order to manage strains a user must first log into the system and click the Admin button. This will direct them to the administrator page. From this page, the user can click the strain of interest and click on the Manage Strains button. This will direct the user to the selected strain’s management page. Figure 2.1 shows the path a user can take in order to reach the strain’s management page. It also shows the path the user must take to reach other pages that will be discussed later in this section.

![Diagram](image_url)

*Figure 2.1 The path a user can take to reach various management pages in the system.*
The strain management page contains all of the information relevant to a particular strain, including its Idd’s. Figure 2.2 shows a snapshot of the strain management page.

![Strain Management Page](image)

**Figure 2.2:** The Strain management page showing the Idd’s associated with the strain. Shown are the different buttons associated with different strain management tasks. On the left is the button used to add a new strain to the system.

The Idd’s associated with the strain are listed at the top of the page as hyperlinks. Each one links to a page displaying the details of the individual Idd. This page also allows a
user to remove and add Ids to the strain. To remove an Id from a strain, select the Id from the drop down box and click Remove. To add an Id to a strain select the Id from the list of available Ids and click the Add button. This will associate the Id with the current strain. To remove the current strain, click the Remove button at the bottom of the screen. Performing this action will remove any references to the strain, as well as the publications associated with the strain. To add a new strain, select the Add Strain button on the left hand of the screen. This will lead you to a page where the user can enter the necessary strain information.

2.1.2 Id Management

Analogous to the process of managing strains, one can also manage Ids. To do so, the user must select the Id of interest from the admin page and click on the button labeled Manage Ids. Upon doing so, the user will be directed to an Id management page, similar to the strain management page described above. Figure 2.3 shows a typical Id management page.
Figure 2.3 The Idd management Page showing the genes associate with the Idd. Shown are the different buttons associated with different Idd management tasks. On the left is the button used to add a new Idd to the system.

The tasks associated with Idd management include:

- Remove a gene from the Idd
- Add a gene to the Idd
- Remove the Idd from the system
• Add a new Idd to the system

The genes associated with the Idd are listed as hyperlinks at the top of this page. Clicking on a gene will lead to a page that displays the details of the specific gene. To remove a gene from the current Idd, the user can select the gene from the appropriate drop down list and click on the Remove button. To add a gene to the current Idd, the user can select the gene to be added and clicks on the Add button. To remove the Idd from the entire system, the user can click the Remove button at the bottom of this page. This will remove the Idd and all references to it from the system. This includes removing references strains have to it. To add a new Idd to the system, the user can click on the Add Idd button on the left hand side of the page. This will allow the user to type the new Idd into a text field and add it to the system.

2.1.3 Gene Management

The user may also choose to manage the genes present in the system. This is done by selecting the specific gene from the admin page and clicking on the Manage Genes button. This will direct the user to the gene management page. The user can only perform two tasks on this page: add a gene to the system or remove the current gene from the system. To add a new gene to the system, the user clicks the Add Gene button on the left hand side of the page. Doing so will allow the user to enter the name of the new gene into a text box and add it to the system. In order to remove the current gene from the system, the user clicks the Remove button at the bottom of the page. Once again, this will remove references Idds may have to the gene, and the publications associated with the gene. To summarize, the two tasks the user can complete on the gene management page are:

• Remove the current gene from the system
• Add a new gene to the system.
2.1.4 **Keyword Management**

The user can also manage keywords. Keywords are terms unrelated to strains, Idds, and strains. They serve more like general categories that may be of interest. Keywords examples include “diabetes” or “dendritic cells”. Instead of storing a resource under a strain for example, a user may want to store it under the keyword diabetes. This can arise if the resource is a document relevant to the general nature of the disease, rather than a specific subcategory. The inclusion of keywords provides users with some flexibility out of the prescribed organizational structure.

The user can manage a keyword by selecting a specific keyword from admin page and clicking on the button labeled **Manage Keywords**. This will lead the user to a page similar to the gene management page. The keyword management page will allow the user to remove the selected keyword from the system. To do so, the user clicks the **Remove** button at the bottom of the page. This will also remove the publications associated with the keyword. To add a new keyword, the user can select the **Add Keyword** button on the left hand side of the page. This will allow the user to enter the new keyword into a text box and add it to the system. To summarize, the keyword management page allows the user to complete the following:

- Remove the current keyword from the system
- Add a new keyword to the system

2.1.5 **Literature Data Mining**

The next two tasks are relating to the data mining portion of the system. Data mining is the process by which the system mines publications from PubMed and imports them into the system. The publications mined from PubMed are specified by the strains, Idds, genes, and keywords stored in the system.

On the admin page, one can click the **Mine** button to launch the java mining application. An application window will appear three text boxes and a **Connect** button at the top of the frame. Figure 2.4 shows the mining application interface.
In order to start using the mining application, the user must enter the IP address for the database in the database location text field. The user name and password for the database must also be specified in their appropriate text boxes. Once the necessary information is complete, the user can click on the Connect Button. If the connection is successful, the bottom of the frame will have text displaying “Status: Connected.” Once the application is connected to the database, it will display a list of terms in a box on the left hand side of the application. These are all of the available search terms for which the application can mine PubMed. The search terms come from the keywords, strains, Idds and genes stored in the system. If a user wants to search for a term, he can select it and click on the “Add Selected Terms” button. Multiple selections can be made at once by pressing down the Ctrl key while making selections. To add all of the search terms, the user can select the
Add All Terms button. The selected terms appear in the right hand box. To clear the selected terms, the user can click the Clear All Terms button.

As previously mentioned, the terms that appear on the right hand box are the terms that the application is slated to mine. Once the user is satisfied with the terms that appear in this box, he can click the Mine button. Doing so will start the mining process. The status of the process will be displayed in the monitor box at the bottom of the frame. The application will tell the user the current term being mined and the number of publications attempting to be added to the system. Closing the application will stop the mining process.

When the mining process is complete, the newly added publications will need to be approved in the system. Approving the publications allows a user to filter out irrelevant publications from the system. In order for a publication to be visible to ordinary users, it must first be approved for entry into the system. The approval page can be accessed by clicking the Approval List button from the admin page.

The approval page allows a user to approve newly mined publication. Figure 2.5 shows a hypothetical approval page.
**Figure 2.5:** A typical approval page displaying the mined publication awaiting approval into the system.

The page will list the pending publications along with the option to approve (represented as a "Y" hyperlink) or disapprove (represented as a "N" hyperlink). Clicking the approve option causes the publication to be fully incorporated within the system. This makes the publication accessible to all users under its appropriate category (strain, Idd, gene,
keyword). Disapproving the publication will discard it from the system and users will be unable to access it. While on the approval page, the user may choose to view the publication in more detail, before approving or disapproving it. To do this, one can click on the publication hyperlink. This will direct the user to a page displaying the details of the publication and allow the user to either approve or disapprove it.

2.1.6 Administrator Task Summary

The administrator tasks outlined enable a user to build the organizational structure of the system. It allows the user to specify strains, Idd, genes, and keywords. More importantly it allows the user to establish relationships between different strains, Idd, and genes. As highlighted in the beginning of this section, the connection between these three concepts is the key to the entire system. Without leveraging biological knowledge, this system would fail to do anything novel in the way of resource management. By using the connections between strains, Idd, and genes, the system facilitates efficient information retrieval by a potential user. It also enables the user to explore topics related to his research topic (whether it be relevant Idd, strains, or genes). The user tasks of the next section further illustrate the system’s ability to facilitate knowledge acquisition by a user.

2.2 User Tasks

The user tasks comprise of browsing the system for relevant publications and data, as well as uploading data into the system. As previously mentioned, publications and documents are organized into four main categories: strains, Idd, genes, and keywords. To start browsing the system, the user must first arrive at the search page. This can be done by clicking the Search button the home page, which is displayed after logging in. Figure 3.5 shows the search page—the page that results after clicking the Search button.
Figure 2.5: The search page. Buttons for searching by Idd, strain, gene, and keyword are displayed, as well as the buttons for uploading documents and microarray data.

2.2.1 Strain Search

To search the system by strain, the user selects the strain of interest and clicks the Search by Strain button. This directs the user to a strain search page. This page displays the strain of interest, along with the strain’s relevant genes and Ids. At the bottom of the page will be all mined publications as well as uploaded documents associated with the strain. Figure 3.6 shows a typical strain search page.
2.2.2 Idd Search

To search the system by Idd, the user selects the Idd of interest from the search page and clicks on the Search by Idd button. This directs the user to an Idd detail page that displays the current Idd along with its relevant strains and genes. At the bottom of the page will be all mined publications as well as uploaded documents associated with the Idd.
2.2.3 Gene Search

To search the system by gene, the user selects the gene of interest from the search page and clicks on the Search by Gene button. This directs the user to a gene search page that displays the current gene along with its relevant strains and genes. At the bottom of the page will be all mined publications as well as uploaded documents associated with the gene.

2.2.4 Keyword Search

To search the system by keyword, the user selects the keyword of interest from the search page and clicks on the Search by keyword button. This directs the user to a keyword search page. At the bottom of the page will be all mined publications as well as uploaded documents associated with the keyword.

2.2.4 Microarray Search

In addition to the four categories discussed above, the user can also search microarray data. Due to the complex nature of these experiments, microarray data is not categorized into strains, Idds, genes, or keywords. They reside as a separate group within the system. To browse microarray data, the user can click on the Browse Microarray button on the search screen. The user will be directed to a page that displays all microarray experiments present in the system. To view the details of a particular experiment, one can click on the hyperlink associated with it.

2.2.5 Uploading Documents

The next set of tasks involves uploading documents and microarray data into the system. The user can upload a document by clicking on the Upload Doc button, located on the search page. The system will then direct the user to a page where he can enter the title of the document, the author, and a combination of system references. The system references can be any combination of strains, Idds, genes or keywords. Finally, the user will have to locate the file he wishes to upload and click on the Upload button. Successful completion of this task will make the uploaded document available for other users to browse.
2.2.6 Uploading Microarray data

Uploading microarray data is also possible. Clicking the Upload Array will take the user to a page displaying all of the necessary information required to upload a microarray experiment. The required fields adhere to the MIAME standard and are discussed in a separate section. Upon specifying the required fields, the user can upload the experiment into the system. Successful completion of this task will make the experiment available to other users. (Microarray data is discussed in Chapter 9)

2.2.7 User Task Summary

The user tasks should have demonstrated the ease by which the system connects different types of research information. It should be apparent that an interest in a particular strain, for example, can easily lead to information related to an Idd or gene. Without the use of the system, gathering information about Idds relevant to a particular strain was a time consuming process. It required knowledge of all the relevant Idd’s as well as collections of publications relevant to each of the Idd’s. With the implementation of this system that process is greatly expedited. All of the relations between strains, Idd’s, and genes are already capture, and the navigation among each of these domains is seamless. Making seamless transitions between different kinds of research information was a key requirement of the system, and hopefully, the design and implementation of the system demonstrates this.
Chapter 3
High Level Design Overview

The first two chapters discussed the biological motivation of the system as well as the primary system goals. They described the essence of the system by illustrating the tasks the system facilitates. This section will give a high level overview of the system’s different parts and components. In doing so, it will outline the different modules of the system and how they interact to meet the system’s goals.

The system has three major components, or modules: a MySQL relational database, a series of JSP’s (Java Server Pages) and Java files, and a Tomcat JSP engine/web server. Each component is responsible for a different set of tasks, and each is vital to the overall functionality of the system. A typical user task can illustrate the role of each component.

A typical scenario involves a user searching the system by a specific strain. In order to do this, the user must select a strain of interest from the search page, and click the Search by Strain button. (This process is highlighted in section 2.2.1) The resulting page displays information relevant to the strain, as well as the corresponding publications and uploads.
3.1 JSP Introduction

In order for this simple task to take place, the three components of the system must work together. The pages viewed by the user are JSP's. JSP's are web pages (HTML) that contain embedded Java. The Java allows pages to react to user input and adjust dynamically to different user-fired tasks. In the above example, when the user selects a specific strain, the page he is lead to changes based on his selection. Without the dynamic behavior afforded by Java, each selected strain would require a completely different web page to display it. Instead, the embedded Java allows the page to fetch the selected strain and adjust its content according to the user's selection. In summary, the JSP's provide the raw web pages and Java code for the system.

3.2 Tomcat Introduction

In order for the JSP's to be displayed correctly, a Tomcat JSP engine/web server must be deployed. With a simple web page (HTML), a user can open the file in a web browser and it will correctly display the content of the page. A JSP, however, is not as easily viewed. Because of its dynamic content, JSP's require an engine to interpret them and return a file that can be read by a web browser. A web browser does not have the ability to interpret the Java present in the JSP, and that is why the Tomcat Server is needed. When a user clicks the Search by Strain button in the above example, he is not simply being directed to another page. Instead, the Tomcat JSP engine/web server is taking the appropriate JSP, interpreting it, and returning a web page to the user's web browser. Figure 3.1 illustrates the process by which a user input fires the interpretation of a JSP by the Tomcat Server, returning the correct web page to the user.
3. Returned Web Page

1. User Input

2. Interprets

Tomcat JSP engine/Web Server

JSP

3. Returned Web Page

**Figure 3.1:** The process by which a JSP is interpreted and returned to the user.

### 3.3 MySQL Introduction

The MySQL database is the final and most important part of the entire system. While the JSP’s and Tomcat components are essential for the interface of the system, the MySQL database is responsible for maintaining the organization and information of the entire system. It contains the entire dataset of information regarding strains, Idd’s, genes, keywords, publications, and uploads. It also stores all of the relationship between the different pieces of data. For example, the database has a table which stores the relationships between strains and Idd’s. Its structure and content are the keys to the overall design and functionality of the system.

In the example discussed at the beginning of the section, a user chooses to search the system by strain. In doing so, he is lead to a strain search page that displays the strain’s relevant Idd’s and genes as well as related publications. Up to this point, the piece of the process that has been overlooked is the data retrieval involved in this entire process. How for example, does the system retrieve the correct Idd’s and genes associated with the selected strain?

The MySQL database provides the mechanism by which data retrieval is carried out. The Java embedded in the JSP’s makes appropriate calls (specified by the strain the user selects) to the MySQL and retrieves the necessary information. In essence, the JSP’s and Tomcat engine provide the bridge by which the content of the database is logically displayed to the user. An overview of this entire process is captured in figure 3.2.
illustrates the three major components of the system, and the manner in which they interact.

Figure 3.2: A diagram showing the process by which the DRV responds to a user’s input.

3.4 High Level Design Summary

In summary, the three parts of the system—the JSP’s, Tomcat engine/web server, and the MySQL database—interact with one another in structured manner. A user’s initial request is handled by the Tomcat server. It directs this request to the appropriate JSP. The JSP may have embedded JAVA. If so, the Tomcat server will process make any necessary calls to the MySQL database. In turn, a web page will be returned to the user displaying all the information relevant to his request.

The next three sections examine each of the three system components in more detail. They give implementation details for the database table structure and the JSP hierarchy. Tomcat configuration details are also covered in a subsequent section.
Chapter 4

Database Design

This section delves into the specifics behind the MySQL database. Initially, it gives some background behind the concept of a relational database, and continues by giving details on the particular database implementation for the system. Understanding the benefits of a relational database is the key to understanding the overall operation of our system. This section aims at illustrating the ease by which the relational database model effectively manages the data and publications stored in the system.

The use of a database was a logical choice for the project, because it provided a standardized and flexible way of managing the system’s knowledge. No alternatives (XML files, or text files) provided quite the flexibility or scalability of a relational database. The use of a database also kept the focus of the project on information design rather than on the details involved with data storage. In order to understand the benefits of the database, it is important to grasp the core concepts about relational databases.

4.1 Relational Database Overview

A relational database is a collection of linked tables. A simple example can illustrate this basic idea. Suppose a person is interested in keeping a list of students in a particular class. At first he may only be interested in a person’s name. This translates
into having a database table in which there is one column representing names of the students currently enrolled in the class. Figure 4.1 illustrates this very basic table.

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kia Javanmardian</td>
</tr>
<tr>
<td>Trent Reznor</td>
</tr>
<tr>
<td>Kurt Cobain</td>
</tr>
<tr>
<td>Manny Ramirez</td>
</tr>
<tr>
<td>Johnny Damon</td>
</tr>
</tbody>
</table>

Figure 4.1: A table representing the names of student in a class.

The table in Figure 4.1 represents a table in which the name is a primary key. A primary key differentiates one row from another. In this simple example, the name is the primary key and therefore must be unique from every other entry in the table. Two rows with the same name are not permitted.

Suppose later, the person decides that he also wants to keep track of the other classes each student is taking. In order to do so, he chooses to create another table that links to the first table. This second table will contain a name and a class; therefore, it will have two columns. The primary key of this table will be the combination of name and class. This allows multiple entries for a given person but only one entry for every unique combination of person and class. For example, Kia may appear multiple times in the table, but Kia and 6.001 can only appear once.

The first table now links to the second table via the name. The ability to link is what makes a database “relational”. Figure 4.2 demonstrates this concept.
Figure 4.2: The name table linking to a table containing name and class combinations. This figure demonstrates how two tables can be linked in a relational database.

The ability to link tables makes relational databases very powerful, because it allows one to compound knowledge in a flexible manner. In our system, there are links between strains, Idd’s, and genes. In the future, however, research interests may change, and another category may emerge. By implementing a relational database the system becomes easily extendable. Therefore, a new category of interest may easily be linked to the current dataset. The ability to easily add different types of data made a relational database the ideal solution for the data storage component of our system.

4.2 Database Table Specifications

The relational database selected for the system was the MySQL database implementation. MySQL is an open source implementation of a relational database. It is free to use and it is fairly thoroughly documented. It has also been used by a number of corporations and organizations. [7]

The DRV consists of 11 relational tables that are responsible for storing the data for the entire system. The name of the database is diabetes, and every table in the database is accessed in the form: diabetes.name_of_table. The tables are relational; therefore, pieces of data in a single table may link to data in other tables. This creates a
web of relationships between tables. Users can make calls to specific tables when inquiring about particular information. The linked nature of the tables makes calls made by the system very efficient. The information returned for a particular call will, in theory, not contain irrelevant information. The alternative would be having one table with all the information present within it. Each call would then return a lot of unnecessary information.

Figure 4.3 shows the eleven tables and the interrelations that exist between them. An interrelation signifies a field in a field of one table that maps to a field in another table.

![Figure 4.3 The table hierarchy of the system.](image)

The eleven tables shown in figure 4.3 vary in content. The next part of this section will outline the fields present in each table and the mappings that exist between tables.
4.2.1 The diabetes.gene Table Specification

The diabetes.gene table consists of a single field, Gene. This field is of type varchar(32) and is the primary key for this table. Gene maps to a number of tables, as shown above. The tables include publications, publications_pending, and idds_genes.

4.2.2 The diabetes.idd Table Specification

The diabetes.idd table also consists of a single field, Idd. This field is also of type varchar(32) and serves as the primary key for the table. The diabetes.idd table maps to publications, publications_pending, strain_idd, and idds_genes.

4.2.3 The diabetes.strain Table Specification

The diabetes.strain table consists of the primary key, Strain. This field maps to publications, publications_pending, and strain_idd.

4.2.4 The diabetes.word Table Specification

The diabetes.word table consists of a primary key, Word. This field maps to publications and publications_pending.

4.2.5 The diabetes.users Table Specification

The diabetes.users table consists of two fields: Name and Password. The Name field serves as the primary key for this table; thus, each user is required to have a unique login. This table links to name_recent.

4.2.6 The diabetes.strain_idd Table Specification

The diabetes.strain_idd table consists of two fields: Strain and Idd. Both of these fields are of type varchar(32). These two fields in combination serve as the table’s primary key. Therefore, there may be multiple entries with the same Strain or the same Idd, but there may only be one entry for any Strain and Idd combination. As previously mentioned, the strain and idd table are linked to this table. This means that a Strain/Idd combination must have a strain and Idd that are present in the Idd and strain tables respectively.
4.2.7 The diabetes.idds_genes Table Specification

The diabetes.idds_genes table is analogous to the strain_idd table. Idds and Gene make up the two field names, and in conjunction, serve as the table’s primary key. There may be multiple entries with the same Idd or Gene, but only one unique Idd/gene combination is allowed. This table is linked to the Idd and gene tables. All Idd/gene combination must have a counterpart present in both the Idd and gene tables.

4.2.8 The diabetes.publications Table Specification

The diabetes.publications table consists of 10 fields which specify all the information for a publication or document stored in the system. The Title and Idd fields (both varchars) serve as a combination primary key. The Idd field is misleading, as strains, Idds, genes, and keywords can map to it. The combination primary key between title and Idd make it possible to have a single publication under a number of different strains, Idds, genes, and keywords. This allows a person to upload a document and place it under a number of different categories. The other fields are as follows:

- Abstract (text) (can be null) – A short description of the document or an abstract detailing the contents of the publication.
- Source (varchar(200)) – The source (journal) which the publication came from. If the document was uploaded, the source is the value “uploaded”.
- Link (varchar (255))(can be null)– A link to the actual document if it was uploaded. This field will be left blank if the document is mined from PubMed.
- Volume (varchar(200))(can be null) – The volume from which the publication came from. This is generally blank for uploaded documents.
- PubMed_ID (decimal)(can be null) – This field stores the PubMed ID for every publication mined from PubMed. This field is blank for uploaded documents.
- Author ((varchar(200))(can be null) – The author of the publication or document.
- Date (date)(can be null) – The date the publication was published or the date the document was uploaded into the system.
4.2.9 The diabetes.publications_pending Table Specification

The publications_pending table is identical to the publications table, but only contains mined publications awaiting approval. Uploaded publications do not need approval and are stored directly in the publications table.

4.2.10 The diabetes.name_recent Table Specification

The name_recent table has four fields: Name, Idd, Title, Date. Name, Idd, and Title serve as combination primary keys. This table is responsible for storing the publications recently viewed by a particular user. Each recently viewed publication is stored in this table, paired with the user who viewed it. Name, Idd, and Title are of type varchar and cannot be null. The Date field is of type date and can be null. The value stored under the date field corresponds to the date which the user viewed the publication or document. As previously mentioned, Idd and Title map to the publications table. The name maps to the users table.

4.2.11 The diabetes.micro Table Specification

The final table in the database is the diabetes.micro table. This table stores all of the information relevant to a microarray experiment. If the experiment is to adhere to the MIAME standard, none of the fields in this table should ever be null. The actual table, however, requires that the Experiment Goal field be specified. In conjunction with the date the Experiment Goal serves as the primary key. The other fields are allowed to be null, although, technically they shouldn’t be. Below is an outline of the fields present in the diabetes.micro table:

- Experiment_Goal varchar(200) (Primary Key) – The experiment goal
- Experiment_Description (text)(can be null) – A description of the experiment.
- Keyword (text)(can be null) – A keyword representative of the nature of the experiment (i.e. time course, cell type comparison)
- Experimental_Factors(text)(can be null) – The parameters or conditions tested, such as time, dose, or genetic variation (the use of MGED ontology terms is recommended).
- Experimental Design (can be null) - relationships between samples, treatments, extracts, labeling, and arrays (e.g., a diagram or table).
- Quality Control (can be null) - control steps taken (e.g., replicates or dye swaps).
- Supplemental Steps (can be null) - Links to the publication, any supplemental websites or database accession numbers.
- Biological Origin (can be null) - The origin of each biological sample (e.g., name of the organism, the provider of the sample) and its characteristics (e.g., gender, age, developmental stage, strain, or disease state).
- Protocols (can be null) - Manipulation of biological samples and protocols used (e.g., growth conditions, treatments, separation techniques).
- Experimental Factor (can be null) - value for each experimental factor, for each sample (e.g., 'time = 30 min' for a sample in a time course experiment).
- Hybridization Protocol (can be null) - Technical protocols for preparing the hybridization extract (e.g., the RNA or DNA extraction and purification protocol), and labeling.
- Experimental Control (can be null) - External controls (spikes), if used.
- Hybridization Conditions (can be null) - The protocol and conditions used for hybridization, blocking and washing, including any post-processing steps such as staining.
- Data Extraction Protocol (can be null) - Image scanning hardware and software, and processing procedures and parameters. Normalization, transformation and data selection procedures and parameters.

- Principle Array Organism (can be null)
- Link (can be null) - A Link to the zip file containing the microarray raw data.
- Date (date) (cannot be null) - The date that the experiment was uploaded to the system.
Chapter 5
Tomcat Server

This section outlines the DRV’s use of the Tomcat Server. The chapter begins by describing the purpose of the server and later, delves into the setup and configuration of the server. It should be noted that this section does not have in depth coverage of initial setup of the Tomcat Server. Documentation for initially setting the server up can be found at http://jakarta.apache.org/tomcat/.

The Tomcat/Apache server is a combination web server and JSP engine. As highlighted in the overview of Chapter 3, the Tomcat Server has two major jobs. The first is to act like a web server and serve the entire website to the outside world. The other job is to serve as a JSP engine. An overview and brief explanation of JSP’s can be found in Chapter 3.

We selected Tomcat for our JSP engine and web server, because of its cost and prominent use worldwide. The server is an open source project and is free for use and deployment. Many organizations use the server as a JSP engine and web server. There are many reports of success using it. [9]

The Tomcat Apache Server (version 5.0) can be downloaded via the Jakarta website at http://jakarta.apache.org/tomcat/. The site allows a user to download a binary file that includes an installer which allows for easy installation of the software.
5.1 Tomcat Configuration Specific to the DRV

This subsection discusses the configuration of Tomcat specific to the DRV. Once Tomcat is installed, a number of files and resources must be transferred into appropriate directories to make the server function properly.

Upon installation, there are two key folders vital to correctly setting up Tomcat. The first folder is located under: \Tomcat Home Directory\webapps\ROOT. In this folder one should place all of the JSP files that Tomcat will be serving. In the case of the DRV, all of the JSP’s are stored within one folder named Curation. Therefore, the raw JSP’s can be found under: \Tomcat Home Directory\webapps\ROOT\Curation.

The DRV’s JSP’s use external Java classes for many of its computational processes. The role of Java classes, in the overall functionality of the DRV, is discussed in Chapter 6. Regardless, the Java files used by the JSP’s should be bundled into a jar file and placed under: \"Tomcat Home Directory\"\common\lib. When a jar file is placed in this directory, the server will make the jar available to JSP’s that refer to it. In computer jargon, putting a jar under \"Tomcat Home Directory\"\common\lib adds it to the server’s class path. It is worth noting that the server must be restarted in order to correctly incorporate newly added jar files.

Once the server is started accessing the site is straightforward. One can access the server by typing “http://IP_ADDRESS_OF_SERVER:8080”. The IP address of the server along with the default port 8080 will lead the user to the tomcat index page. If a collection of JSP files have been stored under \"Tomcat Home Directory\"\webapps\ROOT\NEW_FOLDER, one can access the site defined by these group of JSP’s. As discussed in the previous paragraph, the DRV’s JSP’s are stored in a folder named Curation. Therefore, the home page of the DRV can be accessed by typing: http://IP_ADDRESS_OF_SERVER:8080/Curation, into a web browser. This will bring up index.jsp page found in the Curation folder.
Chapter 6
Java Overview

This section discusses the role of external Java files in the overall functionality of the DRV. While the DRV relies upon JSP’s for much of its dynamic processing, the JSP’s rely on external Java classes in order to carry out different computations. This section assumes familiarity with the Java programming language. The site http://java.sun.com contains numerous references and white paper on the Java programming language.

6.1 Java Class Outline

The Java classes are responsible for retrieving and storing data from the MySQL database. They are also responsible for processing retrieved data and making it usable by the JSPs. These Java files are bundled into a single jar file—in the case of the system, the jar file is named Project.jar. Incorporation of Project.jar into the DRV is discussed in Chapter 5. In all, there are seven java files that are involved with various tasks the DRV executes.

- AdminData.java – This file is responsible for most of the database calls made by the system.
- MedParser.java – The backend for the mining application.
- MicroArray.java – A data structure for storing MicroArray information.
- MonthConverter.java – Converts a text representation of a month into a number.
- Publication.java – Data structure for storing publication information.
- Strain.java – Data structure for storing strain information.

A class dependency model is shown below. AdminData depends on four different classes, while PubMedApp only depends on MedParser.

Figure 6.1: A class dependency model between all of the external Java classes used in the DRV.

6.1.1 Justification for External Java Classes

External Java classes are used for the sake of modularity. By encapsulating different computational processes into external classes, future computational changes can be made in a single location. If each JSP rewrote commonly used computational
processes, making a change to a single process, would require changing every JSP file that implemented it.

Modularity also makes adding more computational processes to the DRV a straightforward process. New classes, specifying new computation processes, can be written and incorporated into Project.jar. These new classes become available to the JSP’s of the system, and allow for new functionality within the DRV. Expanding one of the existing classes can also add functionality to the DRV. Expanding an existing class means adding more methods to it. Extending or creating a new Java class is up to the discretion of the system designer. In the case that the structure of the existing classes is to be changed, starting from scratch with a new class will be a better option.

The subsequent sections outline the DRV’s different Java files. Descriptions of the classes as well as input output characteristics are also discussed. Furthermore, a description and specification for each of a class’s different methods are also given.

6.1.2 AdminData.java Specification

AdminData.java is responsible for most of the calls to the MySQL database. Its methods retrieve data from the database as well as modifying data in the database. Examples of this include retrieving publication relevant to a particular Idd. Also included are tasks like associating an Idd with a strain or a gene with an Idd. Below are the methods associated with the AdminData.java class:

- **Constructor:** AdminData() Empty constructor.

- **Methods that Add Data**

  - `addIdd(String idd)` – Takes in the idd and adds it to the system. This affects the idd table in the database.
  - `addStrain(Strain strain)` – Takes the strain and adds it to the system. This affects the strain table in the database.
  - `addGene(String gene)` – Takes the inputted gene and adds it to the system. This affects the gene table in the database.
- **addWord(String word)** – Takes the inputted word and adds it to the system. This affects the word table.

- **addIddToStrain(String strain, String idd)** – Takes the idd and associates it with the strain. This affects the strain_idd table.

- **addGenesToIdd(String idd, String gene)** – Takes a gene and associates with the idd. This affects the idds_genes table.

- **addLogin(String name, String password)** – Adds the name and password to the system by modifying the users table.

- **approvePublication(String idd, String title)** – Takes the publication specified by the idd and title and moves it from the pending_publications table to the publications table. The publication will no longer appear in the pending publications table.

- **addRecentPublication(String name, String idd, String title)** – This method takes the name of a user, the title of a publication and the keyword of a publication and stores it in the name_recent table. This piece of data shows that the user recently viewed a given publication.

**Methods that Remove Data**

- **removeStrain(String strain)** – Attempts to remove all references to the strain from the system. Affects the strain table and strain_idd table.

- **removeIdd(String idd)** – Attempts to remove all references to the idd from the system. Affects the idd table, the strain_idd table, and the idds_genes table.

- **removeGene(String gene)** – Attempts to remove all references to the gene from the system. This affects the gene table and the idds_genes table.

- **removeWord(String word)** – Attempts to remove the word from the system. This affects the word table.

- **removeIddFromStrain(String strain, String idd)** – Disassociates the idd from the strain. Affects the strain_idd table.

- **removeGeneFromIdd(String gene, String idd)** – Disassociates the gene from the Idd. This affects the idds_genes table.
- `disapprovePublication(String idd, String title)` – Takes the publication specified by the idd and title and drops it form the pending-publications table. A reference to this publication will cease to exist within the system.

- `clearRecentPublications(String name)` – Clears the recently viewed publications of a user. This method makes a call to the name_recent table.

- `deletePublication(String title, String idd)` – Removes the publication specified by the title and idd from the publications table.

### Retrieval Methods

- `String[] getWords()` – Returns all the keywords stored in the system.

- `Strain[] getStrains()` – Returns all of the strains stored in the system.

- `String[] getGenes()` – Returns all the genes stored in the system.

- `String[] getIdds()` – Returns all the Idds stored in the system.

- `Publication getPendingPubForTitle(String title, String idd)` – Returns a publication from the pending_publications table which matches the title and specified idd.

- `Publication getPubForTitle(String title, String idd)` – Returns a publication present within the publications table that matches the given title and idd.

- `Publication[] getUploadForStrain(String strain)` – Returns an array of publications that are related to the given strain. These publications must have been uploaded by a user. The returned publications include all documents that are related to the Idds of the strain.

- `Publication[] getPublicationForStrain(String strain)` – Same as above method, except publications are restricted to those that were mined by the system.

- `Publication[] getUploadForGene(String gene)` – Gets all uploaded files that have been placed in the system under the given gene. Returns same thing as `getUploadForWord` and `getUploads`.

- `Publication[] getPublicationForGene(String gene)` – Gets all publications for the given gene that have been mined by the system.
• `Publication[] getUploadForWord(String Word)` – Gets all publications uploaded to the system under the specified word. **Returns same thing as `getUploadForGene` and `getUploads`**.

• `Publication[] getPublicationForWord(String Word)` – Gets all mined publications for the specified word.

• `Publication[] getUploads(String idd)` – Gets all uploaded publications for the given Idd. **Returns same thing as `getUploadForGene` and `getUploadForWord`**.

• `Publication[] getPendingPublications()` – Gets all pending publication from the table pending_publications.

• `Publication[] getPublications()` – Gets all publications that were not uploaded from the publications table.

• `String[] getIddForGene(String gene)` – Gets all idds that are relevant to the given gene. The information for this call is taken from the tabled idds_genes.

• `String[] getStrainsForGene(String gene)` – Gets the strains relevant to the given gene. This method first gets the Ids associated with the strains and retrieves the genes associated with the Ids. The tables involved with this process are idds_genes and strain_idd.

• `String[] getIddForStrain(String strain)` – Gets the strains relevant to the given Idd. This method makes a call to the strain_idd table.

• `String[] getStrainsForIdd(String idd)` – Gets the strains relevant to the given idd. This method makes a call to the strain_idd table.

• `String[] getGenesForIdd(String idd)` – Gets the relevant genes for a given idd. This method makes a call to the idds_genes table.

• `String[] getGenesForStrain(String strain)` – Gets the relevant genes for a given strain. This method makes calls to strain_idd and idds_genes.

• `boolean validateLogin(String name, String password)` – Attempts to validate the name and password combination. Makes a call to the users table.

• `Publication[] getRecentPublications(String name)` – Returns a group of publications recently viewed by the user. These array of publication objects only contains the title and keyword of the publication.
- **MicroArray getMicroArray(String goal, String date)** – Returns a MicroArray object specified by the goal and date of the experiment. This method makes a call to the micro table.

- **MicroArray[] getMicroArrays()** – Returns an array of all microarray experiments. This method makes a call to the micro table.

### 6.1.3 Java Interacting with MySQL

The parts of AdminData.java that communicate with the database are important. In order to successfully access a MySQL database within a Java class, the Java class must initialize a JDBC (Java Database Connectivity) driver and establish a connection with the database. In order to initialize the driver, the driver must be in the class path of the Java class instantiating it. In the DRV, a jar file containing the JDBC driver was placed in the same folder as Project.jar. The DRV uses a MySQL JDBC driver downloaded directly from the MySQL website at http://www.mysql.com.

To make a connection to the MySQL database the following code is needed:

```java
private String driverURL;
private Connection conn;
public AdminData() {
    driverURL = "com.mysql.jdbc.Driver";
    try {
        Class.forName(driverURL).newInstance();
        databaseURL = "jdbc:mysql://localhost/diabetes";
        conn = DriverManager.getConnection(databaseURL, "root", "kiarash");
    }...
}
```

The `driverURL` variable specifies the location of the driver in the external jar file. The driver is then initialized in the second line. Finally, a connection (the `conn` variable) is established in the last line by specifying the `databaseURL`, the user name, and a password.

Once the connection has been established, calls to the database can be made. To do this, a Statement object must be created from the initialized Connection object. Then the Statement can call the method `executeUpdate(String)` to update the database, or `executeQuery(String)` to retrieve data from the database. An example of each is shown below:
The results of a query are stored in a ResultSet object. This object contains all of the data resulting from a query. This object is as a set of rows containing the results of the query. Initially, the pointer of the ResultSet points before the first row of data. Calling the next() method on the ResultSet moves the pointer to the next row. When the pointer is at a certain row, retrieving a column from the row depends on the type of data stored in the column. If the column is storing an integer, the getInt("Column_Name") method retrieves its data. For a String, the getString("Column_Name") method retrieves its data. An example of this process is shown below:

```java
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("Select * from word");
while (rs.next) {
    String word = rs.getString("Word");
    ...
}
```

### 6.1.4 Publication.java Specification

Publication.java is a data representation for a publication. Its purpose is to efficiently store all the pieces of information associated with a publication and allow for the retrieval of different pieces of information. The object stores an idd, strain, gene, or keyword associated with the publication, along with a title, the text of the abstract, the source of the publication, a link to the publication, a volume, an author, and a date the publication was published. The object has a constructor and a number of methods:

- **Constructor:** `Publication(String idd, String title, String abstractText, String source, String link, String volume, String author, String date)` – idd can either be an idd, strain, gene, or keyword.
- **String getDate()** – Returns the date the publication was published.
- **String getIdd()** – Returns the idd, gene, strain, or keyword of the publication.
- **String getTitle()** – Returns the title of the publication.
- **String getAbstract()** – Returns the abstract of the publication.
- **String getSource()** – Returns the source of the publication.
- String getLink() – Returns a link to the publication.
- String getVolume() – Returns the volume of the journal where the publication appears.
- String getAuthor() – Returns the first author of the publication.

### 6.1.5 MonthConverter.java Specification

MonthConverter.java is responsible for converting a three letter String representation of a month and returning a number associated with the month. This class converts Jan into 1, Feb into 2, etc.

- **Constructor:** MonthConverter(String month) – Constructs a converter with the given month. If the month is not a valid input, the constructor sets the month to equal 1;
- int getMonth() – Returns the number representation for the given month.

### 6.1.6 Strain.java Specification

Strain.java is a data structure for storing strain information. It basically encapsulates the two parts of a strain: the strain name and the common reference. This class consists of a constructor and two methods.

- **Constructor:** Strain(String strain, String commonRef) – Constructs a Strain object by using the strain name and common reference of the strain.
- String getStrain() – Returns the strain
- String getRef() – Returns the common reference of the strain.

### 6.1.7 MicroArray.java Specification

MicroArray.java is a data structure for storing microarray experiment information. It has no methods, but the fields used to instantiate it are public and can be accessed outside of the class. Each field in the constructor represents a piece of information required to represent a microarray experiment. More information about these fields can be found in the MicroArray section of the document. The constructor of this class:

**Constructor:** MicroArray(
String experimentGoal,
String experimentDescription,
String keyword,
String experimentalFactors,
String experimentDesign,
String qualityControl,
String supplementalSteps,
String biologicalOrigin,
String protocols,
String experimentalFactor,
String hybridizationProtocol,
String experimentalControl,
String hybridizationConditions,
String dataExtractionProtocol,
String principleArrayOrganism,
String link, String date)

The next two classes make up the mining application. As discussed in Chapter 3, the system mines publications directly from PubMed. This process can be broken down into a backend portion and user interface portion. The interface portion is taken care of by the PubMedApp class, while the MedParser class makes up the back end of the application.

6.1.8 PubMedApp.java Specification

PubMedApp.java is responsible for creating the GUI of the mining publication. It extends the JFrame class, which allows it to create a window where different components can be added for the interface. The class also implements the interface ActionListener, which allows for buttons in the interface to perform a number of different actions. The constructor of the class is an empty one, and there are no visible methods. The source code for the class can be found in the appendix of the document.
The class uses basic swing objects and does nothing out of the ordinary. The only thing of particular interest is the frame’s layout manager. The layout of the frame is taken care of by a SpringLayout (new to JAVA 1.5) layout object. It allows different components on the frame to be placed and spaced relative to one another. This is an improvement on the older layout managers provided by JAVA.

6.1.9 MedParser.java Specification

The MedParser.java class provides the backend for the mining application. The class extends the thread class so that it can mine publications in the background while the interface is in use. If this class was not implemented as a thread, the mining program would stall while publications were being mined. The constructor and methods of the class are listed below:

- **Constructor:** MedParser(String url, String name, String password) – Takes a url (IP address) for the mySQL server housing the database, the user name, and the password necessary for database access.
- **mine(Vector searchTerms)** - This method sets the search terms that the mining application will include in its search.
- **run()** – called to start the thread and begin the mining process.
Chapter 7

JSP Hierarchy

This section outlines the jsp hierarchy of the system. It will first give a flow chart of the JSP hierarchy. An alphabetical listing and description of the JSP content will be given, including a description of each of the files. Figure 7.1 shows the JSP’s related to the user tasks of the system outlined in Chapter 2.

Figure 7.1: A web showing the JSP’s related to user tasks.
Figure 7.2 shows the pages related to the administration portion of the system. It includes the JSPs related to organizing the relationships between strains, Ids, and genes. It also includes the pages related to approving publications. The dotted squares represent processing JSP's. These JSP's are never seen by the user. Instead, they process an action from a previous page and forward the user to a previous page in the system.

Figure 7.2: A web showing the JSP's related to user tasks.

Both figures illustrate a general view of the JSP hierarchy. All cyclical links have been omitted for simplicity, and some processing JSP's have also been omitted for simplicity.
sake. The next subsection will list the JSP’s of the DRV, including a brief description of each.

7.1 JSP File Outline
- AddDoc.jsp – This page comes up when the user wants to upload a document to the system.
- AddGene.jsp – This page comes up when the user wants to add a gene to the system.
- AddIdd.jsp – This page comes up when the user wants to add an Idd to the system.
- AddLogin.jsp – This page comes up when the user wants to add a user and password combination to the system.
- AddLoginError.jsp – This page appears when a user tries to add a login and fails. It notifies the user of the problem and asks him to pick another user and password combination.
- AddMicro.jsp – This page appears when the user wants to add a microarray experiment to the system.
- AddStrain.jsp – This page appears when the user wants to add a new strain to the system.
- admin.jsp – This page appears when the user goes to the admin portion of the system.
- ApprovalList.jsp – This page displays all of the documents awaiting approval by an administrator.
- ApprovePublication.jsp – This page appears when the user selects a document to inspect further from the approval list.
- GeneDetail.jsp – Shows the details for a given gene. This includes Idd’s and strains which contain it.
- GeneSearch.jsp – Appears when a user chooses to search the system via a gene. This shows the Idd’s and strains that contain the gene, as well as documents uploaded and mined for the gene.
- home.jsp – This page appears after a user initially logs into the system. It displays the user’s most recently viewed documents.
- *IddDetail.jsp* – The page shows the details of an Idd and appears in the admin section of the system. It shows the strains and genes relevant to the Idd and it allows the user to add genes to the Idd.
- *IddSearch.jsp* – This page shows the user the relevant genes and strains of an Idd. It also shows any uploaded files and mined publications related to the Idd.
- *index.jsp* – This page is the first page to appear when a user accesses the system. It contains a place for the user to enter his login and password.
- *MicroArrayDetail.jsp* – This page displays the information for a selected microarray experiment. It includes fields for all of the information relevant to the experiment.
- *MicroArrayTable.jsp* – This page displays a table of all relevant microarray experiments.
- *ProcessAddDoc.jsp* – Called from AddDoc.jsp when the user chooses to submit a document addition to the system. This JSP serves as a backend process that makes the necessary calls to the database for this process.
- *ProcessAddGene.jsp* – Called from AddGene.jsp when the user chooses to add a gene to the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.
- *ProcessAddGenelId.jsp* – Called from IddDetail.jsp when the user chooses to associate a gene with an Idd. This JSP serves as a backend process that makes the necessary calls to the database for this task.
- *ProcessAddIdd.jsp* – Called from AddIdd.jsp when the user wants to add an Idd to the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.
- *ProcessAddIddToStrain.jsp* – Called from StrainDetail.jsp whenever the user chooses to associate an Idd with the current strain. This JSP serves as a backend process that makes the necessary calls to the database for this task.
- *ProcessAddLogin.jsp* – Called from AddLogin.jsp when the user requests a new login be added to the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.
- **ProcessAddMicro.jsp** – Called from AddMicro.jsp when the user submits a microarray addition to the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- **ProcessAddStrain.jsp** – Called from AddStrain.jsp when the user submits a strain addition to the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- **ProcessAddWord.jsp** – Called from AddWord.jsp when the user submits a word addition to the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- **ProcessApproveDisapprove.jsp** – Called from MicroArrayTable.jsp when the user wants to approve or disapprove a newly mined publication. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- **ProcessDeletePublication.jsp** – Called from PublicationDetail.jsp and UploadDetail.jsp to remove a document from the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- **ProcessDeleteRecent.jsp** – Called from home.jsp when the user chooses to clear his list of recently viewed publications. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- **ProcessLogin.jsp** – Called from index.jsp to validate a username password combination. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- **ProcessRemoveGene.jsp** – Called from GeneDetail.jsp when the user chooses to remove the current gene from the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- **ProcessRemoveGeneFromIdd.jsp** – Called from IddDetail.jsp when the user wants to disassociate a gene from an Idd. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- **ProcessRemoveIdd.jsp** – Called from IddDetail.jsp when the user wants to remove the Idd from the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.
- ProcessRemoveIddFromStrain.jsp – Called from StrainDetail.jsp when the user wants to disassociate an Idd from a strain. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- ProcessRemoveStrain.jsp – Called from StrainDetail.jsp when the user wants to remove the current strain from the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- ProcessRemoveWord.jsp – Called from WordDetail.jsp when the user wants to remove the current keyword from the system. This JSP serves as a backend process that makes the necessary calls to the database for this task.

- PublicationDetail.jsp – Displays the details for a mined publication stored in the system.

- Search.jsp – This page is the main search page of the system. It appears when the user clicks the search button from home.jsp.

- StrainDetail.jsp – This page shows detailed information of a strain. This information includes relevant Idds and genes. This page also allows the user to associate Idds with the strain. It also allows the user to add new strains and remove the current strain from the system.

- StrainSearch.jsp – This page appears when the user chooses to search the system via a particular strain. This page shows the Idds and genes associated with the strain, as well as the publications and uploads associated with the strain.

- UploadDetail.jsp – This page appears whenever the user chooses to examine an uploaded file in detail. It displays the different pieces of information relevant to the file as well as a link to the actual file.

- WordDetail.jsp – This page shows the details of a keyword. It gives options for adding a keyword and removing the current keyword from the system.

- WordSearch.jsp – This page appears whenever the user chooses to search the system via keyword. It displays the publications and uploads associated with the keyword.
Chapter 8

Literature Data Mining

This section discusses the process by which the mining application retrieves publications from PubMed. The DRV uses a Java powered application to mine publications directly from PubMed to the system. It is written in Java and runs by utilizing Sun’s Webstart technology.

8.1 Java Web Start Technology

This technology allows a user to launch a java application from a website. The application requires a standard executable jar file, and a JNLP file specifying the launch properties of the application. A link to the JNLP file in a web browser will make the application launch. Specifications for JNLP files and Web Start technology can be found at http://java.sun.com. With respect to the DRV, the JNLP file is located in the same directory as the JSP files. Figure 8.1 shows the JNLP file that was written for mining application.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for SwingSet2 Demo Application -->
<jnlp spec="1.0+
codebase="http://localhost/Curation"
Figure 8.1: The JNLP file used to specify the startup properties of the mining application.

The `codebase` attribute at the top of the JNLP file specifies the web directory of the JNLP file. The `href` attribute specifies the name of the JNLP. Together, the `codebase` and `href` specify the location of the JNLP file.

The title, vendor, and description tags provide information about the application being launched. The `<security>` tag is set to allow all permissions to be allowed for the application. `<all-permissions>` is the default value for this tag, and allows the launched application like it would if it were to run locally.

Under the resources tag are specifications for the appropriate version of Java as well as the jar files that are needed to execute the program. The `mysql.jar` contains the MySQL JDBC driver, while the Project.jar contains the executable application. Project.jar has its executable class specified in its manifest file, which allows the JNLP file to initiate the class’s execution.
8.2 Mining Tools Used by the Mining Application

The mining application makes calls to two tools located on the nih.gov website. The first tool is located at:


This tool takes a search term and returns an XML file which contains a list of Ids representing matching publications. There are three parameters present in this URL. The first, db, directs the utility to the database of interest. In our case, the database is pubmed. The second parameter, retmax, sets the maximum number of results. This parameter is set to 10000 in our system. The last parameter, term, is the search term of the query. The appropriate search term should be placed after the equals sign in this URL.

The result of this initial query is presented to the system as an XML file. In order to extract the Ids from the XML file, the system utilizes the XML parser provided by Sun Microsystems. The parser can be downloaded from http://java.sun.com.

The second tool takes the ids returned by the first tool and fetches publications for each one. This tool is located at:


In order to use the second tool, the user must supply four parameters. The first, db, directs the tool to the correct database. In our case db is set to PubMed. The second, id, is the parameter of interest. This parameter uses the Id from the previous tool to retrieve the matching publication. The third parameter, report, gives the report type, XML. The final parameter, mode, gives the mode of output. In our case it is text. While this tool has four possible parameters, the id parameter is the only one that changes from query to query.

The second tool returns an XML document containing all of the information relevant to the publication. The XML file contains the publication title, author, journal,
abstract, etc. Both of the tools discussed above are used in MedParser.java, which is specified in Chapter 6.

8.3 *Mining Application’s User Interface*

The final part of the mining process is the user interface. It allows the user to manage the terms being mined by the application. Details regarding the interface can be found in Chapter 3 of this document.
Chapter 9

The DRV’s Microarray Overview

This section focuses on the microarray portion of the system. Specifically, it delves into the MIAME standard for microarray experiments. MIAME (Minimum Information About Microarray Experiments) is a standard used in storing microarray experiments. Adhering to the standard guarantees that necessary information about the microarray experiment is documented.

Our current system adheres to the current MIAME standard. When a user uploads a microarray experiment, the DRV requires a specific set of information. A checklist of that information is listed in the subsequent portion of this section [10].

9.1 MIAME Standard

- The goal of the experiment – one line maximum (e.g., the title from the related publication)
- A brief description of the experiment (e.g., the abstract from the related publication)
- Keywords, for example, time course, cell type comparison, array CGH (the use of MGED ontology terms is recommended).
Experimental factors - the parameters or conditions tested, such as time, dose, or genetic variation (the use of MGED ontology terms is recommended).

Experimental design - relationships between samples, treatments, extracts, labeling, and arrays (e.g., a diagram or table).

Quality control steps taken (e.g., replicates or dye swaps).

Links to the publication, any supplemental websites or database accession numbers.

Samples used, extract preparation and labeling:

- The origin of each biological sample (e.g., name of the organism, the provider of the sample) and its characteristics (e.g., gender, age, developmental stage, strain, or disease state).
- Manipulation of biological samples and protocols used (e.g., growth conditions, treatments, separation techniques).
- Experimental factor value for each experimental factor, for each sample (e.g., ‘time = 30 min’ for a sample in a time course experiment).
- Technical protocols for preparing the hybridization extract (e.g., the RNA or DNA extraction and purification protocol), and labeling.
- External controls (spikes), if used.

Hybridization procedures and parameters:

- The protocol and conditions used for hybridization, blocking and washing, including any post-processing steps such as staining.

Measurement data and specifications:

- Data
  - The raw data, i.e. scanner or imager and feature extraction output (providing the images is optional). The data should be related to the respective array designs (typically each row of the imager output should be related to a feature on the array – see Array Designs).
The normalized and summarized data, i.e., set of quantifications from several arrays upon which the authors base their conclusions (for gene expression experiments also known as gene expression data matrix and may consist of averaged normalized log ratios). The data should be related to the respective array designs (typically each row of the summarized data will be related to one biological annotation, such as a gene name).

- Data extraction and processing protocols
  - Image scanning hardware and software, and processing procedures and parameters.
  - Normalization, transformation and data selection procedures and parameters.

**Array Design:**

- General array design, including the platform type (whether the array is a spotted glass array, an in situ synthesized array, etc.); surface and coating specifications and spotting protocols used (for custom made arrays), or product identifiers (the name or make, catalogue reference numbers) for commercially available arrays.
- Array feature and reporter annotation, normally represented as a table including:
  - For each feature (spot) on the array, its location on the array (e.g., metacolumn, metarow, column, row) and the reporter present in the location (note that the same reporter may be present on several features).
  - For each reporter unambiguous characteristics of the reporter molecule, including
    - Reporter role – control or measurement
    - The sequence for oligonucleotide based reporters
    - The source, preparation and database accession number for long (e.g., cDNA or PCR product based) reporters
    - Primers for PCR product based reporters
  - Appropriate biological annotation for each reporter, for instance a gene identifier or name (note that different reporters can have the same biological annotation)
  - Principle array organism(s)
Summary and Conclusions

The DRV attempts to organize publications and experimental data in the research domain of T1D. It does so by leveraging the underlying biological concepts behind T1D research and categorizing publications and experimental data into strains, Idd’s and genes. Furthermore, by linking the Idd’s with strains and genes, the DRV is able to relate different categories of knowledge together. Thus, a user can search the DRV via a strain and easily access information relevant to the strain’s Idd’s and genes.

Organization and maintenance of publications and experimental data is the major goal of the DRV. The DRV also completes a number of additional tasks. The DRV allows a user to upload experiment data, documents, and microarray experiment data adhering to the MIAME standard. The DRV also allows a user to direct automated publication mining from PubMed. The DRV’s mining application gives a user the ability to mine PubMed based on a publication’s reference to a specific strain, Idd, gene or custom keywords. The mining application automates much of the process involved with importing publications into the DRV.

The design of the DRV centers around three major components: the JSP’s, Tomcat server, and MySQL database. The first two components are involved with the interface portion of the system, and the database handles the organization of the DRV’s data. Additionally, many of the database calls and data processing is handled by the Java embedded and referenced in the DRV’s different JSP’s. Each one of the three components, as highlighted in their respective sections, is vital to the overall functionality of the DRV.

In summary, the DRV was able to accomplish its goals:
- Organizing publications and experimental data
- Linking different kinds of data together (through connections between strains, Idd’s, and genes)
- Mining publications directly from PubMed
- Allowing users to upload data, documents, and microarray data adhering to the MIAME standard
Thus, the DRV allows TID researchers to store and maintain a complex group of experimental data and publications.

**Future Work**

Many additional features could improve on the DRV's overall usability and usefulness. While the relational database allows for fairly straightforward data extension within the DRV, the interface does not provide adequate flexibility. A major improvement to the system would be the ability to link the custom keywords to the organizational structure between strains, Idd’s, and genes. Having the ability to link the custom keywords to a strain, Idd, or gene would make the system more effective at consolidating information among different research categories.

A sophisticated text search would also be useful. Currently, the DRV leverages the search algorithms of PubMed to correctly categories publications under strains, Idd’s, genes and custom key words. Having a built in text searching tool would allow users to search within different categories of publications and experimental data.

Another useful feature of the DRV would be the ability to create subdirectories and categories under the DRV’s existing categories. Included would be the ability to create user specific folders under different strains, Idd’s, genes or custom keywords. Therefore, a particular user can have a personal folder of resources under Idd1, for example. Making the DRV more user customizable is a key step in extending the system’s overall impact.

While there are many improvements and additional features that can build on the DRV, the DRV provides a conceptual and concrete starting point from which to build. Key to the DRV’s usability is its biological inspiration. By incorporating biological concepts in the overall design of the system, the DRV has given researchers a logical way to manage their research information, and it aimed to reduced the overall learning curve associated with new software systems. While the DRV pertains to TID, the illustrated concepts and design considerations can be directly applied to many other biological research areas.
References

[1] www.jdrf.org


