Appendix To:

Exp-DB: Fast Development of Experimental Information Systems

Nomair A. Naeem

Supervisor:
Professor Bettina Kemme

December, 2002
# Contents

1 Introduction ............................................. 3

2 Architecture ............................................ 3
   2.1 Advantages of Model 1 over Model 2 .................. 6
   2.2 Advantages of Model 2 over Model 1 .................. 6
   2.3 The Old BRI Database ............................... 6
   2.4 The New BRI Database ............................... 7
      2.4.1 Architecture Chosen ............................. 8

3 Database Design ........................................ 8
   3.1 Access Control ...................................... 8
   3.2 Experimental Data ................................... 9
   3.3 Tables in the Database ............................. 11
      3.3.1 Access Control Tables .......................... 11
      3.3.2 WorkFlow Tables ................................. 11
      3.3.3 Sub-Experiment Tables ........................... 13

4 Web Application Structure ............................. 13
   4.1 Login Module ....................................... 13
   4.2 Servlets ........................................... 13
   4.3 Beans ............................................. 14
   4.4 JSPs .............................................. 14

5 A Typical Table Bean .................................. 16

6 An Example JSP ......................................... 24

7 Possible Extensions .................................... 26
   7.1 Access Control Module ............................... 26
   7.2 Automated Input .................................... 27
   7.3 Representation of Data .............................. 27
   7.4 Querying ........................................... 28
   7.5 Navigation ......................................... 29
   7.6 Web Application Structure .......................... 32

8 Conclusion .............................................. 32
1 Introduction

This document serves as an appendix to the paper Exp-DB: Fast Development of Experimental Information Systems. The paper was written as an effort to create a database and web application which could be used by small research laboratories to track their experiments. Important considerations were the startup costs, ease of extensions and flexibility of the system. This report gives details of a prototype developed using the ideas mentioned in the paper. The prototype is based on the current research methodology practiced at the Macromolecular Structural group, Biotechnology Research Institute (BRI), National Research Council, Canada.

The report concentrates on the choice of architecture used for the web application. The design of the database is discussed in detail and explains why the choices that were made make the database more suitable for a fast development system with lots of room for modifications and upgrades. The structure of the application and how it makes life easier for the administrator, to add new features, is discussed. The report also addresses the issue of possible extensions to the prototype to make it suitable for use in a real time environment.

2 Architecture

For the presentation of the information stored in the database, a web application was developed using Java Server Pages, Java Servlets and Java Beans.

Java Server Pages (JSPs), are used to display content on a web browser. Hence in an application which responds to the user by displaying information in a web browser a JSP can be utilized. Often it is said that that JSPs provide the presentation logic i.e. presentation of information, for an application.

Java Servlets, often called Servlets for short, are used to provide control to the flow of data (request response actions) in a web application.

Java Beans provide all the processing power of Java to an application. Business logic i.e. constraints on data within the application and ways the data is allowed to interact within the application is provided by Java Beans.

With these standards two types of architectures have become the most commonly used. They are Model 1 Architecture and Model 2 Architecture (MVC).
The two architectures can best be explained by outlining what happens when a user sends its request from a browser. In the model-1 architecture (see figure 1) a JSP page receives this request. This page might or might not use custom actions and JavaBeans to decide on a response for the request sent. This response could also be the selection of another JSP to which the request is forwarded. Hence in the model-1 architecture the request is processed at the JSP.

Model 1 architecture can also be explained in terms of view and model. The view is responsible for a presentation view of the business domain. The views within the web tier application typically consist of HTML and JSP pages. HTML pages are used to serve static content, while JSP pages can be used to serve both static and dynamic content. Most dynamic content is generated in the web tier. However, some applications may require the need for client-side JavaScript.

The model is responsible for the business domain state knowledge. In a 2-tier application the model classes may be a set of regular Java objects. These objects may be populated manually from a result set returned by a database query. In a more complex enterprise application where the web tier communicates with an EJB server for example, the model portion might be Enterprise Java Beans. These Java Beans are commonly referred to as value objects and are used within the views to build the dynamic content.

In the model-2 architecture (see figure 2), a controller first intercepts the client request. This controller almost always is a Servlet. The trick is to separate the processing, the data handling and the presentation into different components. The controller Servlet, or very rarely the
controller JSP, handles the initial processing of the request and also determines which JSP page to display next. This allows the Servlet to perform front-end processing like authentication and authorizations, and centralized logging. The controller might also initialize some Java Bean that the selected JSP might require. As a summary, in this model the client sends a request to the controller, the controller decides on the JSP to forward the request, possibility initializes some Java Bean and then calls the JSP.

Model 2 architecture can also be explained in terms of model, view and controller. The model and view portions are identical to those of model 1 architecture.

The controller is responsible for controlling flow and state of the user input. The controller portion of the web tier design is generally a Java Servlet. The controller in a web tier application performs the following duties:

1. Intercepts HTTP requests from a client.
2. Translates the request into a specific business operation to perform.
3. Either invokes the business operation itself or delegates to a handler.
4. Helps to select the next view to display to the client.
5. Returns the view to the client.

![Figure 2: Model 2 Architecture](image)

Model 2 can therefore be said to follow the Model-View-Controller pattern. Model-View-Controller [KP88] is an architectural pattern that by itself
has nothing to do with web applications. It deals with separating responsibilities in a web application. By having components within a web application that have very clear and distinct responsibilities, the development and maintenance of an application can be made more efficient.

2.1 Advantages of Model 1 over Model 2

The model-1 approach just uses JSPs and JavaBeans. There are no Servlets. According to some developers this is a relief because managing different components all for one page can be very hectic and requires too much knowledge of Java. The Model-1 architecture might be the best decision for smaller applications that have simple page navigation, no need for centralized features, and are fairly static. The simplicity of the approach results in fast prototyping and deployment, a need for many companies.

2.2 Advantages of Model 2 over Model 1

In the model-1 architecture, the JSP selecting the next page often requires Java code and hence the Model-1 does not really succeed in separating presentation from business logic. Invariably, Java code becomes added into the Java Server Pages to produce what in programmer’s lingo is called spaghetti code. The result being: long pages tough to read and even tougher to debug.

With the introduction of a controller Servlet that provides a single point of entry and also encourages more reuse and extensibility than model-1, model-2 is definitely the approach for larger applications. With the model-2 architecture, there is a clear separation of the business logic, presentation output, and request processing. While the Model-2 architecture might seem overly complicated, it can actually simplify an application greatly. Web applications built using the model-2 approach are generally easier to maintain and can be more extensible than comparable applications built around the Model-1 architecture.

2.3 The Old BRI Database

The choice of architecture for this project has been greatly influenced by why this project was actually taken up. BRI already has a database running which uses Dynamic HTML with client and server side JavaScript to provide
the front-end. The choice of DHTML and JavaScript as the preferred languages was mainly due to its ease of use, the rapid development it provides and the ease with which a non programmer can code in it.

The database was designed around three years back and over the years there have been lots of modifications. Usually these modifications have been small like addition of a field or altering the field’s length etc. But sometimes major alterations and additions have been made. These changes include the changing of primary ids, the addition of a branch in the normal sequence of experiments etc.

With these modifications to the database come the modifications to the front-end. These modifications are usually need driven and usually done on very short notice. It therefore becomes hard to see what effect these modifications have on the complete design. Since code is added later, to provide new features, keeping the modularity of the application also becomes tough. Over the course of time as more and more changes are made the code becomes less and less modular. The web pages tend to include not only the presentation logic (html tags etc) but also database querying, the processing of the data received etc. The current front-end can be best explained as following:

For each of the experimental tables there is ONE html page which takes care of the addition of new experiments (CASE: NEW), the editing of existing experiments(CASE:EDIT), the saving of edited experiments(CASE:SAVE), the listing of experiments(CASE:LIST) and the searching of experiments (CASE:SEARCH).

What has been realized is that this approach tends to make the html pages real long (going into a few thousands of lines sometimes) and also hard to modify and debug.

2.4 The New BRI Database

It is with these things in mind that the group decided to design and implement a new database. A new design was made in collaboration with the project Leader Mirek Cygler, my supervisor at BRI Stephan Raymond, my honors project supervisor Bettina Kemme and a McGill Computer Science Masters Student Xueli Li.

The database has a totally new access control module which works independently from the rest of the front-end. The next big step was to choose the language to code the front-end in. It is here that my honors project comes
in to play. The project aims to develop a prototype of a subset of the new proposed database using JSP and JavaBeans.

2.4.1 Architecture Chosen

There was a choice between two architectures: model 1 (using JSP with JavaBeans only) and model 2 (using JSP and Servlets with JavaBeans). It was decided to use model 2 as the architecture of the web application. The first and foremost reason for this decision is the need to separate the different components. The design contains the following components:

- **Database Access:** This is the actual action required to be performed on the data. All these database accesses use Java Beans.

- **Presentation:** This is how the data retrieved from the database is displayed. It also is the component which interacts with the user asking questions like what data to display. These are coded as JSPs.

- **Decision Making:** What happens when the user says 'xyz'? What to do with users 'abc' input'. What page to display next? These types of questions along with preprocessing of the users request to suit database querying makes a separate component. This could be incorporated within the JSPs but that would be doing exactly what we don’t want: a big file containing both presentation and business logic. These decision making components are therefore code as Java Servlets.

3 Database Design

The database structure required by BRI contains two modules: access control and experimental data. The experimental data contains information related to the research that is carried out at the Institute. The access control module is required to restrict the access to the data depending on the privilege of the user trying to access the data.

3.1 Access Control

Within the group at BRI there is a hierarchy of roles. The roles of the researchers can be project leaders, technicians etc. The group also collaborates with other research groups on different projects/genes. It is therefore
required that the level of access granted to a user depend on his/her role regarding the data trying to be accessed. A role based access control module is therefore the best solution. With this design of the access control the administrator can design customized access control levels. More about role-based access control can be found in [SS94].

3.2 Experimental Data

This module deals with the storage of information related to the research underway. The most important requirement by the Institute was a flexible database in which fields and even tables can be added without a big hassle. The experiments conducted at the Institute consist of a sequence of sub-experiments each of which is referred to as a stage in the process of determining the structure of a protein. The structure of the database that was
decided can best be explained with the help of an entity-relationship diagram [C76]. (See figure 3).

In my project I concentrated on a simplified portion of the experimental data. The access control module has not been implemented at all. The way the prototype is implemented, adding the access control module will be easily feasible. The complex relationship of the crystallization stage was also simplified. Figure 4 shows the part of the database structure that has been implemented in the prototype implemented as my project.

Figure 4: Implemented Entity Relationship Diagram
3.3 Tables in the Database

The next stage in the implementation was the creation of the required tables in the database. Important things to consider were the chaining structure of the experiments and how one stage follows the next. Although most experiments are conducted in the order indicated by the entity-relationship diagram sometimes a stage or multiple stages in the chain can be skipped. The translation of the entity-relationship diagram to the relational schema in the database took care of this. Instead of creating the many-to-many relations between the sub-experiments as separate tables it was chosen that one table be created which just helps in linking the new sub-experiment to a previous sub-experiment. The database tables can be categorized into three types: access control, workflow tables, and sub-experiment tables. Figure 5 shows the structure of the tables implemented in this project.

3.3.1 Access Control Tables

For the project implemented this portion is greatly simplified. There is a users table which stores information about the users with access to the database. A field inside this table defines the access level. The projectuser table provides further access control by assigning users to different projects. This is the portion of the project implemented that will be changed once the role based access control module is ready to be incorporated into the system.

3.3.2 WorkFlow Tables

These tables (experimenttypes, experiment and relation) control the sequence of stages that are possible during research on a particular protein. The experiment table contains general information common to all types of experiments. The relation table helps to link the current experiment to the experiment whose results were used in this experiment or to the experiment immediately preceding this experiment in the sequence of experiments on this particular target. The experimenttypes table helps provide constraints on the workflow. The fields of this table contain parent child pairs which indicate that the "child" experiment can follow from the "parent" experiment and never the other way round. This structure satisfied the requirement of being able to skip stages and that new stages are able to be added with ease.
Figure 5: The Database Tables with Dummy Fields
3.3.3 Sub-Experiment Tables

These are the tables which contain specific information for each stage of the process. Each stage of the procedure has its own table. The tables all contain a link to the experiment table which can be considered as a parent table for all sub-experiment entries. With this structure addition or removal of fields from a sub-experiment requires the corresponding addition or removal of the field from the sub-experiment table and the rest of the database structure does not have to be changed.

Sub-experiment tables can have as many fields as are required. For the purpose of this project dummy fields were added into these tables.

4 Web Application Structure

Figure 6 depicts the structure of the web application. It depicts all the files present in the web application and how files are related to each other. Such a diagram is very important for the administrator for future additions of modules to the application.

4.1 Login Module

The login module is different from the rest of the web application since it uses model 1 architecture. The reason for doing this was the absence of access control from the implemented project. It is thought that a separate access control module will be built and added to this web application. Because of this there was no need for a model 2 implementation of the login procedure as that would be managed by the access control.

4.2 Servlets

The web application contains three Servlets: Dispatcher Servlet, Controller Servlet and Displayer Servlet. Although one Servlet could have been used to control all possible requests it seems better to have different Servlets specialized in handling specific kind of requests. The Dispatcher Servlet deals with requests as soon as the user is logged on. These can include list, search and add actions. The Controller Servlet deals with addition of data into the database. The displayer Servlet deals with processing requests like viewing, updating, editing and deleting of individual records in the database.
4.3 Beans

Apart from some helper beans to provide basic utilities to the JSPs the web application contains specific beans for each of the table in the database. These beans are responsible for the business logic associated with the particular table. For a sample bean see the "a typical bean" section.

4.4 JSPs

These are the presentation of data to the user. The data can be static data as well as data generated using database access through the java beans. The JSPs contain minimum java code and are responsible for the look of the web application. Because of this separation of the processing of the data from the presentation it is possible to change the presentation easily and efficiently without changing what the application actually does.

When the user accesses the application he/she is taken to the login JSP. Once the user enters the login id and password the doLogin bean checks with the database if the information is valid. If so the user is taken to the welcome JSP. From here the user can choose to insert, list or search the database. If the user selects search, the Dispatcher Servlet redirects the user to the searchcriteria JSP. Once the criteria for the search are provided the Controller Servlet initializes the corresponding table bean and then redirects the user to the results JSP. The result JSP utilizes the initialized table bean to access the data.

If the user decides to view an entry of the database control is first given to the Displayer Servlet which decides on which bean to initialize. The user is then redirected to the view JSP which displays information retrieved by the initialized bean.

For the addition of an entry the control first passes to the Dispatcher Servlet. The dispatcher Servlet decides whether the insert is experimental data or non-experimental. Accordingly it redirects the user to either add or expadd JSPs respectively. Once the data has been entered the Controller Servlet initializes the corresponding bean for the table in which the insert is being made and then redirects the user to the insert JSP.
Figure 6: Web Application Structure
Each of the tables in the database has its own java bean. The responsibility of this bean is to provide information to the JSP such as number, name and types of fields in the table, insertion of a tuple into this table, query of this table etc. When the web application administrator wants to add a new sub-experiment it is this type of bean that the programmer has to code for the newly added table to the database. Below is sample code for a typical table bean:

```java
package tables;
import java.io.*;
import java.sql.*;
import java.util.*;
public class plasmid {
    private String transid; // varchar(20) NOT NULL,
    private String plasmid_name; // text NOT NULL,
    private String tag; // text,
    private String cloning_strategy; // text,
    private String expression; // text,
    private int solubility; // int4,
    private String errMsg; // string
    private ResultSet temporaryResult; // resultset
    private String where; // string

    /*
     * The constructor of the bean initializes all the local variables.
     */
    public void plasmid(){
        transid="";
        plasmid_name="";
        tag="";
        cloning_strategy="";
        expression="";
        solubility=0;
        errMsg="";
        where ="";
    }
}
```
This method is used extensively to provide the dynamic nature to the database. Instead of hard coding the jsp pages with individual fields of each table this method can be coded into each table bean. The JSP now only has to call this method to know which are the fields belonging to this particular table. A great advantage of this approach is that when the database table is changed (addition of new field, deletion of existing field or an update on the type of a field this change has to be only made to the Individual bean and not to the JSP. Hence we attain maximum presentation and processing separation.

```java
public Vector getFields(){
    Vector fields = new Vector();
    fields.addElement("transid");
    fields.addElement("plasmid_name");
    fields.addElement("tag");
    fields.addElement("cloning_strategy");
    fields.addElement("expression");
    fields.addElement("solubility");
    return fields;
}
```

With each field a certain type is associated. Also there are certain constraints to be satisfied. For example some of the fields might be automatically generated (primary ids) and we don't want the user to mess with those. Also differentiating between required and not required fields is of prime importance since mostly not required fields don't have to be filled and the required fields (usually primary ids, or linking values) are needed to have a successful insert. I have used a very simple approach to overcome this constraint requirement. I use strings delimited with underscores. The first part of the string tells the JSP whether this field is a text field or an integer field. The second part tells the JSP whether to display the field and if to display whether to point out to the user that this is a required field. The third part of the string is the maximum size to take as input.

/*
public Vector getTypes(){
    Vector types = new Vector();
types.addElement("text_dontdisplay_20");
types.addElement("text_required_20");
types.addElement("text_notrequired_30");
types.addElement("text_notrequired_30");
types.addElement("text_notrequired_30");
types.addElement("int_notrequired_20");
types.addElement("int_notrequired_20");
    return types;
}

/*
Since the bean here is a sub-experiment bean it has considerably
greater amount of processing to do than a non experiment bean. The
reason for this is the need to link the current sub-experiment being
added to its parent. This method is used to enter the actual plasmid
specific data into the database.
*/

public String insertExp(Vector values) throws ClassNotFoundException {
    try{
        Integer temporary;
        transid=(String)values.elementAt(0);
        plasmid_name=(String)values.elementAt(1);
        tag = (String)values.elementAt(2);
        cloning_strategy= (String)values.elementAt(3);
        expression = (String)values.elementAt(4);
        temporary = Integer.valueOf((String)values.elementAt(5));
        solubility=temporary.intValue();

        Connection db;
        // Load the driver
        Class.forName("org.postgresql.Driver");
        db = DriverManager.getConnection("jdbc:postgresql:brimsg",
"brimsg","brimsg01");
        Statement sql = db.createStatement();
        String insert="Insert into plasmid(transid,plasmid_name,"+"tag,

cloning_strategy, expression, solubility) values(';
insert += transid+"','"+plasmid_name+"','"+tag+"','"+
cloning_strategy+"','"+expression+"','"+solubility+"')
sql.execute(insert);
db.close();
}
catch (NumberFormatException num){ //restrict user to integer input
boolean result=cleardb();
if(result)
    return "One of the fields has to have an integer value."+
    "<BR>Error: "+num.getMessage();
else
    return "One of the fields has to be an integer value."+
    "<BR>Error:"+num.getMessage()+"<BR>Couldn't delete upper
    hierarchy. "+"Please report the transid to programmer.''+errMsg;
}
catch (SQLException e) {
    //clear the database of records of this addition(experiment and relation
    boolean result=cleardb();
    if(result)
        return "Could not insert into plasmid<BR>"+ "Error from
        postgresql is "+e.getMessage();
    else
        return "Could not insert into plasmid<BR>Error from
        postgresql" + "is:"+ e.getMessage()+"<BR>Couldn't
delete upper hierarchy "+ "either: Please report the
        transid to programmer.<BR>"+errMsg;
}
return "success";
}

/*
This method implements a small recovery method. If the above insert fails
we have to get rid of the relation and experiment entries in the
respective tables. As an alternative, a transaction mechanism could be
used which would do the series of inserts and would commit only if the
above insert message was successful
*/
private boolean cleardb() throws ClassNotFoundException {
    Connection db;
    // Load the driver
    try {
        Class.forName("org.postgresql.Driver");
        db = DriverManager.getConnection("jdbc:postgresql:brimsg","brimsg",
                                    "brimsg01");
        Statement sql = db.createStatement();
        String delete="delete from relation where child like "+transid+"");
        sql.execute(delete);
        delete = "delete from experiment where transid like "+transid+"");
        sql.execute(delete);
        db.close();
    } catch (SQLException e) {
        errMsg=e.getMessage();
        return false;
    }
    return true;
}

/*
When the user wants to list the entries in this table we require to obtain
a dataset for the particular list. This is basically a select * with no
restrictions. The number of hits are computed by this emthod and returned
to the calling JSP for display
*/
public int getnumHits() throws SQLException,ClassNotFoundException {
    int numHits=0;
    String type="list";
    if(executeQuery(type)==1) {
        while(temporaryResult.next())
            numHits++;
        temporaryResult.close();
    } else
        numHits=666;
    return numHits;
}
/*
   Once the list query has been executed the response has to be sent to the
   user. The representation used is a Vector with vectors in it. The vector
   values contains one vector each for the hits. The tuple vector stores the
   fields of the table into the vector as objects and then inserts itself
   into the values vector. This approach again helps to restrict the
   processing for each table into the bean and not propagate changes of the
   database to the presentation layer (JSPs).
*/
public Vector getList() throws SQLException, ClassNotFoundException {
    Vector values = new Vector();
    String type = "list";
    if (executeQuery(type) == 1) {
        values.addElement("success");
        while (temporaryResult.next()) {
            Vector tuple = new Vector();
            tuple.addElement(temporaryResult.getString(1));
            tuple.addElement(temporaryResult.getString(2));
            tuple.addElement(temporaryResult.getString(3));
            tuple.addElement(temporaryResult.getString(4));
            tuple.addElement(temporaryResult.getString(5));
            tuple.addElement(temporaryResult.getString(6));
            values.addElement(tuple);
        }
        temporaryResult.close();
    } else {
        values.addElement("Could not obtain a ResultSet for the plasmid
                        table");
    }
    return values;
}

/*
   This method is similar to the getnumHits method. The only difference is that
   in this case the search is restricted by a whereClause created by the servlet.
*/
public int getnumSearchHits(String whereClause) throws SQLException, ClassNotFoundException {
    int numHits=0;
    String type="search";
    where=whereClause;
    where=where.replace('$','%');
    if(executeQuery(type)==1) {
        while(temporaryResult.next())
            numHits++;
        temporaryResult.close();
    } else
        numHits=666;
    return numHits;
}

/*
This method is almost identical to the getList method and infact should
have been coded together. This is a possible place for improvement.
*/
public Vector getResults(String whereClause) throws SQLException, ClassNotFoundException {
    Vector values = new Vector();
    String type="search";
    where = whereClause;
    where=where.replace('$','%');
    if(executeQuery(type)==1) {
        values.addElement("success");
        while(temporaryResult.next()) {
            Vector tuple = new Vector();
            tuple.addElement(temporaryResult.getString(1));
            tuple.addElement(temporaryResult.getString(2));
            tuple.addElement(temporaryResult.getString(3));
            tuple.addElement(temporaryResult.getString(4));
            tuple.addElement(temporaryResult.getString(5));
            tuple.addElement(temporaryResult.getString(6));
        }
    } else
        numHits=666;
    return numHits;
}
values.addElement(tuple);
}
temporaryResult.close();
}
else {
    values.addElement("Could not obtain a ResultSet for the plasmid
table"+errMsg);
}
return values;
}

/*
The actual method which interacts with the database to retrieve
information. Both the search and list method above invoke this method to
access the database. The method interacts with the database and returns a
resultset with the hits provided by the database.
*/
private int executeQuery(String type) throws ClassNotFoundException {
    String whereTemp="";
    if(type.compareToIgnoreCase("list")==0)
        whereTemp="";
    else if(type.compareToIgnoreCase("search")==0)
        whereTemp = where;
    Connection db;
    Statement sql;
    String query ;
    query="";
    // Load the driver
    try {
        Class.forName("org.postgresql.Driver");
        db = DriverManager.getConnection("jdbc:postgresql:brimsg","brimsg","brimsg01");
        sql = db.createStatement();
        query="select * from plasmid "+whereTemp;
        temporaryResult=sql.executeQuery(query);
        return 1;
    }
    catch (SQLException e) {

errMsg="could not get resultSet. error from database is "+
e.getMessage()+whereTemp;
return 0;
}
}
}

6 An Example JSP

The web application utilizes JSPs to display the information retrieved by Java Beans. Apart from the presentation code (utilizing HTML tags) the JSP contains calls to Java Beans to extract the information stored in the bean. A typical JSP looks like this:

Defining the header and some variables to be used by the JSP.

<html>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>VIEW</title>
</head>
<body bgcolor="linen">
</body>

Viewing entry of table: <%= request.getParameter("table") %>

The JSP will use this bean which was initialized by the servlet for it

<jsp:useBean id="tableBean" class="tables.tableBean" scope="request" />
<% String whereClause = request.getParameter("where"); %>
<% String restrict = request.getParameter("restrict"); %>

Code for displaying using HTML tags.

<CENTER>
<FORM NAME="displayAction" METHOD="get" ACTION="servlet/displayer">
<% if(restrict.compareTo("yes") !=0) 

24
Getting the information from the Java Bean and displaying it.

```java
int i=0;
javax.util.Vector fields = new javax.util.Vector();
fields = tableBean.getFields(tablename);
javax.util.Vector values = new javax.util.Vector();
javax.util.Vector tuple = new javax.util.Vector();
values = tableBean.getResults(tablename,whereClause);
String result= (String)values.elementAt(0);
```

```
.. truncated display code
...
i++;
}
else
   out.println("Could not execute list query."+result);
```

Click <A href="welcome.jsp">here</A> to go back to the start page.
7 Possible Extensions

The aim of the project was to test the feasibility of a web application written using JSPs and Java beans. The project concentrates on getting the system up and running and hence the web application as it is right now does not cover all the possibilities. There is room for many extensions to the system. I outline some of the extensions that could be made to the web-application.

7.1 Access Control Module

To simplify the implementation of the project no access control module was added to the web application. The original Entity Relationship diagram contains a role based access control mechanism. There are a couple of ways how this module could be appended to the web application with minimum changes required to the current implementation. The first way is to move all the database access into the access control module. Since all database accesses are currently performed by JavaBeans this would mean that the JavaBeans belong to the access control layer.

A second way is to apply the access control not on the database access but on the page displaying the data. In the model 2 architecture, when the Servlet decides to forward a request to a particular page, it forwards enough information which could be used by the access control module to check whether the user is allowed to access such data. This approach is easier to implement. The module could be written as a java class whose access control method is invoked by the Servlet before the response is forwarded to the JSP. Although easier to implement the approach is less intuitive since usually access control is linked to restricting access to the database.
7.2 Automated Input

With the advancement of technology more and more experiments are becoming automated. Use of robots in such experiments results in production of "flat files" that the robot can provide once the experiment is completed. These files are usually specific to the robot used and contain a certain pattern. A researcher could possibly want to extract the information (or a subset of the information) from such files and insert them into the database. Exp-DB currently provides no tool to achieve such a task.

A possible way to retrieve information from a file is to have an option, in the presentation layer, to read and insert into the database from a file. The programmer would have to write the complete read and extraction mechanism for each file pattern. Possibly a Java Bean could be used for this. When the JSP (presentation layer) receives such a request it forwards the request to the Servlet. The controller (Servlet) can initialize a bean which would read the required data from the file and insert it into the database. The main task in this extension is to write the parser for the file. Since each robot usually produces files with its own syntax, robot specific parsers will have to be written. However once a parser has been written for a particular file outputted by a robot the parser can be used over and over again to retrieve the information from the file and store it into the database.

7.3 Representation of Data

Exp-DB only provides a generic representation of the data in the database. It is easy to visualize a need for more complex representation of data (input and output). Here I provide an example how to extend the display supported by Exp-DB to provide a specialized presentation solution.

With the current use of the system each sub-experiment has to be added individually. The addition module for sub-experiments displays input fields where the lab technician is asked to insert the results of this particular sub-experiment (e.g. inserting purification data). However some sub-experiments are conducted in batches. In these cases the single procedure can produce a large set of results all of which need to be inserted into the database for analysis. An example of this is the crystallization stage at the Macromolecular Structural Group, BRI.

A crystallization experiment is conducted once purification has been done. The goal of crystallization is to produce crystals of the purified protein for
X-Ray diffraction experiments. Since it is vital to have good quality crystals a number of different settings are used to try to find the optimum settings for producing good quality crystals. A typical crystallization experiment involves placing the purified protein with different solutions at different concentrations and giving the protein time to crystallize. These solutions are kept in crystallization trays which can have up to 24 wells. Each of these wells can have several drops (3 to 6). The 24 wells can be considered as 24 sub-experiments or a typical approach would be to consider each drop of each well as a separate sub-experiment since each of these could result potentially result in a good crystal. It is therefore essential to maintain a record of what solutions and at what concentrations was used at each drop.

In Exp-DB entering the information that relates to one crystallization experiment would require to enter at least 24 sub experiments, meaning 24 new web-pages that the user has to fill. Clearly, this is a specific problem that has to be solved in a very specific way. Exp-DB could be extended by the administrator by customizing the presentation of individual sub-experiments. For instance in this case a new presentation scheme could be used to insert crystallization sub experiments. Since most of the time wells contain same solutions at varying concentrations use of default values and displaying multiple entries to insert at the same time could ease the task. To complete this task the administrator would have to write a JSP (VIEW) which would now have the new presentation logic. The controller for the addition of experiments could then be made to redirect all requests for addition of crystallization sub-experiments to this new JSP. The bean representing the crystallization sub experiment will also have to be modified to incorporate that now one request from the user involves insertion of multiple sub-experiments.

7.4 Querying

The current implementation of the web application performs queries over one table at a time. Using this querying capability the usual queries would be for instance: select the users involved in a project, give me all experiments conducted by a given user ‘xyz’. However it is considered vital for the web application to have querying capabilities to span multiple tables. Also being able to define custom queries and then storing them is a feature much needed in the system.

To allow users to generate nearly arbitrary queries over the entire data set is not an easy task. The query engine cannot be absolutely independent of
the relational database schema, and might have to be extended if the schema significantly changes its structure. The interface for such a generic query system would be complex to use for users that have little understanding of how data is structured.

Custom queries that are most required by the researchers could be encoded as individual JavaBeans. These could then be invoked whenever the user wants to perform that particular query. Adding new custom queries then involves writing the corresponding Java Bean.

Another approach could be to go ahead and provide the user the option to create and store custom queries. These queries could themselves be stored in a table within the database. A user would retrieve a saved query which would then itself retrieve the data relating to that query.

7.5 Navigation

Currently one can view only the entries related to a particular sub-experiment. It is important for the web application to have a tool to be able to view the sequence of experiments leading to a particular sub-experiment. Such a tool should have the following features:

- Navigation through an experimental workflow. Starting with the view of an initial experiment, the user can navigate to the information about its child experiments. From there he/she can move back to view the parent experiments, or move forward to see the child experiments of the child. In this environment, at any given time, the user sees the information of one experiment and links to its parents and children.

- Viewing an entire experiment workflow. On a higher level, the user can view an entire workflow, or a sub-workflow starting with an initial experiment (for instance, a target experiment). The view is in form of a graph where the nodes represent the individual experiments. They are tagged with the type of the experiment and the key of the experiment. In a similar way, given a specific experiment, the user can activate the workflow in which the experiment is involved, that is, the graph of ancestors and successors of this experiment. By clicking on one of the nodes, the user can view the detailed information of this experiment.

The first feature for the navigation requires retrieval of the parents and children of a given sub-experiment. This information is stored in the ”relation” table within the database. To get the parents a query of the form \texttt{select}
parent from relation where child like "transid of sub-experiment" can be executed. For retrieval of the children a query of the form select child from relation where parent like 'transid of sub-experiment'.

The second feature, navigating through an entire work flow, needs recursive calls to the parent selection and child selection queries. An example follows:

Suppose our workflow looks like the following

![Figure 8: Hypothetical Graph of a Workflow](image)

Suppose the user clicks on sub-experiment D and wants an entire workflow. The first step is to get all the successors. So the children selection query is applied. This returns the id of the sub-experiment F. Now a selection of children of F is made. This returns sub-experiment G. A selection of children of G results in no results hence we know that this is the latest stage in the sequence. At this point we have a graph of the form:

![Figure 9: State of graph after completion of retrieval of successors](image)

The next stage is to get the ancestors of the sub-experiment D. The query to get the parents of D is applied. This results in the sub-experiment C. A
query on the parents of C returns A and B. A query on the parents of A or B yields no result. At this point we have a graph of the form:

It can be seen that the only sub-experiment missing are the siblings of D itself (sub-experiment E). Similarly the siblings of the ancestors of D would not be displayed either. If the navigation tool is to provide a complete picture of the workflow then we need to get the tree of the siblings too. This can be done by applying the "find successors" algorithm described above for each of the parents of the sub-experiment D. However this can result in a substantial amount of queries since the tree could be particularly dense at the other branches. Depending on how the navigation tool is being coded this feature of displaying the siblings and their successors could be added.

The above algorithms only deal with the retrieval of the information of the database. The next task is to provide a user friendly display for the graph. Two approaches are possible. The first approach is to use a JSP for the display. Since the graph is dynamically generated and because HTML does not provide the greatest of designing tools, aligning and presenting the graph as a JSP will be a tough job. Another approach is to use a java applet for this purpose. A java bean could be used to retrieve all the data from the database and forward this to a java applet which would display the graph. The nodes of the graph can be clickable links leading to a JSP displaying the details of that particular experiment.
7.6 Web Application Structure

The current structure of the web application utilizes three Servlets. I used this separation of controllers since each of these Servlets can be considered to be dealing with a separate module of the web application. The dispatcher Servlet deals with requests from the user coming from the home page. The controller Servlet controls the addition of data into the database and the displayer Servlet only worries about displaying the data. Although this creates a nice break of functionality of the web application there is no reason why one controller (Servlet) could not be used to control the entire application.

Other possible extensions to the application could be stronger and dynamic type checking. Currently the java beans test before entering into the database whether the input from the user matches the data type required by that particular field. This information is currently hard coded into the table beans. An approach to have this information dynamic would be to store the restrictions on the fields in the form of META information into the database. Before inserting into the database tables the restriction table could be queried and the field matched with the date being inserted.

Since the current application is only a prototype the presentation is not the most pleasing the eyes. The presentation could be greatly improved by using style sheets and other formatting techniques.

8 Conclusion

This report was aimed to explain in greater detail the development of an application that can be used by research laboratories for data management. Modularity and separation of functionality was the prime concern. Using well established software engineering principals it is seen that the system can provide the flexibility and ease of upgrade and modification required by so many of the research laboratories for their book-keeping needs.

9 References


Communications, 32(9), 1994.
