XML-Based Specification and Automatic Code Generation for Easy Customization of View in the GUS WDK Framework

by

Nivedita P. Kaluskar

(Under the direction of Dr. Eileen T. Kraemer)

Abstract

An automatic custom view generation method for the Genomics Unified Schema (GUS) framework has been designed and implemented in this thesis work. The Genomics Unified Schema is a database schema and application framework used to store, integrate, analyze and present functional genomics data. The *Cryptosporidium* Genome Resource (CryptoDB) [18], [19], [61] is a project developed at the University of Georgia under the GUS framework. CryptoDB offers a number of queries that a user can ask of the database and uses the GUS Web Development Kit (WDK) to generate a default view for the database. Although the earlier WDK offered an easy way of generating a view for any given data model, the view generated was too closely tied to Model and Controller.

The method for generating a custom view enables a more clear separation between the Model, View and Controller in the MVC type application and allows on-the-fly generation of custom views. It uses an XML-based custom view specifications file containing the specifications a user is allowed to make in order to generate a custom view. This gives the user much more flexibility in designing a custom view and allows the View to be less closely tied to the Model and the Controller.

Index words: CryptoDB, GUS, WDK, XML, Model, View, Controller, Custom View

XML-Based Specification and Automatic Code Generation for Easy Customization of View in the GUS WDK Framework

by

Nivedita P. Kaluskar

Bachelor of Engineering, R.V College of Engineering, India, 2003

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

2005

XML-Based Specification and Automatic Code Generation for Easy Customization of View in the GUS WDK Framework

by

Nivedita P. Kaluskar

Approved:

Major Professor:	Dr. Eileen T. Kraemer
Committee:	Dr. John A. Miller
	Dr. Jessica C. Kissinger

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2005

DEDICATION

To My Family

Acknowledgments

I would like to take this opportunity to express my heartfelt gratitude to my advisor Dr. Eileen Kraemer for her valuable guidance and continuous support throughout my Master's program. It has been a rich learning experience and rewarding hands-on research exercise as I hoped my Master's would be. I would also like to thank Dr. Jessica Kissinger and Dr. John Miller for their time and guidance during the many fruitful meetings and discussions during which the seeds of this research were sown. Finally, it has been a pleasure working and studying with everyone at the Computer Science Department at UGA. Thanks everyone for making it a great experience.

Table of Contents

List of Figures

CHAPTER 1

INTRODUCTION

As the prevalence of web-based computing continues to increase, the focus of interaction design is shifting from traditional desktop applications toward web-based applications. A number of web application frameworks are available today, including Jakarta Struts [20], Ruby on Rails [25] and Spring [53], to name a few. These frameworks provide a complete development environment for creating a web application. They typically rely on a Model-View-Controller architecture, which aims at separating the Model, View and Controller parts of an application. The Model is the core of the application. It maintains the state and data that the application represents. When significant changes occur in the model, the model updates all of its views. The Controller is responsible for processing user requests and mapping requests to views. A user request could be a query to the database for retrieving data, for example. The View is the user interface, which displays information about the Model to the user. Web development frameworks interact with standard data access technologies such as the Java Database Connectivity (JDBC) [55] API and Enterprise JavaBeans (EJB)[56] to provide the Model and with standard presentation systems and technologies such as JavaServer Pages (JSP)[34] and JSP Standard Tag Library (JSTL)[35] to provide the View for an application.

The field of Bioinformatics is a good forum for the development of such web applications due to the large amount of biological data that needs to be stored, analyzed and viewed in a suitable form and rendered to users worldwide. The *Cryptosporidium* Genome Resource (CryptoDB) [18], [19],[61] is a project developed at the University of Georgia under the Genomics Unified Schema (GUS) [60] framework. The Genomics Unified Schema is a database schema along with associated application framework and has been designed to store, integrate, analyze and present functional

genomics data. The GUS Application Framework has an object-relational layer and a Plugin API which is used to create data loading programs for diverse data sources. The GUS Web Development Kit (WDK) is a rich environment for efficiently designing query-based websites [60].

The *Cryptosporidium* genome resource offers a number of queries that a user can ask of the database and uses the GUS Web Development Kit to generate a default "view" for the database by representing the queries in a model file and using JSP technology to generate the view for the data model.

Although version 1.3 of the WDK (the latest version at the time the work presented in this thesis began) offered an easy way of generating a view for any given data model, the view generated by the WDK was a direct representation of the model and was too closely tied to it. For example, a model may have different queries organized in different question sets, and each of the questions may further have parameters. The default view generated by the WDK for such a data model required the user to select a question from a particular question set using a pull-down menu, and then displayed the question on a second page, in which the user had to enter the question parameter values, before actually submitting the question to retrieve a summary of results.

The main disadvantage of such a view is that it requires too many steps and selections by the user to get to the results page. Such a view also hides information and thus does not give the user a clear idea of what he/she can query the database for and the nature of results that may be retrieved by a particular question.

One solution to this is to have a view that renders all the questions and their parameters on a single front page, so that the user knows exactly what his options for querying the database are, and the nature of results that may be retrieved from a particular query. Such a "flattened" out view would then require just one click to get to the results page, without having to go through a large number of selections and steps.

Though good from the usability standpoint, such a view was not very easy to create and required a good knowledge and understanding of the various technologies used in building the application. The Struts web application framework which is used in this project uses the Model-

View-Controller architecture to organize the site. The MVC architecture tries to separate the Model, View and Controller parts of an application and ideally any modification of the View should not entail with it a modification in the Model or the Controller. However, this was not the case, and in order to change the default structured view to a more usable flattened out one, it was necessary to make modifications in the Controller as well. Thus a simple change in the View required changes in multiple places, making the process of creating a “custom” view extremely time-consuming and complicated.

In order to make the process of creating “custom” views for a site easy and less time-consuming, an automatic view generation method based on custom view specifications has been designed and implemented and is the main focus of this thesis work. This method allows a more clear separation between the View and the Controller parts of an application, and thus allows changes to the View without requiring any modifications to the Controller or the Model. The chapters that follow discuss this method in greater detail, and show how it can be used to easily generate custom views.

The rest of the thesis is organized as follows. Chapter 2 describes the related work and the various web application frameworks available and a more detailed description of the Jakarta Struts framework used here. Chapter 3 describes the implementation details and the method used to generate a custom view. Next, Chapter 4 shows some evaluation results of the method implemented and how it can be used to create custom views. Finally, Chapter 5 discusses the future work.

CHAPTER 2

RELATED WORK

Several frameworks are available to create web applications. Spring [53], Ruby on Rails [25], Struts [20], etc. are some of these frameworks, and all are aimed at enabling easy creation of web applications which have databases at the backend. The Struts framework has been used to design the application here. A brief background of these frameworks as well as the WDK has been provided below.

2.1 THE APACHE STRUTS WEB APPLICATION FRAMEWORK

The Struts framework is designed to provide an open source framework for building Java web applications. The framework has a control layer and uses standard technologies including Java Servlets [27], JavaBeans [28], ResourceBundles [29] and XML [30], and various Jakarta Commons packages [31]. Applications developed using Struts generally follow the Model 2 approach [21], which is a variation of the classic Model-View-Controller (MVC) design paradigm [22]. Both the Model 2 approach and the MVC paradigm are discussed in the next section.

An application based on the MVC design paradigm can be developed using the Struts framework in the following way: To create the Model, standard data access technologies such as JDBC and EJB, as well as many third-party packages including Hibernate [26], iBATIS [32] or Object Relational Bridge [33] can be used. The View can be created by using several presentation systems such as JavaServer Pages [34], including JSTL [35] and JavaServer Faces (JSF) [36], as well as Velocity Templates [37], and XSLT [38]. For the Controller, the Struts Controller component can be used.

2.1.1 THE JSP MODEL 1 AND MODEL 2 ARCHITECTURES

JavaServer Pages technology allows developers to create dynamic and information rich websites rapidly and easily. JavaServer Pages technology extends the Java Servlet technology. Servlets are server-side modules and are platform-independent. JSP technology and Servlets allow separation of logic from display [39].

JSP uses XML-like tags which contain the logic that generates the content for a page. The JSP page accesses the application logic objects stored on the server using these tags. JSP separates the page logic from its presentation, thus making it faster and easier to build Web-based applications.

The JSP technology offers two approaches for building applications. The approaches are known as the JSP Model 1 and Model 2 architectures and differ in the location at which most of the request processing takes place [21]. These architectures are discussed in the following sections.

The Model 1 Architecture:

In this architecture, request processing is handled by the JSP page alone and it sends the response back to the client when the processing is done. The control flow in this architecture, depicted in Figure 2.1, is as follows:

1. The user first submits a request through the web browser. The request is sent directly to a JSP page.

2. The JSP page then communicates with JavaBeans to access data.

3. JavaBeans access the data sources and get the requested data. JavaBeans represent the data objects and contain most of the Java code. Thus a JavaBean is like an object in any object-oriented environment and is comprised of two primary things: data and methods that act on the data. The data part of a bean completely describes the state of the bean, whereas the methods provide a means for the bean's state to be modified and for actions to be taken accordingly.

4. The beans then send the requested data back to the JSP.

5. The JSP page then selects the next page to be displayed and sends the response back to the browser.

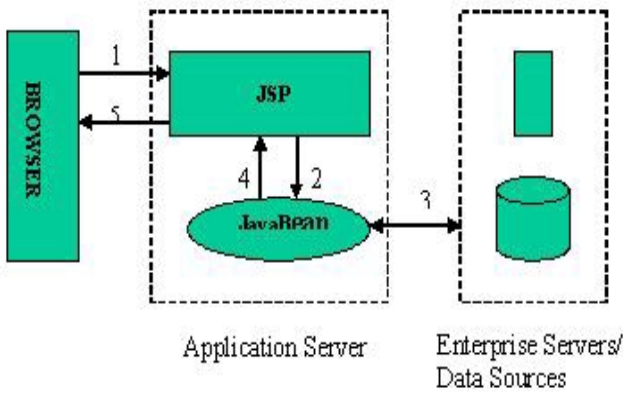


Figure 2.1: The JSP Model 1 Architecture(adapted from [21])

Thus there is a separation between presentation (view) and content (model), since all data access is performed through beans. This architecture is more suitable for simple applications than for complex applications.

The Model 2 Architecture:

The Model 2 architecture is as shown in Figure 2.2. This model combines the use of both servlets and JSP and uses the predominant strengths of both these technologies. It uses JSP to generate the view or the presentation layer and the servlet which acts as the controller, to perform any request processing and to create beans or objects used by the JSP. The servlet also decides which JSP page to forward the request to depending on the user's actions. Thus there is no processing logic in the JSP and it is only responsible for retrieving any objects or beans that may have been created by the servlet. The control flow in this architecture seen in Figure 2.2, is as follows:

1. The user request is first intercepted by a servlet, referred to as a controller servlet.

2. This servlet handles the initial processing of the request and creates any beans that may be used by the JSP. As shown in Figure 2.2, the request is never sent directly to the JSP page. This allows the servlet to perform front-end processing.

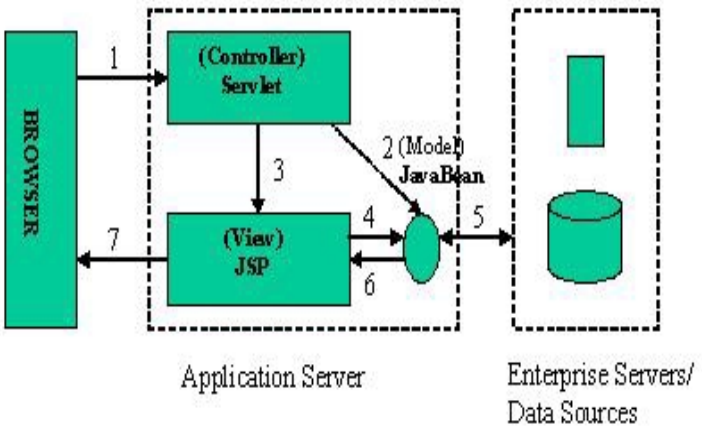


Figure 2.2: The JSP Model 2 Architecture(adapted from [21])

3. Once request processing is complete, the servlet directs the request to the appropriate JSP page.

4. The JSP page communicates with the JavaBean components to access the data.

5. JavaBeans represent the data objects and access the data sources to get the requested data.

6. The beans then send the requested data back to the JSP.

7. The JSP sends the response back to the browser by displaying the results in an appropriate view.

Thus there is a more clear separation between request processing (controller) and presentation (view) in this architecture than the previous model 1 architecture. This architecture is more suitable than Model 1 for complex applications.

2.1.2 The MVC Design Pattern

The main idea behind design patterns is to extract and abstract out high level interactions among objects and reuse their behaviors from application to application. In the MVC design pattern, a central Controller handles the application flow. The Model represents the application specific logic.

Control is forwarded to the appropriate View through the Controller using a set of mappings from a configuration file. This provides a clear separation between the Model and the View, thus making applications easier to create and maintain [22].

The Model:

The Model portion of an MVC-type system contains the application-specific or business logic of the system. The Model is implemented using a set of JavaBean components. JavaBeans contain properties which are used to retrieve information from the data source components of the system which could be a database, an Entity Enterprise JavaBean, etc.

The View:

The View portion of a Struts application is generally constructed using JavaServer Pages (JSP) technology. JSP pages can contain static HTML as well as standard action tags to include dynamic content in the page. The standard action tags are used to access the JavaBean components and include tags such as

```
<jsp:useBean>, <jsp:getProperty>, etc.
```

In addition to the built-in actions, the developer can also define his/her own tags, and these are organized into “custom tag libraries”.

The Controller:

The Controller portion of the application receives requests from the client, decides the application-specific logic function to be performed, and then forwards control to an appropriate View component. In Struts, the primary component of the Controller is a servlet of class Action-Servlet. The Controller uses a set of action mappings specified in the struts-config.xml to map an incoming request URI to a fully qualified class name of an Action class and forwards control to the appropriate View. The Action class is subclassed from org.apache.struts.action.Action.

2.1.3 The Struts Control Flow

The control flow in a Struts application is as follows:

```
1 <action-mappings>
2 <action
3    path="/showQuestion"
4     type="org.gusdb.wdk.controller.action.ShowQuestionAction"
5    scope="session"
6     name="questionSetForm"
7    input="/questionSets.jsp">
8    <forward name="show_question" path="/question.jsp"/>
9 </action>
10 </action-mappings>
```

Figure 2.3: Action-mappings in struts-config.xml

1. On initialization, the controller parses a configuration file called struts-config.xml and deploys other objects of the control layer using it. The configuration file contains the Action Mappings [org.apache.struts.action.ActionMappings] for an application.

2. The Struts controller servlet uses these ActionMappings to route HTTP requests to the components in the framework. Requests are either forwarded to JavaServer Pages or Action subclasses. The controller turns the HTTP requests into application actions using the mappings provided in the struts-config.xml.

For example, consider a simple HTML form containing user input fields to input an email address and a password. When the user submits this form, the appropriate action class (according to the mappings defined in the struts-config.xml) is called and its execute method is invoked. This method retrieves the user input from the form fields and performs a check to determine if the data entered is valid and accordingly returns a value which is mapped in the struts-config.xml to an appropriate JSP page.

An example of an action mapping in the struts-config.xml is as shown in Figure 2.3.

Thus, as shown in line 3 in Figure 2.3, the path or the request URI is /showQuestion. The action mapping specifies the Action subclass ShowQuestionAction to act upon when the request is sent. In addition, the action mapping may also contain other properties such as scope, the name of the form bean and the input JSP page from which the request is sent. The form bean extends the ActionForm class and represents the data submitted by the user. The Action object, which extends the Action class, can handle the request and respond to the client (usually a Web browser) or indicate that control should be forwarded elsewhere.

3. In a Struts application, most of the business or application-specific logic resides in JavaBeans. An action calls the properties of a JavaBean, but requires no knowledge about its internal workings. The ActionForm is used to represent user input data as well as data retrieved from a request. When the request is used to submit user input data, the controller servlet passes the ActionForm bean instance to the Action object which then forwards the data to the application logic. When the request is used to create an input page, the Action object populates the form bean with any data required by the input page.

Thus the six basic steps involved in using Jakarta Struts are the following:

In Struts, the data contained in a form (HTML form) is submitted to a URL of the form blah.do. The blah.do is an address that is mapped to the corresponding action class (BlahAction.java) in the struts-config.xml. This class contains the execute method which handles the incoming request and populates the properties of an ActionForm bean with the incoming form data. The Action object then invokes application and data-access logic, and places the results in normal beans that represent the results of the business logic and contain getter and setter methods to access the data. These beans do not extend any class, unlike the form beans which extend the ActionForm class and are stored in request, session, or application scope. A request scope object is an object containing the client request information and provides data such as parameter name, value and attributes [48]. A session-scope object exists for a session and is created at the beginning of each new session and is destroyed at the end of the session. Similarly an application scope is the context for the current application. There is one context per web application [48]. They represent the output of

a computational process and not the input as represented by the form beans. The Action uses mapping.findForward to return a condition, and the mappings in struts-config.xml are used to map the returned condition to various JSP pages. The request is forwarded to the appropriate JSP page, and the JSP action tags such as the bean:write tag can used to output properties of the form bean and results beans [23].

The six basic steps are summarized as follows:

1. Specify in struts-config.xml: The Action classes which handle requests, the URLs and form beans need to be specified in the struts-config.xml. The server has to be restarted whenever this file is modified, since it is read only when the Web application is first loaded.

2. Define the form bean: This bean extends the ActionForm class and represents the data submitted by the user. The form bean gets automatically populated with the input data when the form is submitted.

3. Create result beans: Struts uses the MVC architectural style in which the application-specific logic creates the results that are presented by the JSP pages. Results beans need to have getter and setter methods, but need not extend any particular class and also do not have to be declared in struts-config.xml. Beans are used to transfer the results from one layer to the other. The beans are stored in request, session, or application scope with the setAttribute method of HttpServletRequest, HttpSession, or ServletContext, respectively. The HttpServletRequest is an interface that extends the ServletRequest interface and provides request information for HTTP servlets. The HttpServletRequest object created by the servlet container is passed as an argument to the servlet's methods [45]. With the HttpSession interface, a user can be identified across more than one page request and information about a user can be stored. This interface is used by the servlet container to create an HTTP session between a client and a server. This interface also allows servlets to view and manipulate information about a session [46].The ServletContext interface defines a set of methods which a servlet uses to communicate with its servlet container. The ServletContext object is contained within a servlet configuration object and is used by a servlet container to pass information to a servlet during initialization [47].

4. Create an Action object: The struts-config.xml contains mappings that designate Action objects to handle requests for various URLs. Action objects invoke the appropriate data-access logic and store the results in beans. The struts-config.xml then uses its mappings to decide which JSP page should be displayed.

5. Create an HTML form: The user submits his/her request through an HTML form on the web page. This form specifies the action that must be invoked when the form is submitted. This action is in the form of *.do (e.g., blah.do) and is mapped to a Controller Action class by the struts-config.xml (e.g., BlahAction.java for a blah.do specification in the HTML form).

6. Display results in JSP: JSP pages usually use the jsp:getProperty, the jsp:bean:write or the JSTL c:out tag to output the results from the beans. JavaBeans interact with the data source and get the requested data. This data is displayed on the JSP page using the jsp:getProperty, or the jsp:bean:write tags.

Pros and Cons: Struts uses JSP technology for the View and the tag mechanism offered by JSP allows code re-usability and helps to abstract out Java code from JSP. A single XML based configuration file separates the Java code from the configuration details. Thus changes can be made to the configuration without affecting the Java code. Struts has a higher learning curve. It requires knowledge of JSP and other technologies used by Struts in order to develop an MVC based application.

2.2 Ruby on Rails

Ruby is a pure object-oriented programming language with a clean syntax. Rails is an open source Ruby framework for creating web applications that have a database at the backend [41]. Rails takes full advantage of the simplicity of Ruby. What makes this framework more efficient and much faster to learn and use than other frameworks is the efficient use of the Ruby programming language. The two guiding principles of Rails are: less software and convention over configuration [41]. Less software means writing fewer lines of code and this is possible due to the use of Ruby.

Convention over configuration implies that Rails does not use any XML-based configuration files, but uses programming convention along with reflection and discovery.

A typical Rails application directory structure: Rails has a helper script, which if used to create an empty application, creates the entire directory structure for the application. Rails knows where to find things it needs within this directory structure and does not require any configuration files. The directory structure contains the following subdirectories:

- The controllers subdirectory contains the controller classes and Rails looks for them in this directory. The controller is responsible for handling requests from the user.

- The views subdirectory contains display templates which are used to represent the data of an application and are returned to the user in the form of HTML.

- The models subdirectory contains the model classes which wrap the data stored in the applications database.

- The helpers subdirectory contains helper classes which are used to assist the Model, View and Controller classes.

Controllers and URLs: Controller classes are responsible for handling requests from the user and map the URL of the request to a controller class and a method within it. A controller class can be created using the Rails helper script. For example, in order to create a controller class called MyTest, the following command could be given:

```
ruby script\generate controller MyTest
```

This creates a file named

```
my_test_controller.rb
```

containing an outline definition for the class MyTestController which looks like:

```
class MyTestController <ApplicationController
end
```

In order for the application to be opened in a browser, the URL must contain the name of the controller class. Rails tries to find an action named index in the controller. Thus defining the index method in the following way will render the text Hello World in the browser.

```
class MyTestController <ApplicationController
         def index
                  render_text Hello World
         end
   end
```

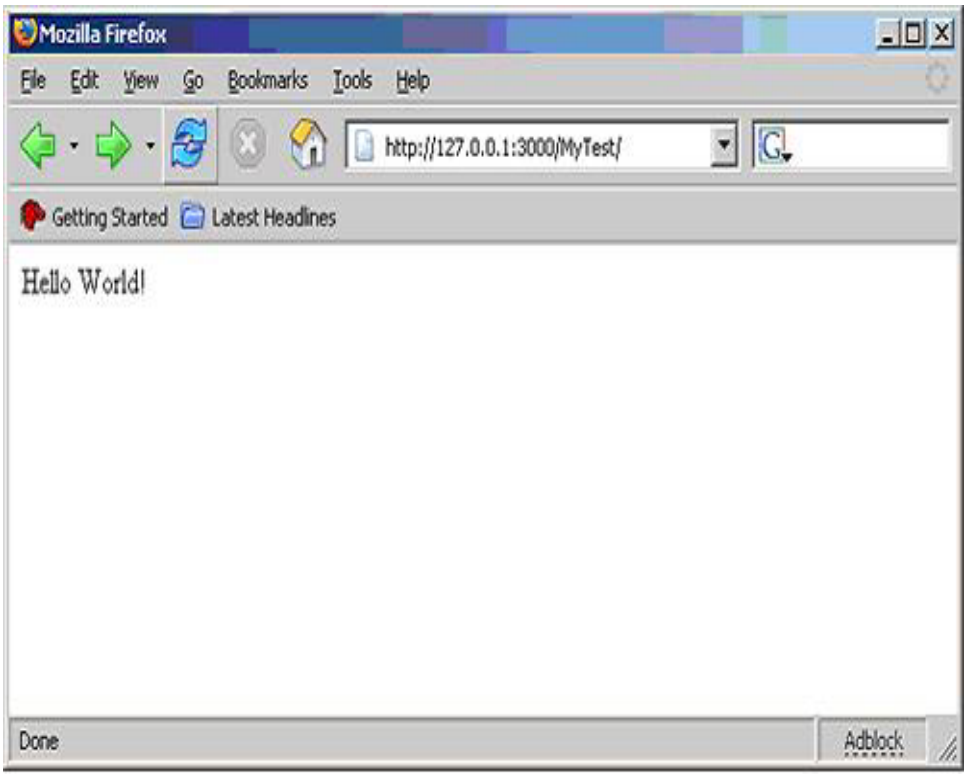


Figure 2.4: Creating the Controller [41]

Creating the Model: As with the Controller classes, the Rails' helper script can be used to generate Model classes also. Thus to create a model class called Test, the following command can be given:

```
ruby script\generate model Test
```

This creates a file named test.rb containing an outline definition for the Test class. The empty class looks like follows:

```
class Test <ActiveRecord::Base
end
```

This class definition is the Test business or application object that Rails maps to a tests table in the database. Rails also dynamically populates the Test class with methods for accessing the rows and columns in the tests table. In order to have a dynamic connection between the Test class and the tests table in the database, it is necessary to create a test controller with actions to manipulate the table in the database with operations like create, read, update, and delete. The controller class can be created by using the Rails helper script as described before, with the addition of the line scaffold:test in it. Thus the Test Controller class would look like:

```
Class TestController <ApplicationController
          scaffold:test
end
```

This single line of code enables all the database operations including create, read, update and delete.

Creating Actions and Views: The default view generated by Rails can be changed by defining a method in the Controller class. The HTML view templates need to be placed in the view directory so that the test controller can display them. For example, to change the view of the listing of a database, the list method can be added in the controller as follows:

```
Class TestController <ApplicationController
          scaffold:test

          def list
                    @tests = Test.find_all
          end

end
```

Rails then no longer uses the default view, and looks for a template in the views directory with the name list.rhtml. This is an HTML file with Ruby code embedded in it. The line

```
 @tests = Test.find_all
```

asks the Test class for a collection of all rows from the database assigning the collection to the instance variable @tests.

Pros and Cons: Ruby is a very easy to use object-oriented programming language with a clean syntax. This makes it very easy to create applications with many fewer lines of code than with traditional programming languages. As can be seen from above, Rails eliminates the need for using configuration files and uses code convention instead to map requests to classes. This is however not the case with most web application frameworks like Jakarta Struts, etc., which use an XML-based configuration file to define the mappings. Having multiple files or configurations for an application makes it very difficult to maintain and use it.

2.3 SPRING FRAMEWORK

Spring is another framework used to design web applications. A web application typically uses an M-V-C type of architecture to organize the various components. Spring particularly focuses on management and configuration of business objects of an application. Spring's MVC model is almost similar to that of Struts, but it provides a cleaner division between the Model, View and Controller. Spring's MVC model is very flexible and is entirely based on interfaces. It is possible

to configure almost every part of the Spring MVC framework using one's own implementation of an interface. Spring MVC also allows a custom view mechanism by giving the user the flexibility to use any view technology and does not restrict him to using only JSP, Velocity or XLST for example. The architecture of the Spring framework is based on an Inversion of Control container which is based on the use of JavaBean properties. This container is discussed in more detail below:

Inversion of control (IoC) container: Spring represents an Inversion of Control container through its bean factory concept. The basic idea behind the IoC is that the user does not create objects, but rather specifies how they should be created. Components and services are not directly connected together, but the connections are described in a configuration file. The IoC container creates the objects and wires them together by setting their properties and determines the methods to be invoked [54]. Thus instead of the user's code calling a traditional class library, an IoC framework calls the users code. The core of the Spring framework is the org.springframework.beans package, and is designed to work with JavaBeans. This package is not used directly by users, but defines most of the functionality of the Spring framework [53].

Spring Aspect-oriented Programming: Aspect Oriented Programming(AOP) is one of the key components of the Spring framework. The AOP paradigm complements Object Oriented Programming (OOP) by providing another way of thinking about program structure. AOP decomposes programs into aspects or concerns, whereas OOP decomposes an application into a hierarchy of objects. This enables modularization of concerns that fall under various divisions of responsibility, such as logging and transaction management. Re-usability is facilitated by encapsulating common behavior of multiple classes using Aspect, which is the core construct of AOP [53].

2.3.1 The Spring Architecture

The Spring framework consists of several modules and has a layered architecture. Each of these is a stand-alone module, but can also be implemented along with other modules. The Spring framework has a total of seven modules; six of these modules are built on top of a core container module, which is called the Spring Core Container.

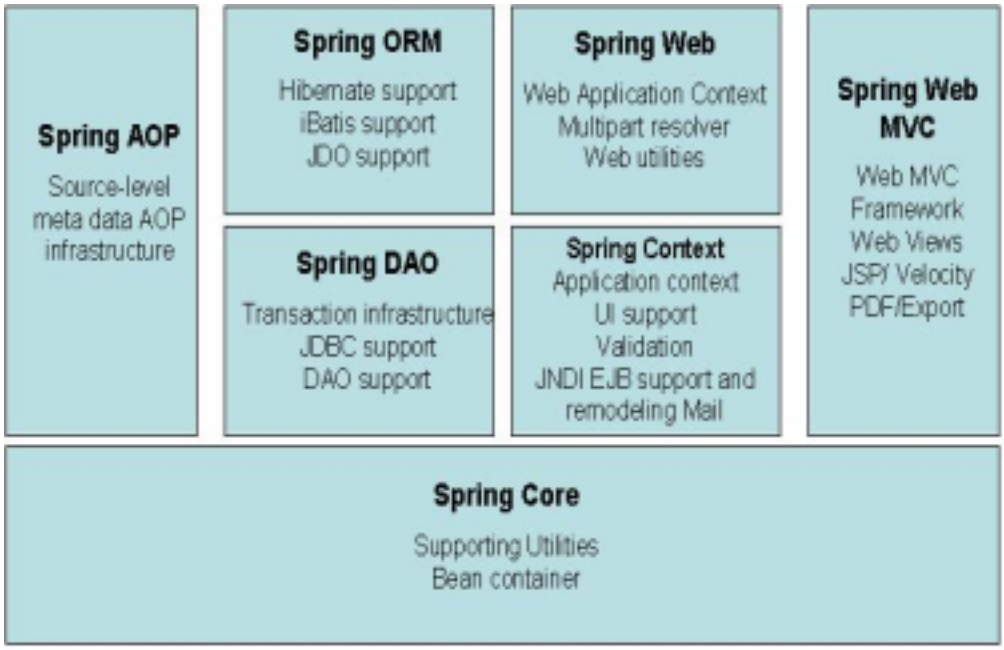


Figure 2.5: The Spring Architecture [54]

Figure 2.5 shows the Spring architecture along with the seven modules:

1. Spring Core: This module provides the main functionality to the Spring framework. One of its main components is the BeanFactory which implements the Inversion of Control pattern and thus separates the application's configuration and specification of dependencies from the actual application code.

2. Spring Aspect Oriented Programming (AOP): This component integrates Aspect-oriented programming into the Spring framework. With this, any object used by the framework can be easily AOP enabled. This module also provides transaction management services for objects in any application based on the Spring framework.

3. Spring Object/relational mapping integration module (ORM): The Spring framework provides support for using Hibernate [26], iBatis [32], Java Data Objects [57] through this module.

4. Spring Data Access Object (DAO): This module provides a JDBC [55] abstraction layer and an exception hierarchy which defines a meaningful way to handle exceptions and errors thrown

by several database vendors. It also reduces the amount of code to be written to handle these exceptions.

5. Spring Web: This module provides context for web-based applications and sits on top of the application context module. Jakarta Struts applications can thus be integrated into the Spring framework.

6. Spring Context: This is a configuration file and is responsible for providing context information to the framework. It also includes support for various enterprise services such as Java Naming Directory Interface (JNDI) [58], EJB [56] etc.

7. Spring Web MVC: This module provides an MVC implementation for building web applications. It includes support for various presentation technologies such as JSP [34], Velocity [37] etc.

Pros and Cons: Spring allows a custom view mechanism by allowing the user to use any view technology and does not restrict them to using any particular technology. The use of Inversion of Control Container and the BeanFactory pattern enables separation of an applications configuration and dependency from the actual application code. Encapsulation of common behavior in common classes using the Spring AOP allows re-usability.

The framework is configuration intensive, with substantial XML [30] code. This is in direct contrast to the Rails [25] framework which minimizes the use of configuration and uses code convention over configuration. Less code reduces complexity and makes the application simple and easy to use.

2.4 AJAX

Ajax stands for Asynchronous JavaScript And XML. Ajax is a web development technique for creating dynamic web applications. Ajax is not a technology by itself, but uses a combination of several technologies including HTML and CSS (for presenting information), and JavaScript to dynamically display and interact with the data, and the XMLHttpRequest object to exchange data asynchronously with the web server [40].

Comparison to traditional web applications: In traditional web applications, a user sends a request to the web server by submitting a form. The web server processes this request and sends a new web page back to the user. Thus for every user request, the web server has to send back a web page and this can increase the response time significantly. The difference between Ajax and the traditional web application model is that a user request takes the form of a JavaScript call to the Ajax engine instead of an HTTP request as in the case of the traditional model. The Ajax engine handles simple user requests such as data validation, editing data in memory that do not need to be sent to the server and asynchronously handles other requests that require processing by the server, and thus does not stall the users interaction with the application [40]. The Ajax engine is written in JavaScript and gets loaded at the start of the session. The engine renders the interface to the user and acts as an intermediary between the user and the server. Users interactions with the application happen asynchronously, and are not dependent on the communication with the server [42]. Figure 2.6 shows how a traditional web application handles a user request and compares it to Ajax showing how it handles the request.

As shown in Figure 2.6, in the classic or the traditional web application model, every user request goes to the web server for processing and the presentation data is returned by the server in the form of an HTML or CSS page. As a result the applications run slowly, since every time a new page has to be sent from the server it increases the wait time for the user significantly. In contrast, in the Ajax model, as shown in the figure, the user request is intercepted by an Ajax engine which is responsible for sending the presentation data (in the form an HTML or a CSS page) to the user and does simple data processing tasks which do not need to be sent to the server. This reduces the processing time significantly.

Example: In a traditional application, a request is made to the server via a normal HTTP POST or GET request, whereas an Ajax script makes a request to the server by using the JavaScript XMLHttpRequest object. The XMLHttpRequest object retrieves information from the server in an invisible manner without requiring a page refresh [51].

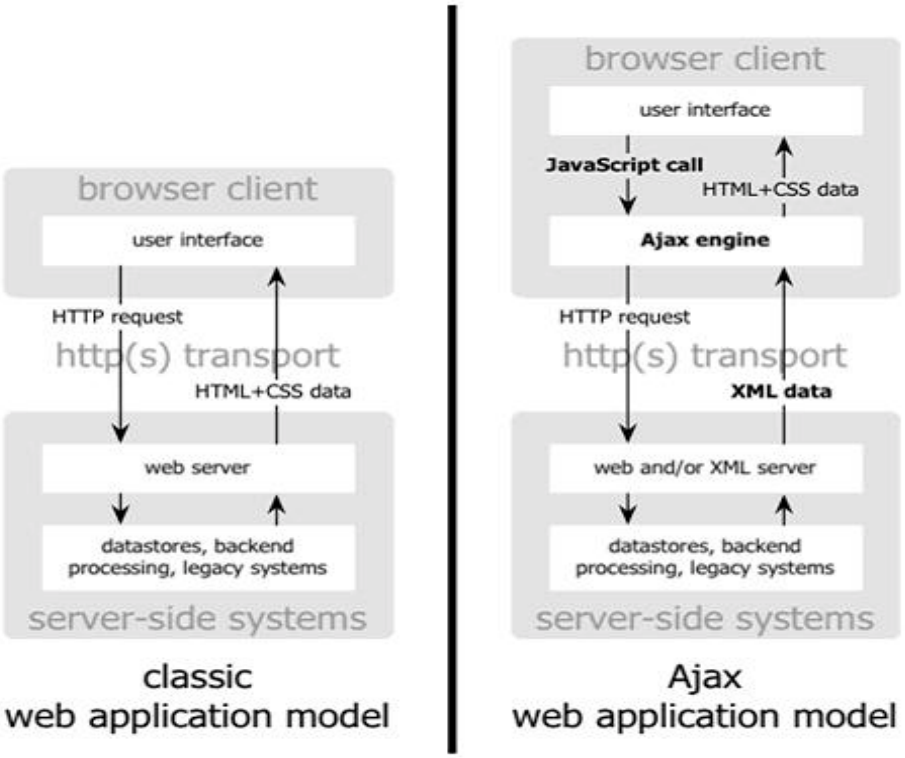


Figure 2.6: Comparison between the classic web application model and the Ajax model [40].

Consider a simple web application with a database server storing the zip code, city and state and an HTML form with input text fields at the front-end. The application makes use of Ajax technology and demonstrates how it can be used to save the users substantial time and typing. Instead of requiring the user to type in the details of all the three fields, the data in the city and state text fields gets automatically displayed when the user enters the zip code [52].

As mentioned earlier, Ajax technology makes use of the XMLHttpRequest object which is a JavaScript object and offers a convenient way for webpages to get information without requiring pages to refresh themselves.

The steps involved in creating the application are as follows:

1. Creating a simple HTML form: This is required to create input text fields for the user to enter data.

Figure 2.7: An Ajax Example [52]

2. Adding the event handler: The event handler is called whenever the zip code field loses focus and is responsible for asking the server the city name and the state for a given zip code. In the Figure 2.8 below, the updateCityState() (Lines 5-6) is an event handler which is called when the text input field loses focus (Line 11).

```
1 <html>
2 …….
3 …….
4 <script language="javascript" type="text/javascript">
5 function updateCityState() {

6 }
7 </script>
8 …….
9 <form action="post">
10 …….
11 <input type="text" name="zip" id="zip" onblur="updateCityState();"/>
.……
……..
……..
……..
```

Figure 2.8: Event Handler [52]

3. Creating the XMLHttpRequest Object: The object could be created using a function named getHttpObject(). Line 10 in the getHttpObject() function in Figure 2.9 creates a new XMLHttpRequest object and assigns it to the xmlhttp variable.

```
1 <html>
2 …….
3 <script language="javascript" type="text/javascript">
4 function updateCityState() {
5  …
6 }

7 function getHttpObject() {
8  var xmlhttp;
9  ….
10  xmlhttp=new XMLHttpRequest();
11  ….
12  return xmlhttp;
13 }
14   var http = getHTTPObject();
15 </script>
16 …….
17 <form action="post">
18 …….
19 <input type="text" name="zip" id="zip" onblur="updateCityState();"/>
.……
……..
……..
……..
```

Figure 2.9: Creating an XMLHttpRequest Object [52]

4. Specifying the URL: The URL for the server-side script is specified in Line 4 in Figure 2.10 below. This URL will have the zip code appended to it and will be sent to the server using the GET method (Line 15, Figure 2.10). Before the HTTP request is sent, the onreadystatechange property is specified to handleHttpResponse(). Thus every time the HTTP ready state has changed, the function handleHttpResponse() is called (Line 16, Figure 2.10). The function handleHttpResponse() keeps waiting idle and when it gets called, it checks to see if readState is equal to 4. If it is, then it means that the request is complete and the response text is obtained and unpacked and the city and state form fields are set to the returned values (Lines 5-12, Figure 2.10).

Pros and Cons: The biggest advantage of using Ajax is the short response time after the user submits a request. Data is manipulated without having to reload the entire page again and it allows

dynamic web page updates. Another advantage is that Ajax technologies are supported in all JavaScript enabled browsers irrespective of the operating system.

A major disadvantage of using Ajax is the fact that the functionality of the browser Back button may be broken due to the dynamic loading of web pages. Another issue related to the dynamic updating of the web pages is the fact that it may become difficult to bookmark a particular state of the application [40].

A brief summary of the comparison between the web application frameworks discussed in this section has been provided in Figure 2.11.

2.5 The Web Development Kit

Introduction: The WDK is a system designed to allow easy creation of “query-based” websites. The system works with any relational database system and on any schema [60]. The main purpose of the WDK is to define and create a data model specifying the types of questions a user can ask of the database and the kind of answers that can be retrieved. Secondly, the WDK also helps to define a “view” of the data model using JavaServer Pages technology. The main objectives of the WDK are to make it easy to offer numerous canned questions and display and manage the question results. The WDK is also aimed at allowing some user operations on the data results such as set operations, query history, sorting of results, etc.

Question Sets, Questions, Summaries and Records: The WDK uses a Question Set-Question-Summary-Record structure to organize data on a website. Users are provided with question sets. Each of the question sets contains questions to choose from. Users ask a particular question by specifying its parameters to obtain a “summary” of the entities found by the query. Each element in the summary has a link to the full “record” for the entity.

WDK Design: The WDK uses the Model-View-Controller architectural pattern. This allows clear separation of the view, from the model and the controller.

The Model: The Model describes the questions, summaries and records that appear on the website. The model is defined in an XML file.

```
1 <html>
2 …….
3 <script language="javascript" type="text/javascript">
4  var url="getCityState.php?param=";

5  function handleHttpRequest() {
6    if (http.readyState == 4) {
7      results = http.responseText.split(",");
8      document.getElementById('city').value = results[0];
9      document.getElementById('state').value = results[1];
10
11   }
12 }

13 function updateCityState() {
14     var zipValue = document.getElementById("zip").value;
15     http.open("GET", url + escape(zipValue), true);
16     http.onreadystatechange = handleHttpResponse;
17     http.send(null);
18  …
19 }

20 function getHttpObject() {
21  var xmlhttp;
22  ….
23  xmlhttp=new XMLHttpRequest();
24  ….
25  return xmlhttp;
26 }

27   var http = getHTTPObject();
28 </script>
29 …….
30 <form action="post">
31 …….
32 <input type="text" name="zip" id="zip" onblur="updateCityState();"/>
.……
…….
…….
…….
```

Figure 2.10: A Complete Ajax Example [52]

The View: The WDK view creates a “view” for the model. It makes use of the JavaServer Pages (JSP) technology for generating the view and represents the questions and records listed in the model.

The Controller: The Controller is a set of Java classes used to control the site. It functions internally to the WDK and is based on the standard Model 2 architecture.

2.5.1 STRUTS BASED WDK MVC ARCHITECTURE

The Struts-based WDK MVC architecture is as shown in Figure 2.12. The Control Flow when the user submits a request in a typical WDK application is as follows:

1. The browser sends a request to the server in the form of an HTTP request. The request is intercepted by the ActionServlet on the server. The user may submit a request to retrieve the summary of results for a particular question as shown in Figure 2.13, for example.

2. The Action servlet creates the Action Form bean instance which extends the Action Form class and represents the data submitted by the user and may represent the parameter values for a particular question (Figure 2.14).

3. The Action servlet uses the mappings defined in the struts-config.xml (e.g. Figure 2.15) to route the user request to the appropriate Action object (e.g. Figure 2.16).

4. The Action object reads the user input data from the Action Form bean.

5 and 6. The Action object uses WdkModelBeans to access the database and retrieve data. The Model beans interact with the database and act as wrappers around the database objects.

7. Once data is retrieved, the Action object represents this data in the form of an ActionForm which is later used by the JSP to represent data.

8. Control returns to the ActionServlet.

9. The ActionServlet uses the mappings in struts-config.xml to determine the forward JSP page.

10. Control goes to the appropriate JSP page according to the mappings specified in the struts-config.xml (eg:showQuestion.jsp)

11. The appropriate JSP page accesses the ActionForm bean to display the results to the user.

Comparison Criteria	Jakarta Struts	Ruby on Rails	Spring
1. Learning curve	High	Low	High
2. Language used	Java	Ruby	Java
3. Architecture/ technology	Model 2 (MVC) Model: Integrates with standard technologies like JDBC and EJBs View: JSP technology Controller: Struts provides its own Controller component	Model 2 (MVC) The Rails helper script is used to create the MVC components (typically with a single line of code)	Model 2 (MVC) Seven modules of layered architecture. One of them is the Spring MVC web module which is highly configurable and is entirely based on interfaces. It allows the use of custom view technologies
4. Configuration files/additional files required	Yes (to map requests to Controller action mappings)	None	Yes (to map requests to actions as well as components to their services)
5. Guiding/key principles	JSP Model 2 design	Code convention over configuration	Inversion of Control Container and Aspect Oriented Programming
6. Creating a typical application	One line of Ruby code is equivalent to approx 10 lines of code in Struts or Spring	The Rails helper script can be used to create the Model, View and Controller with a single line of code.	Similar to that of Struts, but requires additional specification in the configuration file to connect components and services together
7. Pros/Cons	Pros: -Struts uses JSP technology for the View and the tag mechanism offered by JSP allows code re-usability and helps to abstract out Java code from JSP - XML based single configuration file. This allows changes to the configuration without affecting the Java code Cons: - Struts has a higher learning curve - Knowledge of JSP and other technologies is necessary in order to use MVC with Struts	Pros: -Eliminates the use of configuration files -Thus code complexity is reduced -Complexity is further reduced due to the clean and simple Ruby syntax Con: Although the ActiveRecord in the Model class in Rails takes care of retrieving model objects from the database easily, it makes the user deal with databases and the user must make sure that the database is set up and that all the tables are created in the database	Pros: -Allows custom view mechanism - Allows separation of application configuration from the actual code Con: -Substantial XML code for configuration

Figure 2.11: Comparison of web application frameworks

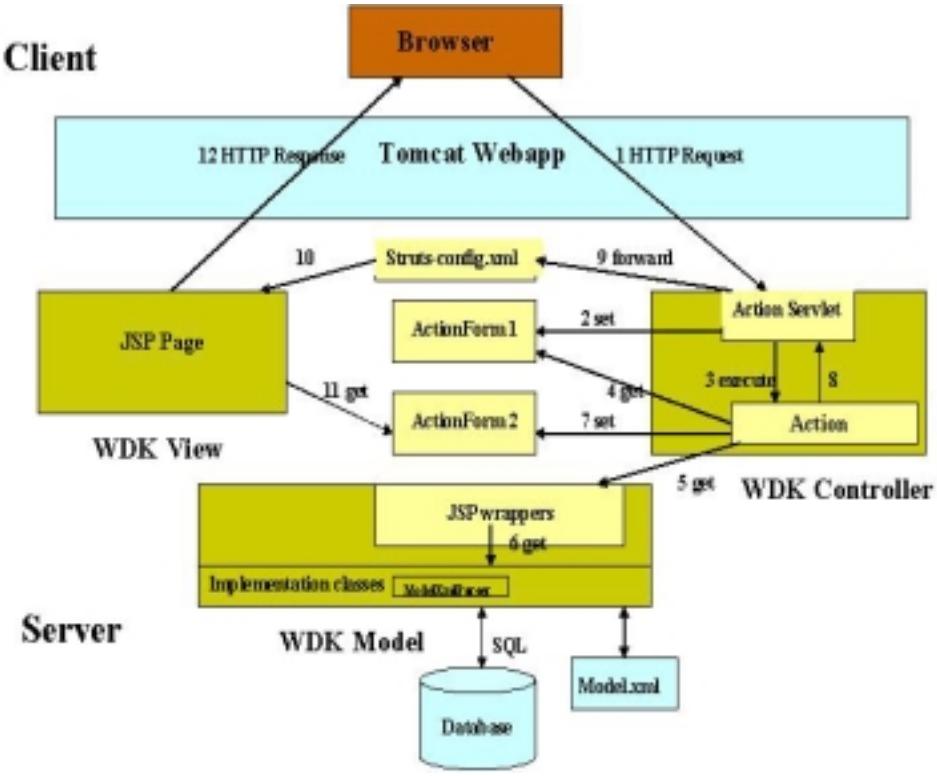


Figure 2.12: The WDK MVC Architecture

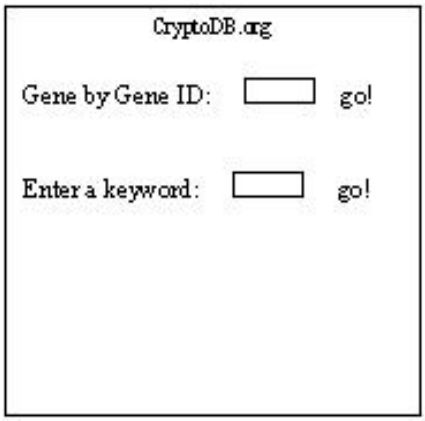
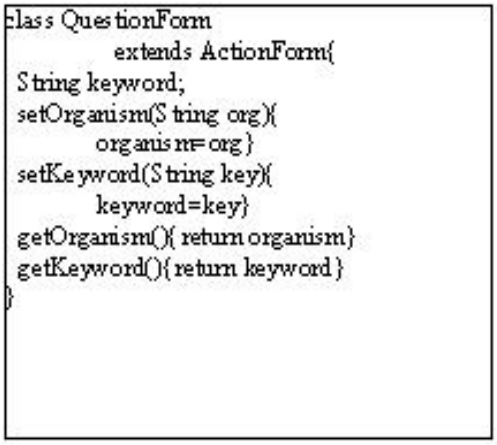


Figure 2.13: Index.jsp



```
class QuestionForm
        extends ActionForm{
 String keyword;
 setOrganism(String org){
        organism=org}
 setKeyword(String key){
        keyword=key}
 getOrganism(){return organism}
 getKeyword(){return keyword}
}
```

Figure 2.14: ActionForm

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>

 <form-beans>
  <form-bean name="questionForm"
        type="org.gusdb.wdk.controller.action.QuestionForm"/>
 </form-beans>

 <global-forwards type="org.apache.struts.action.ActionForward">
  <forward  name="home" path="/index.jsp"/>
 </global-forwards>

 <action-mappings>

  <action
    path="/processQuestion"
    type="org.gusdb.wdk.controller.action.ProcessQuestionSetsFlatAction"
    scope="session"
    name="questionForm"
    input="/question.jsp">
    <forward name="goToThis" path="/showThis.jsp"/>
    <forward name="goToThat" path="/showThat.jsp"/>
  </action>

 </action-mappings>

</struts-config>
```

Figure 2.15: Action Mappings (struts-config.xml)

```
class ProcessQuestionSetsFlat
          extends Action{
 execute/perform(ActionMapping mapping,
         ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response) {

   QuestionForm qf = (QuestionForm) form;
   String org=qf.getOrganism();
   String key=qf.getKeyword();
   String res=DBquery(org, key);
   ActionForward f;
   if(res==this)
     f= mapping.findForward("goToThis");
    else  f= mapping.findForward("goToThat");
  }
}
```

Figure 2.16: Action Object (ProcessQuestionSetsFlatAction.java)

CHAPTER 3

APPROACH

In the version of the WDK that was the latest version (1.3) at the time this work began, the query pages generated by the WDK were highly structured and hierarchical. That is, the user's view of, and interaction with, the database was directly tied to the structure of the model. The model consisted of questions sets, question sets consisted of questions, questions involved one or more parameters, and the submission of a question, at last, would return a record. Thus, the user was forced to:

- select a question from a pull-down list of questions belonging to a question set
- press the "Show Question" button
- set each parameter value, by filling in text boxes and going through pull-down lists
- submit the question, and then finally go to the record page

While such a site is straightforward to generate, given a model, it is less than ideal from the user perspective: too much clicking, too much hidden information, too much navigation, and loss of context. In generating the CryptoDB site for release to the public in March 2005, we were forced to make substantial changes not only to the JSP files, but also to the controller, in order to get a "flattened out" site with less hidden information, less navigation and more desirable parameter displays. That is, we created a site in which all of the questions appear on one page, and in which the user can glance at the page and know what the choices are for all of the parameters, without having to look at multiple pull-down lists to understand the capabilities of the queries (questions) listed there. Below are some sample screen shots of the default WDK-generated site, Figure 3.1, and the completely "flattened out" view, Figure 3.2:

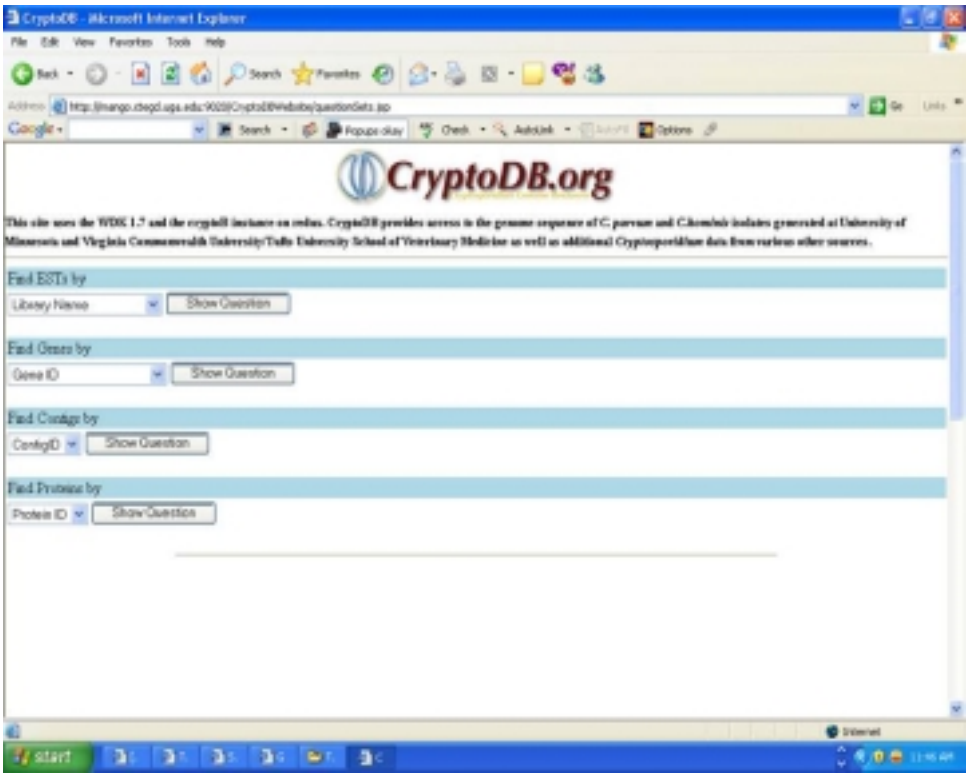


Figure 3.1: Default WDK View

3.1 The Solution

The first approach towards this problem involved giving the user an option between choosing a completely “flat” or a completely structured site. This was accomplished by:

- updating the controller to handle invocations from different steps along the “path” (from question set down to parameters for a given question).

- creating two index.jsp files - one for flat and one for structured.

- allowing the user to set a flag in the build program to select between the structured and flat approaches.

Since this solution was initially implemented, this functionality has been integrated into the WDK: sites are now flat rather than structured, with all questions listed on one page. Still, a

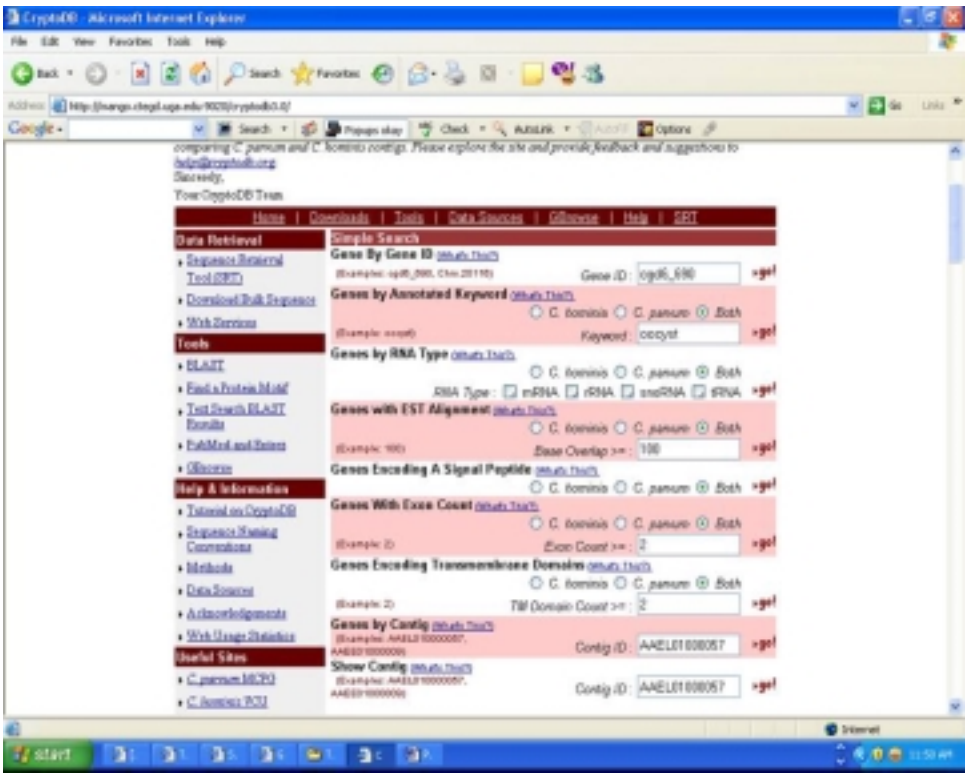


Figure 3.2: Completely flattened View (CryptoDB release 3.0)

problem remains: as the number of possible questions becomes large, site designers may wish to pick and choose which questions appear “flat” on the first page (likely the most commonly used questions or questions typically issued by the most novice users) and which questions should be accessed in a more hierarchical manner. Also, there’s a tradeoff in usability / viewability between showing all parameter choices for a question directly on the screen and hiding some of that visual clutter in a pull-down menu or on a secondary page. The site designers will need to weigh the pros and cons at design time. The ability to further customize the display (alignment, fonts, other parameter displays, default selections, etc.) is also desirable. Further, site designers may also wish to change the order in which the questions are displayed and may desire to group questions differently from the way in which they were grouped in the Model. Facilitating the task of easy custom site design within the WDK is the goal of this thesis project. The basic idea behind the approach

presented here is to have an XML-based view specification to enable custom view specifications for each of the question sets and questions in the model.

1. The first step is the generation of an XML-based view-specification file, which lists all the questions in the model and also the various custom specifications that the user may make on them. This requires iterating through the model and generating XML tags for the custom view specifications. We currently achieve this through a simple JSP file, as follows:

```
-----------------------------------------------------------------------
Simple JSP file to generate an XML-based view-specification.
-----------------------------------------------------------------------
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3 <%@ page contentType="text/xml;charset=ISO-8859-1" %>

4 <c:set var="wdkModel" value="${applicationScope.wdkModel}"/>
5 <c:set value="${wdkModel.displayName}" var="wdkModelDispName"/>

6 <Specifications>
7    <Defaults>
8           <QuestionType>Flat</QuestionType>
9           <FlatVocabFormat>pulldown</FlatVocabFormat>
10           <NonFlatVocabFormat>text</NonFlatVocabFormat>
11           <ParamAlign>right</ParamAlign>
12     </Defaults>

13   <Page name="First">
14       <PageDefaults>
15           <QuestionType>Flat</QuestionType>
16           <FlatVocabFormat>multibox</FlatVocabFormat>
17           <NonFlatVocabFormat>text</NonFlatVocabFormat>
18           <ParamAlign>right</ParamAlign>
19        </PageDefaults>

20   <c:set value="${wdkModel.questionSets}" var="questionSets"/>
21    <c:forEach items="${questionSets}" var="qSet">
22       <c:set value="${qSet.name}" var="qSetName"/>
23     <QuestionSetName name="${qSetName}" type="Structured"
         prompt="" fontSize="" fontFace="" newGroup="true">
24       <c:set value="${qSet.questions}" var="questions"/>
25        <c:forEach items="${questions}" var="q">
```

```
26          <c:set value="${q.name}" var="qName"/>
27          <c:set value="${q.displayName}" var="qDispName"/>
28          <QuestionFullName name="${qDispName}"
             hiddenName="${qSetName}.${qName}" fontSize="" fontFace="">
29           <c:forEach items="${q.params}" var="qP">
30            <Parameter name="${qP.name}"/>
31           </c:forEach>
32          </QuestionFullName>
33         </c:forEach>
34       </QuestionSetName>
35     </c:forEach>
36   </Page>

37   <Page name="Second">
38   </Page>

39   <Page name="Third">
40   </Page>

41 </Specifications>
```

```
-------------------------------------------------------------------------
End JSP file
-------------------------------------------------------------------------
```

Line 1 in the above JSP file listing is the XML declaration and defines the XML version and the character encoding used in the document. Lines 2 and 3 in the file above are page directives that define attributes that apply to an entire JSP page. Line 3 specifies that content of the page generated by the JSP file is of type XML and allows the use of user-specific tags within the page. Custom tags are described in detail in the next section. The custom view specifications file contains some default specifications for the site and individual pages and the default values for these have been specified in the above JSP file (Lines 7-12 and 14-19). By default, all the question sets and questions appear within the “Page First” tags. Lines 20 and 21 iterate through the question sets in the model and set the “name” attribute within the QuestionSetName tag to the name of the question set (Line 23). The default type of the question set is also set to “Structured” and the prompt, which is the display name for the question set, can be specified here. The font size and the font face of the question set prompt can also be specified here. Further, the “newGroup” attribute gives the site designer a

choice to group together various question sets and is set to “true” by default. Lines 25-28 iterate through the questions in each question set and set the name attribute within the QuestionFullName tag to the display name of the question and the hidden name to the full question name, which includes the name of the question set to which the question belongs. The font size and font face for the question display name can be specified here. Lines 29-30 iterate through all the parameters of the question and display the parameter name within the Parameter custom specification tag. Lines 37-38 and 39-40 contain empty “Page” tags and the user can copy and paste the questions listed within the “Page First” tags for a multi-page display.

2. To begin the view specification process, the user would deploy and start the application from the default configuration. This has the effect of storing the model in the application scope. Opening the above JSP file in a browser renders the XML-based view specifications containing all the questions in the model and the allowed customizations for them. An example generated XML file would look as follows:

```
-----------------------------------------------------------------------
Generated XML-based view specifications file
-----------------------------------------------------------------------
1 <?xml version="1.0" encoding="ISO-8859-1" ?>

2 <Specifications>

3  <Defaults>
4   <QuestionType>Flat</QuestionType>
5   <FlatVocabFormat>pulldown</FlatVocabFormat>
6   <NonFlatVocabFormat>text</NonFlatVocabFormat>
7   <ParamAlign>right</ParamAlign>
8  </Defaults>

9  <Page name="First">

10  <PageDefaults>
11   <QuestionType>Flat</QuestionType>
12   <FlatVocabFormat>multibox</FlatVocabFormat>
13   <NonFlatVocabFormat>text</NonFlatVocabFormat>
14   <ParamAlign>right</ParamAlign>
15  </PageDefaults>
```

```
16  <QuestionSetName name="EstQuestions" type="Structured" prompt=""
    fontSize="" fontFace="" newGroup="true">

17   <QuestionFullName name="By Library Name" hiddenName=
     "EstQuestions.EstsByLibrary" fontSize="" fontFace="">
18    <Parameter name="libraryId" />
19   </QuestionFullName>

20   <QuestionFullName name="Overlapping a Gene" hiddenName=
     "EstQuestions.EstsWithGeneOverlap" fontSize="" fontFace="">
21    <Parameter name="libraryId" />
22    <Parameter name="bp_overlap_gte" />
23    <Parameter name="inNotIn" />
24   </QuestionFullName>

25  </QuestionSetName>

26  <QuestionSetName name="GeneQuestions" type="Structured" prompt=""
    fontSize="" fontFace="" newGroup="true">

27  <QuestionFullName name="By Gene ID" hiddenName=
    "GeneQuestions.GeneByLocusTag" fontSize="" fontFace="">
28    <Parameter name="locus_tag" />
29  </QuestionFullName>

30  <QuestionFullName name="By Annotated Keyword" hiddenName=
    "GeneQuestions.GeneByAnnotatedKeyword" fontSize="" fontFace="">
31    <Parameter name="organism" />
32    <Parameter name="keyword" />
33  </QuestionFullName>

34  <QuestionFullName name="By RNA Type" hiddenName=
    "GeneQuestions.GeneByType" fontSize="" fontFace="">
35    <Parameter name="organism" />
36    <Parameter name="rnatype" />
37  </QuestionFullName>

38  <QuestionFullName name="With EST Cluster Overlap" hiddenName=
    "GeneQuestions.GenesByESTClusterOverlap" fontSize="" fontFace="">
39    <Parameter name="organism" />
40    <Parameter name="bp_overlap_gte" />
41  </QuestionFullName>

42  <QuestionFullName name="Encoding A Signal Peptide" hiddenName=
```

```
    "GeneQuestions.ProteinsWithSignalP" fontSize="" fontFace="">
43    <Parameter name="organism" />
44  </QuestionFullName>

45  <QuestionFullName name="By Exon Count" hiddenName=
    "GeneQuestions.GenesByExonCount" fontSize="" fontFace="">
46    <Parameter name="organism" />
47    <Parameter name="num_exons_gte" />
48   </QuestionFullName>

49  <QuestionFullName name="Encoding TM Domains" hiddenName=
    "GeneQuestions.GenesByTMCount" fontSize="" fontFace="">
50    <Parameter name="organism" />
51    <Parameter name="num_tm_gte" />
52  </QuestionFullName>

53  <QuestionFullName name="By Contig ID" hiddenName=
    "GeneQuestions.GeneByContig" fontSize="" fontFace="">
54    <Parameter name="contig" />
55  </QuestionFullName>

56  <QuestionFullName name="By Protein MW Range" hiddenName=
    "GeneQuestions.ProteinsByMolWt" fontSize="" fontFace="">
57    <Parameter name="organism" />
58    <Parameter name="upperMolWtBound" />
59    <Parameter name="lowerMolWtBound" />
60  </QuestionFullName>

61 </QuestionSetName>

62 <QuestionSetName name="ContigQuestions" type="Structured"
   prompt="" fontSize="" fontFace="" newGroup="true">

63  <QuestionFullName name="Show Contig" hiddenName=
    "ContigQuestions.ContigLookup" fontSize="" fontFace="">
64    <Parameter name="contig" />
65  </QuestionFullName>

66 </QuestionSetName>

67 <QuestionSetName name="ProteinQuestions" type="Structured"
   prompt="" fontSize="" fontFace="" newGroup="true">

68  <QuestionFullName name="By Protein ID" hiddenName=
    "ProteinQuestions.ProteinLookup" fontSize="" fontFace="">
```

```
69    <Parameter name="protein_id" />
70  </QuestionFullName>

71 </QuestionSetName>

72  </Page>

73<Page name="Second" />

74<Page name="Third" />

75</Specifications>

-----------------------------------------------------------------
End view specifications file.
-----------------------------------------------------------------
```

The body of the view-specifications.xml file as shown in the listing above appears within the Specifications tags. The first section defines the Defaults (Lines 3-8) which are the defaults for the entire site and apply to all the pages on the site. The default specifications set the defaults for the question type, the display widgets (pulldown, scrollpane, radio buttons etc.) for parameters (FlatVocabParameters and NonFlatVocabFormatParameters) and the parameter alignment. FlatVocabFormatParameters refer to question parameters having multiple value selections, whereas NonFlatVocabParameters have only one possible value choice for a parameter. The next section in the specifications file is the “Page” section. This defines all the question sets and questions that appear on a page. The first section (Lines 10-15) defines the defaults, which are the “PageDefaults” and are applicable only to the questions sets and questions on that page. If specified under a Page section, the values specified here override those specified under the site defaults. Further, each of the question sets listed in the page section have the attributes “name”, “type”, “prompt”, “fontSize”, “fontFace” and “newGroup”. “Name” refers to the name of the question set, “type” refers to either “flat” or “structured”, “prompt” gives the user a choice to give a “display” name to the question. “FontSize” and “FontFace” allow the user to specify the size and type of font for the question set prompt. “NewGroup” allows the user to group together various question sets in the display. If set to

true, the question set appears as a different group. Questions have the attributes “name”, “hidden-Name”, “fontFace” and “fontSize”. “Name” refers to the question name and “hiddenName” refers to the full name of the question, including its question set name. “FontSize” and “fontFace” allow the user to specify size and type of font for the question display name. Each question has a list of parameters listed below it within the “Parameter” tags. Custom specifications for the parameters such as display type, alignment, etc. can be specified here, and by default the specifications take the values as specified in the site or page defaults sections. If the specifications for the parameters of a particular question are desired to be different from those specified in the defaults, then they need to be specified here. Further, the custom view specifications also allow the site designer to rearrange and re-group the questions in an order or grouping that is different from the Model. By default, the XML specifications file lists the questions and question sets in the order in which they appear in the Model. However, the order and grouping of questions and question sets can be changed by simply cutting and pasting the questions in the desired order and group in the specifications file (Appendix A.2 contains the listing of a “custom” view-specifications.xml file). Questions are displayed in the order and group in which they appear in the XML specifications file. In the earlier versions of the WDK, it was not possible to re-arrange the questions within a question set, or across different question sets, without changing the Model. The order of questions would follow exactly the same order of the Model. This made the “View” very tightly coupled to the “Model” and gave less flexibility to the site designer to create his/her custom view. The site designer may wish to arrange the most frequently used or the most common questions at the top of the site page, and less frequently used questions at the bottom. In other cases, it may be desirable to group together all questions that require one parameter, and have a 2-step page display for questions requiring more than one parameter. Thus, various permutations and combinations of the display are possible and giving the site designer the choice to make his custom view, without having to make changes in the Model and the Controller, makes the task much easier.

3.The above XML file is validated against a Relax NG (RNG)[59] schema file. A schema file specifies the grammar or vocabulary that the XML based document instances will represent. Vali-

dation is the process of verifying the grammar of an XML document against its schema. Relax NG is a pattern-based, user-friendly XML schema language [49]. The view-specifications.rng schema file (part) is as follows:

```
-----------------------------------------------------------------
View Specifications validation .rng file
-----------------------------------------------------------------

<?xml version="1.0" ?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">

<start>
<element name="Specifications">

 <zeroOrMore>
          <element name="Defaults">
             <element name="QuestionType">
             <text/>
             </element>
             <element name="FlatVocabFormat">
             <text/>
             </element>
             <element name="NonFlatVocabFormat">
             <text/>
             </element>
             <element name="ParamAlign">
             <text/>
             </element>
          </element>
   </zeroOrMore>


   <oneOrMore>
   <element name="Page">
       <attribute name="name">
         <text/>
       </attribute>

        <zeroOrMore>
          <element name="PageDefaults">
             <element name="QuestionType">
             <text/>
             </element>
```

```
    <element name="FlatVocabFormat">
    <text/>
    </element>
    <element name="NonFlatVocabFormat">
    <text/>
    </element>
    <element name="ParamAlign">
    <text/>
    </element>
 </element>
</zeroOrMore>


 <zeroOrMore>
 <element name="QuestionSetName">
      <attribute name="name">
         <text/>
      </attribute>
      <attribute name="type">
         <text/>
      </attribute>
      <attribute name="prompt">
         <text/>
      </attribute>
      <attribute name="fontSize">
         <text/>
      </attribute>
      <attribute name="fontFace">
         <text/>
      </attribute>
      <attribute name="newGroup">
         <text/>
      </attribute>
           <oneOrMore>
            <element name="QuestionFullName">
               <attribute name="name">
                  <text/>
               </attribute>
                <attribute name="hiddenName">
                  <text/>
               </attribute>
                <attribute name="fontSize">
                 <text/>
                </attribute>
                <attribute name="fontFace">
```

```
                        <text/>
                       </attribute>
                        <zeroOrMore>
                         <element name="Type">
                            <text/>
                         </element>
                         </zeroOrMore>
                         <oneOrMore>
                         <element name="Parameter">
                               <attribute name="name">
                                  <text/>
                               </attribute>
                                  <zeroOrMore>
                                  <element name="FlatVocabFormat">
                                     <text/>
                                  </element>
                                  </zeroOrMore>
                                   <zeroOrMore>
                                  <element name="NonFlatVocabFormat">
                                     <text/>
                                  </element>
                                  </zeroOrMore>

                                   <zeroOrMore>
                                  <element name="Align">
                                     <text/>
                                  </element>
                                   </zeroOrMore>
                         </element>
                         </oneOrMore>
                   </element>
                   </oneOrMore>
          </element>
          </zeroOrMore>
   </element>
   </oneOrMore>
</element>
</start>
</grammar>
-----------------------------------------------------------------------
End rng validation file
-----------------------------------------------------------------------
```

The schema file specifies the names of the tags in the XML file and the type of value each XML tag in the specifications file can hold. Most of the tags here allow text data. Some tags could appear repeatedly in the XML file and this is specified in the schema file using the oneOrMore tag, which indicates that one or more elements with the same tag name could appear in the XML file. Similarly the zeroOrMore tag allows an element tag to be absent from the XML specification.

4. Once the user makes his custom view specifications in the XML file, the file needs to be parsed in order to extract the custom data from it, and hence to generate a custom view. A SpecificationsParser class based on the Digester package (org.apache.commons.digester) has been written in order to achieve this (Appendix B.1). The Digester component from Jakarta Commons reads an XML file and acts on it using a set of pre-defined rules. In essence, Digester is an XML to Java object mapping package [50]. This contains all the registered element matching patterns along with the processing rules that are fired when the pattern is recognized in the input document. Finally, the digester.parse() method parses the XML document and a JSP file uses this extracted custom view data to generate a custom view. Below is a brief description of the standard rules used by Digester: ObjectCreateRule: This rule creates an object of the specified class using its default constructor and pushes it onto the stack. It is popped when the element completes. The class to instantiate can be given through a class object or the fully-qualified class name. Example: The following statement creates an object of the SpecificationsParser class.

```
digester.addObjectCreate("Specifications", SpecificationsParser.class);
```

SetPropertiesRule: This rule sets one or several named properties in the top-level bean using the values of named XML element attributes. Attribute names and property names are passed to this rule in String[] arrays. It is typically used to handle XML constructs like

```
<Page name="First">
```

The rule for this construct would look as follows:

```
digester.addSetProperties("Specifications/Page","name", "name");
```

BeanPropertySetterRule: This rule sets a named property on the top-level bean to the character data enclosed by the current XML element. Example:

```
<QuestionType>Flat</QuestionType>
```

CallMethodRule: This rule calls an arbitrary named method on the top-level bean. The method may take an arbitrary set of parameters. Example: For the XML element,

```
<QuestionType>Flat</QuestionType>,
```

the rule will look like:

```
digester.addCallMethod("Specifications/Defaults/QuestionType",
                                                "setQuestionType",0);
```

AddSetNextRule: This rule is used to add an element to a collection. Example:

```
digester.addSetNext("Specifications/Page/QuestionSetName/
                           QuestionFullName", "addQuestionFullName");
```

The above statement adds the object QuestionFullName to the collection when the specified pattern is seen.

5. The SpecificationsParser class specifies the methods to be invoked when a pattern in the XML document is recognized. The methods are defined in JavaBeans and the corresponding “set” method of an element is invoked when a matching pattern is seen by the Digester parser. The JavaBeans also define an add method which is invoked when a repeating pattern is seen by the parser and the element is added and stored in an ArrayList.

The SpecificationsParser.java listed in Appendix B.1 uses Digester properties described above to register the XML elements and to invoke appropriate methods in the JavaBeans. After all the patterns are registered, the input XML file is parsed using the Digester parse() method. In the specifications file, we allow PageDefaults to override the Site Defaults, and the individual question specifications to override the PageDefaults. Each of the Array List data structures created to hold the question sets, the questions and their parameters under each page is iterated through and if

the values for the question type, the parameter type for FlatVocab and NonFlatVocab parameters and their alignment is found to be empty, then these values are set to the values specified in the PageDefaults, if it is not empty, else it is set to the value specified in the Site Defaults.

The JavaBeans created and used here are Defaults.java, PageDefaults.java, Page.java, Parameter.java, QuestionFullName.java and QuestionSetName.java and contain the “set” and “get” methods to hold the data extracted for the corresponding attributes of the elements defined in the XML document. Appendix B contains the full listing of the classes.

6. Controller classes ShowQuestionAction.java and ProcessQuestionSetsFlatAction.java allow questions to be displayed as either “structured” or “flat” respectively. NextPageAction.java allows a multi-page display and forwards control to either Page “Second” or Page “Third” depending on the request. The struts-config.xml contains additional mappings to handle the multi-page display. The ApplicationInitListener class (Appendix C.2) is initialised at the start of the web application and it ensures that all global resources are available to all contexts which require them. The class uses reflection to invoke the parseSpecFile() method of the SpecificationsParser class.

7. The next step is to read these specifications in the JSP file (index.jsp for example) and thus enable a custom view generation. (Restarting the application is required at this point). To accomplish the customization, the default index.jsp file contains a number of if-else constructs. These handle the various options from the custom specifications. The control flow (for flat and structured), the type of display, etc. are defined within these constructs, thus enabling an easy customization of the view for the user. The question sets and questions are iterated through from the specifications file and this enables questions to be displayed in the order in which they appear in the specifications file and thus allows an ordering and grouping that differs from the order of questions in the Model.

The Struts-based WDK MVC architecture with the modifications for generating a custom view as described above has been shown in Figure 3.3.

An example JSP file for this purpose would look like this (only a part of the file is provided here, Appendix D.1 contains the full listing):

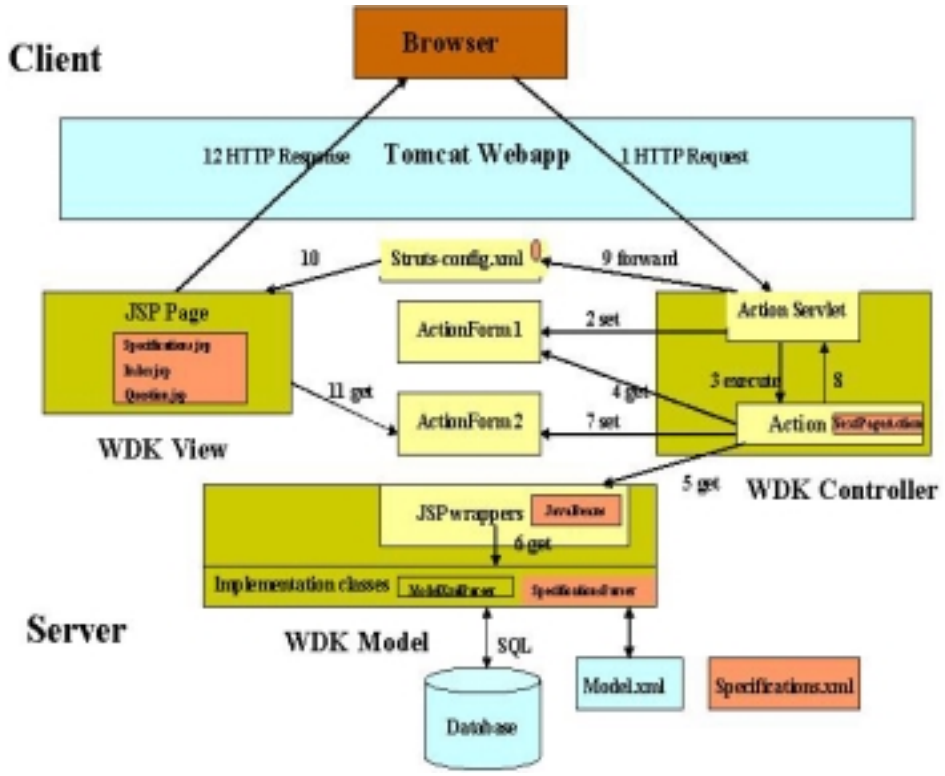


Figure 3.3: The modified WDK MVC Architecture

```
-------------------------------------------------------------------------
Example custom view jsp file (part)
-------------------------------------------------------------------------

1 <%@ taglib prefix="site" tagdir="/WEB-INF/tags/site" %>
2 <%@ taglib prefix="wdk" tagdir="/WEB-INF/tags/wdk" %>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4 <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
5 <%@ taglib prefix="nested" uri="http://jakarta.apache.org/
                                    struts/tags-nested" %>
6 <%@ taglib prefix="html" uri="http://jakarta.apache.org/
                                   struts/tags-html" %>

7 <!-- get wdkModel saved in application scope -->
8 <c:set var="wdkModel" value="${applicationScope.wdkModel}"/>
```

```
9 <c:set var="parser" value="${applicationScope.specRaw}"/>
10 <c:set var="pageList" value="${parser.pageList}" />

11 <!-- get wdkModel name to display as page header -->
12 <c:set value="${wdkModel.displayName}" var="wdkModelDispName"/>
13 <site:header banner="${wdkModelDispName}" />

14 <!--loop 1: loop through all pages  -->
15 <c:forEach items="${pageList}" var="page">
16    <c:if test="${page.name == 'First'}">

17 <!-- loop through all the question sets on page 1
              from specification file -->

18   <c:forEach items="${page.questionSetList}" var="questionSet">
19        <!-- loop through question sets from the model -->

20           <c:if test="${questionSet.type == 'Flat'}">

21     <!-- loop through each question set from specification file -->
22      <c:forEach items="${questionSet.questionList}" var="question">
23       <!-- loop through each question set from model file -->
24         <c:set value="${wdkModel.questionSets}" var="qSets"/>
25            <c:forEach items="${qSets}" var="qSet2">
26             <c:forEach items="${qSet2.questions}" var="q">
27               <c:if test="${question.name == q.displayName}">

28           <c:set value="${qSet2.name}" var="qSetName"/>
29            <c:set value="${question.type}" var="qType"/>
30      <!-- If question type is flat, display all the parameters -->
31       <c:if test="${question.type == 'Flat'}">
32       <html:form method="get" action="/processQuestionSetsFlat.do">
                 ........
                 ........
                 ........
                 ........

33         <!-- Added part for structured diplay here -->

34             <c:if test="${question.type == 'Structured'}">
35                  <html:form method="get" action="/showQuestion.do">

           .........
           .........
           .........
```

```
            .........

-------------------------------------------------------------------
End custom view jsp file (part)
-------------------------------------------------------------------
```

The above partial file shows how the customizations from the XML-based view specifications are used in the JSP file to generate a custom view. For example, in order to generate a “flat” or a “structured” view for the questions, the appropriate control flow needs to be specified in the if-else construct. The user should not have to make changes in the Controller for this. Thus in the above code snippet, from lines 31 and 34, it is seen that if the question type is specified as “flat”, the control goes to “processQuestionSetsFlat.do” and if it is specified as “structured”, the “showQuestion.do” action is executed. ProcessQuestionSetsFlatAction and ShowQuestionAction are classes defined in the Controller, which enable a flat or structured view of questions, respectively. The control flow is defined in the struts-config.xml, and hence the forward page from the action is determined from here. That is, the struts-config.xml contains action-mappings which map the “.do” extension names specified in the JSP file to action classes in the Controller. Thus “showQuestion.do” is mapped to “ShowQuestionAction.java” and “processQuestionSetsFlat.do” is mapped to the “ProcessQuestionSetsFlatAction.java” class in the Controller. The JSP similarly reads the other custom specifications by accessing methods in JavaBeans and defines the code for different parameter displays, alignment, etc. within if-else constructs, thus enabling easy custom view generation.

We realize that the code within the index.jsp file is somewhat complex for users to view and deal with. A tag library could be created to handle the code within the various “if-else” constructs, thus reducing the complexity and making it easier to write the JSP file to generate a custom view.

CHAPTER 4

EVALUATION

Below are some screen-shots that show how various custom views can be generated by making custom specifications in the view specifications file. The method was used to re-create the CryptoDB site (release 3.0) and also to create a custom view for release 3.1 of CryptoDB.

1. A completely structured view (similar to the WDK default view): The view shown in Figure 4.1 is similar to the default view generated by the WDK. All question sets are specified as "Structured" in the configuration file. Thus the question sets appear as "pull-down" menus listing the questions in each question set. After selecting a question from a particular question set, the user has to go to a second page to make parameter selections. The display widgets of parameters can be specified in the configuration file for each question individually, or for the entire page or site in the Page Defaults and Site Defaults sections of the configuration file respectively. Though the view has a somewhat clean look, it hides much of the information in pull-down menus and does not give the user a clear idea of the nature of results that can be retrieved from the database.

2. Flat/Structured view for questions: The configuration shown in Figure 4.2 allows some questions to be "flat" and some to be "structured". It may not be desirable to have a "flattened" out view for all the questions in the question sets, since questions requiring more than one input parameter will require more space and tend to make the site look busy. Thus in this configuration, all the questions requiring simple text input parameters have been specified as "Flat" in the custom specifications and those questions requiring a multiple choice or more than one parameter have been specified as "Structured" in the specifications file. Since all the question sets have been specified as "Flat", there is no hidden information in pull-down menus, and it is clear to the user what the available options are right on the front page.

Figure 4.1: A Fully Structured View (Configuration 1)

3. Creating custom Question Sets: Additional question sets can be created in the custom specifications file, and this gives the site designer more flexibility and allows him to re-group the questions differently from the grouping specified in the Model. In this configuration (Figure 4.3), a new question set called “More Gene Questions” has been created and as an example, all the questions requiring more than one parameter, or a parameter requiring multiple choice have been grouped together under this question set. The question set has been specified to be of type “Structured”, hence it appears as a pull-down menu. Further, since these questions belong to the same main category of “Gene Questions”, it would be logical to group this newly created question set along with the other Gene questions, rather than as a separate question set separated by a horizontal rule from other question sets. The “newGroup” attribute for a Question set in the specifications allows this

Figure 4.2: Flat or Structured View (Configuration 2)

flexibility and if set to “true” separates the question set from other sets with a horizontal rule, and if set to “false”, a question set is displayed in the same group as the question set that follows it.

4. Re-ordering of questions: It may be desirable to display questions in an order that differs from the way it is specified in the Model. For example, it may be desirable to display questions requiring simple input parameters, or the more frequently asked questions first. In this configuration (Figure 4.4), the “Show Contig” and “By Protein ID” questions have been re-ordered in the specifications file in order to display them before the “More Gene Questions” and the “EST Questions”. Normally such a change in the View would require changes in the Model and would make the process time consuming and complicated. Users can simply cut and paste the questions and place them in the desired order in the custom specifications file and the questions will be displayed

Figure 4.3: Custom Question Sets (Configuration 3)

in the order specified in the view specifications file, and not the order in which they are specified in the Model.

5. PageDefaults/Parameter displays: The configuration in Figure 4.5 shows how custom parameter displays can be specified for individual questions. The PageDefaults in the following configuration specify a “pulldown” type of display for questions having multi-choice parameters (FlatVocabType). However, if the user wants to depart from this and specify a different parameter format for a particular question, he/she can specify it separately for that question and PageDefaults specifications will be overridden by the individual question specifications. Thus in the configuration below, the parameter type for the “Annotated Keyword” question appears as a “scrollpane” and not as a “pulldown” menu as specified in the PageDefaults since the display type has been explicitly specified as “scrollpane” for the question. For the questions for which a display type has not been

Figure 4.4: Re-ordering of questions (Configuration 4)

specified, the display is of type specified in the PageDefaults. Thus for example, for the “Number of TM Domains” question, since no specific parameter format has been specified for the question, the parameter display type is of type specified in the PageDefaults.

6. Multi-page display: The configuration in Figure 4.6 shows how a multi-page display can be specified in the custom specifications, where some questions are specified to appear on the second/third page. By default, the custom specifications template lists all the questions under the

```
<Page name="First">
```

tag. However the user can simply copy and paste questions from this listing under the

```
<Page name="Second"> or <Page name="Third">
```

Figure 4.5: PageDefaults/Parameter displays (Configuration 5)

tags in order to have a multi-page display. In the view below, the five questions displayed are listed under the

```
<Page name="Second">
```

tag in the view specifications file. Further, the user can specify different default specifications for this page. The user simply has to make these default specifications under the “PageDefaults” section for this page which holds for all the questions on a page, unless specific customizations have been specified for a particular question. Thus, the default parameter display type for multi-choice parameters (FlatVocabFormat) has been specified as “radio” in the PageDefaults section for this page. This is different from the specifications for the first page (Figure 4.5), where the FlatVocab-

Figure 4.6: Multi-page display (Configuration 6)

Format has been specified as “pull-down”. Thus the organism parameter appears as “radio” on the second page and in a “pull-down” menu on the first page (Figure 4.5).

CHAPTER 5

CONCLUSION

An automatic “custom” view generation method has been designed and implemented in this thesis work. This method enables a clear separation between the Model, View and Controller of an MVC type application and thus allows changes to the View without requiring any substantial changes in the Model and the Controller. The method uses a custom configuration file that contains the specifications a user is allowed to make in order to generate a custom view. This gives the user much more flexibility in designing a custom view, since the View is no longer closely tied to the Model and the Controller.

In a typical WDK application, the View consists of a set of questions that the user can ask of the database. Each of the questions has one or more parameters for which the user specifies a value in order to retrieve the result. As mentioned earlier, the default View generated by the WDK was highly “structured” and contained much hidden information in pull-down menus, etc. It did not give the user a clear idea of what he/she could query the database for and the nature of results that may be returned for a particular question. In order to give the user more clear options and in order to generate a more user-friendly view, a completely “flattened” out view of the questions was created in which all the questions along with their parameters were displayed on the front page, and it was possible for the user to retrieve the results with just one click. Though good from the usability point of view, creating this view was a highly tedious and time consuming process with substantial modifications to the Controller and the Model. Further, the View was tightly coupled to the Model in that it did not give the site designer enough flexibility to design a more “custom” view. The order and grouping of the questions for example could not be changed and they could not be re-grouped or re-ordered differently from the way they were listed and grouped in the Model. Also,

the default view generated a completely “structured” view and the improved view was completely “flat”. However, it is possible that the site designer would want some of the questions to be listed as “flat” and some as “structured”, have a different ordering and grouping of questions from the Model, have different types of parameter displays for the questions, etc. Thus it became necessary to have a method of custom view generation that would give more flexibility to the site designer and have the View as loosely tied to the Model and the Controller as possible.

The method of having a custom specifications file to generate a custom view, as designed and implemented in this work, was used to design a custom view for the CryptoDB site (release 3.1). The method allows “on-the-fly” generation of custom views by making changes in the specifications file. This was very useful in evaluating and comparing the usability of the different possible views for the website. Questions were re-grouped and re-ordered in different question sets in the various configurations, along with “flat” or “structured” specifications for them and different parameter display types. Re-ordering and re-grouping of questions can be done in the specifications file by simply cutting and pasting the questions in the desired order, and can thus be specified and displayed differently from the order and group specified in the Model. The view is also not tied to being fully “structured” or fully flattened as in the earlier two versions and gives the site designer the flexibility to specify “flat” and “structured” for individual question sets and questions. This was particularly useful to avoid “visual clutter” on the CryptoDB site, since some questions have single input parameters and some have multiple parameters, requiring multiple choices. Having a “flattened” out view for all the questions would give the site a very “busy” look and would be confusing to the user. Hence it was desired to have a “structured” view for all questions requiring more than one input parameter and a “flattened” view for questions requiring a single input text parameter. This allowed a clean view and still did not hide any information in pull-down menus, etc. “Structured” questions were still displayed on the front page, but required the user to go to a secondary page to input the parameters. Questions with a “flattened” view allowed the user to retrieve the answer in one-click.

5.1 Future Work

With the current implementation, users have to make their custom specifications in an XML-based specifications file. In the future, it may be a good idea to have a more “user-friendly” version and have a graphical user interface which would include a “drag and drop” feature to allow users to simply drag and drop the question sets and questions on a page along with their custom selections. For example, a question could be specified as “structured” by simply dragging and dropping it on the page without listing its parameters. If it is desired to be “flat”, then the parameters could also be listed along with custom selections for their display widgets.

Secondly, a number of “if-else” constructs have been included in the JSP file in order to facilitate a custom view generation. This makes the code somewhat complex and a “custom” tag library could be created to reduce this complexity.

BIBLIOGRAPHY

[1] Andre van der Hoek. Integrating Configuration Management and Software Deployment.

[2] Andrei Erdoss. Exploring the Portability and Extensibility of XML Schema Generated Web Applications.

[3] Susan Dart. Spectrum of Functionality in Configuration Management Systems. 1990, Software Engineering Institute:Pittsburgh, Pennsylvania.

[4] Carzaniga, A., Rosenblum, D.S., and Wolfe A.L. Design and Evaluation of a Wide-area Event Notification Service. ACM Transactions on Computer Systems, 2001.

[5] Conradi, R. and Westfechtel, B. Version Models for Software Configuration Management. ACM Computing Surveys, 1998. 30(2):p. 232-282.

[6] Dashofy, E.M., van der Hoek, A., and Taylor, R.N., A Highly-Extensible, XML-Based Architecture Description Language. Design and Evaluation of a Wide-area Event Notification Service. Working IEEE/IFIP Conference on Software Architecture. 2001 (to appear).

[7] Hall, R.S., Heimbigner, D.M., and Wolfe A.L. Approach to Support Software Deployment Using the Software Dock Proceedings of the 1999 International Conference on Software Engineering. 1999,ACM Press. p. 174-183.

[8] Branko Milosavljevic, Milan Vidakovic, Zora Konjovic. Automatic code generation for database-oriented web applications. ACM International Conference Proceeding Series (Dublin, Ireland, 2002). National University of Ireland, Maynooth, Ireland, 2002, 59-64.

[9] Jia Zhang, Jen-Yao Chung, Carl K. Chang. Towards increasing web application productivity. Symposium on Applied Computing, Proceedings of the 2004 ACM symposium on Applied computing (Nicosia, Cyprus, 2004).

[10] Estublier J. A Configuration Manager: The Adele Data Base of Programs. Proceedings of the Workshop on Software Engineering Environments for Programming-in-the-Large, pages 140-147. June 1985.

[11] Babich, W. Software Configuration Management. Addison-Wesley, 1986.

[12] Bersoff, E. H., Henderson, V.D., and Siegel, S.G. Software Configuration Management. Prentice-Hall, 1980.

[13] Ploedereder, E. and Fergany, A. A Configuration Management Assistant. Proceedings of the Second International Workshop on Software Version and Configuration Control, pages 5-14. ACM, USA, October 1989.

[14] Leblang, David B. and McLean, Gordon D., Jr. Configuration Management for Large-Scale Software Development Efforts. GTE Workshop on Software Engineering Environments for Programming in the Large, pages 122-127. June 1985.

[15] Tichy, Walter F. RCS:A System for Version Control. Software: Practice and Experience, pages 637-654. July 1985.

[16] Dirk Draheim and Gerald Weber. Specification and Generation of Model 2 Web Interfaces.

[17] Chuck Cavaness. Programming Jakarta Struts. O'Reilly and Associates, Inc. 2003.

[18] D. Puiu, S. Enomoto G.A. Buck, M. Abrahamsen and J.C. Kissinger. (2004) CryptoDB: The *Cryptosporidium* genome resource. Nucleic Acids Research 32:D329-331.

[19] Mark Heiges, Haiming Wang et al. CryptoDB: A *Cryptosporidium* Bioinformatics Resource Update. Nucleic Acids Research - NAR-02075-2005, in Press.

[20] Apache Struts http://struts.apache.org/

[21] Govind Seshadri. Understanding JavaServer Pages Model 2 architecture. Exploring the MVC design pattern. http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

[22] http://struts.apache.org/userGuide/introduction.html

[23] Demystifying Jakarta Struts. An Introductory Jakarta Struts Tutorial. http://www.coreservlets.com/Apache-Struts-Tutorial/

[24] Apache Struts http://struts.apache.org

[25] Ruby on Rails http://www.rubyonrails.org

[26] Hibernate http://www.hibernate.org

[27] Java Servlet Technology http://java.sun.com/products/servlet/

[28] Java Beans http://java.sun.com/products/javabeans/

[29] http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html

[30] Extensible Markup Language (XML) http://www.xml.org/

[31] http://jakarta.apache.org/commons/

[32] iBATIS http://ibatis.apache.org/

[33] Object Relational Bridge (OJB) http://db.apache.org/ojb/

[34] JavaServer Pages http://java.sun.com/products/jsp/

[35] JSP Standard Tag Library http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html

[36] Java Server Faces http://java.sun.com/j2ee/javaserverfaces/

[37] Velocity http://jakarta.apache.org/velocity/

[38] http://www.w3.org/TR/xslt

[39] http://java.sun.com/products/jsp/overview.html

[40] Ajax http://en.wikipedia.org/wiki/AJAX as on December 5, 2005, 14:45

[41] Curt Hibbs. Rolling with Ruby on Rails. http://www.onlamp.com/pub/a/onlamp/2005/01/20/rails.html?page=1

[42] Jesse James Garrett. Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/publications/essays/archives/000385.php

[43] Red Johnson. Introduction to the Spring Framework. http://www.theserverside.com/articles/article.tss?l=SpringFramework

[44] http://java.boot.by/wcd-guide/ch03s02.html

[45] http://java.sun.com/j2ee/sdk$_1$.3/$techdocs/api/javax/servlet/http/HttpServletRequest.html$

[46] http://java.sun.com/products/servlet/2.2/javadoc/javax/servlet/http/HttpSession.html

[47] http://java.sun.com/j2ee/sdk$_1$.3/$techdocs/api/javax/servlet/ServletContext.html$

[48] http://livedocs.macromedia.com/coldfusion/6.1/htmldocs/ssacti14.htm

[49] http://java.sun.com/j2se/1.5.0/docs/api/javax/xml/validation/package-summary.html

[50] Vikram Goyal. Jakarta Commons Online Bookshelf: XML parsing with Digester. Part 1. http://webreference.com/programming/jakarta/

[51] http://www.mousewhisperer.co.uk/ajax$_p$$age.html$

[52] Bill Bercik. Guide to Using AJAX and XMLHttpRequest from WebPasties. http://www.webpasties.com/xmlHttpRequest/xmlHttpRequest$_t$$utorial_1.html$

[53] Spring Framework http://www.springframework.org/

[54] Naveen Balani. The Spring series, Part 1: Introduction to the Spring framework. A first look at Spring AOP and the IOC container. http://www-128.ibm.com/developerworks/library/wa-spring1/

[55] Java Database Connectivity (JDBC) http://java.sun.com/products/jdbc/

[56] Enterprise JavaBeans Technology http://java.sun.com/products/ejb/

[57] Java Data Objects (JDO) http://java.sun.com/products/jdo/

[58] Java Naming and Directory Interface (JNDI) http://java.sun.com/products/jndi/

[59] RELAX NG http://www.relaxng.org/

[60] The Genomics Unified Schema (GUS) http://www.gusdb.org/

[61] The *Cryptosporidium* Genome Resource (CryptoDB) http://www.cryptodb.org/

APPENDIX A

VALIDATION (VIEW-SPECIFICATIONS.RNG) AND CUSTOM VIEW SPECIFICATION (VIEW-SPECIFICATIONS.XML) FILES

A.1 VIEW-SPECIFICATIONS.RNG

```
<?xml version="1.0" ?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0">

<start>
<element name="Specifications">

 <zeroOrMore>
          <element name="Defaults">
             <element name="QuestionType">
             <text/>
             </element>
             <element name="FlatVocabFormat">
             <text/>
             </element>
             <element name="NonFlatVocabFormat">
             <text/>
             </element>
             <element name="ParamAlign">
             <text/>
             </element>
          </element>
   </zeroOrMore>


   <oneOrMore>
   <element name="Page">
       <attribute name="name">
         <text/>
       </attribute>

        <zeroOrMore>
```

```
 <element name="PageDefaults">
    <element name="QuestionType">
    <text/>
    </element>
    <element name="FlatVocabFormat">
    <text/>
    </element>
    <element name="NonFlatVocabFormat">
    <text/>
    </element>
    <element name="ParamAlign">
    <text/>
    </element>
 </element>
</zeroOrMore>


 <zeroOrMore>
 <element name="QuestionSetName">
      <attribute name="name">
         <text/>
      </attribute>
      <attribute name="type">
         <text/>
      </attribute>
      <attribute name="prompt">
         <text/>
      </attribute>
      <attribute name="fontSize">
         <text/>
      </attribute>
      <attribute name="fontFace">
         <text/>
      </attribute>
      <attribute name="newGroup">
         <text/>
      </attribute>
           <oneOrMore>
            <element name="QuestionFullName">
               <attribute name="name">
                  <text/>
               </attribute>
                <attribute name="hiddenName">
                  <text/>
               </attribute>
               <attribute name="fontSize">
                  <text/>
```

```
                        </attribute>
                        <attribute name="fontFace">
                           <text/>
                        </attribute>
                          <zeroOrMore>
                           <element name="Type">
                              <text/>
                           </element>
                           </zeroOrMore>
                           <oneOrMore>
                           <element name="Parameter">
                                 <attribute name="name">
                                    <text/>
                                 </attribute>
                                    <zeroOrMore>
                                    <element name="FlatVocabFormat">
                                       <text/>
                                    </element>
                                    </zeroOrMore>
                                     <zeroOrMore>
                                    <element name="NonFlatVocabFormat">
                                       <text/>
                                    </element>
                                    </zeroOrMore>

                                     <zeroOrMore>
                                    <element name="Align">
                                       <text/>
                                    </element>
                                     </zeroOrMore>
                           </element>
                           </oneOrMore>
                     </element>
                     </oneOrMore>
            </element>
            </zeroOrMore>
   </element>
   </oneOrMore>
</element>
</start>
</grammar>
```

A.2 VIEW-SPECIFICATIONS.XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Specifications>


<!-- To specify values other than the defaults,
specify your choice by including the template
for each question as below -->
<!-- Example template -->

  <Defaults>

          <QuestionType>Flat</QuestionType>
          <FlatVocabFormat>scrollpane</FlatVocabFormat>
          <NonFlatVocabFormat>text</NonFlatVocabFormat>
          <ParamAlign>right</ParamAlign>

   </Defaults>


  <Page name="First">

    <PageDefaults>

          <QuestionType>Flat</QuestionType>
          <FlatVocabFormat>pulldown</FlatVocabFormat>
          <NonFlatVocabFormat>text</NonFlatVocabFormat>
          <ParamAlign>right</ParamAlign>

    </PageDefaults>


     <QuestionSetName name="GeneQuestions" type="Flat"
         prompt="Gene Questions" fontSize="-1"
         fontFace="Arial,Helvetica" newGroup="true">

          <QuestionFullName name="By Gene ID"
                   hiddenName="GeneQuestions.GeneByLocusTag"
                   fontSize="-1" fontFace="Arial,Helvetica">
           <Parameter name="locus_tag"/>
          </QuestionFullName>

          <QuestionFullName name="Contig ID"
               hiddenName="GeneQuestions.GeneByContig"
               fontSize="-1" fontFace="Arial,Helvetica">
           <Type>Flat</Type>
            <Parameter name="contig">
              <NonFlatVocabFormat>text</NonFlatVocabFormat>
```

```
      <Align>right</Align>
      </Parameter>
   </QuestionFullName>


   <QuestionFullName name="By Annotated Keyword"
    hiddenName="GeneQuestions.GeneByAnnotatedKeyword"
    fontSize="-1" fontFace="Arial,Helvetica">
    <Type>Structured</Type>
    <Parameter name="organism"/>
    <Parameter name="keyword"/>
   </QuestionFullName>



    <QuestionFullName name="Encoding TM Domains"
    hiddenName="GeneQuestions.GenesByTMCount"
    fontSize="-1" fontFace="Arial,Helvetica">
     <Type>Structured</Type>
    <Parameter name="organism"/>
    <Parameter name="num_tm_gte"/>
    </QuestionFullName>

   <QuestionFullName name="Encoding A Signal Peptide"
    hiddenName="GeneQuestions.ProteinsWithSignalP"
    fontSize="-1" fontFace="Arial,Helvetica">
      <Type>Structured</Type>
      <Parameter name="organism">
         <FlatVocabFormat>pulldown</FlatVocabFormat>
         <Align>right</Align>
      </Parameter>
</QuestionFullName>


    <QuestionFullName name="With EST Cluster Overlap"
     hiddenName="GeneQuestions.GenesByESTClusterOverlap"
     fontSize="-1" fontFace="Arial,Helvetica">
     <Type>Structured</Type>
    <Parameter name="organism">
         <FlatVocabFormat>pulldown</FlatVocabFormat>
         <Align>right</Align>
      </Parameter>
    <Parameter name="bp_overlap_gte">
         <NonFlatVocabFormat>text</NonFlatVocabFormat>
         <Align>right</Align>
      </Parameter>
    </QuestionFullName>
```

```
          <QuestionFullName name="By RNA Type"
           hiddenName="GeneQuestions.GeneByType"
           fontSize="-1" fontFace="Arial,Helvetica">
            <Type>Structured</Type>
           <Parameter name="organism">
            <FlatVocabFormat>radio</FlatVocabFormat>
            <Align>right</Align>
           </Parameter>
          <Parameter name="rnatype">
             <FlatVocabFormat>multibox</FlatVocabFormat>
           <Align>right</Align>
           </Parameter>
         </QuestionFullName>

       <QuestionFullName name="By Exon Count"
          hiddenName="GeneQuestions.GenesByExonCount"
          fontSize="-1" fontFace="Arial,Helvetica">
             <Type>Structured</Type>
           <Parameter name="organism">
                 <FlatVocabFormat>pulldown</FlatVocabFormat>
                 <Align>right</Align>
            </Parameter>
           <Parameter name="num_exons_gte">
                  <NonFlatVocabFormat>text</NonFlatVocabFormat>
                  <Align>right</Align>
            </Parameter>
         </QuestionFullName>

        <QuestionFullName name="By Protein MW Range"
          hiddenName="GeneQuestions.ProteinsByMolWt"
          fontSize="-1" fontFace="Arial,Helvetica">
          <Type>Structured</Type>
          <Parameter name="organism"/>
          <Parameter name="upperMolWtBound"/>
          <Parameter name="lowerMolWtBound"/>
         </QuestionFullName>


</QuestionSetName>

 <QuestionSetName name="EstQuestions" type="Flat"
       prompt="EST Questions" fontSize="-1"
       fontFace="Arial,Helvetica" newGroup="true">

       <QuestionFullName name="By Library Name"
       hiddenName="EstQuestions.EstsByLibrary"
```

```
          fontSize="-1" fontFace="Arial,Helvetica">
          <Type>Structured</Type>
          <Parameter name="libraryId">
          <FlatVocabFormat>multibox</FlatVocabFormat>
          </Parameter>
        </QuestionFullName>

        <QuestionFullName name="Overlapping a Gene"
          hiddenName="EstQuestions.EstsWithGeneOverlap"
          fontSize="-1" fontFace="Arial,Helvetica">
          <Type>Structured</Type>
          <Parameter name="libraryId">
          <FlatVocabFormat>multibox</FlatVocabFormat>
         </Parameter>
         <Parameter name="bp_overlap_gte"/>
         <Parameter name="inNotIn"/>
       </QuestionFullName>

</QuestionSetName>


    <QuestionSetName name="ContigQuestions" type="Flat"
        prompt="Contig Questions" fontSize="-1"
        fontFace="Arial,Helvetica" newGroup="true">

        <QuestionFullName name="Show Contig"
           hiddenName="ContigQuestions.ContigLookup"
           fontSize="-1" fontFace="Arial,Helvetica">
            <Type>Flat</Type>
            <Parameter name="contig">
                 <NonFlatVocabFormat>text</NonFlatVocabFormat>
                 <Align>right</Align>
              </Parameter>
           </QuestionFullName>

       </QuestionSetName>

    <QuestionSetName name="ProteinQuestions" type="Flat"
         prompt="Protein Questions" fontSize="-1"
         fontFace="Arial,Helvetica" newGroup="true">

         <QuestionFullName name="By Protein ID"
             hiddenName="ProteinQuestions.ProteinLookup"
             fontSize="-1" fontFace="Arial,Helvetica">
             <Parameter name="protein_id"/>
          </QuestionFullName>
```

```
      </QuestionSetName>
  </Page>

  <Page name="Second">
  </Page>

  <Page name="Third">
  </Page>

</Specifications>
```

APPENDIX B

MODEL CLASSES

The following is a listing of the Model classes that were created for handling custom specifications.

B.1 SPECIFICATIONSPARSER.JAVA

```
package org.gusdb.wdk.model.implementation;

import org.apache.commons.digester.*;
import org.xml.sax.SAXException;
import java.io.File;
import java.io.IOException;

import org.gusdb.wdk.model.QuestionSetName;
import org.gusdb.wdk.model.QuestionFullName;
import org.gusdb.wdk.model.Parameter;
import org.gusdb.wdk.model.Page;
import org.gusdb.wdk.model.Defaults;
import org.gusdb.wdk.model.PageDefaults;

import java.lang.*;
import java.util.ArrayList;

import org.gusdb.wdk.model.WdkModelException;

import java.net.URL;

import org.xml.sax.ErrorHandler;
import org.xml.sax.InputSource;

import com.thaiopensource.util.PropertyMap;
import com.thaiopensource.util.SinglePropertyMap;
import com.thaiopensource.validate.ValidateProperty;
import com.thaiopensource.validate.ValidationDriver;
import com.thaiopensource.xml.sax.ErrorHandlerImpl;
```

```
/** SpecificationsParser.java
* Created for Custom specifications
* @author Nivedita Kaluskar
**/

/**
 * Parses the contents of specifications XML file.
 *
 */


public class SpecificationsParser
{
 private ArrayList pageList;
 private static SpecificationsParser specification;
 private static final String DEFAULT_SCHEMA_NAME = "specifications.rng";

 public SpecificationsParser() {
    pageList = new ArrayList();
 }
 public void addPage(Page page) {
    pageList.add(page);
 }
 public ArrayList getPageList() {
        return pageList;
 }

 /**
  * Configures Digester rules and actions, parses the XML file specified
  * as the first argument.
  *
 */
 public static void main(String[] args)
 {
   URL specXmlURL=null, specSchemaURL=null;
   try {
     SpecificationsParser abp =  parseSpecFile(specXmlURL, specSchemaURL);
     System.out.println(abp.toString());
     } catch (Exception e) {System.err.println(e.getMessage());
   }
 }
 public static SpecificationsParser parseSpecFile(URL specXmlURL,
   URL specSchemaURL) throws IOException, SAXException, WdkModelException {

   // NOTE: we are validating before we substitute in the properties
   // so that the validator will operate on a file instead of a stream.
   // this way the validator spits out line numbers for errors
```

```
 if (!validModelFile(specXmlURL, specSchemaURL)) {
    throw new WdkModelException("Spec validation failed");
 }
SpecificationsParser abp = null;
// instantiate Digester and disable XML validation
try {
  Digester digester = new Digester();
  digester.setValidating(false);

// instantiate SpecificationsParser class
 digester.addObjectCreate("Specifications", SpecificationsParser.class);
// instantiate Defaults class
digester.addObjectCreate("Specifications/Defaults", Defaults.class);
digester.addObjectCreate("Specifications/Page", Page.class);
digester.addObjectCreate("Specifications/Page/PageDefaults",
                         PageDefaults.class);
digester.addObjectCreate("Specifications/Page/QuestionSetName",
                         QuestionSetName.class);
digester.addObjectCreate("Specifications/Page/QuestionSetName/
                         QuestionFullName", QuestionFullName.class);
digester.addObjectCreate("Specifications/Page/QuestionSetName/
                         QuestionFullName/Parameter", Parameter.class);
digester.addSetNext("Specifications/Page/QuestionSetName/
                         QuestionFullName/Parameter",
                           "addParameter");
digester.addSetProperties("Specifications/Page/QuestionSetName/
                         QuestionFullName/Parameter");

digester.addCallMethod("Specifications/Defaults/
                        QuestionType","setQuestionType",0);
digester.addCallMethod("Specifications/Defaults/
                        FlatVocabFormat","setFlatVocabFormat",0);
digester.addCallMethod("Specifications/Defaults/
                        NonFlatVocabFormat", "setNonFlatVocabFormat",0);
digester.addCallMethod("Specifications/Defaults/
                        ParamAlign","setParamAlign",0);

digester.addCallMethod("Specifications/Page/PageDefaults/
                        QuestionType","setQuestionType",0);
digester.addCallMethod("Specifications/Page/PageDefaults/FlatVocabFormat",
                       "setFlatVocabFormat",0);
digester.addCallMethod("Specifications/Page/PageDefaults/NonFlatVocabFormat",
                       "setNonFlatVocabFormat",0);
digester.addCallMethod("Specifications/Page/PageDefaults/
                       ParamAlign","setParamAlign",0);

digester.addCallMethod("Specifications/Page/QuestionSetName/
```

```
                    QuestionFullName/Parameter/FlatVocabFormat",
                    "setFlatVocabFormat",0);
  digester.addCallMethod("Specifications/Page/QuestionSetName/
                    QuestionFullName/Parameter/NonFlatVocabFormat",
                    "setNonFlatVocabFormat",0);

  digester.addCallMethod("Specifications/Page/QuestionSetName/
                      QuestionFullName/Parameter/Align","setAlign",0);
  digester.addSetProperties("Specifications/Page","name", "name" );
  digester.addSetProperties("Specifications/Page/
                           QuestionSetName","name", "name" );
  digester.addSetProperties("Specifications/Page/
                           QuestionSetName","type", "type" );
  digester.addSetProperties("Specifications/Page/
                           QuestionSetName/QuestionFullName",
                          "name", "name" );
  digester.addSetProperties("Specifications/Page/
                          QuestionSetName/
                          QuestionFullName","hiddenName",
                          "hiddenName" );

  // set different properties of Contact instance using specified methods
  digester.addCallMethod("Specifications/Page/QuestionSetName/
                       QuestionFullName/Type","setType", 0);
  digester.addCallMethod("Specifications/Page/QuestionSetName/
                       QuestionFullName/Page","setPage", 0);

  digester.addSetNext("Specifications/Page", "addPage");
  digester.addSetNext("Specifications/Page/
                      PageDefaults", "addPageDefaults");
  digester.addSetNext("Specifications/Page/
                      QuestionSetName", "addQuestionSetName");
  digester.addSetNext("Specifications/Page/QuestionSetName/
                    QuestionFullName","addQuestionFullName");

// now that rules and actions are configured, start the parsing process

   File input = new File("/usr/local/tomcat/webapps/
                       WDKToySite1.7/WEB-INF/wdk-model/
                       config/specifications.xml");
   abp = (SpecificationsParser)digester.parse(input);
   System.out.println("abp " + abp);
   System.out.println("Parse done in new SpecificationsParser");
   System.err.print(abp.toString());


        } catch(IOException ioex) { System.err.println(ioex.getMessage());
```

```
     } catch(SAXException saex) {System.err.println(saex.getMessage());
     }
for (int i=0;i<abp.pageList.size();i++)
     {
      Page page = (Page)abp.pageList.get(i);

    PageDefaults pageDefaults= new PageDefaults();

    if (page.getPageDefaults().size() != 0)
      pageDefaults=(PageDefaults)page.getPageDefaults().get(0);

     for (int j=0;j<page.getQuestionSetList().size();j++)
       {
        QuestionSetName qSetName = (
                    QuestionSetName)page.getQuestionSetList().get(j);
       for (int k=0;k<qSetName.getQuestionList().size();k++)
        {
         QuestionFullName qName = (
                  QuestionFullName)qSetName.getQuestionList().get(k);

         System.out.println(page.getName() + " Page Default "+ pageDefaults);

         if (qName.getType() == "" && pageDefaults != null)
           qName.setType(pageDefaults.getQuestionType());
         if (qName.getType() == "")
           qName.setType(Defaults.getQuestionType());

          for (int m=0;m<qName.getParameters().size();m++)
           {
           Parameter pName = (Parameter)qName.getParameters().get(m);
           if (pName.getFlatVocabFormat() == "" && pageDefaults != null) {
            pName.setFlatVocabFormat(pageDefaults.getFlatVocabFormat());
           }

           if (pName.getFlatVocabFormat() == "") {
            pName.setFlatVocabFormat(Defaults.getFlatVocabFormat());
           }
            if (pName.getNonFlatVocabFormat() == "" && pageDefaults != null)
            pName.setNonFlatVocabFormat(pageDefaults.getNonFlatVocabFormat());
           if (pName.getNonFlatVocabFormat() == "")
            pName.setNonFlatVocabFormat(Defaults.getNonFlatVocabFormat());

            if (pName.getAlign() == "" && pageDefaults != null)
            pName.setAlign(pageDefaults.getParamAlign());
           if (pName.getAlign() == "")
            pName.setAlign(Defaults.getParamAlign());
           }
```

```
         }
        }
       }
      return abp;
    }
private static boolean validModelFile(URL specXmlURL, URL specSchemaURL)
                      throws WdkModelException {

  System.setProperty("org.apache.xerces.xni.parser.XMLParserConfiguration",
                  "org.apache.xerces.parsers.XIncludeParserConfiguration");

     try {
       ErrorHandler errorHandler = new ErrorHandlerImpl(System.err);
       PropertyMap schemaProperties = new SinglePropertyMap(
                         ValidateProperty.ERROR_HANDLER, errorHandler);
       ValidationDriver vd = new ValidationDriver(schemaProperties,
                            PropertyMap.EMPTY, null);
       vd.loadSchema(ValidationDriver.uriOrFileInputSource(
                            specSchemaURL.toExternalForm()));

       InputSource is = ValidationDriver.uriOrFileInputSource(
                              specXmlURL.toExternalForm());
            //            return vd.validate(new InputSource(modelXMLStream));
       return vd.validate(is);

      } catch (SAXException e) {
            throw new WdkModelException(e);
        } catch (IOException e) {
            throw new WdkModelException(e);
        }
    }
}
```

B.2 DEFAULTS.JAVA

```
package org.gusdb.wdk.model;

/** Defaults.java
* Created for Custom specifications
* @author Nivedita Kaluskar
**/

public class Defaults {
```

```
  private static String questionType="";
  private static String flatVocabFormat="";
  private static String nonFlatVocabFormat="";
  private static String paramAlign="";

  public void setQuestionType(String newQuestionType) {
        questionType = newQuestionType;
  }

  public static String getQuestionType() {
    return questionType;
  }

  public void setFlatVocabFormat(String newFlatVocabFormat) {
       flatVocabFormat = newFlatVocabFormat;
 }

 public static String getFlatVocabFormat() {
       return flatVocabFormat;
 }

 public void setNonFlatVocabFormat(String newNonFlatVocabFormat) {
        nonFlatVocabFormat = newNonFlatVocabFormat;
  }

public static String getNonFlatVocabFormat() {
       return nonFlatVocabFormat;
 }
public static String getParamAlign() {
        return paramAlign;
 }

 public String setParamAlign(String newParamAlign) {
        return paramAlign = newParamAlign;
 }

  public String toString() {
      String newline = System.getProperty( "line.separator" );
      StringBuffer buf = new StringBuffer(
                         "Question type: questionType='"
                        +  questionType + "'" + newline
                        + "FlatVocabFormat='"  + flatVocabFormat
                        +   "'" + newline + "NonFlatVocabFormat='"
                        + nonFlatVocabFormat + "'"  + newline
                        + "ParamAlign='" + paramAlign + "'");

      return buf.toString();
```

```
    }

}
```

B.3 PAGEDEFAULTS.JAVA

```
package org.gusdb.wdk.model;

/** PageDefaults.java
* Created for Custom specifications
* @author Nivedita Kaluskar
**/


public class PageDefaults {

   private String questionType="";
   private String flatVocabFormat="";
   private String nonFlatVocabFormat="";
   private String paramAlign="";

   public void setQuestionType(String newQuestionType) {
         questionType = newQuestionType;
   }

   public String getQuestionType() {
     return questionType;
   }

   public void setFlatVocabFormat(String newFlatVocabFormat) {
        flatVocabFormat = newFlatVocabFormat;
  }

  public String getFlatVocabFormat() {
        return flatVocabFormat;
  }

  public void setNonFlatVocabFormat(String newNonFlatVocabFormat) {
         nonFlatVocabFormat = newNonFlatVocabFormat;
   }

 public String getNonFlatVocabFormat() {
        return nonFlatVocabFormat;
  }
```

```
  public String getParamAlign() {
         return paramAlign;
  }

  public String setParamAlign(String newParamAlign) {
         return paramAlign = newParamAlign;
  }

   public String toString() {
       String newline = System.getProperty( "line.separator" );
       StringBuffer buf = new StringBuffer("Question type: questionType=’"
                                           + questionType + "’" + newline
                                           + "FlatVocabFormat=’"
                                           + flatVocabFormat + "’"
                                           + newline + "NonFlatVocabFormat=’"
                                           + nonFlatVocabFormat
                                           + "’"  + newline
                                           + "ParamAlign=’"
                                           + paramAlign + "’");

       return buf.toString();
    }

}
```

B.4 PAGE.JAVA

```
package org.gusdb.wdk.model;
import java.util.ArrayList;

/** Page.java
* Created for Custom specifications
* @author Nivedita Kaluskar
**/


 public class Page
    {
        private String name;
        private ArrayList questionSetList;
        private ArrayList pageDefaultsList;

        public Page() {
              questionSetList = new ArrayList();
              pageDefaultsList = new ArrayList();
```

```
        }

        public void setName(String newName)
        {
            name = newName;
        }
        public String getName()
        {
            return name;
        }

      public void addQuestionSetName(QuestionSetName questionSetName)
        {

            questionSetList.add(questionSetName);
        }

        public ArrayList getQuestionSetList()
        {
            return questionSetList;
        }

       public void addPageDefaults(PageDefaults pageDefaults)
        {
            pageDefaultsList.add(pageDefaults);
        }

       public ArrayList getPageDefaults()
        {
           return pageDefaultsList;
        }

       public String toString() {
       String newline = System.getProperty( "line.separator" );
       StringBuffer buf = new StringBuffer("Page: name='" + name + "'");

       return buf.toString();
    }


 }
```

B.5 QUESTIONSETNAME.JAVA

```
package org.gusdb.wdk.model;
```

```
import java.util.ArrayList;

/** QuestionSetName.java
* Created for Custom specifications
* @author Nivedita Kaluskar
**/


 public class QuestionSetName
    {
        private String name;
        private String type;
        private String prompt;
        private String fontSize;
        private String fontFace;
        private String newGroup;
        private ArrayList questionList;
        private Page page;

        public QuestionSetName() {
              questionList = new ArrayList();
        }
        public void setName(String newName)
        {
            name = newName;
        }
        public String getName()
        {
            return name;
        }

        public void setType(String newType)
        {
            type = newType;
        }
        public String getType()
        {
            return type;
        }

        public void setPrompt(String newPrompt)
        {
            prompt = newPrompt;
        }
        public String getPrompt()
        {
            return prompt;
```

```
  }

  public void setFontSize(String newFontSize)
  {
      fontSize = newFontSize;
  }
  public String getFontSize()
  {
      return fontSize;
  }

   public void setFontFace(String newFontFace)
  {
      fontFace = newFontFace;
  }
  public String getFontFace()
  {
      return fontFace;
  }

  public void setNewGroup(String newGroupVal)
  {
      newGroup = newGroupVal;
  }
  public String getNewGroup()
  {
      return newGroup;
  }

public void addQuestionFullName(QuestionFullName questionFullName)
  {
      questionFullName.setPage(page);
      questionList.add(questionFullName);
  }

  public ArrayList getQuestionList()
  {
    return questionList;
  }


 public String toString() {
 String newline = System.getProperty( "line.separator" );
 StringBuffer buf = new StringBuffer(
                        "QuestionSetName: name='"
                       + name + "'"
                       + newline
```

```
                            + "Type='"
                            + type + "'");

       return buf.toString();
    }
 }
```

B.6 QUESTIONFULLNAME.JAVA

```
package org.gusdb.wdk.model;
import java.util.ArrayList;
import org.gusdb.wdk.model.Defaults;
import org.gusdb.wdk.model.PageDefaults;
import org.gusdb.wdk.model.Parameter;

/** QuestionFullName.java
* Created for Custom specifications
* @author Nivedita Kaluskar
**/

 public class QuestionFullName
    {
        private String name;
        private String hiddenName;
        private String fontFace;
        private String fontSize;
        private Defaults defaults;
        private Parameter parameter;
        private String type="";
        private Page page;
        private ArrayList parameters;


        public QuestionFullName() {
             parameters = new ArrayList();
             parameter = new Parameter();
        }

        public void setName(String newName)
         {
            name = newName;
         }
        public String getName()
        {
```

```
    return name;
}

public void setHiddenName(String newHiddenName)
{
   hiddenName = newHiddenName;
}

public String getHiddenName()
{
    return hiddenName;
}

public void setType(String newType)
{
    type = newType;
}
public String getType()
{
    return type;
}

public void setFontSize(String newFontSize)
{
    fontSize = newFontSize;
}
public String getFontSize()
{
    return fontSize;
}

public void setFontFace(String newFontFace)
{
    fontFace = newFontFace;
}
public String getFontFace()
{
    return fontFace;
}

public void setPage(Page newPage)
{
    page = newPage;
}

public Page getPage()
{
```

```
            return page;
        }

       public void addParameter(Parameter parameter)
        {
            parameters.add(parameter);
        }

        public ArrayList getParameters()
        {
             return parameters;
        }

       public String toString() {
       String newline = System.getProperty( "line.separator" );
       StringBuffer buf = new StringBuffer(
                              "QuestionFullName: name='"
                              + name + "'"
                              + newline
                              + "Type='"
                              + type + "'");

       return buf.toString();
    }


 }
```

B.7 PARAMETER.JAVA

```
package org.gusdb.wdk.model;

import org.gusdb.wdk.model.Defaults;
import org.gusdb.wdk.model.PageDefaults;

/** Parameter.java
* Created for Custom specifications
* @author Nivedita Kaluskar
**/

public class Parameter {

   private String name;
   private String format="";
```

```
 private String flatVocabFormat="";
 private String nonFlatVocabFormat="";
 private String align="";

 public Parameter() {

  }

public void setName(String newName) {
       name = newName;
 }

 public String getName() {
   return name;
 }

 public void setFormat(String newFormat) {
    format = newFormat;
 }

 public String getFormat() {
     return format;
 }

 public void setFlatVocabFormat(String newFlatVocabFormat) {
      flatVocabFormat = newFlatVocabFormat;
}

public String getFlatVocabFormat() {
      return flatVocabFormat;
}

 public void setNonFlatVocabFormat(String newNonFlatVocabFormat) {
      nonFlatVocabFormat = newNonFlatVocabFormat;
}

public String getNonFlatVocabFormat() {
      return nonFlatVocabFormat;
}

public void setAlign(String newAlign) {
       align = newAlign;
 }

public String getAlign() {
       return align;
}
```

```
  public String toString() {
       String newline = System.getProperty( "line.separator" );
       StringBuffer buf = new StringBuffer(
                                  "Parameter: name='"
                                  + name + "'"
                                  + newline
                                  + "Format='"
                                  + format + "'"
                                  + newline
                                  + "Align='"
                                  + align + "'");

       return buf.toString();
    }

}
```

APPENDIX C

CONTROLLER CLASSES

The following is a listing of the Controller classes that were created for handling custom specifications.

C.1 NEXTPAGEACTION.JAVA

```
import java.util.HashMap;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionServlet;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.gusdb.wdk.controller.CConstants;
import org.gusdb.wdk.model.jspwrap.WdkModelBean;
import org.gusdb.wdk.model.jspwrap.EnumParamBean;
import org.gusdb.wdk.model.jspwrap.QuestionBean;
import org.gusdb.wdk.model.jspwrap.AnswerBean;
import org.gusdb.wdk.model.jspwrap.BooleanQuestionLeafBean;

/** NextPageAction.java
* Created to handle multi-page display for custom specifications
* @author Nivedita Kaluskar
**/

public class NextPageAction extends Action {
    public ActionForward execute(ActionMapping mapping,
                                 ActionForm form,
                                 HttpServletRequest request,
                                 HttpServletResponse response)
                          throws Exception {
```

```
        ActionForward forward = null;
        String  submitAction = request.getParameter(
                            CConstants.PQ_NEXT_PAGE_KEY);
      System.out.println("submitAction : " +submitAction);

        if (submitAction.equals(CConstants.PQ_NEXT_PAGE)){
            forward = mapping.findForward(CConstants.PQ_NEXT_PAGE_MAPKEY);
        }
        else if (submitAction.equals(CConstants.PQ_PAGE2)){
           forward = mapping.findForward(CConstants.PQ_PAGE2_MAPKEY);
       }
     forward = mapping.findForward(CConstants.PQ_PAGE3_MAPKEY);
       }


        return forward;
    }
}
```

C.2 ApplicationInitListener.java

```
package org.gusdb.wdk.controller;

import org.gusdb.wdk.model.RDBMSPlatformI;
import org.gusdb.wdk.model.WdkModel;
import org.gusdb.wdk.model.QuestionFullName;
import org.gusdb.wdk.model.implementation.SpecificationsParser;
import org.gusdb.wdk.model.jspwrap.WdkModelBean;
import org.gusdb.wdk.model.WdkModelException;

import java.io.IOException;
import java.lang.reflect.Method;
import java.net.MalformedURLException;
import java.net.URL;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.jsp.jstl.core.Config;
import javax.sql.DataSource;


/**
 * A class that is initialised at the start of the web application.
 * This makes sure global resources
 * are available to all the contexts that need them
```

```
 *
 */
/** Modified to handle custom specifications
 *  @author Nivedita Kaluskar
 **/

public class ApplicationInitListener implements ServletContextListener {

    private RDBMSPlatformI platform;

    public void contextDestroyed(ServletContextEvent sce) {
        try {
            getPlatform().close();
        } catch (WdkModelException exp) {
            throw new RuntimeException(exp);
        }
    }

public void contextInitialized(ServletContextEvent sce) {

        ServletContext application  = sce.getServletContext(  );

        String configXml = application.getInitParameter(
                        CConstants.WDK_MODELCONFIGXML_PARAM);
        String modelXml = application.getInitParameter(
                        CConstants.WDK_MODELXML_PARAM);
        String specXml = application.getInitParameter(
                        CConstants.WDK_SPECXML_PARAM);
        String schema = application.getInitParameter(
                        CConstants.WDK_MODELSCHEMA_PARAM);
        String specSchema =  application.getInitParameter(
                        CConstants.WDK_SPECSCHEMA_PARAM);
        String props = application.getInitParameter(
                        CConstants.WDK_MODELPROPS_PARAM);
        String parserClass = application.getInitParameter(
                        CConstants.WDK_MODELPARSER_PARAM);
        String specParserClass = application.getInitParameter(
                        CConstants.WDK_SPECPARSER_PARAM);
        String customViewDir = application.getInitParameter(
                        CConstants.WDK_CUSTOMVIEWDIR_PARAM);

        initMemberVars(configXml, modelXml, specXml, schema, specSchema,
             props, parserClass, specParserClass, customViewDir, application);

        //Config.set(application, Config.SQL_DATA_SOURCE, dataSource);
    }
```

```
    public static boolean resourceExists(
                    String path, ServletContext servletContext) {
        try {
            URL url = servletContext.getResource(path);
            return url !=null;
        }
        catch (MalformedURLException exp) {
            RuntimeException e = new RuntimeException(exp);
            throw e;
        }
    }

    private static URL createURL(String param,
                String defaultLoc, ServletContext application) {

        if (param == null) {
            param = defaultLoc;
        }

        URL ret = null;
        try {
            ret = application.getResource(param);
if (ret ==null) {
                RuntimeException e = new RuntimeException("Missing resource.
                                        Unable to create URL from "+param);
                throw e;
            }
        }
        catch (MalformedURLException exp) {
            RuntimeException e = new RuntimeException(exp);
            throw e;
        }
        return ret;
    }


    protected RDBMSPlatformI getPlatform() {
        return platform;
    }
    protected void setPlatform(RDBMSPlatformI platform) {
        this.platform = platform;
    }

    private void initMemberVars(String configXml, String modelXml,
                                String specXml, String schema,
                                String specSchema,
                                String props, String parserClass,
```

```
                                String specParserClass, String customViewDir,
                                ServletContext application) {
        URL schemaURL = null;
        URL specSchemaURL = null;

        if (schema != null) {
            schemaURL = createURL(schema, null, application);
        } else {
            throw new RuntimeException(
                       "can not start application because schema is absent");
        }
       //adding here for spec schema

       if (specSchema != null) {
            specSchemaURL = createURL(specSchema, null, application);
        } else {
            throw new RuntimeException(
                 "can not start application because spec schema is absent");
        }

  if (parserClass == null) {
            parserClass = CConstants.DEFAULT_WDKMODELPARSER;
        }
        if (specParserClass == null) {
            specParserClass = CConstants.DEFAULT_SPECMODELPARSER;
        }
        if (customViewDir == null) {
            customViewDir = CConstants.DEFAULT_WDKCUSTOMVIEWDIR;
        }

        URL modelURL = createURL(modelXml,
                        CConstants.DEFAULT_WDKMODELXML, application);
        URL configURL = createURL(configXml,
                        CConstants.DEFAULT_WDKMODELCONFIGXML, application);
        URL propsURL = createURL(props, CConstants.DEFAULT_WDKMODELPROPS,
                       application);
        URL specURL = createURL(specXml, CConstants.DEFAULT_SPECMODELXML,
                      application);


        System.out.println("modelXML: " + modelURL);
        System.out.println("specXML: " + specURL);

        // read config info
        try {

            Class parser = Class.forName(parserClass);
```

```
            Method build = parser.getDeclaredMethod("parseXmlFile", new Class[] {
                            URL.class, URL.class, URL.class, URL.class});
            System.out.println("Build :" + build);
            WdkModel wdkModelRaw = (WdkModel) build.invoke(null, new Object[] {
                           modelURL, propsURL, schemaURL, configURL});

            //System.out.println("wdkModelRaw: " + wdkModelRaw);
            WdkModelBean wdkModel = new WdkModelBean(wdkModelRaw);


            setPlatform(wdkModelRaw.getRDBMSPlatform());
            application.setAttribute(CConstants.WDK_MODEL_KEY, wdkModel);
            application.setAttribute(
                        CConstants.WDK_CUSTOMVIEWDIR_KEY, customViewDir);
        } catch (Exception exp) {
            exp.printStackTrace();
            throw new RuntimeException(exp);
        }
         // read spec info

           try {

            Class parser2 = Class.forName(specParserClass);
            System.out.println("Parser: " + parser2);
            Method build2 = parser2.getDeclaredMethod(
                         "parseSpecFile", new Class[] {URL.class, URL.class});
            System.out.println("Build2 : " + build2);
            SpecificationsParser specRaw = (SpecificationsParser)
                       build2.invoke(null, new Object[]{specURL, specSchemaURL});
            //WdkModelBean wdkModel = new WdkModelBean(wdkModelRaw);

            //setPlatform(wdkModelRaw.getRDBMSPlatform());
            application.setAttribute(CConstants.SPEC_MODEL_KEY, specRaw);
            //application.setAttribute(
                         CConstants.WDK_CUSTOMVIEWDIR_KEY, customViewDir);
        } catch (Exception exp) {
            exp.printStackTrace();
            throw new RuntimeException(exp);
        }


    }
}
```

C.3 CCONSTANTS.JAVA

```
package org.gusdb.wdk.controller;

/**Modified to handle custom specifications
 * @author Nivedita Kaluskar
 **/

public class CConstants {
    private CConstants() {
        ; // no-op
    }
    //key for objects in cache, used in Action/ActionForm classes and
    //maybe jsp pages
    public static final String WDK_RESULTFACTORY_KEY = "wdkResultFactory";
    public static final String WDK_MODEL_KEY = "wdkModel";
    //added here
    public static final String SPEC_MODEL_KEY = "specRaw";
    //adding from original WDK1.7 CConstants file.
    public static final String WDK_CUSTOMVIEWDIR_KEY = "wdkCustomeViewDir";
    public static final String WDK_CUSTOM_QUESTIONSETS_FLAT_PAGE =
                                                "customQuestionSetsFlat.jsp";
    public static final String WDK_CUSTOM_QUESTIONSETS_PAGE =
                                                 "customQuestionSets.jsp";
    public static final String WDK_CUSTOM_QUESTION_PAGE =
                                                   "customQuestion.jsp";
    public static final String WDK_CUSTOM_SUMMARY_PAGE =
                                                    "customSummary.jsp";
    public static final String WDK_CUSTOM_RECORD_PAGE =
                                                   "customRecord.jsp";
    public static final String WDK_CUSTOM_HISTORY_PAGE =
                                                  "customQueryHistory.jsp";

    public static final String WDK_QUESTION_KEY = "wdkQuestion";
    public static final String WDK_QUESTION_PARAMS_KEY =
                                                  "wdkQuestionParams";
    public static final String WDK_ANSWER_KEY = "wdkAnswer";
    public static final String WDK_RECORD_KEY = "wdkRecord";
    public static final String NEXT_QUESTION_OPERAND =
                                               "nextQuestionOperand";
    //public static final String QUESTIONSETFORM_KEY = "questionSetForm";
    public static final String QUESTIONFORM_KEY = "questionForm";
    public static final String BOOLEAN_QUESTION_FORM_KEY =
                                                   "booleanQuestionForm";
    public static final String BOOLEAN_SEED_QUESTION_KEY =
                                                "booleanSeedQuestionName";
```

```
  public static final String CURRENT_BOOLEAN_ROOT_KEY =
                                                  "currentBooleanRoot";
  public static final String BOOLEAN_OPERATIONS_PARAM_NAME = "booleanOps";
  public static final String DOWNLOAD_RESULT_KEY = "downloadResult";

  //key for finding action forward, from struts-config.xml,
 //used in Action classes
  public static final String SHOW_QUESTION_MAPKEY = "show_question";
 public static final String SHOW_QUESTIONSETS_MAPKEY =
                                                    "show_questionsets";
  public static final String SHOW_QUESTIONSETSFLAT_MAPKEY =
                                                "show_questionsetsflat";
  public static final String PROCESS_QUESTIONSETSFLAT_MAPKEY =
                                              "process_questionsetsflat";
  public static final String SHOW_SUMMARY_MAPKEY = "show_summary";
  public static final String SHOW_RECORD_MAPKEY = "show_record";
  public static final String PQ_SHOW_SUMMARY_MAPKEY = "pq_show_summary";
  public static final String PQ_START_BOOLEAN_MAPKEY = "pq_start_boolean";
  public static final String PBQ_GET_BOOLEAN_ANSWER_MAPKEY =
                                                    "pbq_get_boolean_answer";
  public static final String PBQ_GROW_BOOLEAN_MAPKEY = "pbq_grow_boolean";
  public static final String GROW_BOOLEAN_MAPKEY = "grow_boolean";
  public static final String GET_BOOLEAN_ANSWER_MAPKEY =
                                                     "get_boolean_answer";
  public static final String CONFIG_DOWNLOAD_MAPKEY = "config_download";
  public static final String GET_DOWNLOAD_RESULT_MAPKEY =
                                              "get_download_result";
 // Added from original WDK1.7 CConstants file.
public static final String SHOW_QUERY_HISTORY_MAPKEY =
                                         "show_query_history";
public static final String DOWNLOAD_HISTORY_ANSWER_MAPKEY =
                                       "download_history_answer";
public static final String DELETE_HISTORY_ANSWER_MAPKEY =
                                              "delete_history_answer";
public static final String PROCESS_BOOLEAN_EXPRESSION_MAPKEY =
                                     "process_boolean_expression";


  /* Adding here for multi page display */

  public static final String PQ_NEXT_PAGE_MAPKEY = "pq_next_page";
  public static final String PQ_PAGE2_MAPKEY = "pq_page2";
  public static final String PQ_PAGE3_MAPKEY = "pq_page3";

  public static final String PQ_NEXT_PAGE_KEY = "nextPage";
  public static final String PQ_NEXT_PAGE = "Next Page";
```

```
public static final String PQ_PAGE2 = "Page 2";
public static final String PQ_PAGE3 = "Page 3";

/* End additional stmts for multi page display */

//button click detectors, used in action, action forms, and jsp pages
//match question.jsp
public static final String PQ_SUBMIT_KEY = "questionSubmit";
//match question.jsp
public static final String PQ_SUBMIT_GET_ANSWER = "Get Answer";
//match question.jsp
public static final String PQ_SUBMIT_EXPAND_QUERY = "Expand Question";
//match booleanQuestion.jsp
public static final String PBQ_SUBMIT_KEY = "process_boolean_question";
 //match booleanQuestion.jsp
 public static final String PBQ_SUBMIT_GET_BOOLEAN_ANSWER =
                                          "Retrieve Answer";
 //match WEB-INF/includes/booleanQuestionNode.jsp
public static final String PBQ_SUBMIT_GROW_BOOLEAN = "Expand";
//match summary.jsp
public static final String PD_CHOOSE_KEY = "chooseFields";
//match downloadConfig.jsp
public static final String DOWNLOAD_INCLUDE_HEADER = "includeHeader";
public static final String YES = "yes"; //match downloadConfig.jsp
public static final String ALL = "all"; //match downloadConfig.jsp
//Adding from original WDK1.7 CConstants.
 //match queryHistory.jsp
 public static final String USER_ANSWER_ID = "user_answer_id";

//used in action, action forms, and jsp pages
//match WEB-INF/includes/booleanQuestionNode.jsp
public static final String NEXT_QUESTION_OPERAND_SUFFIX =
                                        "_nextQuestionOperand";
//match WEB-INF/includes/booleanQuestionNode.jsp
public static final String NEXT_BOOLEAN_OPERATION_SUFFIX =
                                        "_nextBooleanOperation";

//name of webapp init params, from web.xml, used in
// ApplicationInitListener.java
protected static final String WDK_MODELCONFIGXML_PARAM =
                                        "wdkModelConfigXml_param";
protected static final String WDK_MODELXML_PARAM =
                                            "wdkModelXml_param";
//added here
  protected static final String WDK_SPECXML_PARAM = "wdkSpecXml_param";
protected static final String WDK_MODELSCHEMA_PARAM =
                                                "wdkModelSchema_param";
```

```
    //added for spec schema
    protected static final String WDK_SPECSCHEMA_PARAM =
                                                    "wdkSpecSchema_param";

    protected static final String WDK_MODELPROPS_PARAM =
                                                    "wdkModelProps_param";
    protected static final String WDK_LOGFILE_PARAM = "wdkLogFile_param";
    protected static final String WDK_MODELPARSER_PARAM =
                                                  "wdkModelParser_param";
    //added here
    protected static final String WDK_SPECPARSER_PARAM =
                                                    "wdkSpecParser_param";

    protected static final String WDK_CUSTOMVIEWDIR_PARAM =
                                                "wdkCustomViewDir_param";

    //default value of webapp init params, from web.xml, used in
    //ApplicationInitListener.java
    protected static final String DEFAULT_WDKMODELCONFIGXML =
                                  "/WEB-INF/wdk-config/wdkModelConfig.xml";
    protected static final String DEFAULT_WDKMODELXML =
                                       "/WEB-INF/wdk-config/wdkModel.xml";
    protected static final String DEFAULT_WDKMODELSCHEMA =
                                     "/WEB-INF/wdk-config/wdkModel.rng";
    protected static final String DEFAULT_WDKMODELPROPS =
                                      "/WEB-INF/wdk-config/wdkModel.props";
    protected static final String DEFAULT_WDKMODELPARSER =
                        "org.gusdb.wdk.model.implementation.ModelXmlParser";
  //added here
 protected static final String DEFAULT_SPECMODELXML =
                                   "/WEB-INF/wdk-config/specifications.xml";
 protected static final String DEFAULT_SPECMODELPARSER =
                           "org.gusdb.model.implementation.SpecificationsParser";
   protected static final String DEFAULT_WDKCUSTOMVIEWDIR = "/customPages/";
}
```

APPENDIX D

VIEW

D.1 INDEX.JSP

```
<%@ taglib prefix="site" tagdir="/WEB-INF/tags/site" %>
<%@ taglib prefix="wdk" tagdir="/WEB-INF/tags/wdk" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib prefix="nested" uri="http://jakarta.apache.org/
                                struts/tags-nested" %>
<%@ taglib prefix="html" uri="http://jakarta.apache.org/
                                struts/tags-html" %>

<!-- get wdkModel saved in application scope -->
<c:set var="wdkModel" value="${applicationScope.wdkModel}"/>
<c:set var="parser" value="${applicationScope.specRaw}"/>
<c:set var="pageList" value="${parser.pageList}" />

<!-- top banner and introduction -->
<!-- get wdkModel name to display as page header -->

<c:set value="${wdkModel.displayName}" var="wdkModelDispName"/>
<site:header banner="${wdkModelDispName}" />

<!-- end top banner and introdction -->

<!-- horizontal tool bar-->
<table width="640" align="center">
<tr>
  <td>
    <jsp:include page="WEB-INF/includes/announcements.html"/>
   </td>
</tr>
<tr>
  <td>
    <jsp:include page="WEB-INF/includes/toolbar.html"/>
  </td>
```

```
</tr>
</table>
<!--end horizontal tool bar-->

<!-- main table -->
<table width="640" align="center" border="1" cellpadding="0"
 cellspacing="0">
<tr>
 <!-- side bar-->
 <td width="25%" valign="top">
   <jsp:include page="WEB-INF/includes/sidebar.html"/><br>
   <center><img src="images/CmurisSEM1_color_tiny.gif"/></center>
 </td>
 <!-- end side bar -->
 <!--space between sidebar and main body-->
 <td width="1%"> </td>
 <!-- end space -->
 <!-- Start of the main body  -->
 <td width="74%" valign="top">

 <!--THE SIMPLE SEARCH TABLE-->
 <table width="100%" border="1" cellspacing="0" cellpadding="0">
    <c:set value="${wdkModel.questionSets}" var="questionSets"/>
    <c:set var="i" value="0"/>
<!--loop 1: loop through all pages  -->

<c:forEach items="${pageList}" var="page">
 <c:if test="${page.name == 'First'}">
 <tr class="primary2">
<!-- loop through all the question sets on page 1 from specification file -->
  <c:forEach items="${page.questionSetList}" var="questionSet">
   <c:if test="${questionSet.type == 'Flat'}">

    <td colspan="100%"><font size="${questionSet.fontSize}" face=
    "${questionSet.fontFace}"><b><jsp:getProperty name="questionSet"
     property="prompt"/></b></font>
     </td>
 </tr>
 <jsp:useBean scope="request" id="helps" class="java.util.HashMap"/>
 <!-- loop through each question set from specification file -->
   <c:forEach items="${questionSet.questionList}" var="question">
 <!-- Loop through all the questions in the Model for each question
      from the specifications file to allow regrouping -->
     <c:set value="${wdkModel.questionSets}" var="qSets"/>
     <c:forEach items="${qSets}" var="qSet2">
      <c:forEach items="${qSet2.questions}" var="q">
       <c:if test="${question.name == q.displayName}">
```

```
        <c:set value="${qSet2.name}" var="qSetName"/>
        <!--SET UP HELP FOR EACH QUESTION -->
        <jsp:useBean id="helpQ"  class="java.util.HashMap" />
        <c:set value="${q.name}" var="qName"/>
        <c:set value="${q.displayName}" var="qDispName"/>
        <!-- Added for alternate row color -->
        <c:choose>
        <c:when test="${i % 2 == 0}"><tr class="rowLight"></c:when>
        <c:otherwise><tr class="rowMedium"></c:otherwise>
        </c:choose>
<!-- If question type is flat, display all the parameters -->

        <c:if test="${question.type == 'Flat'}">
          <html:form method="get" action="/processQuestionSetsFlat.do">
          <html:hidden property="questionFullName"
          value="${question.hiddenName}"/>
          <c:set value="0" var="numParams"/>
          <c:set value="false" var="needOrg"/>
          <c:forEach items="${q.params}" var="qP">
          <c:set value="${numParams+1}" var="numParams"/>
          <c:if test="${qP.name =='organism'}">
          <c:set value="true" var="needOrg"/>
          </c:if>
          </c:forEach>
<!--SET UP HELP FOR EACH QUESTION -->

          <c:set value="Help for question: ${qDispName}"
          var="fromAnchorQ"/>
          <c:set var="anchorQp" value="HELP_${fromAnchorQ}"/>
          <c:set target="${helps}" property="${fromAnchorQ}" value="${q}"/>
          <td colspan="100%" align="left" nowrap>
          <a name="${fromAnchorQ}"></a>
          <font size="${question.fontSize}" face="${question.fontFace}">
          <b>${qDispName}</b></font>
          <a href="#${anchorQp}">
          <font size="-2">
           (What's This?)
          </font>
          </a>
          </td>
          </tr>
          <c:set var="noSpace" value="false"/>
          <c:set value="${qSetName}_${qName}" var="qSetNameName"/>
          <c:forEach items="${q.params}" var="qP">
           <c:set value="${qP.name}" var="pNam"/>
           <c:set value="${qSetName}_${qName}_${pNam}" var="pNamKey"/>
           <c:forEach items="${question.parameters}" var="para">
```

```
            <c:if test="${para.name == qP.name}">
             <c:if test="${para.align == 'right'}">
              <c:choose>
              <c:when test="${i % 2 == 0}">
              <tr class="rowLight"></c:when>
              <c:otherwise><tr class="rowMedium"></c:otherwise>
              </c:choose>
              <td align="right">
              <font size="-1">
              <i><jsp:getProperty name="qP" property="prompt"/></i> :
              </font>
             </c:if> <!-- end right alignment for organism -->

             <c:if test="${para.align == 'left'}">
              <c:choose>
              <c:when test="${i % 2 == 0}">
              <tr class="rowLight"></c:when>
              <c:otherwise><tr class="rowMedium"></c:otherwise>
              </c:choose>
              <td align="left">
              <font size="-1">
              <i><jsp:getProperty name="qP" property="prompt"/></i> :
              </font>
             </c:if>

<c:choose>
 <c:when test="${qP.class.name eq
 'org.gusdb.wdk.model.jspwrap.FlatVocabParamBean'}">
          <c:if test="${para.flatVocabFormat == 'radio'}">
            <font size="-1">
            <c:forEach items="${qP.vocab}" var="flatVoc">
            <html:radio property="myMultiProp(${pNamKey})"
            value="${flatVoc}">
            <i><c:out value="${flatVoc}"/></i>
            </html:radio>
            </c:forEach>
            </font>
          </c:if>

          <c:if test="${para.flatVocabFormat == 'pulldown'}">
            <font size="-1">
            <html:select property="myMultiProp(${pNamKey})">
            <c:forEach items="${qP.vocab}" var="flatVoc">
            <html:option value="${flatVoc}">${flatVoc}</html:option>
            </c:forEach>
            </html:select>
          </c:if>
```

```
              <c:if test="${para.flatVocabFormat == 'scrollpane'}">
                <html:select  property="myMultiProp(${qSetNameName}_${pNam})"
                multiple="1">
                <c:set var="opt" value="${opt+1}"/>
                <c:set var="sel" value=""/>
                <c:if test="${opt == 1}"><c:set var="sel" value="selected"/>
                </c:if>
                <html:options property="values(${qSetNameName}_${pNam})"
                labelProperty="labels(${qSetNameName}_${pNam})"/>
                </html:select>
               </c:if>

               <c:if test="${para.flatVocabFormat == 'multibox'}">
                 <c:set value="${qP.name}" var="pNam"/>
                 <c:set var="mp" value="0"/>
                 <c:if test="${qP.multiPick}">
                 <c:set var="mp" value="1"/>
                 </c:if>
                 <c:set var="opt" value="0"/>
                 <c:forEach items="${qP.vocab}" var="flatVoc">
                 <html:multibox property="myMultiProp(${qSetNameName}_${pNam})">
                 <c:out value="${flatVoc}" />
                 </html:multibox>
                 <c:out value="${flatVoc} "/>
                 </c:forEach>
               </c:if>

</c:when>
<c:otherwise>
              <c:if test="${para.nonFlatVocabFormat == 'text'}">
                <font>
                <html:text property="myProp(${qSetNameName}_${qP.name})"
                 value="${qP.default}" size="14"/>
                </font>
               </c:if>
</c:otherwise>
</c:choose>

</c:if> <!-- end if test for para name = qP name -->
</tr>
</c:forEach> <!-- end for question.parameters -->

              <c:if test="${numParams==1}">
                <html:hidden property="questionFullName"
                 value="${question.hiddenName}" />
                 <c:set var="noSpace" value="true"/>
```

```
             </c:if>
 </td>
</c:forEach> <!-- end loop through all params -->
            <td colspan="4" align="right">
            nbsp;
            <html:hidden property="questionSubmit" value="Get Answer"/>
            <input name="go" value="go" type="image" src="images/go.gif"
             border="0" onmouseover="return true;">
            </td>
            </tr>

  </html:form>
 </c:if>   <!-- end if question type is flat -->


 <c:if test="${question.type == 'Structured'}">
     <c:choose>
     <c:when test="${i % 2 == 0}"><tr class="rowLight"></c:when>
     <c:otherwise><tr class="rowMedium"></c:otherwise>
     </c:choose>
     <td><!-- list of questions in a questionSet -->
     <!-- start question name and show question row -->

 <html:form method="get" action="/showQuestion.do">
     <c:set value="${qSet.name}" var="qSetName"/>
     <c:set value="${qSet.questions}" var="questions"/>
     <font size="${question.fontSize}" face="${question.fontFace}"><b>
     <jsp:getProperty name="q" property="displayName"/></b></font>

     <html:hidden property="questionFullName" value="${question.hiddenName}"/>
     <td colspan="6" align="right">
     <html:hidden property="questionSubmit" value="Get Answer"/>
     <input name="go" value="go" type="image" src="images/go.gif"
      border="0" onmouseover="return true;">
     </td>
</html:form>
    </td>
  </tr> <!-- end question name and show question row -->
</c:if>

 </c:if>
 </c:forEach>
</c:forEach> <!-- end iterating thru questions from spec first -->

<c:set var="i" value="${i+1}"/> <!-- for row color -->

</c:forEach> <!--end for each question in a question set-->
```

```
<c:if test="${questionSet.newGroup == 'true'}">
 <tr class="primary2">
</c:if>
<c:if test="${questionSet.newGroup == 'false'}">
 <tr>
</c:if>

</c:if> <!-- close if question set is flat -->


<c:if test="${questionSet.type == 'Structured'}">

   <td colspan="100%"><font size="${questionSet.fontSize}"
    face="${questionSet.fontFace}"><b><jsp:getProperty
    name="questionSet" property="prompt"/></b></font></td>
  </tr>

 <c:choose>
     <c:when test="${i % 2 == 0}"><tr class="rowLight"></c:when>
     <c:otherwise><tr class="rowMedium"></c:otherwise>
 </c:choose>

<td colspan="2" align="right">
 <html:form method="get" action="/showQuestion.do">
    <html:select property="questionFullName">
    <!-- c:forEach items="${questions}" var="q" -->
    <!-- changed the for loop to allow custom question sets -->
    <c:forEach items="${questionSet.questionList}" var="question">
    <c:set value="${question.name}" var="questionDispName"/>
    <c:set value="${question.hiddenName}" var="questionName"/>
    <font size="${question.fontSize}" face="${question.fontFace}">
    <html:option value="${questionName}">${questionDispName}
    </html:option>
    </font>
    </c:forEach>
    </html:select>
 </td>
 <td colspan="3" align="right"><html:hidden property="questionSubmit"
  value="Show Question"/>
 <input name="go" value="go" type="image" src="images/go.gif"
  border="0" onmouseover="return true;">
 </td>
 </html:form>
</tr></td>
 <c:set var="i" value="${i+1}"/> <!-- for row color -->
```

```
<c:if test="${questionSet.newGroup == 'true'}">
 <tr class="primary2">
</c:if>

<c:if test="${questionSet.newGroup == 'false'}">
 <tr>
</c:if>

</c:if> <!-- close if question set is structured -->
</c:forEach> <!-- end for each question set -->

</c:if>
</c:forEach> <!-- end if and forEach for page from spec file -->

</table><!--END OF SIMPLE SEARCH TABLE -->

<html:form method="get" action="/nextPage.do">
   <html:submit property="nextPage" value="Page 2" />
</html:form>

</table> <!-- end of main body table -->
   <!--footer-->
   <table width="640" align="center" border="0">
    <tr><td><hr></td></tr>
     <tr>
 <td>
    <jsp:include page="WEB-INF/includes/footer.html"/>
</td>
 </tr>
</table>


<site:footer/>
```

D.2 QUESTION.JSP

```
<%@ taglib prefix="site" tagdir="/WEB-INF/tags/site" %>
<%@ taglib prefix="wdk" tagdir="/WEB-INF/tags/wdk" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="html" uri="http://jakarta.apache.org/
    struts/tags-html" %>

<!-- get wdkQuestion; setup requestScope HashMap to collect help
     info for footer -->
```

```
<c:set value="${sessionScope.wdkQuestion}" var="wdkQuestion"/>
<c:set var="parser" value="${applicationScope.specRaw}"/>
<c:set var="pageList" value="${parser.pageList}" />

<jsp:useBean scope="request" id="helps" class="java.util.HashMap"/>

<c:forEach items="${pageList}" var="page">
 <c:if test="${page.name == 'First'}">

<c:forEach items="${page.questionSetList}" var="questionSet">
  <c:forEach items="${questionSet.questionList}" var="question">
    <c:if test="${question.name == wdkQuestion.displayName}">

<!-- Logo with title and horizontal toolbar -->
<table align='center' width="640" border='0'>
<tr><td align="right" width="20">
  <img src="images/oocyst_bg.gif" border='0' height='60' alt='logo'></td>
  <td align="center" valign="middle"><b><font face="Arial,Helvetica" size="+3">
  ${wdkQuestion.displayName}</font></b></td>
  <td align='right' width="20">
  <img src="images/CmurisSEM1_color_tiny.gif" alt='log'></td></tr>
<tr><td colspan="3"><jsp:include page="WEB-INF/includes/toolbar.html"/>
</td></tr>
<tr><td colspan="3"><hr></td></tr>
</table>

<!-- show all params of question, collect help info along the way -->
<c:set value="Help for question: ${wdkQuestion.displayName}" var="fromAnchorQ"/>
<jsp:useBean id="helpQ" class="java.util.HashMap"/>
<!-- put an anchor here for linking back from help sections -->

<A name="${fromAnchorQ}"></A>

<html:form method="get" action="/processQuestion.do">

<!-- show error messages, if any -->
<wdk:errors/>

<c:set value="${wdkQuestion.params}" var="qParams"/>
<c:forEach items="${qParams}" var="qP">
<c:forEach items="${question.parameters}" var="para">
  <c:if test="${para.name == qP.name}">
<!-- an individual param (can not use fullName, w/ '.', for mapped props) -->
 <c:set value="${qP.name}" var="pNam"/>

<table align='center' width="640" border='0'>
<tr><td align="left" width="200"></td><td align="left" width="370"></td>
```

```
<td align="right" width="70"></td></tr>
  <tr><td><b><jsp:getProperty name="qP" property="prompt"/></b></td>

<!-- choose between flatVocabParam and straight text or number param -->
<c:choose>
 <c:when test="${qP.class.name eq
  'org.gusdb.wdk.model.jspwrap.FlatVocabParamBean'}">
  <td>
  <c:set var="opt" value="0"/>
  <c:if test="${para.flatVocabFormat == 'scrollpane'}">
    <html:select  property="myMultiProp(${pNam})" multiple="1">
    <c:set var="opt" value="${opt+1}"/>
    <c:set var="sel" value=""/>
    <c:if test="${opt == 1}"><c:set var="sel" value="selected"/></c:if>
    <html:options property="values(${pNam})" labelProperty="labels(${pNam})"/>
    </html:select>
  </c:if>

  <c:if test="${para.flatVocabFormat == 'pulldown'}">
    <html:select  property="myMultiProp(${pNam})">
    <c:set var="opt" value="${opt+1}"/>
    <c:set var="sel" value=""/>
    <c:if test="${opt == 1}"><c:set var="sel" value="selected"/></c:if>
    <html:options property="values(${pNam})" labelProperty="labels(${pNam})"/>
    </html:select>
  </c:if>

  <c:if test="${para.flatVocabFormat == 'multibox'}">
    <c:set value="${qP.name}" var="pNam"/>
    <c:set var="mp" value="0"/>
    <c:if test="${qP.multiPick}">
    <c:set var="mp" value="1"/>
    </c:if>
    <c:set var="opt" value="0"/>
    <!-- begin select -->
    <c:forEach items="${qP.vocab}" var="flatVoc">
    <html:multibox property="myMultiProp(${pNam})" >
    <c:out value="${flatVoc}" />
    </html:multibox>
    <c:out value="${flatVoc} "/>
    </c:forEach>
  </c:if> <!-- close if to check multibox format -->

  <c:if test="${para.flatVocabFormat == 'radio'}">
    <c:forEach items="${qP.vocab}" var="flatVoc">
    <html:radio property="myMultiProp(${pNam})" value="${flatVoc}">
    <i><c:out value="${flatVoc}"/></i>
```

```
    </html:radio>
    </c:forEach>
  </c:if> <!-- close if to check radio format -->

 </td>
</c:when>
<c:otherwise>
  <td>
   <html:text property="myProp(${pNam})"/>
  </td>
 </c:otherwise>
</c:choose>

      <td>    </td>
 </c:if>
</c:forEach> <!-- closing tags for iteration thru spec file -->

</c:forEach>
<c:set target="${helps}" property="${fromAnchorQ}" value="${helpQ}"/>

 <td> </td>
<td>
 <html:submit property="questionSubmit" value="Get Answer"/>
</td>
</table>
</html:form>

</c:if>
</c:forEach> <!-- closing tags for spec -->

</c:forEach> <!-- added for iteration thru questionSetList -->
</c:if>
</c:forEach> <!-- added for iteration thru pageList -->

<table align="center" width="640">
<tr><td><hr></td></tr>
<tr><td><jsp:include page="WEB-INF/includes/footer.html"/></td></tr>
</table>

<site:footer/>
```

D.3 WEB.XML

```
<?xml version="1.0" ?>
<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
```

```
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>Sample GUS WebApp</display-name>
  <description>
    This is a simple webapp to demonstrate the new GUS WebDevKit (WDK).
  </description>

<!--
  <context-param>
    <param-name>wdkLogFile_param</param-name>
    <param-value>/www/wdktoysite-strutsdev/webapp/WEB-INF/
                              logs/webapp.log</param-value>
    <param-value>/usr/local/tomcat/5.0.24/logs/
                    wdktoysite-strutsdev.log</param-value>
  </context-param>
-->

  <context-param>
    <param-name>wdkModelConfigXml_param</param-name>
    <param-value>/WEB-INF/wdk-model/config/cryptoModel-config.xml
    </param-value>
  </context-param>
  <context-param>
    <param-name>wdkModelProps_param</param-name>
    <param-value>/WEB-INF/wdk-model/config/cryptoModel.prop</param-value>
  </context-param>
   <context-param>
   <param-name>wdkSpecXml_param</param-name>
   <param-value>/WEB-INF/wdk-model/config/specifications.xml</param-value>
  </context-param>
  <context-param>
    <param-name>wdkModelXml_param</param-name>
    <param-value>/WEB-INF/wdk-model/config/cryptoModel.xml</param-value>
  </context-param>
  <context-param>
    <param-name>wdkModelSchema_param</param-name>
    <param-value>/WEB-INF/wdk-model/lib/rng/wdkModel.rng</param-value>
    </context-param>
   <context-param>
    <param-name>wdkSpecSchema_param</param-name>
    <param-value>/WEB-INF/wdk-model/lib/rng/specifications.rng</param-value>
  </context-param>
  <context-param>
    <param-name>wdkModelParser_param</param-name>
    <param-value>org.gusdb.wdk.model.implementation.ModelXmlParser
```

```
    </param-value>
  </context-param>
 <context-param>
   <param-name>wdkSpecParser_param</param-name>
   <param-value>org.gusdb.wdk.model.implementation.SpecificationsParser
   </param-value>
  </context-param>
 <context-param>
    <param-name>wdkCustomViewDir_param</param-name>
    <param-value>/customPages/</param-value>
  </context-param>
  <listener>
    <listener-class>org.gusdb.wdk.controller.ApplicationInitListener
    </listener-class>
  </listener>

  <servlet>
    <servlet-name>wdkToySiteStruts</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>wdkToySiteStruts</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>

  <!-- The Welcome File List -->
  <welcome-file-list>
     <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
<!-- Error Page Mappings -->
  <error-page>
    <exception-type>java.lang.RuntimeException</exception-type>
    <location>/error.jsp</location>
  </error-page>
  <error-page>
    <exception-type>java.lang.IOException</exception-type>
    <location>/error.jsp</location>
  </error-page>
  <error-page>
    <exception-type>javax.servlet.ServletException</exception-type>
    <location>/error.jsp</location>
  </error-page>
```

```
  <taglib>
    <taglib-uri>/WEB-INF/tld-third-party/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/tld-third-party/struts-html.tld
    </taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>/WEB-INF/tld-third-party/pager-taglib.tld</taglib-uri>
    <taglib-location>/WEB-INF/tld-third-party/pager-taglib.tld
    </taglib-location>
  </taglib>

   <taglib>
    <taglib-uri>/WEB-INF/struts-layout.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-layout.tld</taglib-location>
  </taglib>

   <taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>

   <taglib>
    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
  </taglib>
</web-app>
```