13.4 DESIGN ASPECTS OF THE COMET MULTIMEDIA DATABASE

Stephen A. Drake *
UCAR/COMET, Boulder, CO

Eileen Kuo
UCAR/COMET, Boulder, CO

1. INTRODUCTION

The COMET Multimedia Database (CMDB) utilizes a Model/View/Controller (MVC) architecture. MVC design is widely recognized in the software development community as a preferred way to loosely couple basic application components. Depending on available resources and projects goals, employed software technologies will vary in how they implement an MVC design. For this project, we utilize a relational database to contain model data and Java Server Pages (JSPs) to define the view. For the controller, we utilize the Jakarta Struts framework, extending Struts action classes to further enhance application behavior. In this paper, we delineate the CMDB software design and examine utility the Struts controller in this context.

2. APPLICATION DESIGN

To facilitate both ease of use and the capability to specify a complex query, we created three different web-based interfaces that a user can manipulate. The "SimpleQuery" interface contains a text field in which a user can enter an alphanumeric string. With an "AdvancedQuery" the user can search for two different strings and apply logical operators that define a relationship between them. The user can also configure the resultant display to some degree in the AdvancedQuery. With the third query type, a "BrowseQuery" a user can apply operators relevant for the data type that will be searched in the database. Although we designed a different user interface for each of these query types, they all utilize synonymous round-trip processes to generate a response on the user's browser.

2.1 THE CONTROLLER

The controller is a set of Java classes housed on the web server that act like a switchboard to direct the flow of an incoming request. Figure 1 is a snippet from the CMDB Struts controller configuration file, "struts-config.xml", that defines an action element. It associates a destination class as given by the "type" attribute with a URL as given by the "path" attribute. So, when the user submits a form to the "simpleQuery" URL, the Struts controller forwards this request to the SimpleQueryAction class. This action element also defines the name of a FormBean, called "simpleQueryForm", which is a Struts-specific JavaBean (also called a bean). This FormBean contains the information typed in by the user. The "scope" attribute defines the longevity of the FormBean and the "validate" attribute indicates whether or not the input bean should be checked for valid values. The "forward" elements define which views to direct invalid or valid responses.

```
<!-- Simple Query -->
<action    path="/simpleQuery"
type="edu.ucar.comet.struts.SimpleQueryAction"
               name="simpleQueryForm"
             scope="request"
          validate="true"
             input="/simpleQuery.jsp">
      <forward name="failure"
path="/simpleQuery.jsp"/>
      <forward name="success"
path="/simpleQuery.jsp"/>
</action>
```

Figure 1. Code snippet from a "struts-config.xml" file.

The diagram in Figure 2 shows an overview of the round trip activity that occurs when a user activates an HTML form submission. When this occurs, the user's browser sends an HTTP request to the web server hosting the application. The web server directs this request to a servlet engine which, in turn, forwards it to the Struts controller. The controller parses the request and populates a JavaBean associated with the submitted form, in this case a SimpleQueryForm bean. Since the "scope" attribute in "struts-config.xml" is set to "request", the SimpleQueryForm bean is placed into "request" scope. If the "validate" attribute in "struts-config.xml" is set to "true" (as in Figure 1), the controller then invokes the validate() method in the SimpleQueryForm bean that checks the form values. If the submitted information is not valid, the Struts controller generates a response containing an error message and directs this response to the "failure" JSP defined as in "struts-config.xml". A message resource file called "ApplicationResources.properties" defines this error message, as well as other text contained in the HTML form. (You can write a different message resource file for each language that you want to support.)

*Corresponding author address:
Stephen A. Drake UCAR/COMET, P.O. Box 3000, Boulder, CO 80307, email: drake@comet.ucar.edu

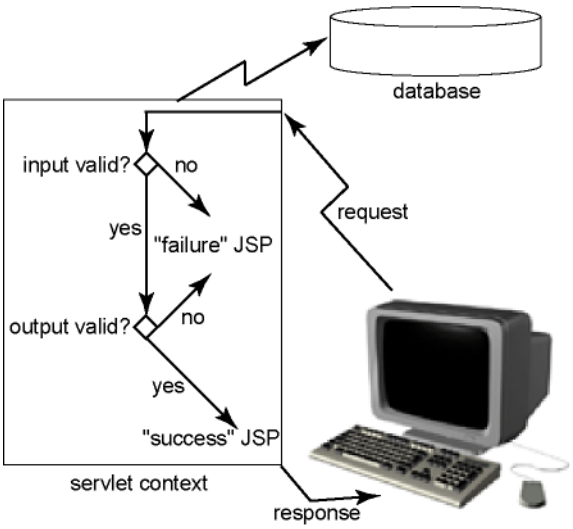

Figure 2. Request/Response overview.

If, on the other hand, the user entered valid information into the HTML form, processing continues with a class associated with the query type. As indicated in Figure 1, the Struts controller passes the request to the action "type" which in this case is the "edu.ucar.comet.struts.SimpleQueryAction" class. The SimpleQueryAction class merges information stored in the SimpleQueryForm with a QueryObject built from information in the file "database.xml" to create an SQL query. If the result of this SQL query is not valid, the application forms an error message and forwards a response to the "failure" JSP. Otherwise, the result set is repackaged into a result set bean and forwarded to the JSP defined by the "success" attribute as in Figure 1.

2.2 THE VIEW

JSPs maintain information relevant to displaying information that the user enters as well as the response from a given form submission. To do this, each JSP keeps a copy of the JavaBean containing the user submission, a locale object to keep track of the preferred language, and a result set bean that contains the database values from a successful query.

For example, the "simpleQuery.jsp" file contains commonly used HTML tags as well as JSP constructs to store and display information. It also contains Struts HTML tags that provide additional behavior and obviate the need to embed Java scriptlet code. The <bean:message> tag indicates where in the response to embed a piece of text defined in the application resource file mentioned earlier called "ApplicationResources.properties". By using this tag, you can avoid embedding any language specific text and thus create a JSP that supports more than one language.

We utilized many other tags in the JSPs composed for this application and a full description would be overly exhaustive. Suffice it to say that these tags add capability to standard HTML but in a tag format that is consistent with HTML.

2.3 THE MODEL

In MVC architecture, the term "model" is often used interchangeably with the word "data". In this application, we have several data sources - the backend database, the configuration information required to initialize the application, and the information the user entered.

The backend relational database is opaque to both the user and software developer. When the user submits a query, the database creates a result set which is then packaged into a result set bean and returned to the JSP. All that the JSP developer needs to know is the name of the database attributes that may be contained in the result set bean. This information is fully defined by the database schema. The information required to form an SQL query for each query type is defined in a file called "database.xml". This file defines a syntax for forming SQL queries that permits variable SQL constructs utilized in different database implementations. For example, one can see from the "database.xml" code snippet for a "BrowseQuery" in Figure 3 that an sqlClause element contains attributes that define operator names used in an SQL query. Since these attribute values are defined in an editable configuration file, this syntax allows the CMDB to be utilized with a variety of backend databases.

```
<select name="sqlSelect"
     queryType="browseQuery"
     command="select"
     defaultAttribute="*">
    <sqlClause searchAttribute="media_id"
               attributeType="integer"
               attributeValue="1"
               attributeOperator="="
               operatorType="contains"
               sqlOperator="where"/>
  </select>
```

Figure 3. Elements and attributes that define an SQL query and are contained in the "database.xml" file.

Application initialization information for the CMDB is stored in a file called "struts-config.xml" as previously mentioned. This file contains information about how to connect to the backend relational database and it also specifies various action mappings. The example in Figure 4 shows how to connect to an Informix database using a minimum of five and a maximum of ten keep-alive connections in a connection pool. Connection pooling in the CMDB enables faster database access by eliminating the time required to create a database connection for a given query. The Java classes that comprise the connection pool are available in an optional Struts jar file.

```
<data-source
   autoCommit="false"
   description="Informix Data Source"
   driverClass="com.informix.jdbc.IfxDriver"
   maxCount="10"
   minCount="5"
   password="myPassword"
   url="jdbc:informix-
sqli://myServer:myPort/mydb:informixserver=me"
   user="me"
/>
```

Figure 4. Database connection information.

Data that the user enters comprises the final component of the model. This data is kept in a FormBean relevant to the JSP that contains it. The Struts controller instantiates this bean and it only exists for the lifespan of a given request. Utilizing request scope (rather than session scope) for this bean enhances the scalability of the application since the servlet engine need not waste memory for objects with a longer lifespan - typically, session beans are maintained for 30 minutes.

Struts utilizes the concept of reflection to match components in an HTML form with fields in a JavaBean. For example, the text field in the SimpleQueryFormBean has a "property" called "queryString", as shown in Figure 5. By using reflection, Struts invokes a setter method in the SimpleQueryFormBean called "setQueryString()", inserting the value that the user entered into the method argument list.

```
<html:text property="queryString" size="16">
```

Figure 5. Struts HTML tag for a text field.

3. CONCLUSIONS

Despite a steep learning curve, the Struts controller has proven to be a very solid foundation upon which to build a web-based application. Whereas some web-based applications rely exclusively upon inheritance to implicitly control application behavior, Struts defines a framework in the real sense in that it is a Java-based controller to explicitly route predefined actions.

With the Struts JSP tags, we defined HTML widgets and FormBeans that the Struts controller automatically populated with values entered by the user. We utilized an inbuilt error catching mechanism to verify the user input and catch errors at any point in the thread of execution. We also took advantage of Struts tags that allowed us to locate language-specific text in a message resource file, enabling multi-language browsing capability.

With the "struts-config.xml" file we explicitly mapped URLs to actions. In this file we defined relationships between classes that defined behavior and FormBeans that contained the user input data. We also defined database connection information in this file.

Several peripheral aspects of Struts also added required functionality. For example the Digester that comes as part of Struts (and has been winnowed out into its own project) is very useful for dynamically generating objects. You define initial values for a target object in an XML file and the Digester instantiates and populates the object in memory. We utilized this capability was to create the QueryObjects for each query type.

As its name implies, Struts gave us a framework upon which to build the CMDB and concentrate our efforts on designing and coding core business logic. Especially in an HTTP environment that lacks events and callbacks, such as found in common desktop applications, we were thankful to find a system that could emulate these functional requirements. We anticipate that others requiring the capabilities provided by Struts will find it a sound framework on which to build.

4. ACKNOWLEDGEMENTS

We gratefully acknowledge Craig R. McClanahan, the initial author of the Struts framework. We also thank Heidi Godsil for drafting Figure 2.

This paper was funded by cooperative agreements from the National Science Foundation (NSF) and the National Oceanic and Atmospheric Administration (NOAA). The views expressed herein are those of the authors and do not necessarily reflect the views of NSF or of NOAA or any of its sub-agencies.

5. WORLD WIDE WEB LINKS

The COMET Multimedia Database:
http://archive.comet.ucar.edu/moria/index.jsp
COMET Multimedia Database Documentation:
http://www.comet.ucar.edu/appdoc/software/apps/moria/index.html
The Jakarta Struts Project:
http://jakarta.apache.org/struts/index.html

6. REFERENCES

Bergsten, H., 2001: *JavaServer Pages*, O'Reilly & Associates, Sebastopol, CA, 552pp.

Hunter, J., Crawford, W., H., 1998: *Java Servlet Programming*, O'Reilly & Associates, Sebastopol, CA, 510pp.