



# 5

## bonForum Chat Application: Use and Design

**T**HIS CHAPTER INTRODUCES YOU TO BONFORUM, the Web chat application that will be the major subject of the rest of the book. bonForum was designed as a tool to explore each of the subjects of this book, XML, XSLT, Java servlets, Java applets, and JavaServer Pages, while solving some real Web application problems.

### 5.1 Installing and Running bonForum

You can understand the remainder of this book much more easily if you have installed and tried out the Web chat application that it features. Therefore, we begin this chapter with instructions for installing and running bonForum.

#### 5.1.1 A Preview of the Rest of the Book

After helping you install and try bonForum, this chapter gives you some hints about how you can customize bonForum and develop it further yourself. After that, we discuss the design process. This chapter ends with some additional material about using XML data in Web applications.

The next chapter continues this overview of the entire bonForum project and begins by describing the implementation that our design led us to develop. That chapter ends with highlights of some of the major problems that we encountered, together with solutions found and solutions planned.

In Chapters 7–11, we cover in detail the code that we developed as we created this prototype for a multiuser, server-based Web application. Each of those chapters focuses on a different software technology that we applied to create the bonForum Web chat and uses excerpts from the source code to illustrate the discussion.

At the end of the book, you will find a listing of Web sources that might help you as you further explore the subjects of this book or as you try to fill in the gaps in its coverage of those subjects. Appendix A, “CD-ROM Contents,” shows you what is on the book’s accompanying CD. Finally, in the back of this book you’ll find the full source code for the bonForum project, with all its warts and wrinkles.

### 5.1.2 Checking Tomcat Server Availability

If you have read this far in the book, you likely have realized that to develop Web applications using Tomcat Server, you will need to have one available that you are free to use as a developer. Quite a few network and machine setups exist—some quite complex—that enable you to develop and test Tomcat Web applications. Some installations, for example, might feature several Tomcat Servers being used by developers, even while other Tomcat Servers are running deployed applications over the Internet or an intranet.

To avoid introducing such complexity into the discussion, we usually assume that you have full access rights to a Tomcat Server that is on the same domain as the browsers that you will be using to test the bonForum Web application. We usually will not be giving URL examples that include domain names. We always assume that you have the freedom to stop and restart the Tomcat Server and to edit and add files to its directory space.

Instructions for getting, installing, and running Tomcat Server are covered in Chapter 3, “Java Servlets and JavaServer Pages—Jakarta Tomcat.” You will need to have a suitable Java SDK installed on the Tomcat Server machine to use JSP and, therefore, to use the bonForum Web chat application.

#### Port Number for Tomcat

Throughout this book, we assume that your Tomcat Server is configured to use its default port number, port 8080. If that is not the case, you will need to change that in the examples given to the port number that you are using. If you are accessing Tomcat through another Web server, such as Apache Server, you might need no port number at all.

#### Trying the Tomcat Examples

First, be sure that you have a Tomcat Server installed and running on your system. You can verify this by using your browser to display the HTML document that gives access to the JSP and Java servlet examples packaged with Tomcat. If the browser is on the same host as Tomcat, first try to browse the following URL:

```
http://localhost:8080
```

If the browser that you are using is on a different host from Tomcat, you should instead use one similar to this one for a host named Freedom:

```
http://freedom:8080
```

### Using IP Address Instead of Hostnames

With Internet Explorer 5.X, you can use a URL with the hostname even for the local host. However, we tried without success in our Netscape browser to use both of the previous address examples to browse the bonForum Web application. After that failure, we ran the ntconfig.exe program from an NT command window. That gave us the IP address for the machine on which Netscape was running, and we put that in the URL instead. That gave us an address like the following:

```
http://192.168.165.99:8080
```

That URL worked for Netscape and brought up the Examples Web page for the Tomcat Server. At least we knew then that the problem was a result of using the domain name form for the address. However, the bonForum application then ran only until it needed to use frames on a page, when it displayed instead our (nonfunctional) noframes.html link, which is for browsers that are not frames-capable or that have frames capability turned off.

To avoid wasting effort, we decided to postpone handling cross-browser compatibility issues until we had settled on a final user interface. We stayed with our plan to make Internet Explorer 5.X the only browser until after extensive testing of a prototype.

### When Tomcat Server Is Not a Standalone Server

You might be running another Web server besides Tomcat—for example, Apache Server or Microsoft IIS. If so—and if you have configured it to use the Tomcat Server to handle JSP and Java servlet requests, and if you have also configured it to use Tomcat for requests whose paths begin with /examples—then you can test the Tomcat Server by requesting its Examples Web app through the “main” Web server. To do so, you browse a URL like one of the following:

```
http://localhost/examples
http://freedom/examples/
```

Many possible ways exist to set up Tomcat as the Java servlet container for another Web server. Therefore, the best advice that we can give you is that you should review the information about setting up Tomcat with Apache Server, found in the document “Tomcat—A Minimalistic User’s Guide.” That very helpful guide is supplied with the Tomcat Server download in the file TOMCAT\_HOME\doc\uguide\tomcat\_ug.html.

TOMCAT\_HOME is the path to the folder where you installed Tomcat Server. On our system, it is c:\jakarta-tomcat. Of course, you will need to substitute your own Tomcat Server home folder path in the previous URL, as in many others in this book.

### 5.1.3 Installation as a Tomcat Web App

The most convenient way to install a Web application for Tomcat is as a single compressed file containing all the required files. In reality, such a file is just a .zip file, but the convention is for it to have a filename extension of .war. We have provided our chat room example on the accompanying CD in a .war file called bonForum.war.

Installing bonForum could not be simpler. First, make sure that there is not a folder with the name TOMCAT\_HOME\webapps\bonForum on the server with Tomcat. If this folder exists, then it must be deleted before the new bonForum.war distribution file is used.

Copy the bonForum.war file into the Webapps folder under your TOMCAT\_HOME folder. For example, if your Tomcat was installed with a TOMCAT\_HOME of c:\jakarta-tomcat, then you should end up with c:\jakarta-tomcat\webapps\bonForum.war.

If the Tomcat Server is running, shut it down using the shutdown.bat command file in the TOMCAT\_HOME\bin folder. When you restart Tomcat, it will find the .war file and automatically unzip it under the Webapps folder into its own folder named bonForum. On the NT command window for Tomcat Server, you now should find a line something like the following:

```
2001-03-09 02:11:55 - ContextManager: Adding context Ctx( /bonForum )
```

Tomcat assumes that a .zip archive file that it finds in its Webapps folder is an archived Web application and automatically installs the context for it to run. There is also an alternate way to install a Web application, by entering some information about it into the server.xml Tomcat configuration file, which is found in the TOMCAT\_HOME\conf folder. You will need to use this nonautomatic installation method for either or both of two reasons: First, you want to install a nonarchived Web application. Second, you want to specify values for some Web application settings (such as docBase) that differ from the default values.

### 5.1.4 Running bonForum

You should now be able to begin your tour of the bonForum Web chat by browsing a URL something like one of the following (with or without the 8080 port number):

```
http://localhost:8080/bonForum
http://balderdash:8080/bonForum
http://192.168.165.99:8080/bonForum
```

That should display in your browser the default document for the bonForum Web application, the file TOMCAT\_HOME\webapps\bonForum\index.html. Click the bonForum logo to start the Web application on your browser. The bonForum user interface is simple enough, so little help is needed for you to try it out now. If you need help, you can look ahead to Chapter 7, “JavaServer Pages: The Browseable User Interface.”

Here are a few tips to get you started:

- Don't expect a polished application or user interface. This is an experimental prototype, and you will find problems to solve.
- First try becoming a host by starting a chat; otherwise, you might find no chats listed for you to join.
- Start up a second browser, on the same host or another that can access the Web app. Become a guest on the chat you are hosting.
- Nicknames in bonForum must be unique. If one is rejected, try again. There are no messages yet to explain the rejection.
- You may or may not need to download or configure the Java plug-in. Enabling Java Console for Plug-in might provide help.
- Using the browser's Back arrow is not always convenient here. Exit or reenter a new nickname instead.
- You may get frequent errors regarding .gif files on the console output. This is a harmless problem often reported with Tomcat 3.2.

### 5.1.5 First JSP Requests Require Patience

Be patient. Especially if you are running your computer at nearly its limit for memory and machine cycle resources, be patient. You will experience user-unfriendly delays. Your computer might need to find and start up a new Java virtual machine and then load all the classes it needs to compile JSP files into Java `HTTPServlet` class files. Each new JSP file that it encounters will need to be compiled, which takes time. After compilation, JSP can output HTML to your browser at a much more exciting tempo.

Here are two thoughts to keep in mind whenever you first try using any Web application that uses JSP technology:

- Be patient! Compiling takes time, and a JavaServer Page will not thrill you with its speed the first time a browser requests it.
- Users will not experience JSP compilation delays if you make sure to visit all the JSP documents in the application when you deploy it.

### 5.1.6 One Java Applet and the Sun Java Plug-In

Perhaps you already have the Sun Java Plug-in installed on the machine with the browser, for example, from this book's CD-ROM. In that case, there will be some delay while it starts up the BonForumRobot Java applet that is used by bonForum. This delay happens only once—each time you start the bonForum application on the server, which happens automatically whenever Tomcat Server is restarted. Depending on how your Sun Plug-in is configured, starting up the applet can take quite some time.

However, if you do not yet have the Sun Plug-in installed on the machine on which the browser is running, you will be asked to approve its download from the Sun Web site. Before you OK that, be aware that this might require quite some time if you

have a slow connection to the Internet. You will not be able to run this version of the bonForum Web chat without having the Sun Java plug-in available. For more on that, see Chapter 9, “Java Applet Plugged In: BonForumRobot.”

### Non-Applet Version of bonForum

Some people have objected to the use of an applet in bonForum and would rather see a purely server-side chat solution. That would require replacing our applet-based mechanism for refreshing user interface content (chat messages and so on) with different mechanisms that are not based upon an applet. In fact, our first version of bonForum did work without an applet (using the refresh Pragma), but the flashing of the refresh bothered us, so we went to the BonForumRobot applet solution.

#### 5.1.7 Frames and Tables Required

The browser that you use to enter bonForum must be capable of displaying HTML tables and frames. Again, we “certify” bonForum use only with the IE5.X browsers, in which that is not a problem. It would be possible to have a version of bonForum that does not require tables or frames. In fact, we also began the project without either tables or frames, but we found the results to be less than satisfactory.

#### 5.1.8 Problems Running bonForum

Perhaps the most common problem encountered while trying to install and run a new Java application is that it throws the `java.lang.NoClassDefFoundError` exception. If you did not tell Tomcat where to find the Apache Xerces XML package correctly, for example, you will not get far into bonForum before you encounter such an exception. Such exceptions should be politely shown to the user on an error page, and Tomcat has a facility for doing that. We did not add “polite” error handling to bonForum yet, so you will get the following rude message on your browser instead:

```
Error: 500
Location: /bonForum/servlet/BonForumEngine
Internal Servlet Error:
java.lang.NoClassDefFoundError: org/apache/xerces/framework/XMLParser
at java.lang.Class.newInstance0(Native Method)
at java.lang.Class.newInstance(Class.java:237)
at org.apache.tomcat.core.ServletWrapper.initServlet(
ServletWrapper.java:298)
```

The result of this error is that you cannot proceed; you must quit the application and fix the classpath problem.

## 5.2 Changing the bonForum Web Application

Although you might want to wait until you have read the rest of this book before editing and recompiling the source for the bonForum project, we feel certain that you will be sorely tempted to do so at some point. The software contains many loose ends and potential bugs that will no doubt aggravate you, and fixing these can be valuable learning experiences. (We would like very much to hear of these—you can email us at [email@bonforum.org](mailto:email@bonforum.org)).

### 5.2.1 Compilation of Java Source

See Chapter 2, “An Environment for Java Programming,” for help in setting up the necessary tools to compile this Web application. All the Java source code files for the `de.tarent.forum` package are found in the folder `TOMCAT_HOME\webapps\bonForum\web-inf\src\`.

You can configure your IDE to compile these and place the compiled class files into the folder where they will be used. An alternative is to run the `BonMakeIt.bat` command file provided in the source folder. The compiled `de.tarent.forum` package (but not the `bonForumRobot` applet class) goes in the folder `TOMCAT_HOME\webapps\bonForum\web-inf\classes\`.

The Java source code files can be compiled in the following order, among others:

```
BonForumUtils.java
BonLogger.java
BonForumTagExtraInfo.java
OutputPathNamesTag.java
OutputChatMessagesTag.java
OutputDebugInfoTag.java
NoCacheHeaderTag.java
Xalan1Transformer.java
Xalan2Transformer.java
TransformTag.java
NodeKey.java
BonNode.java
ForestHashtable.java
BonForumStore.java
BonForumEngine.java
```

These Java files are not all there are, however. The source for the `BonForumRobot` applet source file can also be found in the folder `TOMCAT_HOME\webapps\bonForum\web-inf\src\`. Compile it after the others, and arrange to have its two compiled class files stored in the folder `TOMCAT_HOME\webapps\bonForum\jsp\applet\`.

### 5.2.2 Editing of JSP Files

To be accessed by Tomcat Server as part of the bonForum Web application, the JSP files for bonForum must be located in the folder TOMCAT\_HOME \webapps\bonForum\jsp\forum.

We have found the Textpad editor from Helios Software Solutions to be very convenient for editing the JSP files. A trial version has been included on the CD for this book, under the \tools folder. You can find out more about this editor at the following URL:

<http://www.textpad.com>

If you have already requested any JSP files from Tomcat Server using a browser, you can look in its work folder, which is called work (unless this default name has been changed in its server.xml configuration file). You will find a folder there for each context. For example, for the examples that come with Tomcat, you will find the folder TOMCAT\_HOME\work\localhost\_8080\examples.

Inside these Work subfolders, you will see some Java class files with long, strange names, such as this one:

```
_0002fjsp_0002fsnp_0002fsnoop_0002ejspnoop_jsp_0.java
_0002fjsp_0002fsnp_0002fsnoop_0002ejspnoop.class
```

These are the .java files and compiled .class files created by Tomcat from the JSP files. The first time that each JSP file is requested, it gets compiled and placed here, where it can then serve requests for the JSP file. If you make any changes to the JSP file, Tomcat creates a new .java and .class file, changing the numbers that are embedded in the long filenames. It is very instructive to look at the Java files that are produced in the Work subfolder in your favorite editor because you can experiment with using JSP files. Doing so can also help you understand the error messages that you get from JSP compilation because they have line numbers that you can look up in the source here.

#### Some Problems Found with JSP

A few times we found that Tomcat could not compile a JSP file. Then, strangely enough, it sometimes used not the most recent successfully compiled class file, but the next older one! In these cases, stopping and restarting Tomcat fixed the problem.

Another useful trick required at times has been to stop Tomcat, delete the entire Work folder for the bonForum project, and then restart Tomcat. Sometimes it has also been necessary to restart the browser (note that you always must do that if you change and recompile an applet class). In one case, we even needed to reboot the NT Server before we could get the new JSP functioning.

You should definitely keep backups of any JSP files that you do not want to lose. For a while, our Internet Explorer was fond of changing the JSP file into an HTML file—in Unicode and full of browser-specific stuff. It somehow did so without even changing the file attributes. These JSP files became noneditable and had to be replaced by the backups that we had luckily made.

Lest you think that you are in for an unpleasant development experience, we hasten to add that the latest versions of Tomcat and the other software that we use have proven themselves very robust and stable. Hopefully, you will not need these tricks that we mention!

### 5.2.3 Modifying Style Sheets

The XML and XSL files for the bonForum Web application (plus a few batch files for testing things) are found in the folder `TOMCAT_HOME\webapps\bonForum\mldocs` (the “ml” stands for “markup language”).

You can experiment quite easily with the `chatItems.xml` style sheet document to change the appearance and even the functionality of the list of available chats that is displayed for a user who is looking for a chat to join. Alternatively, you can come up with a new version of `chatGuests.xml` to change the way the list of guests in a chat is presented to its host for rating changes. Read the last section of Chapter 10, “JSP Taglib and Custom Tag—Choice Tag,” for help with XSLT as it is applied in the bonForum Web application.

### 5.2.4 Using Logs to Develop and Debug

The best and most inexpensive way to debug what a servlet does is by having it create a log file for you. Our log files are built up by the accumulated results of calls to a method in our `BonLogger` class. They are created in the folder `TOMCAT_HOME\webapps\bonForum\WEB-INF\logs\`.

Our crude implementation of logging in the project could definitely be improved, but it helped enormously anyway. You can control its output destination by setting the logging init parameter in the `web.xml` configuration file to `none`, `all` or `file`.

This time-honored technique from lower-level programming practice should not be underestimated. We routinely log lots of output from our Java servlets.

#### Periodic Maintenance of Log Files Required

A few of the many calls to logging methods (in the source code) have been left in the source code because they give indications of errors that have occurred. Unless you have turned off logging in `web.xml`, the resulting log files (which, for now, are created in the `TOMCAT_HOME\logs` folder) will continue to grow, requiring eventual deletion by a human operator. Unlike all those Java class instances that you can leave lying around for the garbage collector, these log files will stick around until you—or someone else—delete them. In the future, the task of managing the growing log files could be assigned to a background task.

## 5.3 Using XML to Design Web Applications

Before we designed and developed the bonForum chat application, we spent some time using XML to model the structure, dynamics, and data that exist in a simple marketplace. The possibilities that this approach opened up were exciting. We wanted to simulate a marketplace by using that XML-based model for a Web application, but we knew that a simpler case would make a better first choice for a prototype and experimentation.

At about the same time, we stumbled upon the “Cluetrain Manifesto” on the Web, found at the following URL:

<http://www.cluetrain.com>

Although the entire manifesto is fascinating, it was the first statement of the manifesto that really struck us: “Markets are conversations.”

We had also just been checking out a chat Web site. This simple statement instantly made clear to us that our marketplace-modeling project should be preceded by a simpler chat-modeling project. A model of a conversation would intrinsically include the essence of a market. A model of a forum could be extended into a model of a marketplace.

### 5.3.1 What Is Meant by an XML Design

We followed one simple design rule: The model that we built was to be representable as an XML document. The root of this document, in the case of the marketplace model, was named bonMarketPlace, where *bon* is Latin word for “root,” meaning “good.” The root element of the new forum project could have been bonChat, but bonForum seemed to better encompass the greater possibilities inherent in conversations, such as commerce. Conversations—that is, chats—are only one commodity of those that could be exchanged using the bonForum Web application framework.

In the succeeding months, we found that by developing an application based upon an XML-representable design, we gained both simplicity and power. This simple development model kept us from creating an architecture and implementation that were overly complex, which is a common pitfall in many software projects. Just as important, the data that our application needed to handle became active—the data participated in the design of the application from the very beginning.

These are some of the real benefits of XML-based technologies. XML is not just a way to mark up and organize data. XML also can—and should—guide the definition and design of the Web application itself.

Too often, the architecture and logic of an application determine its input and output requirements. However, just as JSP has inverted the Java servlet, XML should invert the Web application. Both of these inversions can be used for the same purpose: to enable human (or robot) interaction, in one case with the servlet and in the other case with the Web application.

In this part of this chapter, we discuss the process of designing the bonForum Web application. Some of the ideas that we cover were used in the project; others were left out for one reason or another.

### 5.3.2 Actors, Actions, and Things

The children of the root element in the bonMarketPlace XML model are named Actors, Actions, and Things. The Actors element has children such as Buyer and Seller. The Actions element has children such as Sells and Buys. The Things element has many children, such as House, Car, Pizza, and Beer. With this simple model, we can model such market realities as Seller Sells Car and Buyer Buys Lemon.

Let's see how a similar framework can reflect the elements to be found in a highly simplified chat forum. There are two actors in our simple forum, one a chat host and the other a chat guest. They are both engaged in one action: They talk. The thing they talk about is the topic of the chat. We can diagram this forum and its mapping in our Actors-Actions-Things XML framework, as shown in Figure 5.1.

### 5.3.3 XSLT in XML Web Applications

XML technologies are still evolving, and many variations and extensions of the basic idea already exist. We can say that today one central and exciting area is the use of XSLT to map XML on a server to HTML on a browser. In very simple terms, we can diagram an XML Web application based on XSLT transformation in the manner shown in Figure 5.2.

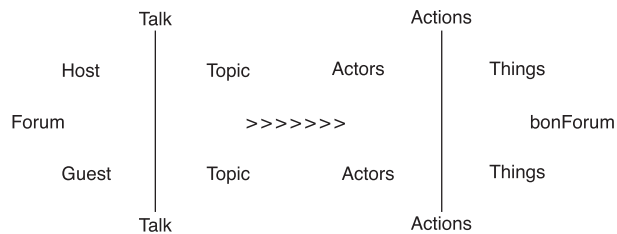


Figure 5.1 The forum and its users are reflected in the bonForum model.

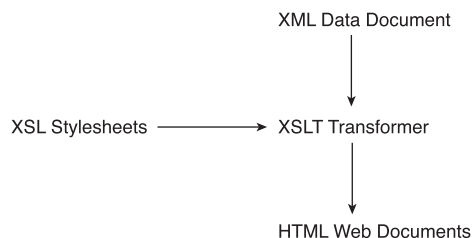


Figure 5.2 The bonForum model is transformed into the bonForum Web application.

This technology enables you to use XSL to design dynamic user interfaces. If our Actors-Actions-Things XML model has succeeded in capturing the static and dynamic elements of the Web application, it can be transformed into the HTML browser representation of the application with the help of XSLT and style sheets.

### 5.3.4 A Model of the Interaction Between Users

Our design must also take into consideration the interaction between the users of the application. The users of a multiuser Web application are represented in our XML-based model as children of the Actors element. Usually we think of these users as people sitting at a Web browser anywhere in the world, but they could just as easily be robots or client applications.

User interaction is obviously essential to any Web chat application. In Figure 5.3, we again include only two representative bonForum actors in this other context that our XML-to-reality mapping must encompass.

### 5.3.5 No UML and Data Management Tools Used

Usually at this point in the design process, we would have used a UML modeling tool to design our application. We also would have selected a database management system because handling chat data is an obvious job. We decided against that for several reasons. One is that we did not want to assume that all our readers are familiar with these professional tools, a thorough discussion of which is beyond the scope of the book. A more important reason is that a major goal of our project was exploration and experimentation. We wanted to find new approaches to designing and implementing Web applications. Furthermore, we wanted to build a learning platform for experimenting with servlets and JSP, applets, XML, and XSLT. If you are primarily interested in finding a real-world example that follows standard software engineering practice, you might think that we are being too academic. However, we feel strongly that the best way to learn about tools is to play around with them for a while.

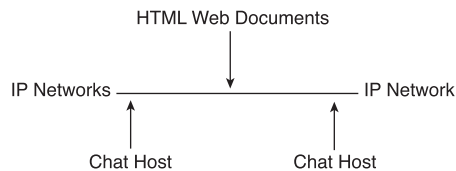


Figure 5.3 The bonForum Web application involves and connects the forum users.

### 5.3.6 No Interface and Class Design Used

Of course, we could not avoid at least considering the analysis of our application from an object-oriented perspective. We felt compelled to look for the Java interfaces and classes that we would build. Would not `Host` and `Guest` make good classes? Could they not share an `Actor` interface?

An `Action` interface could act as the verb in Actor-Action-Thing statements. Then `Start`, `Join`, `Execute`, and `Exit` would implement this `Action` interface, and their instances would handle specific types of actions. Perhaps instead it would be better to put `Start`, `Join`, `Execute`, and `Exit` methods in the `Action` interface. We spent some time analyzing chat forums along these lines, coming up with designs such as the one represented in Table 5.1.

Table 5.1 **Alternative Interface-and-Class Design for bonForum**

Interface	Class
Actor	Visitor
	Host
Action	Guest
	System
	Start
	Stop
	Join
	Execute
	Exit
Thing	Identity
	Subject
	Chat
	Message
	Forum

Again, however, we turned away from familiar design methodology. We decided to stay with our XML-based application design principle. The classes that we were busy identifying would instead become child elements of the `Actors`, `Actions`, and `Things` elements in our XML document.

We do not mean to say that we wanted to exclude objects from the application. Certainly, XML and Java representations are complimentary, not exclusive. As we will detail further, the capability of XML to model the world of a forum was to be complemented by the capability of JSP to generate servlet classes mapped to that model and to provide an extensible, Web-based view of that model.

### 5.3.7 A More Developed View of bonForum

We continued designing the tree that would become our XML document. Each node in the tree would become an XML element node. Actually, Figure 5.4 has been updated to resemble more closely the elements that we ended up using in the project.

As you can see, we are now including “key” nodes that related their parent elements to other nodes in the tree. To get the host that had started a chat, we used the `HostKey` child of the chat node. Not shown in this diagram is another aspect of the design: Each XML element has a “key” attribute that uniquely identified it. As we shall later discuss in detail, these keys enable us to store and retrieve the XML representation to and from a relational database.

### 5.3.8 An Early JSP-Based Experiment

We wanted to start experimenting with the design as something that could represent the bonForum as data that changes with the passage of time. In other words, we wanted to have a way to create events that would change the content of the XML data.



Figure 5.4 Designing bonForum XML using a tree diagram.

Using the elements of this early design, that meant being able to represent, for example, `host starts chat` as an event created by a user, which would add a `host` element, a `starts` element, and a `chat` element to our XML data and the various subelements that these required.

We decided to create a JSP document that would make this possible. The page, called `choose.jsp`, would display HTML on a browser to enable a user to select an Actor, an Action, and a Thing, using three select controls that were filled using the XML data. For example, the user could select `host starts chat`.

The user would have another select control available that would list all the subject items in the data. The selected subject item would become the subject of the chat. Experimenting with this simple JSP was very valuable and greatly influenced the implementation of the project.

### 5.3.9 Experimenting with Cookies

We needed some design-testing experiments that took into consideration the fact that `bonForum` was to be a multiuser application. On the Web, one way to keep track of users is with cookie objects. A cookie object packages information so that the server can send it to a browser. The browser keeps the cookie and then sends it back to the server along with its next request. Having a globally unique identifier associated with each browser allows the server to keep track of all the browsers—at least, as long as the browser user agrees to enable cookies.

Other information included in the cookie allows the server software to maintain a context that includes multiple visits by one or more browsers. The HTTP protocol by itself is “stateless.” Requests coming from browsers to a server are disconnected events. One way to connect the various requests is to use information stored in the cookies that a browser includes with each request. That can connect browsers in a Web application, and it can also connect requests coming from a given browser.

#### Displaying Cookies in JSP

Just in case you want to experiment with cookies from a JSP page, here is some code that we used to display their name/value pairs on the browser page:

```
<%
    Cookie[] cookies = request.getCookies();
    Cookie cookie = null;
    out.println("<H3>Cookies in request:</H3><BR>");
    for (int i = 0; i < cookies.length; i++) {
        cookie = cookies[i];
        out.println("\t<li>" + cookie.getName() + " = " + cookie.getValue() + "</li>");
    }
%>
```

Experiments with cookies led us to create a design that was later discarded but that nevertheless began to clarify the problems that would have to be solved to deal with

multiple-user contexts. This design was based on using cookies to control the XSLT process that was illustrated previously in Section 5.3.3, “XSLT in XML Web Applications.”

One example can illustrate the plan. When a user started a chat in the bonForum, the browser would include in its response to that action a cookie that had two key/value pairs, one with the new user status and the other with a key to the chat that was created. That would look like this:

```
Cookie: Status="host", Key="8734568345"
```

When the browser sent its next request—say, to send a message to the chat—the server-side software knew from the cookie that the user’s status was “host.” The key value identified one element in an XML file. With that key, the application did not need to use some complex XPATH expression to find that element; it got it directly using the key. Also, the XSL to apply to the XML data using the XSLT processor was determined according to the cookie value.

It turned out that we did not use that design, but it was through experimentation such as this that we found out the real problems that we had to solve in any design. Although these problems could have been solved by manipulating cookies directly, we instead availed ourselves of the more complete, robust, and user-friendly session management offered by the Tomcat Servlet engine (which itself uses cookies and URL parameters to maintain state). If you want to explore this fascinating subject in depth, we suggest studying what the Jakarta Servlet API 3.2 documentation has to say about the cookie, `HttpSession` class, and related interfaces and classes. Then study the source code that implements this API, which is in the package `org.apache.tomcat.session` in the Tomcat source.

### 5.3.10 And the Winner Is JSP Design

Initially, we were using JSP as a convenient way to write server-side Java code that understood the HTTP game of request and response, application contexts, and so on. Our focus was on using XSLT and cookies to design our application. Gradually, however, we started realizing that JSP could play a much more direct role in bringing our XML-based design to the Web.

The main reason for the increased role of JSP was the ease of establishing a relationship between a series of Web pages that a user traverses as they change states in the application, and between a series of JSP pages that create the HTML for those Web pages.

#### **Sending Three-Part Commands to the Server**

At first, our idea was to send our Actor-Action-Thing statements to a Java servlet, which would interpret them and control an XSLT engine. That XSLT engine would thus create HTML as an application-dependent response to each Actor-Action-Thing statement. We started calling these statements “three-part commands.” We created a

simple prototype Web page that could be used to send such commands to a server. It posted each of the three parts of a command in separate input fields of an HTML form element. Here are the contents of that file:

```
<html>
<body>
<h1>Test the bonForum:</h1>
<h2>Enter combinations of actor, action and thing, for testing!</h2>
<form method="POST" action="forum" >
  <input type="text" name="actor">
  <input type="text" name="action">
  <input type="text" name="thing">
  <input type="submit" name="forum">
</form>
</body>
</html>
```

### To *POST* or to *GET*, That Is the Question

We would rather use a *POST* operation because we do not want to have all the parameters and values appended to the URL, especially if we start sending encoded XML parameters. The URL displayed in the browser is part of the page, and it affects the appearance. However, there is a price to pay for this aesthetic decision. *POST* operations have two very large drawbacks: They require extra user input to refresh, and they expire in the browser, limiting the user's ability to navigate with browser controls. Also, the decision to use *POST* operation was made before our project began using frames. The ugly URLs of *GET* operations are of less importance now, so perhaps we will revise that decision.

### Forwarding Each Request to a JSP

At about the same time, we realized that it was easier to create a different JSP page to handle each three-part command than it was to continually revise a Java servlet so that it could parse each command and act accordingly. The servlet task could then be to simply forward the request to the correct JSP, which would be named after the three-part command. For example, if the command were `visitor creates chat`, then the servlet would forward the request that it received from the form *POST* to a JSP file named `visitor_creates_chat.jsp`.

We had now found the central control mechanism for the Web chat application. This seed became the `bonForumEngine` class, the “central station” of the Web application. It was the natural way to implement `bonForum` further, as discussed in more detail later in this chapter. However, before proceeding to create the many JSP documents that would be needed, we had a couple more problems to solve.

### 5.3.11 Choice of Bean or Custom Tag

Each JSP would have application-related work to do on the server side. We did not want to put that code on the JSP pages because that would obscure their function to represent the XML-inspired structure we had created. JSP documents should be kept easy to maintain and change—it is one of their strong points that they allow changes to be easily made to a Web application.

The code would go into methods in Java server-side objects. There were two possibilities to explore. To add functionality to our JSP pages in a manner that did not obfuscate the design, we could either create Java Beans and access their methods via JSP, or create a tag library and use JSP custom tags.

We liked the idea of including a JSP tag library in the project because it presents a very friendly way to add method calls to a JSP document. However there is an even more important point to consider, and that is a basic distinction between the use of beans and custom tags on JSP. A custom tag can affect the HTML that is output by the JSP, whereas a bean, by default, cannot do so. That means the decision of which to use should be based on the nature of the task at hand. Processing that does not affect the JSP page output should use a bean, whereas processing that does affect the output should use a custom tag.

To save some time, we needed a quick way to prototype the various functions that would be required by the interaction of the JSP pages with the other server-side components. We decided to temporarily house all the methods required in the one Java Servlet class that we already had, `bonForumEngine`. Then we decided to put all our JSP-side functionality, regardless of its effect upon the JSP output, into one bloated, temporary chameleon tag.

The plan was to use this setup only as a testbed. After deciding which tags and beans were to be used and what their requirements were, we would break the code out into the many files that would be required. Unfortunately, when the first edition of this book was published in Germany, we had not yet broken down those two huge classes. That is a regretful situation but nevertheless better than the alternative, which was to ship the book's project in a nonfunctioning but better designed state. Thus, our ugly testbed classes have become immortalized in print, further strengthening our opinion that most technical books are obsolete because of their slow refresh rate.

### 5.3.12 Original Plan: Follow the XML Design

At first, we were quite religious about having the XML data faithfully reflect both the data and the dynamics of the forum. After all, a major purpose of the project was to explore the consequences of designing software in this way. As we proceeded, we discovered that some parts of the original XML design needed to be simplified for the sake of both understandability and performance.

However, our initial strict adherence to the original plan turned out to be a valuable exercise: It helped identify the complex, unnecessary, and redundant elements in

the design. We will give here one example to illustrate how our first trial implementations manipulated the XML data. This example will explain how the statement `visitor starts chat` was implemented by the software.

A user of the application would enter and become a visitor. To represent the user in that state, the software would create a `visitor` element in the XML data as a child of the `Actors` element. Then an `identity` element would be added as a child of the `Things` element. This `identity` element would contain the nickname and age given by the visitor. An `actorKey` attribute in this `identity` node would link this information with the user's `visitor` element.

If that visitor then started a chat, the software would first add a `start` element to the `Actions` element. Then it would add a `chat` element to the `Things` element. After getting the topic for the chat from the user via an HTML form, it would add a `host` element to the `Actors` element and change the value of the `actorKey` attribute of the user's `identity` node so that it was linked to the new `host` node. The `visitor` node and the `start` elements would then be removed from the XML data.

### 5.3.13 Simplifying the Grand Plan

In our initial designs, we were trying to find a kind of Lego™ set that would serve to express all possible states of the chat forum. Each Actor, Action, and Thing entity in the real chat forum would be represented by an element in the XML database. Each combination of these three types of entities would generate a “three-part statement” that would have meaning to the application. Each of these statements would be mapped to a JSP document that would create the appropriate response to the user.

However, even the very simple models that we started with quickly generated a complex matrix of “Actor-Action-Thing” statements. The steps required to dynamically modify the XML database to reflect the changing state of the forum also were numerous. It was time to simplify the design.

We started by throwing out many of the “Actor-Action-Thing” statements that were initially in the plan. One example was the statement `host joins chat`. In early plans, we included this sequence of statements:

```
visitor starts chat
visitor becomes host
host joins chat
```

This sequence of statements was replaced by just the first one, `visitor starts chat`. The visitor still becomes a host and joins the chat, but all this occurs without any formal representation in the three related architectural spheres: XML data, JSP documents, and three-part commands.

Our next major simplification involved the XML database. Although the various Action elements (`starts`, `joins`, `executes`, and `exits`) still play a part in the three-part commands and JSP filenames, they are no longer represented in the XML database. We did keep the `Actions` element, because in future versions of `bonForum`, we might add

some Action elements to handle added functionality, such as a `send` element that will control a background email-sending process.

We took yet another simplifying step. We had planned to include all the features from the best of chat Web sites on the Web today. However, it was necessary to avoid doing that. The freedom to experiment and explore new models and technologies would be overwhelmed by such a long list of product requirements at such an early stage in the game.

Many of the chat programs on the Web today rely on client-side programming to achieve their complex user interfaces and feature lists. If we included similar interfaces and features as requirements in the first version of bonForum, we would probably have to rely on the same client-side, operating system code libraries that have allowed JavaScript or VBScript to create such rich Web applications. We would perhaps please the bonForum user, but we would miss our goal of prototyping a server-side Web application framework.

So, the red pencil came out, and we went to work shortening the feature list for the bonForum project. There would be no private chat rooms. The software would not remember which banner ads you clicked on the last time you were there and present you with new ones custom-picked according to your interests. The software would now have no answers to many situations that arise in real chat rooms—for example, what happens if the host of a moderated chat exits for good?

Much more needed to be done before bonForum could become a competitive Web chat solution. However, that has not been our goal. Instead, we are exploring techniques to lay down a novel framework, one that can possibly engender never-before-seen features. Best would be if the framework that we eventually develop becomes “boiled down” enough to become reusable in the context of various different Web applications, such as e-commerce, collaborative engineering, knowledge management, or online billing and product delivery. Unfortunately, the design as presented is not scalable to any of these applications, all of which would certainly need real data persistence (for example, a real database), fault-tolerant design, and so on. At this point, it may be more believable to present this as an exercise than to propose that it form the base of a future array of products.

#### **5.3.14 Some Important Functionality Left Out**

Before deploying the bonForum application, many things must be considered that we are not taking care of here in the book project. Some very important Web application features will be left out of our example project. These include some that make Web applications fast enough and scalable enough for real use.

Consider, for example, the scalability of our application. We could try to design distributed pools of bonForum components, communicating data with each other. A good reason to leave out such things in this prototype is that there are better ways of providing features like this. The new enterprise Java packages from Sun will give you much better ways to connect a large number of users to the Web application.

We will also pretend that load balancing, security, and encryption issues are all handled in some manner not discussed. The Web application that we develop will not take into account such real-world issues as speed and bandwidth. If the result is too slow for testing our hypotheses, we will try using faster hardware and fatter network connections.

### **The Need to Scavenge Old Chat Data**

As we will discuss in detail later, one way that we simplified our programming requirements was by establishing the rule that a user client can remove an element from the XML data only if that element was added by the same client during the same HTTP session. We ensure this in a simple manner: The key of the element is made up of the session ID value followed by the element name. But what happens to all those entries in bonForum XML that are connected to sessions that are finished? According to our rule, no client can remove these orphan elements.

The bonForum Web application will thus require the addition of a background daemon thread whose sole job is to remove old elements from bonForumXML. To know which are old, the application will need to track all the session IDs that take part in bonForum. Perhaps a certain period of inactivity by a given session will be defined as sufficient reason to remove all elements connected to that session.

### **Persistent Data Storage**

In addition, we will keep some of the data that the Web application requires in XML files. This is not the best way to do it, particularly if we want speed and scalability to be optimized. However, we will just imagine that our XML file can instead be XML-streamed from some new database, or that we have mapped the XML representation of our data into another form (say, within one or more relational database tables) that can be much more efficiently processed than XML files.

Again, this choice is related to the question of where best to learn the skills required for handling data for a server-based application. As we see it, you can find good books and sources of information that will show you how to use JDBC, for example, to persist data for a Web application in your favorite relational database tables. These resources will show you how to convert relational data into XML form. But if your favorite database does not already do that for you, it soon will. So, we will leave that work to someone else and instead focus on experimentation for the sake of learning.

### **Security Issues**

Java has well-developed tools for ensuring the security of personal information in Web applications. These are all beyond the scope of this book. They are not necessarily beyond the scope of the Web application when it is deployed, though, depending on how much personal information it includes. Furthermore, plans to prevent the theft or destruction of data in bonForum have not received consideration and definitely should.

### 5.3.15 Other Uses for the bonForum Design

The JSP pages and other parts of the bonForum can be used to generate all the frames of a multipanel GUI. For example, this could be one of those extensive control panels that are found on large industrial installations. The same isolation that has been built into the Web application so that multiple users can execute commands will help establish a robust interface in other multiuser and multifunction environments.

## 5.4 XML Data Flows in Web Applications

We believe that passing XML data within a Web application will turn out to be as important as passing XML data between systems and applications. The latter use of XML is much discussed and heralded in particular as a great benefit in connecting legacy applications to modern Web-based applications.

Within a Web application, passing even one parameter that contains XML data can be a simple yet powerful way to pass a lot of structured information. Passing a long list of `name=value` attributes is cumbersome, by comparison. Let's look at various possibilities for creating XML data flows between the typical components of a Java-based Web application.

Please note that, unlike most of this book, this section is not based upon examples taken from our bonForum Web chat project. Although we certainly pass request parameters around in bonForum, they do not contain XML data. We are excited by our preliminary research into this use of XML. These techniques are included in our future development plans for bonForum. We think that this information may be useful to present here, even before we back it up with "real Web application experience," as we prefer.

### 5.4.1 Sending XML from an HTML Form

Many of the examples that follow involve putting XML data as a string into an `HttpRequest` parameter from a browser. If you need to send XML in a request parameter from HTML documents, then you can put it in a string attribute value of an input element within a form element, as in this example:

```
<input type="hidden" name="fragment"
value="&lt;tree&gt;&lt;topic&gt;Chess Players
Chat&lt;/topic&gt;&lt;moderator&gt;Harvey
Wilkinson&lt;/moderator&gt;&lt;/tree&gt;">
```

Notice that the ampersand character (&) must be escaped twice. You have to escape the escape! The first replacement will produce the characters that are to be replaced with the "less than" character (<).

### 5.4.2 XML from Browser to Servlet

You can send XML from a browser to a Java servlet by putting the XML as a string into a request parameter. You can test this by putting it into an HTML form input element. Try pasting “doubly escaped” XML strings like the one used in the previous example into a form input element and posting that to your servlet.

Your Java servlet must then do something like the following to get the XML back into a string:

```
String sXML = (String)request.getParameter("paramXML");
```

In the next sections, we discuss ways to use the XML passed in from a browser, including servlet control and XSLT processing. Notice that those same ideas can be applied either to XML passed from a browser to a Java servlet, or from a browser to JSP.

### 5.4.3 XML from Browser to JSP

Remember that a JSP is essentially a way of turning a servlet inside-out so that its contents can be written using Java as a scripting language. A new JSP causes a servlet container, such as Tomcat, to create a newly compiled instance of an `HTTPServlet` class. This servlet will have available a `_jspService` method containing Java code that depends on the scripting elements that you put into the JSP.

The service method in a JSP servlet has access to the `HttpServletRequest` object, which can have parameters. You can pass XML to the servlet via one or more of these parameters. You can process that XML using Java code that you add to the JSP script.

#### JSP Applies XSLT to XML from Browser

We are indebted to Steve Muench for information about passing XML from a browser to a JSP, which he posted on the `xsl-list` hosted by `mulberrytech.com`. From his mail we learned the following code fragment, needed to get the XML string transformed by an XSLT processor:

```
<%
// more code goes here...
java.io.ByteArrayInputStream bytesXML = new java.io.ByteArrayInputStream(
    sXML.getBytes());
InputSource xmlInputSource = new InputSource(bytesXML);
// more code goes here...
%>
```

To see how to use JSP “page import” elements to access the needed Java classes, as well as how to create the XSLT processor to process this `InputSource` and an XSL style sheet, you can refer to the code we used to do that in the `bonForum` project. (Note that to use Xalan 2.0 with that code, you will need to make use of its “compatibility jar,” as described in the Xalan 2.0 documentation.) That code is discussed in Chapter 10, “JSP Taglib and Custom Tag: Choice Tag.”

#### 5.4.4 Controlling Java from a Browser with XML

Web-based server-object control could be accomplished by passing the XML from the browser request to an XSLT processor along with an XSL document. The XML `InputStream` can be used to fire custom tag extensions in the XSL document. In this way, you can put the flow of processing inside Java servlet methods under the indirect control of browser-originated XML content.

Clearly, XSLT is useful to control the display of XML data streams. For this, XSL data streams are the controlling, declarative script. What is less obvious at first is that XSLT also allows XML data streams to control programs. If you can pass a data stream “into” a program at runtime (request time), then you can control that program with it. This is fertile ground for Web application designers, and not just for the ones working with embedded systems. (Think of the Internet—all that software is embedded now!)

Another similar experiment that we would like to try is feeding an XML `InputSource` from a browser request parameter into a SAX parser. We could then use the contents of the XML to fire Java classes via the SAX event handlers. Could these classes access the whole JSP context? What could be done within Java objects that are controllable via XML from a browser?

#### 5.4.5 XML from Servlet to JSP

To send XML from an `HttpServletRequest` to a JSP page, you can override any one of the several servlet methods that have access to the `HttpServletRequest` object (`doGet`, `doPost`, or `service`). Inside the method, you get a `RequestDispatcher` to forward the request and response to the JSP page. All you need is to know the URL for the JSP. Be sure to take into consideration the Web application configuration file, (`web.xml`) and the Tomcat servlet container configuration file (`server.xml`).

To see how to do this, read the file `TOMCAT_HOME\examples\jsp\jspotoserv\stj.txt`. There you will find the source code of the `servletToJsp` servlet. As you can see, the servlet overrides the `doGet` method and adds these two relevant lines of code:

```
request.setAttribute ("servletName", "servletToJsp");
getServletConfig().getServletContext().getRequestDispatcher("/jsp/jspotoserv/hello.
=jsp").forward(request, response);
```

So, what about passing XML? You can add that to the request object as one or more parameters, just as we did in the browser-to-servlet and browser-to-JSP examples discussed earlier.

#### 5.4.6 XML from JSP to Servlet, or JSP to JSP

It is also possible to send XML from a JSP page either to a Java servlet or to another JSP. Simply use a form element (as shown earlier) or some other means to get the XML into a request parameter, and then use a `jsp:forward` element to send the request to the desired destination servlet or JSP.

Here is a simple example that you can try. Create a JSP page, called TOMCAT\_HOME\webapps\bonForum\jsp\forwardToSnoop.jsp. Put in this file only the following lines:

```
<html>
<%
request.setAttribute("hello",
"&lt;?xml version=&quot;1.0&quot; encoding=&quot;ISO-8859-
1&quot;?&gt;&lt;doc>Hello&lt;/doc>");
%>
<jsp:forward page="/snoop"/>
</html>
```

Find the web.xml file for the bonForum Web app, in the folder TOMCAT\_HOME\webapps\bonForum\WEB-INF. Make sure that the file has a servlet element for snoop, like the following (if not, you can copy and edit the one in the Tomcat Examples Web app):

```
<servlet>
  <servlet-name>
    snoop
  </servlet-name>
  <servlet-class>
    SnoopServlet
  </servlet-class>
  <init-param>
    <param-name>
      fooSnoop
    </param-name>
    <param-value>
      barSnoop
    </param-value>
  </init-param>
</servlet>
<servlet>
```

Copy the SnoopServlet.class file from the Tomcat Examples Web app into the bonForum Web app. You should find the class file in the folder TOMCAT\_HOME\webapps\examples\WEB-INF\classes. Copy it to the folder TOMCAT\_HOME\webapps\bonForum\WEB-INF\classes.

Now try browsing (with Tomcat running) your forwardToSnoop.jsp page using this (or your similar) address:

```
http://localhost:8080/bonForum/jsp/forwardToSnoop.jsp
```

When you try this example, you should get a page full of detailed information about the HTTP request on your browser. (By the way, this works with only the SnoopServlet, not the snoop.jsp example.) The browser display should include the following lines:

```
Request attributes:
  hello = &lt;?xml version=&quot;1.0&quot; encoding=&quot;ISO-8859-
1&quot;?&gt;&lt;doc>Hello&lt;/doc>;
```

Of course, this is the XML sent from JSP to the servlet:

```
hello = <?xml version="1.0" encoding="ISO-8859-1"?><doc>Hello</doc>
```

Yet, nowhere in all the snoop information can you see anything that would reveal the original receiver of the browser request—namely, the JSP `forwardToSnoop.jsp`.

### 5.4.7 Displaying HTML or XML Using JSP

As a final tidbit, here is a way to display an XML document using JSP. Putting this line on a JSP

```
<%= "<B><I>hello</I></B>" %>
```

displays the text in bold and italics:

```
hello
```

This second excerpt can display an XML or an HTML element. Putting this one line on a JSP

```
<%= "&lt;B&gt;&lt;I&gt;hello&lt;/I&gt;&lt;/B&gt;" %>
```

displays these tags instead:

```
<B><I>hello</I></B>
```