Better Architecture with MVC and Struts

By Phil Coulthard and George Farr,

Here we introduce the next step in the RPG to J2EE roadmap. So far, we've covered the RPG and Cobol tools that comprise the Better Tools step, as well as the IBM WebFacing Tool and the IBM Host Access Transformation tool (HATS) for the Better User Interface step. Now we turn our attention to the Better Architecture step. At this step, we consider the model-view-controller (MVC) design pattern, the Struts framework for Web application design, and the IBM iSeries Web tools.

Model-View-Controller Architecture

In Better Architecture, we move into creating a new application, properly architected, that uses modular RPG or Cobol to drive a new Web user interface. One of your first questions will be, "What do you mean by properly architected?" We mean that the application follows a model-view-controller (MVC) separation, whereby the RPG or Cobol business logic is the model, the WebSphere servlets are the controller, and the Java ServerPages (JSPs) are the view. The goal is to reuse the most important piece, the model. To accomplish this, we need it to be as de-coupled from the view and controller as possible, so we can use the model in different scenarios in the future, such as to drive a personal device or do business-to-business by way of Web services.

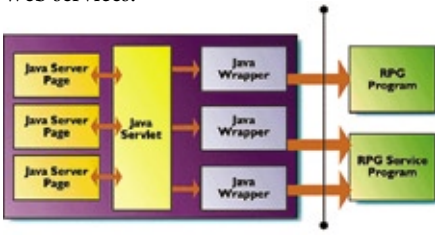

Figure 1: Web application architecture

Figure 1 shows an example of the application architecture we are looking for. Basically, we have the model code on the right, in the third tier, and the controller code on the left, in the second tier, running inside WebSphere Application Server. These are logical tiers, in that they could be on the same physical system or on different systems. The key is a choice that can be changed immediately because the two parts of the application are not tied together.

As we mentioned, you want to design the model logic to be independent of the controller logic. That means it will not explicitly decide screen flow, for example. Rather, that is done by the controller code in the servlet, which also decides what piece of business logic to invoke for a given request from the user (that is, when the user presses a button).

A servlet is merely a Java class, but one that extends a particular base class such that it inherits servlet behavior; this means that a certain method is called each time the user presses a Submit button on a Web form that is coded to call the servlet. We recommend a single servlet for the whole application. Its job is to read the user-entered data from the form, and call the appropriate business logic, passing it that data.

For example, if we are calling RPG or Cobol, that data is usually passed as input parameters, and data is returned in output parameters. The business logic has to be accessible through Java, so if it is RPG or Cobol, we need a Java wrapper for it. There are many ways to call RPG or Cobol from Java, including through Java Native Interface, through Program Call Markup Language (PCML) in the iSeries Toolbox for Java, as a stored procedure through JDBC, and through MQSeries. All of these work well, but PCML is the most popular, followed closely by stored procedures. Upon return from the business logic, the servlet next decides which JSP page to show next, perhaps by analyzing the data returned from the business logic, and then transfers control to that JSP, passing to it the data returned from the business logic. A JSP is merely HTML with embedded special tags for extracting data from a simple Java bean and inserting that data into the Web page, at run time, much as Workstation Data Manager does with display files and application data. The resulting Web page is passed back to the waiting Web browser (which we call tier 1).

There is much to be said about what else is in a JSP. For example, today we don't often code HTML tags directly; rather, we use tag libraries, which are user-defined tags. The JSP specification lets people create their own tags by supplying Java code that will produce HTML and JavaScript at runtime, when the JSP engine encounters the tag. The tags can take attributes that affect what is produced.

Several common libraries of tags are available today, and the Java community is settling on a standard set. Within the Web tools of WebSphere Development Studio Client, extensions to the tag library standard can make these tags visible in the What You See Is What You Get (WYSIWYG) Page Designer editor. A number of tags are supplied within palettes, for your convenience, including a number that are uniquely for you as iSeries developers, as we will see. You can use these by simply dragging and dropping them, and you can use the attributes view to configure them.

JavaScript is code that runs in a Web browser. It's somewhat similar looking to Java, but it's significantly different. JavaScript is reasonably easy to learn and use, but the tag libraries are increasingly able to generate what we need. You might

also want to learn a bit about Cascading Style Sheets (CSS); they make it possible to assign global fonts and styles to particular constructs in your Web pages. Changing these once immediately affects all pages in your application. They make maintenance significantly easier.

While it's fun and easy to learn HTML, JavaScript and CSS (collectively known as Dynamic HTML or DHTML), this is something you can defer because the tools are mature enough now that you can go a reasonable distance with little or no such knowledge. Further, the final finessing of the Web pages is often something that is left to individuals with a particular interest and flair for this type of art, especially if you are building Web pages for public consumption versus just in-house use.

The Struts Framework

Going back to our architecture in **Figure 1**, we have some good news. Enough people have written enough well-architected Web applications that some common ground has been established in the form of a framework that you can use as the starting point of your new application. This framework is known as Struts, and it is an open source initiative that you can learn about at http://jakarta.apache.org/struts/index.html. Struts is a base you can build on, which encourages and rewards a proper architecture in the second tier. It has nothing to say or contribute to the third tier, but that's fine because you know how to code RPG and Cobol, right? **Figure 2** shows how Struts affects the architecture that we are building.

Struts gives us the single servlet that we need, as well as the base classes to simplify creating the Java wrappers to the business logic, which are called Actions in Struts. We also get a base class for creating form beans, which are simple Java beans that contain the user-entered data from a Web page on input, and contain application data to initialize a Web page on output.

At runtime, when the user clicks Submit, the Struts servlet gets control first, finds the appropriate form bean for the current input page, and populates it with the user-entered data. It optionally validates the data if you added validation into your form bean. If validation is skipped or passes, the servlet then finds the Action associated with the input page and invokes it, passing the form bean to it. The Action does whatever it needs to (it's your code, so presumably it invokes RPG or Cobol on the third tier), and then it returns to the servlet the name of the output JSP page to show next. If data is to be passed to the output page, the Action will find the associated form bean for the JSP page and populate it, placing it where the JSP page can find it (technically, in the request or session object), before returning control to the servlet. The servlet then simply passes control to the identified output JSP page, which substitutes any data from its pre-populated form, and the result is the output page to the user. Then the cycle starts again.

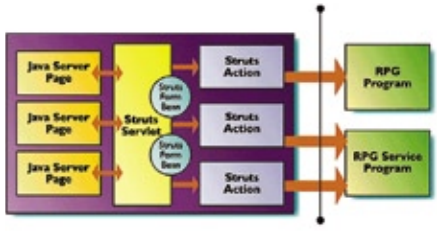

Figure 2: Struts Web app architecture

How do the servlet and the Actions know which form bean is associated with each input and output page? They know this from the Struts configuration file (struts-config.xml), in which you identify the form bean class for each JSP page. Further, you also use this file to map each input JSP page to the Action that will process it, and to identify for each Action which output JSP pages it might possibly forward to. (It is possible to map a JSP page to multiple Actions depending on which button was pressed.) Each Action is given an alias name, which is what your Action code will return. This way, you can decide to show a different output page without changing your code – you need only change which Action is associated with a particular alias name, in configuration file.

Struts helps with some other things too, such as internationalization and error handling. For internationalization, you typically have a Java resource bundle (merely a translated string lookup table) that contains the translated strings, and your JSP pages refer to entries in the bundle using keys instead of hard-coding the text and labels shown on the page. For other internationalization requirements, such as date and time formatting, the locale of the user's Web browser is captured and made easily available to your Java action code and form beans so they can use the Java locale support for localization.

For error handling, if an error is thrown by your code or by Java (it's like running the command SNDPGMMSG (Send Program Message) on the iSeries) during any of the processing, control is passed to a globally registered error JSP page, which can be unique for each type of error. Of course, these are registered in the Struts configuration file. Further, your Action code and form validation code can return instances of an Error class to indicate errors are to be shown to the user. To show them, a special error tag is available for your JSP pages.

In addition to all of this support, to encourage an MVC pattern where the business logic response for each page or button's request is encapsulated in an Action, Struts also comes with a couple of tag libraries, which you can use in your JSPs. Although these are not always needed or used, they are available for your convenience. You will see that for iSeries developers, we supply an alternative set of tags that are designed to leverage your SDA skills.

What About Tools?

In WebSphere Development Studio Client, you get a lot of tools for building Web applications. These tools include the Page Designer for WYSIWYG editing of JSPs, a considerable number of Struts tools, individual tools

Upcoming Articles in this RPG to J2EE Series:

November 2005 – "Web Tools For iSeries Programmers – The Details"

January 2006 – "More Web Tools for iSeries Programmers"

March 2006 – "Web Services Tools for iSeries Programmers"

for logos, animation, CSS, and more. These are all available in a dedicated perspective, for working with your Web projects.

Of course, there are also rich tools for Java development. However, in our case we are interested in allowing you to easily create Struts-based Web applications that ultimately invoke RPG and Cobol code. As much as possible, we try to defer the need for you to learn Java and Web technologies, by generating as much of the second-tier code as we can, and even the first pass at the Web pages (JSPs), if you like. This is done by augmenting the many Web tools that are inherited from WebSphere Studio Site Designer with additional tools specifically for iSeries developers. These tools will generate a Struts Action and configuration file information, given details about how to call your program or procedure, and optionally the input page to prompt for input parameters to your business logic, and an output page to show the output parameters. These pages are internationalized.

There are also iSeries tags for creating your own Web pages, with familiar attributes such as data type and length, edit code or edit word, and even validity checking information. You can even create a Web subfile with RPG or Cobol logic. Of course, you do have to write that RPG or Cobol code, and it really should be good code. For that task, there is the Remote System Explorer and iSeries Projects tools we described in the Better Tools step. You see how these steps build on each other?

The Bottom Line

The bottom line is, the iSeries Web tools make it possible to build functional Web applications with existing skills, and by writing only RPG or Cobol business logic. Sound good? Indeed it does. Tune in for our next installment, and we'll show how these tools leverage the elegance of Struts, the power of J2EE, and the knowledge you have to make building Web applications today practical and even fun. TUG

This article was first published in iSeries News magazine.

Phil Coulthard *works at the IBM Toronto lab, where he is the lead architect for application development tools and languages on iSeries.*

George Farr *works at the IBM Toronto lab, where he is the technical development manager for the RPG and VisualAge for RPG languages, as well as the new RPG and Cobol tools in WDSc. Phil and George are frequent speakers at many conferences and user groups worldwide, and their books Java for RPG Programmers, 2nd Edition and Java for S/390 and AS/400 COBOL Programmers, are available from Penton and MC Press.*