JSF and MVC

JSF and MyFaces Core Concepts How JSF State Management works

Over the last decade or so, the development of applications with graphical user interfaces has pretty much standardised. Whether you are developing with Gnome, KDE, MFC, VisualBasic, Delphi, or any similar library the process is the same:

- a tree of widgets is built, either by code or by loading a configuration file
- callbacks are attached to the widgets
- the GUI framework is left to render and manage the user interface.

When the user activates a widget (clicks a button, chooses a menu option, etc) the appropriate callback is invoked. This callback inspects the requested action, and may choose to report an error, or to update some kind of data-structure according to the user input. Then control is either returned to the UI framework with the same set of widgets, or a new set of widgets (a new "screen" is built).

The data-structure here is the "model"; it may be a model of a building, of an economy, of a text-document, etc. The logic which decides whether to update the model or report an error, and whether to redisplay the same set of widgets (same screen) or a different set (new screen) is the "controller". And the widgets themselves (which may have custom drawing logic) are the "view".

This approach has been vastly successful in systems where all the processing is on one physical computer system. However in our internet-connected world, accessing remote systems in interactive ways is becoming more and more common. Having to deploy applications onto client systems is becoming more and more clumsy.

One solution is the Java WebStart approach, where client logic is dynamically downloaded. The client can then be implemented much like a traditional MVC application, with the addition of communication back to the original host. However downloading logic to the client computer is not generally popular; it requires significant computer power and memory on the client side, and a large startup delay as the logic is transferred.

JSF instead is a different way to bring the successful MVC programming approach to the internet. It is designed to assume that the client has some software capable of rendering a reasonable range of graphical widgets, and of transmitting information back to the server about changes in the state of those widgets. JSF does not assume, however, that it can download any kind of logic to the remote system other than a basic description of the screen layout. Of course there is a widely-spread application that fits the description of a JSF client perfectly: the Web Browser.

Like the previous MVC frameworks (Gnome, KDE, MFC, etc) JSF is based around defining a tree of widgets ("components" in JSF terminology), where some of these components have callbacks associated with them. JSF ensures that when data is received back from the remote client the appropriate callbacks are invoked, without (in most cases) user code having to deal with things like HTML forms or urls. Instead, controller code simply responds to events, updates data-models, and then builds an appropriate view to render the new application state.

In practice, JSF component trees (widget trees) are almost always defined via a "template file", and the most popular template file formats happen to be JSP and XHTML files. Purists will limit themselves to defining only widgets in such files, but many people also take the opportunity to encode fragments of traditional HTML or XHTML in such files, and JSF implementations traditionally merge the two to generate the output that is actually sent to the remote client.

You will of course have noticed that while the effect is similar to older web-application frameworks like Apache Struts, the approach is somewhat different. The HTML and Web are abstracted away to a great extent. This allows a number of nice features, like being able to simply change configuration to render HTML, XHTML, WML, and potentially things like PDF or Mozilla-XUL.

Another nice feature that results from this design is that it is easy to add more types of components (widgets). Simply add a library into the classpath, update your views to reference some of these new components, and if necessary attach callbacks to handle the actions that these components can trigger. The !MyFaces Tomahawk, Trinidad and Tobago libraries are component collections that can be used in exactly this way to enhance your JSF applications, and there are more libraries available from many sources.

One of the issues that people notice when moving from more html-centric frameworks to JSF is that there is some loss of control over exactly what urls are generated, what appears in the query parameters, etc. Part of this is due to handing over of control of these aspects to a framework rather than directly managing it; other parts are due to the fact that JSF is still a reasonably young technology and still has a few rough edges. And partly it is due to the fact that JSF is optimized for very interactive applications; it can do things that are extremely complex to do with other approaches, but does not always perform quite as elegantly in the simpler cases.

JSF and MyFaces Core Concepts How JSF State Management works