XAL APPLICATION FRAMEWORK AND BRICKS GUI BUILDER*

Thomas Pelaia II, Oak Ridge National Lab, Knoxville, TN 37830, U.S.A.

Abstract

The XAL [1] Application Framework is a framework for rapidly developing document based Java applications with a common look and feel along with many built-in user interface behaviors. The Bricks GUI builder consists of a modern application and framework for rapidly building user interfaces in support of true Model-View-Controller (MVC) compliant Java applications. Bricks and the XAL Application Framework allow developers to rapidly create quality applications.

INTRODUCTION

The XAL Application Framework is a mature framework for rapid development of applications with a common look and feel and with many features that users expect from modern applications. Over four dozen applications have been written using this framework. The Bricks application is a graphical user interface layout application which assists the developer in creating application views.

APPLICATION FRAMEWORK

Introduction

The application framework provides the foundation for applications which have a graphical user interface. It provides end users with applications that share a common look and feel while providing developers an environment for rapid development of document based applications.

Overview

The framework is event driven and manages the processing of several events which are common to document based applications. An application participates in the framework by providing three classes that subclass abstract framework base classes and three resources that provided data needed by the framework. The three classes override and implement methods which get called by the framework during different phases of the application's life cycle. This allows a division of responsibility between the framework and the application. The framework shoulders as much responsibility as it can for behaviors common across all applications while allowing the developer flexibility for customization. The application framework is in the XAL package: **gov.sns.application**.

Resources

Resources are supporting data files that reside in the "resources" subfolder of an application. The framework provides a mechanism for easily loading resource files. There are three special resource files which the developer typically provides and are processed automatically by the framework. These files are the application information file, the main help file and the menu definition file. These files each have a specific name and well defined format.

The application information file is a simple properties file named "About.properties" which provides information that will be formatted and displayed in the application's "About Box." This information includes the application's name, version, description, a list of authors, the developer's affiliation and the date of the release.

The main help file is named "Help.html" and it is an HTML file that will be loaded and displayed for the user in a standard help window should the user select the "Help" menu item. The developer may provide additional supporting files in the resources folder such as images and other pages as desired. If the help file is omitted then the help menu item is disabled.

The menu definition file is a simple properties file named "menudef.properties" which allows the developer to customize the menu bar and the toolbar which appear with a document. The developer may specify menu bar menus, toolbar buttons, menu items for each menu and sub menu, labels for the menu items and toolbar buttons, groups for radio style menu items and toolbar buttons and an action key for each menu item or toolbar button. The action key allows the developer to assign an action to the menu item or button at runtime. A separator can be specified simply by supplying a dash. The menu definition file is optional and if omitted, the standard menu bar and toolbar will be used.

The standard menu bar includes the File, Edit, View, Window and Help menus. Each menu has the standard suite of menu items that users have become accustomed to expect from using commercial applications.

Principle Classes

The three principle custom classes which the developer provides for their application are the application adaptor, document and document window. The framework provides abstract base classes with abstract methods the subclasses should implement. Other methods may be optionally overridden for further customization.

*ORNL/SNS is managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract DE-AC05-00OR22725

The application adaptor is a subclass of the framework's **ApplicationAdaptor** base class and provides application wide callbacks and information about the application. Only one instance of this class is instantiated for an application. Typically, it is also the main class responsible for launching the application along with receiving the command line arguments if any. Table 1 lists the methods that are typically overridden by the application adaptor.

Table 1: Application Adaptor Methods

Method	Responsibility
readableDocumentTypes	Provide a list of document types the application can open
writableDocumentTypes	Provide a list of document types the application can write
newEmptyDocument	Create a new empty document
newDocument	Create a new document given a URL to the document source
applicationName	Provide the application's name
applicationFinishedLaunching	Optional callback which is called when the application has finished launching
customizeCommands	Implement actions for menu definition items

The document subclass specifies document properties and implements document callbacks. An application may have multiple instances of a document class and even (though rarely) have more than one document class. The document is responsible for reading and writing the file that supports a document as well as managing its associated main window. In the MVC pattern, the document takes the role of a controller and handles instantiating both the model and the main view (document window). The document also typically implements the actions for the menu items defined in the menu definitions file.

The document has a method for setting whether the document has unsaved changes. If a document has unsaved changes, the framework enables the various save buttons and menu items and appends an asterisk to the end of the document's path in the title bar.

Table 2: Common Document Hooks

Method	Responsibility
makeMainWindow	Instantiate a document's window
customizeCommands	Implement actions for menu definition items

The document window subclass is primarily responsible for creating and managing the views within the document's associated main window. Providing a custom window class is now optional since the introduction of the bricks package as discussed later in this paper.

Event Model

The application framework has an event driven model that calls methods in the application's supplied classes to either get information about the principle components (application, document or window) or allow those components to perform a task. For example, a custom document class has a method to load that document from a file. The framework defines a menu item for opening a document. When the menu item is selected, the framework calls a method in the document to determine the supported file types, present the standard file open dialog box with the appropriate filters for the acceptable file types, handle the user's selection and if the user has selected one or more files, calls a method in the application adaptor to open a new document which is then responsible for loading the data. The framework automatically updates the document's title bar to display both the name of the application and the document's file path in response to opening a document.

BRICKS

Motivation

A challenge when developing applications in Java is the effort required to build a user interface while adhering to the MVC design pattern. In a typical XAL application, the developer would write the window class using the Java Swing packages. Typically, that code would describe the layout of the views within the window, define the actions of controls (e.g. buttons) and populate the container elements (e.g. tables and lists). In practice, this makes it

very difficult to decouple the view and the controller as required by MVC. Also, the developer is forced to spend time laying out elements in code only to find that the display of the window doesn't match their expectation. This results in less time spent on the actual purpose of the application and less than ideal user interfaces.

Many Integrated Development Environments (IDEs) for Java include graphical interface design editors, but they share common shortcomings. They typically generate Java source code for each window. While these tools address the time that it takes to build a user interface, they do nothing to address MVC compliance and they often make the problem even worse. When the user edits the code for the window, they must now navigate through complex and often proprietary code that is generated by the tool while being careful not to edit certain parts of the code reserved by the tool. Furthermore, this approach ties development to a single proprietary IDE choice and does little to help with building user interfaces for scripts written in Java based scripting languages such as Jython [2] and JRuby [3].

Bricks is an XAL application which allows the developer to rapidly construct user interfaces while naturally adhering to MVC compliance. This design for *Bricks* is inspired by OpenStep's™ Interface Builder™ [4]. A *Bricks* document saves the resulting window definitions in a XML file. A *Bricks* package provides the runtime methods for generating the views from a *Bricks* document when it is time to run the target application. These methods are typically called in the controller.

The main window for a *Bricks* document displays a hierarchy of windows and their views for an application. A palette of views is provided so the developer can simply select and drop views into the view hierarchy. A preview window updates to reveal how the window will appear when open in its application. Notional data is provided for tables and lists in the preview so they have a nontrivial appearance. An inspector allows for quick editing of many properties for a selected view. Copy, Cut and Paste support along with drag and drop support make it easy to rearrange views in the view hierarchy window. The built in code assistant helps developers avoid errors by allowing them to paste references to views right into their code independent of their IDE.

Once the views are constructed, the developer needs a way to access the views from within the controller classes and display the window. The *Bricks* package has a method to instantiate a window reference from a bricks document. The window reference has a method to get any view within the window by passing a tag that the developer assigned to the view during construction. Because *Bricks* uses this runtime approach to generate the views, no compilation is required which makes it ideal for use with Java based scripting languages.

The XAL application framework also has explicit support for the *Bricks* runtime. Convenience methods in both the application adaptor and document base classes make it easy to load the bricks definition file located in an application's resource folder. This means that the main document window subclass for an application no longer needs to be provided when using the application framework. This results in less time spent coding user interfaces and more time available for developing an application's model.

SUMMARY

The XAL application framework and the *Bricks* tools can be used together or independently of each other to rapidly build Java applications that offer a great user experience.

REFERENCES

[1] http://neutrons.ornl.gov/APGroup/appProg/xal/xal.htm

[2] http://www.jython.org

[3] http://jruby.codehaus.org

[4] Nancy Craighill, *OpenStep for Enterprises*, John Wiley & Sons, Inc., New York, NY, 1997.