Design for Reuse

by Theresa Ray of Tensor Information Systems

Sponsored by Apple Computer, Inc.
Worldwide Developer Relations Group

Design for Reuse

by Theresa Ray of Tensor Information Systems

On-line Technical Documentation

One of the best resources for all WebObjects programmers, regardless of experience level, is Apple's on-line technical documentation. The documentation provided there, specifically the section on Creating Reusable Components, has an excellent introduction to the topic of design for reuse and the terminology necessary for continued discussions. This survival guide assumes that the reader is familiar with the information presented there. Apple's technical documentation can be found at http://developer.apple.com/techpubs/enterprise/WebObjects/WebObjectsTOC.html

Reuse

One of the biggest challenges facing today's programmers is the issue of "design for reuse". Every application has a specific set of requirements and goals which requires some customized code, but starting the code from scratch on every project is not the best use of your company's available resources. Obviously, the average application will consist of both reusable and application-specific code, but how do you determine the best way to segment your code? The best application design starts with a good project architecture. Design patterns, such as the Model-View-Controller design pattern, can help you structure your application in the most efficient way.

Model-View-Controller Design Pattern

Design patterns, such as Model-View-Controller, help to categorize the code being written. The Model-View-Controller design pattern separates classes into one of three categories – model, view or controller.

Model

The model classes contain your application-independent business logic and tools. Applications storing information in a database have a set of classes corresponding to the entities defined in the database. These classes, known collectively as an object model, include logic such as validation rules and formatting of data. Any application

using that database uses the object model and is therefore subject to a single set of rules for storage and manipulation of data. It does not matter whether it is a command-line Yellowbox/OpenStep application, an NT-deployed client-server application developed with Appkit, or a WebObjects application. Any type of application developed can use the object model framework developed.

For example, you want to store a customer's name and address information in a table known as CustNameAddress. Your object model will have an entity which corresponds with the database entity. In addition to the traditional accessor methods required by this class, you may include the following method:

```
public Exception validateForSave() {
      if (firstName == null) {
            System.err.println("Input
            Error: The first name field may not
            contain numeric data.");
            return new Exception();
            }
      if (stringManipulator.hasNumericIn(firstName)) {
            System.err.println("Input
            Error:  The first name field may not
            contain numeric data.");
            return new Exception();
            }
      // Do the same thing for all other required fields
      return null;
      }
```

or, in Objective-C terminology:

```
- (NSException *) validateForSave {
      if (!firstName || [firstName isEqual:@""])
            return [NSException
            exceptionWithName:@"Input Error"
            reason:@"The first name field is
            required." userInfo:nil];
      if ([stringManipulator
      hasNumericIn:firstName])
            return [NSException
            exceptionWithName:@"Input Error"
            reason:@"The first name field may not
            contain numeric data." userInfo:nil];

      // Do the same thing for all other
      required fields
      return nil;
      }
```

This method is called during saveChanges or tryToSaveChanges and is covered in depth in the EOF documentation. In the case presented here, if no value is entered for the firstName field, an exception is raised (the application must include an exception handler to appropriately deal with the raised error, but by including a useful reason in the exception, that reason may be shown to the user directly). Similar code would exist for all required fields in the entity. By storing the validation logic in the object model, all applications are guaranteed to use the same set of business rules in updating the database, since the validateForSave method is called in each case.

Notice the second if statement – if ([stringManipulator hasNumericIn:firstName]). The method hasNumericIn: is not a standard NSString method, nor was it defined in the CustNameAddress entity (self was not the recipient of the message). Instead of writing the specific code to verify that firstName did not contain any numeric characters in the validateForSave method or even within the object model, that method can be defined in a set of classes independent of the object model and might include other methods such as hasCharacterIn:, etc.

This other set of classes would exist as a company-specific framework similar to the Foundation framework provided by Apple, and should be made available to any application on the entire system. Instead of writing code to check for a numeric character in the firstName field and duplicating that code for the lastName, address, and city fields, you have started a company-specific framework which ensures that no one else needs to write the code to determine if any NSString contains a numeric character. In addition to the validateForSave method, the accessor method itself might contain business logic. For example, the firstName accessor method might look like:

```
public String firstName() { return firstName; }
public void setFirstName(String value) {
   firstName=value.uppercaseString;
   }
```

or, in Objective-C terminology:

```
- (NSString *) firstName { return firstName; }
- (void) setFirstName:(NSString *) value {
   value=[[value uppercaseString] retain];
   [firstName autorelease];
   firstName=value;
   }
```

In this example, business logic dictates that the first name should be stored as an uppercase string. By formatting the value directly in the accessor method, all applications using the object model are ensured of following the business logic definitions. Obviously, this example is trivial and could be implemented with formatters. More complex business logic (such as restricting the suffix field to a subset of allowable values, or running a ZIP code against software to validate it) would be implemented in

a similar manner.

View

The view classes contain reusable code for displaying information to the user. For example, you might develop a WebObjects component for a login panel. The standard layout – two text fields, a reset button and a submit button – is not likely to change much from one application to another. Instead of creating these four elements in the Main component of every application you develop, you can define them within a LoginPanel component and include just the LoginPanel on each of your Main pages. To make the LoginPanel even more generic, the submit and reset buttons might be active images whose source image is passed as an attribute to the LoginPanel component, allowing each application a slightly different "look and feel" while still reusing a single common component.

For a different type of reuasable view component example, consider a component named TISSmartString. This component would have an attribute called isEditable and contain logic to display a WOString if the value of isEditable is NO, or a WOTextField if the value of isEditable is YES. This component allows user-friendly formatting of text based on the level of access provided to the current user. Instead of showing a WOTextField to a user who does not have edit priveleges on the data, this component would display the information as regular text, removing any confusion for your users, yet keeping your code generic.

Obviously, due to the nature of the application interface, view components are not typically reusable between client-server and web-based user interfaces. Whereas the model components may be used by any application, reuse of view components is usually restricted to the interface type of the application that the view component was originally designed for. Typically, your company will construct a set of reusable components for WebObjects applications, and a separate set of reusable AppKit components (subclasses of NSTableView or NSScrollView, for example).

Controller

The controller classes contain application-specific logic which tie the model and view components together. After the submit button on the LoginPanel component is clicked and the user entity in the database is checked for validity of the login, which page is displayed to the user next?

Logic used for navigation of the web site is the most common use for code found within a WebObjects controller class. But WebObjects controller code is essentially all the logic contained within your WebObjects script file for the page being displayed (it may be Objective-C or Java code as opposed to scripted). It is unusual to find page-level logic that is reusable from one instance to another. If you are able to reuse any of the code

from your page component in multiple other pages or applications, chances are you have incorporated code that really should be made into a common component.

Application Reuse

Sometimes entire applications are reusable. For example, you are developing a system which requires a data model with thirty different entities. A dozen or so of these entities contain lookup information that will be used by one or more applications. You have identified the need for a web-based table maintenance application. Your object model already contains the validation logic for each entity, so what you really need is a user interface to the object model for data entry.

Instead of developing hard-coded pages for each entity to be accessed, you could structure a very generic application for use with ANY data model. By initializing the application with a list of entities allowable for data entry, the application could read the object model "on the fly", intelligently display a data entry field for each attribute in that entity and update each attribute appropriately upon submission (see the EOF Developer's Guide for more information on this technique). Not only are you eliminating the need for a separate component to handle data entry for each entity in the model, but the entire application is reusable from one project's object model to the next. Validation logic is handled by the object model itself, and with an exception handler wrapped around the saveChanges method, any data entry errors can be cleanly displayed to the user.

Documentation

Documentation and availability of reusable code is ESSENTIAL. So you've created this really cool set of frameworks to do all sorts of good things with strings and date and numbers. You need to "sell" it to your fellow developers by documenting it well. Application developers are notorious for "reinventing the wheel". No one writes code exactly like the next person, so there is a strong tendency to rewrite code instead of reuse it. Anobject-oriented development environment only intensifies this problem. Because the user of a framework doesn't have access to the implementation of a method, he or she may not feel comfortable using it. If the documentation is thorough, however, the user's confidence level in the framework usually rises significantly and overcomes the rewrite tendency.

Reusable Objective-C code

There are a few "safety tips" to note when writing Objective-C frameworks which will be used by both WebObjects containing scripted components and non-WebObjects

applications. The first is to always implement your methods to return objects instead of BOOL, int, float, etc. The scripting language within WebObjects does not support these simple data types, and you will encounter difficulty using your framework.

Similarly, all method arguments should be objects as opposed to simple data types. Additionally, the use of (void *) as a data type is not supported by the WebObjects scripting language and should be avoided in frameworks to be called from script.

Every possible entry point into a common class should be wrapped in an exception handler in order to enhance WebObjects' default exception handling mechanism. For example:

```
- (void) someMethod {
      NS_DURING
      {
            //-- whatever code, however benign
      }
      NS_HANDLER
      {
            //-- some handler code goes here
      }
      NS_ENDHANDLER
}
```

The usual rules for code reuse pertinent to development with other programming languages also apply to programming with Yellowbox/OpenStep and WebObjects. Don't assume any particular data size (will int always be 32 bit? No.). Don't use any platform-specific functionality in your code (sometimes you have to include certain libraries in the Makefile for a particular platform). Check your reusable code for memory leaks frequently, and fix any leaks you find immediately.

Conclusion

Obviously, developing reusable code requires thought at the front end of the design. It also requires a good understanding of other development efforts planned or under way at your company. The first few projects undertaken may suffer a slight cost or schedule impact from the time taken to study these issues, but the moderate additional effort is money well spent in the long run.

Copying and pasting code into ten different applications and finding out later that a logic error exists in that code, results in spending a significant effort tracking down and fixing the code that has been replicated. It is a rare company indeed is organized enough to know exactly where all the copied code has been deployed. If that code had been reused from a common framework, however, you could update the framework and KNOW that all occurrences of that error were fixed in a relatively short amount of time.

References

http://gemma.apple.com/techinfo/techdocs/enterprise/WebObjects
WebObjects Developer's Guide
Enterprise Objects Framework Developer's Guide
http://www.omnigroup.com/MailArchive/WebObjects
http://www.omnigroup.com/MailArchive/eof
http://www2.stepwise.com/cgi-bin/WebObjects/Stepwise/Sites
ftp://dev.apple.com/devworld/Interactive_Media_Resources
http://www.apple.com/developer
http://developer.apple.com/media
http://enterprise.apple.com/NeXTanswers

About the Author

Theresa Ray is a Senior Software Consultant for Tensor Information Systems in Fort Worth, TX (*http://www.tensor.com*). She has worked as a consultant on WebObjects projects for a wide variety of clients including the U.S. Navy, the United States Postal Service, America Online, and Proctor and Gamble. Her experience spans all versions of WebObjects, from 1.0 to 4.0 beta, several versions of EOF, from 1.1 to 3.0 beta, AppKit, NEXTSTEP 3.1 to OPENSTEP 4.2, Rhapsody for Power Macintosh, and yellow-box for NT. In addition, she is an Apple-certified instructor for WebObjects courses.

Tensor Information Systems is a Apple partner providing systems integration and enterprise solutions to its customers. Tensor's employees are experienced in all Apple technologies including OPENSTEP, NEXTSTEP, Rhapsody, EOF and WebObjects. Tensor also provides Apple-certified training in WebObjects, Oracle consulting and training, as well as systems integration consulting on HP-UX.

You may reach Theresa by e-mail: *theresa@tensor.com*

Interactive Media Resources Include:

- **Interactive Media Guidebooks**
- **Market Research Reports**
- **Survival Guides—Technical "How To" Guides**
- **Comarketing Opportunities**
- **Special Discounts**

Apple Developer Connection

As Apple technologies such as QuickTime, ColorSync, and AppleScript continue to expand Macintosh as the tool of choice for content creators and interactive media authors, the Apple Developer Connection continues its commitment to provide creative professionals with the latest technical and marketing information and tools.

Interactive Media Resources

Whether looking for technical guides from industry experts or for market and industry research reports to help make critical business decisions, you'll find them on the Interactive Media Resources page.

Apple Developer Connection Programs and Products

ADC programs and products offer easy access to technical and business resources for anyone interested in developing for Apple platforms worldwide. Apple offers three levels of program participation serving developer needs .

Membership Programs

Online Program—Developers gain access to the latest technical documentation for Apple technologies as well as business resources and information through the Apple Developer Connection web site.

Select Program—Offers developers the convenience of technical and business information and resources on monthly CDs, provides access to prerelease software, and bundles two technical support incidents.

Premier Program—Meets the needs of developers who desire the most complete suite of products and services from Apple, including eight technical support incidents and discounts on Apple hardware.

Standalone Products

Apple offers many standalone products that allow developers to choose their own level of support from Apple or enhance their Select or Premier Program membership. Choose from the following products and begin enjoying the benefits today.

Developer Connection Mailing—Subscribe to the Apple Developer Connection Mailing for the latest in development tools, system software, and more.

Technical Support—Purchase technical support and work directly with Apple's Worldwide Developer Technical Support engineers.

Apple Developer Connection News—Stay connected to Apple and developer-specific news by subscribing to our free weekly e-mail newsletter, *Apple Developer Connection News*. Each newsletter contains up-to-date information on topics such as Mac OS, Interactive Media, Hardware, Apple News and Comarketing Opportunities.

Macintosh Products Guide

The most complete guide for Macintosh products! Be sure to list your hardware and software products in our *free* online database!

http://developer.apple.com/media

The ultimate source for creative professionals.