PhotoRequest: Understanding Information Management in a Publishing Environment

CPSC 490, Spring 2004

Nathan Francis
nathan.francis@yale.edu

Advisor: Avi Silberschatz

Abstract

How can we integrate many different technologies to create a tool for information management? Newspapers demand a paradox of computing: complex systems that journalists can use. Using Objective-C, Cocoa, the SMYSQL Objective-C framework, MySQL database, BSD sockets and an SMTP server, we create a native application complex enough to handle a real newspaper environment while maintaining the simplicity of pen and paper.

The application developed through this project will be used in a newspaper environment by forty simultaneous users to create and track assignments. By combining many existing technologies into a single solution, we have replaced the previous paper-based system with an electronic analog that provides superior functionality. At the same time, the structure of the system allows for future development of alternative interfaces to the data we are now collecting.

Introduction

Accurate, fast information is critical for success in the news publishing industry. Photography editors constantly rely on third-party information to ensure that there will be photographs in the newspaper the next morning.
Photo editors deal with hundreds of pieces of identical paper, all entitled “Photo Request Form” which are vital to their role as editors. Yet despite the importance of these forms, many become destroyed, misplaced, contain errors in dates and critical information, and occasionally cause a delay in publication of news stories until the requisite photograph can be acquired.

This real-world problem gives us the chance to replace the paper-based system with a database powered electronic system. By developing an electronic solution, we can force the users to enter accurate information while adding functionality through built-in interfaces to different technologies like mail servers.

When Apple acquired NeXT in 1996, they also inherited the rich NSFoundation and NSApplication APIs for Objective-C originally developed for NeXTStep/OpenStep. With the first release of Mac OS X in 2001, Apple integrated these frameworks and the Objective-C language into their free development tools as the default language of choice for Mac OS X development. Retooling the NeXT Interface Builder for Mac OS, Apple retained the proprietary .NIB format for dynamically loaded interface elements controlled through Objective-C while developing their own development tool Xcode, based on gcc, gdb, cvs and other standard UNIX development tools.

Serge Cohen developed an open source framework SMySQL to convert the standard MySQL C API into an Objective-C framework compatible with Mac OS X development. With the use of this framework, it is possible to extract data from MySQL databases as NSObject typed objects compatible with the Cocoa API.

Standard ANSI-C can be interspersed with Objective-C, allowing us to develop unique ways to send data to other sources from without our Objective-C based application.

Design Goals
Below is a diagram of the photography workflow. The shaded
area represents manually-performed tasks that will be handled by our system:

Our application will have a three-panel interface representing the three main stages of this process: creating a new request, scheduling a photograph for assignment, and assigning photo requests. Instead of manually photocopying a photo request, we will integrate the application with an SMTP server so that requests can be emailed to photographers.

**Implementation**

The application will be implemented in Objective-C. Apple’s Cocoa API will be used for the creation of project classes. All classes developed for the project will be subclasses of NSObject, the default object type for all NSApplication or NSFoundation projects. Serge Cohen’s SMySQL framework will be used to provide an Objective-C interface for the MySQL database powering the application. We will use C interspersed with Objective-C to write a BSD socket that will push a well-formatted row from the database to an SMTP server.

The Model-Controller-View derived from Smalltalk-80 is strongly pushed by Apple as the development model of choice for Cocoa/Objective-C. Under this paradigm, there are three types of objects,
each with well-defined boundaries. The following diagram shows the relationships between these three types of objects:

Our view objects will be created through Interface Builder and handle user input, while the model and controller objects will access the database and format the input and output to and from the GUI.

The GUI representing the View type is developed in Apple Interface Builder, a graphical interface builder that allows the user to drag and drop interface elements. The interesting aspect of IB is that the user can create classes and instantiate objects all within Interface Builder. Objects can be linked together by dragging connections to and from outlets, establishing communication channels between instantiated objects and graphical elements.

We have two controller objects: NewRequestController and ViewRequestController. NewRequestController handles taking user input from the View objects and formatting it properly for entry into the MySQL database. A large part of the NewRequestController code is error-checking and string comparison routines to ensure the user cannot issue a request until all information has been properly entered. Specific interface elements appear based on the desk selection, allowing the user
to enter additional information relevant to the desk issuing the request. If all information is complete, the information is extracted from the view objects and packed into an array which is sent to RequestHandler.

RequestHandler initiates a connection to the MySQL server via TCP and issues an INSERT statement into the database. The user is notified by a dialog box displaying the message returned from the RequestHandler on the outcome of the insert command. NewRequestController sends the view objects messages to clear their contents in anticipation of a subsequent request.

ViewRequestController is responsible for too much functionality. Breaking with the M-V-C paradigm, ViewRequestController assumes some responsibilities of the model type object by communicating with the database directly. This happened as a result of the creation of NSTableView objects to handle displaying the data, and a need for frequent database updates. Many of the view level objects act as WHERE clauses on SELECT statements used to fill the tables. Instead of passing messages from the view level, to the control level, to the model level, I moved the database calls into the controller level to avoid the passing of a second message.

The ViewRequestController class has methods that handle table formatting, DELETE requests, refresh requests from the user, implements the filter criterion for the “View Request” panel, full view of requests in a separate window, and handles the email functionality. A majority of this code is again, error checking, string comparison, and generation of SQL statements based on user input and selection. The menus at the top of the window allow the user to select multiple filters to perform queries such as “show all requests for the Next Issue for the News desk that have not been assigned.”

Email functionality allows the editor to press “Email” and have the request information automatically sent to the photographer. A method within the ViewRequestController was created that uses a socket to open a connection to mail.yale.edu on port 25. Cocoa allows
conversion from NSString objects to standard C strings, including buffer-overflow protection. Using BSD `read()` and `write()` system calls, the buffered strings can be sent to the photographer in a human-readable format.

ViewRequestController also controls the Assign Request panel. An UPDATE statement is issued when a request is highlighted and the user has chosen a camera and photographer. As with the Issue Request panel, the user cannot issue a request until all appropriate information has been entered.

Conclusion

While in some sense this project is a case of “reinventing the wheel,” it does so through new technologies and methods. Objective-C is not a widely known language, yet it has some interesting properties that make it appealing—including the ability to mix in ANSI C code and its memory management features which allow for complex manipulation of strings and other high-level data types like dictionaries and arrays of objects of varying type.

MySQL proved to be an excellent data server to use for this application because of its active open source development team and multiple programmer interfaces. As with the implementation of the socket connection to the SMTP server, the MySQL and the SMysQL framework allowed multiple open source technologies to be used in conjunction with one another. While PhotoRequest will undergo revision before entering use as a production tool, the underlying foundation of the project shows how the combination of many technologies can be used to create a functional tool that can be understood by users who have no concept of the underlying foundation. The user sees a clean interface that can easily be manipulated without understanding the different technologies that are responsible for its functionality.
Screenshots
Readings


