THERE AND BACK AGAIN: LEVERAGING iOS DEVELOPMENT ON MAC OS X*

Michael P. Rogers
Northwest Missouri State University, Maryville, MO
309-825-6454
mprogers@mac.com

ABSTRACT

More and more universities have been offering courses on iOS Development. With this knowledge, students can develop apps on iPhones, iPads, and iPod Touches, but they can do more: most of what they have learned applies directly to Mac OS X development, and so by expending just a little more effort they can expand their horizons into a novel and empowering realm. This paper describes the relationship between iOS and Mac OS X, a transition path for iOS-savvy students wishing to learn how to write Mac OS X apps, and the motivation for undertaking such a journey.

1. INTRODUCTION

With the commercial success and huge amount of publicity that has surrounded the release of the iPhone, iPad, and iPod Touch, it is not surprising that numerous universities are now offering courses on iOS development [1-4]. Students enrolled in these courses face a steep learning curve. The majority likely have no experience with the language used to develop iOS apps, Objective-C, as it remains one of the more obscure dialects of C [5]. Likewise, the frameworks (Cocoa Touch) and Integrated Development Environment (IDE) of choice (Xcode) are also likely to be unfamiliar.

Once these topics have been mastered, however, students are able to develop apps in any number of areas, and enjoy the recognition and possible financial gain that comes from having them accepted to and distributed via the App Store.

By virtue of the platform on which they run, though, these apps are somewhat constrained. They run on smaller screens, on slower processors, and employ keyboards

---

that, while ingenious, are less than ideal for extended text input [6,7]. Finally, the apps, while running on a UNIX variant, do so in a sandbox, and consequently cannot tap into much of the power of the underlying OS.

Some students may find themselves chafing at these limitations, and yet remain eager to develop using the same IDE, frameworks, and languages. It is not simply that they will have invested so much time in acquiring, and therefore be reluctant to set aside, their hard-won knowledge. The technology is intrinsically appealing: the combination of the Cocoa Touch frameworks and the IDE (Xcode and Interface Builder) make it possible to wire together powerful apps with great ease and alacrity.

Happily, there is a route for these students, and it is, in a sense, a return to roots: iOS is an offshoot of Mac OS X, and there are compelling reasons for encouraging these students to come full circle, to explore iOS's progenitor.

In this paper, we explore these reasons. We begin by outlining the differences between iOS and Mac OS X, from a student developer's perspective. We then consider in detail the numerous benefits of developing for the latter platform. Finally, we describe the creation of a single app, TyperTimer, for both platforms, to illustrate just how much overlap there is between the two APIs.

2. A COMPARISON OF iOS AND MAC OS X: A STUDENT DEVELOPER'S PERSPECTIVE

For many Computer Science students, languages are a major concern, so it is therefore fortuitous that iOS and Mac OS X share one. Objective-C - a superset of ANSI C that includes object-oriented extensions based on SmallTalk [8] - is Apple's preferred language for writing general-purpose Mac OS X applications, and the only rigorously supported language for iOS development.

Moving from iOS to Mac OS X, students will discover at least one reason to celebrate: garbage collection, which is disabled on iOS device runtimes (for the sake of battery life and performance), is supported on Mac OS X. For those who dread Objective-C's seemingly byzantine memory management system, this will come as a particularly welcome change.

There is just one set of software development tools for both iOS and Mac OS X development, namely Xcode (an IDE), Interface Builder (for laying out interfaces), and Instruments (for performance monitoring). When a student launches Xcode, they will merely have to choose a Mac OS X template rather than an iOS template, to begin creating a Mac OS X project. The project organizer, editor, debugger and documentation bookshelves are identical. Xcode includes 5 Mac OS X templates, and just 2 iOS templates, an inkling that there are more avenues open to Mac OS X developers

In iOS, student developers spend most of their time interacting with the Cocoa Touch frameworks, which incorporate two key sub-frameworks. Foundation is comprised of the most commonly used non-visual classes (NSStrings, NSNumbers, NSArrays, NSThreads, etc.). UIKit consists of classes that have a visual representation on the screen (UIView, UIImageView, UIButton, UITextField, etc).

Shifting from iOS to Mac OS X, student developers will find that existing Foundation classes are largely unchanged [9]. Several additions appropriate for a computer-based application have been added, the most significant of which is bindings: controls can be linked either programmatically or in Interface Builder, so that a change in one control automatically changes the other.

Changes in the user interface classes are more extensive. The Mac OS X framework, known as AppKit rather than UIKit, can display multiple windows simultaneously; menu support is built-in; an entire sub-framework devoted to managing documents is included; NSTableViews can support multiple columns (UITableViews, in use on devices with smaller screens, support only one); even coordinate systems are oriented differently (the origin is at the bottom-left in an NSWindow, as opposed to top-left in a UIWindow).

Adjusting to these changes is not as daunting as it might sound, because for the most part they add functionality. As we discuss in section 4, a simple iOS app, consisting of a single view with multiple controls, can be migrated over to Mac OS X with little more than a few changes in class names and methods. The most difficult challenge is to design an application that can take advantage of all the extra features available on Mac OS X.

Up until recently, one huge advantage of iOS over Mac OS X development had been the existence of the App Store. It provides a convenient system for app distribution and a means for students (and instructors) to obtain an outside and rigorous [10] assessment of their efforts. Apple has now introduced a Mac OS X analogue. Students can still distribute Mac OS X applications via other channels, but the Mac App Store provides greater visibility, an implicit endorsement by Apple, and an exciting target for students to aim for.

3. THE BENEFITS OF MAC OS X DEVELOPMENT

3.1 HARDWARE BENEFITS

The most obvious hardware benefit of Mac OS X development can be seen at a glance: larger screens mean that more data or detailed imagery can be portrayed in its entirety, and, if necessary, windows can be expanded or multiple windows displayed simultaneously. When it is necessary to have multiple applications open at once, the screen size makes it possible to view windows in each, and transfer data back and forth.

In contrast, on iOS devices there is a single, fixed, small window. Large amounts of data must be divided into several views, and some mechanism provided so that the user can page through these views; images may be enlarged to show more detail, and iOS device displays are high resolution, but then only a small fraction of the image is visible. Everything feels slightly confining. iOS 4 does support multitasking, but since the devices only support one window, transferring data among applications takes more effort.

iOS devices are marvels of engineering, but they are not nearly as speedy as the computers that run Mac OS X. Table 1 [11] shows that, in Geekbench 2 Tests, the iPad is much slower than even the MacBook Air.

Device	Processor	Score
MacBook Pro (13", Early 2010)	Intel Core 2 Duo P8800, 2.66 GHz	3655
MacBook Air (13", Late 2010)	Intel Core 2 Duo L9400 1.86 GHZ	2695
iPad	Apple A4	453

Table 1

In spite of all this, iOS devices typically do not feel sluggish, nor do their screens feel inadequate: successful developers design and target their apps for the most appropriate hardware. A developer considering a word processing app would probably be better off on a computer; a developer considering a GPS-related app would likely prefer an iOS device (one with a built-in GPS).

One final hardware-related advantage of Mac OS X development is that getting an app to run on an actual device is easy; the program that is built in Xcode is ready to execute and distribute. For an iOS app, however, the last steps, in order to transfer an app to a physical device, require the student become a registered iOS Developer ($99/year), download several certificates from Apple, upload one of these certificates to the device, and code sign their app [12].

To summarize, students interested in creating an app with the sort of immersive experience that require larger screens; that is more computationally demanding; in which data management is key; which lends itself naturally to multitasking; and requires no extra effort to run on its targeted platform, would be better off with a Mac OS X device.

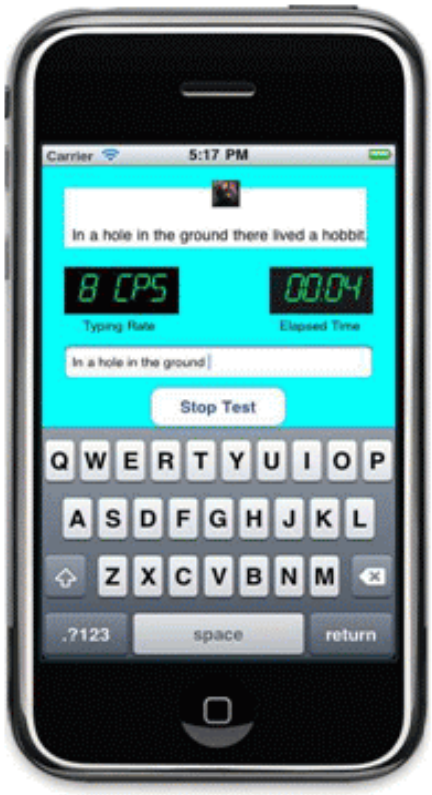
3.2 SOFTWARE BENEFITS

Mac OS X comes with more mature and diverse frameworks than iOS. Students studying computer graphics will appreciate that a full OpenGL implementation is available (iOS ships with a pared down version, OpenGL ES); they may also have reason to use Accelerate, a powerful framework for doing image manipulation and vector-based mathematical calculations in hardware. Students studying parallel computing can solve embarrassingly parallel problems using Xgrid[13], a technology that, because it ships with all Mac OS X computers, makes it possible to set up a computer cluster in a lab setting with very little effort. At a lower level, Mac OS X supports a distributed objects architecture that allows transparent inter-process and inter-machine communication: writing servers and clients using NSDistributedObjects is straightforward. Students in operating systems have the option to write daemons (system-wide background processes) and agents (background processes designed for a particular user). Finally, all students will enjoy NSSpeechSynthesizer and NSSpeechRecognizer, two versatile but delightfully easy-to-use classes that are lacking (and sorely missed) in iOS. An aural version of "Hello, world" takes just two lines:

```
NSSpeechSynthesizer * speaker = [[NSSpeechSynthesizer alloc]init];
[speaker startSpeakingString:@"Hello, world!"];
```

Migrating from iOS to Mac OS X development, students will find the platform more open, allowing access to a plethora of UNIX tools, shells and languages. Students who are familiar with the likes of awk (a programmable text processing system), bc (a scriptable calculator), grep (a regular expression processor) and scripting languages (e.g., perl, python, and ruby), will instantly recognize and appreciate the tremendous power that this puts at their disposal.

Access to all this power comes through NSTask and NSPipe classes. These make it possible to execute a particular task, and pipe the output from one task to another, respectively. NSTasks are not just limited to UNIX tools. Many Mac OS X applications have a command-line interface (Podcast Producer, MySql) as well, and they can be controlled in a similar fashion. None of this is possible with iOS devices. iOS apps run in a tightly controlled sandbox, and have no means to invoke any applications directly.



4. A SAMPLE APPLICATION

In order to illustrate just how readily a student, versed in the iOS APIs, could adapt, we developed TyperTimer, a small app to assess a user's keyboarding prowess. We wrote it first in iOS and then ported it to Mac OS X. We did so by literally pasting the code from the completed iOS app into the skeleton-code of like-named classes in a Mac OS X application, and then began editing. The process was enlightening: encouraging students to perform a similar task with one of their iOS apps would make for an excellent project.

Code that did not involve GUI - calls to NSString, NSDate, NSTimer and NSArray objects - functioned identically on both platforms, without change. We chose to disable garbage collection on Mac OS X to be able to make the preceding statement: we could have saved a few lines had we done otherwise.

There no major conceptual changes in the GUI code, but adjustments were necessary. There were numerous class and method name changes, a few of which are noted in Table 2:

	Classes				
	UITextField	UIButton	UIAlertView	AVAudio Player	UILabel
Mac OS X	NSTextField	NSButton	NSAlert	NSSoun d	NSTextField

	Methods				
iOS	text, setText	setTitle: forState:	initWithTitle: message:delegate: cancelButtonTitle: otherButtonTitles:otherButtonTitles	play	text, setText
Mac OS X	stringValue setStringValue	setTitle:	alertWithMessageText: defaultButton: alternateButton:otherButton:informativeTextWithFormat:	play	stringValue setStringValue

Table 2

This small sampling of a few classes and methods may make the changes seem cosmetic and perhaps even arbitrary; in fact, the designers of the iOS APIs have carefully streamlined and refined the classes to make them easier to use, without alienating Mac OS X developers.

Most iOS and Mac OS X apps use Interface Builder to lay out the user interface in what are known as .xib files. These .xib files contain an XML specification of the interface's objects, including their attributes and connections to other objects. One side effect of the redefinition of classes is that these .xib files must be recreated from scratch - one cannot simply copy an iOS .xib file into a Mac OS X project.

The iOS APIs use properties, invoked with dot notation, much more so than in the Mac OS X APIs. This tends to result in pithier code. For instance, to assign a value to a textfield, we would write, in iOS and Mac OS X, respectively:

```
iOSTextField.text = @ "Hello, Bilbo";                    // UITextField * iOSTextField
[macOSXTextField setStringValue:@"Hello, Bilbo"]; // NSTextField * macOSXTextField
```

Other changes include slight variations in the life cycle of UI objects created in .xib files; and a diminution of the importance of view-controllers (in iOS, UIViews and UIViewControllers are intertwined, fundamental, and created automatically in Xcode templates; in Mac OS X, the default templates provide an NSWindow, but the user must supply all other necessary classes).

5. CONCLUSIONS

Students writing iOS apps, by virtue of their position in front of a Mac OS X computer running Xcode, have all that they need to begin exploring a variety of exciting new dominions, from numerical computation to computer graphics to distributed computing, using languages, IDEs and frameworks that they have already mastered. The transition is relatively painless, and by exploring Mac OS X they will also learn more about iOS; they will see the changes that expert API designers have implemented; and by porting their own iOS apps over to Mac OS X, they will, in one project, learn more about the importance of MVC than they would by any other means.

6. REFERENCES:

[1] Zardon. University Classroom Courses, 2010, www.stanford.edu/class/cs193p/cgi-bin/drupal/.

[2] Chen, B.X. UC Davis Teaches iPhone App Development, Too, 2009, www.wired.com/gadgetlab/2009/12/uc-davis-teaches-iphone-app-development-too/.

[3] iPhone Programming, 2010, www.uh.edu/continuingeducation/professional/iphone.php.

[4] Johnson, P.D. University of Maryland Introduces iPhone Programming Course, 2010, www.newsdesk.umd.edu/uniini/release.cfm?ArticleID=2077.

[5] TIOBE Programming Community Index for November 2010, 2010, www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[6] Gyford, P. Pen v keyboard v Newton v Graffiti v Treo v iPhone, 2010, www.gyford.com/phil/writing/2010/01/18/input.php.

[7] Hoggan, E., Brewster, S.A., Johnston, J. 2008. Investigating the effectiveness of tactile feedback for mobile touchscreens. *CHI 2008 Proceedings*, 26, 1573-82, 2008.

[8] Kochan, S.G. *Programming in Objective-C 2.0.* Upper Saddle River, NJ: Addison-Wesley, 2009.

[9] Apple Inc., Migrating from Cocoa, 2010, tinyurl.com/2byjpct.

[10] Apple Inc., App Store Review Guidelines, 2010, developer.apple.com/appstore/resources/approval/guidelines.html.

[11] OSXDaily, New MacBook Air 11" and 13" Benchmarks, 2010, osxdaily.com/2010/10/25/new-macbook-air-benchmarks/.

[12] Apple Inc., 2010, iOS Developer Program, 2010, developer.apple.com/programs/ios.

[13] Apple, Inc., 2010, developer.apple.com/library/mac/#documentation/MacOSXServer/Conceptual/Xgrid_Programming_Guide/Introduction/Introduction.html.