CONCEPTS AND PRACTICAL CONSIDERATIONS OF PLATFORM-INDEPENDENT DESIGN OF MOBILE MUSIC ENVIRONMENTS

Jong Wook Kim
University of Michigan
EECS, Ann Arbor, USA
jongwook@eecs.umich.edu

Georg Essl
University of Michigan
EECS and Music, Ann Arbor, USA
gessl@eecs.umich.edu

ABSTRACT

UrMus is a mobile music environment that provides a live and interactive design and programming platform for multi-touch mobile devices. Platform independence and cross-device development and code migration are integral to allow flexible design of musical interactions and networked performances. This paper discusses software architecture considerations that enhance device-independence in the design of urMus as well as practical aspects of porting it to Android with implications for porting other similar environments.

1. INTRODUCTION

Mobile music performers and programmers usually do not want to be concerned with the specifics of the hardware, but rather develop and perform on the hardware that is available. Hence it is helpful if mobile music environments can hide or mitigate the dependency on hardware as much as possible. Unfortunately mobile smart phones have a diverse operating system and programming paradigm landscape. Apple devices use iOS and Objective-C, while Android devices use Android and primarily Java. Windows Mobile 7 and RIM have their own setups, and there are a number of smaller or legacy systems in use as well.

Unlike desktop where some standard of cross-device programming emerged such as Posix, which both fixed some aspects of the API and the language (variants of C), the situation for mobile devices is more diverse.

UrMus [5] proposes numerous solutions to creating mobile music environments, including live patching and interface programming. Additionally it follows a layered architecture. Part of the role of the lowest layer of this design is system abstraction as well as on-device programming and cross-device code migration. For this purpose urMus uses Lua, a light-weight, efficient embeddable scripting language that is especially suitable for on-the-fly and modifiable programming[7].

The purpose of this paper is to present the design considerations for platform independence, cross-device programming and code migration, as well as document the process of implementing interoperability between iOS and Android platforms in the case of urMus.

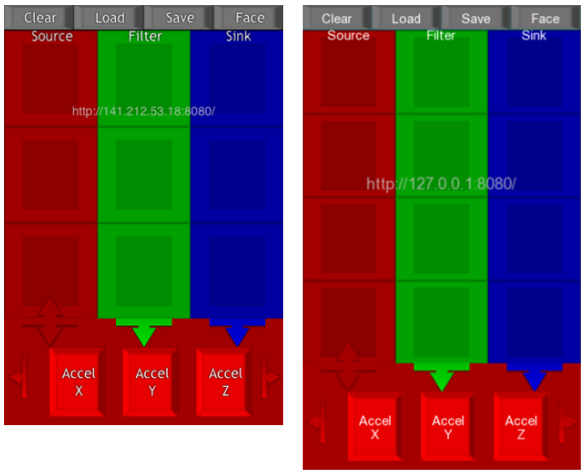

Figure 1. Comparison of default urMus interface on iPhone (left) and Android (right)

2. RELATED WORK

That sound synthesis environments transcend platforms is typical on desktop and laptop computers. Often these environments provide their own programming language and functional abstractions as well to further provide independence from more system level programming as well as provide language concepts and encapsulations more directly suitable for the development needs of musicians. Live coding in particular has been made possible by such environment as SuperCollider [8] and ChucK [11]. Similar environments for mobile smart-devices are only emerging or are experimental, and often only run on a specific mobile platform such as Apple's iOS devices.

There are a number of approaches to cross-platform mobile application development, which resembles numerous cross-platform frameworks developed back in 1990s. Such approaches are typically based on an abstraction layer that lies between web browser and the platforms. Using this layer developers can write applications in languages like Javascript and HTML to run the application on any supported platform. A comprehensive review of the following existing systems can be found in [9, 10]. Rhodes offers a Ruby-based framework that allows web applications to run on a large number of smartphone platforms

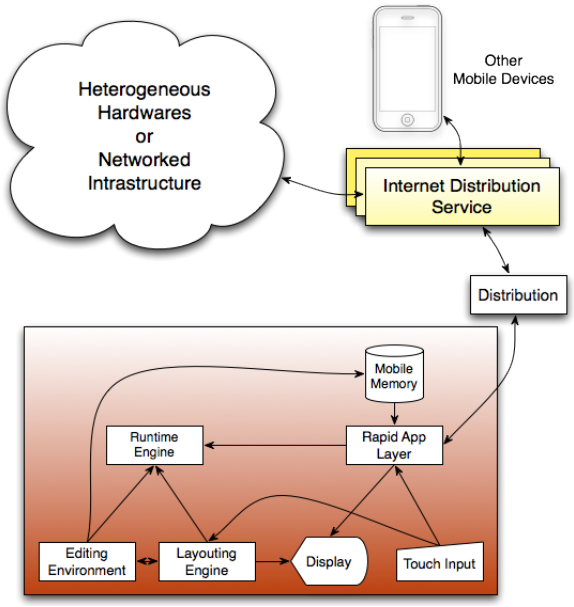

Figure 2. Cross-platform programming and code migration of urMus

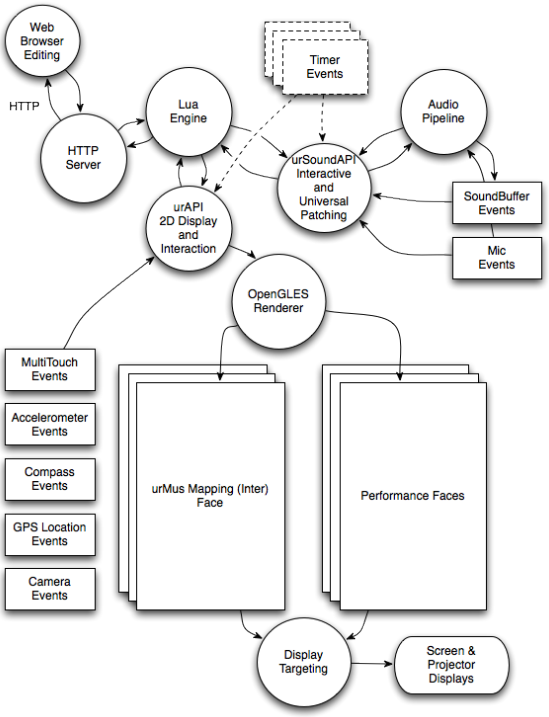

Figure 3. The functional structure of urMus

without any modification. Titanium mobile supports development of native applications through typical web development languages including Javascript, PHP, Python, Ruby and HTML. PhoneGap allows developers to write cross-platform applications in Javascript and HTML with Javascript access to hardware components such as accelerometer, camera and sound. MoSync is a platform-independent software development kit (SDK) in C/C++.

There are approaches to move existing cross-platform to mobile realm. Bedrock is based on J2ME. Qt Mobility is a mobile version of Qt. Adobe Flash and Flash Lite enable Flash player to run on mobile platforms, and Adobe AIR allows developers to run Flash applications as standalone applications.

WidgetPad provides a web-based development environment for HTML5 web applications. jQTouch is an extension to jQuery javascript library for mobile web applications. Processing.js is a javascript version of Processing, a JAVA-based open source programming language for electronic arts and design. Corona from Ansca Mobile is a Lua- and OpenGL ES-based platform that supports iOS and Android, and targeted for rapid cross-platform development.

UrMus is in some sense comparable to many of these existing cross-platform solutions. It too offers its own abstraction, though rather than using HTML or some web based scripting system, it utilizes Lua instead. Lua is known for being an exceptionally fast script language and hence helps to ensure high performance even for time-critical interactive media heavy applications such as mobile musical instruments. UrMus is in many ways different than these platforms. Most importantly it was explicitly developed to support mobile music interaction design in a platform independent manner. But urMus has a range of other considerations embedded in its design as well that are relevant and to be discussed in the next section.

3. CROSS-DEVICE ARCHITECTURE OF URMUS

Numerous considerations went into the design of urMus[5]. A number of the design decisions have direct implications for cross-platform portability and more importantly cross-platform use. urMus implements the ability to access all of its functionality in the fast embeddable script language Lua [7]. Hence rather than having to content with platform centric language such as Objective-C for iOS or Java for Android, now one language is used across all platforms. This is not the primary motivation for introducing a new language. Rather the script language layer has a number of desirable properties. For one it allows to alter the mobile program development cycle. Traditionally programming of mobile devices happens not on the device itself, but rather on some standard desktop computer or laptop. The program is compiled, simulated, tested, then provisioned and uploaded to the mobile device. If problems occur, changes happen offline, are recompiled and so forth. One of the goals of urMus is to support programming that can in principle happen on the device alone [6], as well as provide seemless support of interactive coding and code sharing between devices. Hence this development cycle needs to be altered to make the desktop no longer a required part of the process. The development cycle implemented in urMus has the character depicted in Figure 2.

The main advantage of this architecture is that it sup-

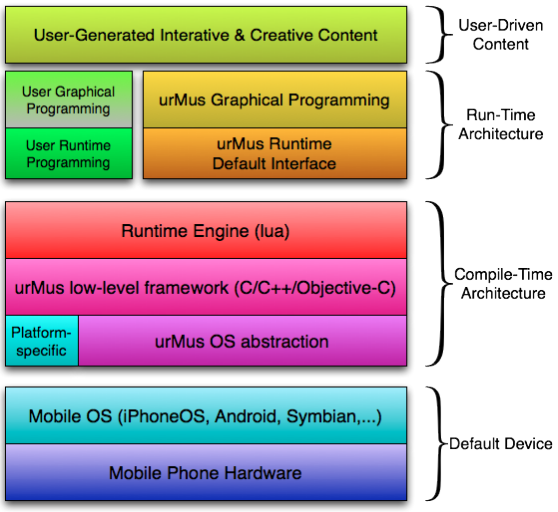

Figure 4. The layered software design of urMus

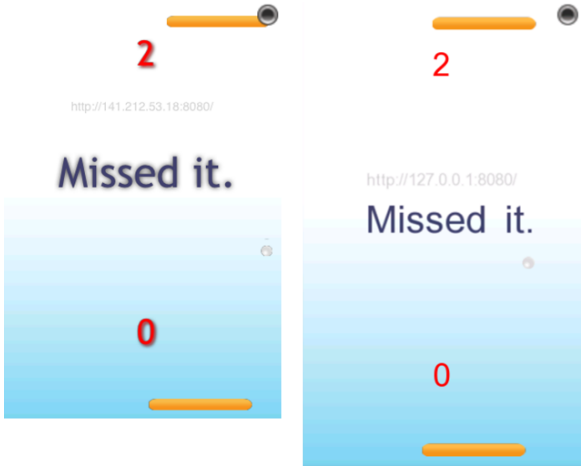

Figure 5. Comparison of the same face running on iPhone and Android

ports true code sharing and code migration between devices and projects. For example, a musical piece can transmit programs or program segments over the network and these can be processed by participating devices, whether mobile or otherwise in a democratic fashion, hiding the specifics of the architecture.

4. DEVELOPMENT

UrMus was originally developed for iOS devices. However, since its design Android has emerged as an important mobile operating system. Hence it became paramount to be able to support these different platforms and offer a shared environment. The resulting development of the Android version of urMus is based on Android SDK 2.2 and Android NDK r4b, and tested with Google Nexus One running Android 2.2 Froyo.

A big portion of the urMus codebase is written in a platform independent way, which made it possible to port to Android without rewriting the entire program. The code base that was already platform-independent included Lua, urMus Lua API, urSound, STK, HTTP Server and good parts of the graphical rendering code using OpenGL ES. Most components shown in Figure 2 are by design cross-platform. Exceptions include event handlers, and access to low level hardware such as the audio pipeline or multi-touch interactions.

4.1. Gaining Platform Independence

In order to make urMus cross-platform, iOS specific parts of urMus were rewritten or replaced with cross-platform code.

Regions of urMus was implemented using Texture2D, which is an Objective-C class provided by Apple that represents a 2-dimensional OpenGL ES surface with texture and/or texts on it. This class is replaced with urTexture, a C++ class that represents OpenGL ES Surface in a platform independent way. urMus also utilized FreeType project [4] and `ofTrueTypeFont` class of openFrameworks [1], rather than Apple's Core Graphics, to read true type fonts and render texts on the OpenGL Surface. Sound and image files were being processed by Apple's Core Audio and Core Image library. This has been replaced with wave file I/O capabilities of STK [3], device dependent low level audio access abstractions, and libpng.

Touch handling code originally included in Objective-C class was separated to platform-independent C/C++ functions and Objective-C wrappers. This helped the Android version to have identical touch behavior by calling those functions with parameters given by Android OS. These corrections on urMus codebase made it more cross-platform and ready to be ported to other mobile operating systems, except for the parts accessing the interface hardware.

4.2. Porting to Android

Android is considerably different from most mobile operating systems. It is designed to natively run an efficient Java virtual machine and offer Java as primary programming language. However additionally Android does allow programmers to use C/C++ code using Android NDK. In order to maintain portability, all parts of the Android version urMus were compiled using Android NDK. User interface code and the basic application skeleton had to be written in Java due to the particulars of the Android architecture. The Java code receives events and signals from the Android operating system and invokes native C++ methodsthat uses existing urMus code.

In addition, due to the lack of full C++ support in Android NDK, the project had to include STLport, a cross-platform implementation of C++ Standard Template Library. This resolved compilation issues of STK, which relies heavily on C++ STL.

Unlike iOS platforms, bundled resourses such as .wav, .png and .ttf files of an Android application resides within

its zipped package and accessible only from Java level, not allowing C/C++ codes to access the resources directly. This problem was resolved by installing the resources to the document directory on the first launch of the application.

The audio layer has also been replaced by Android's audio library, using the `AudioTrack` class.

Figure 1 and 4.1 shows the screenshots of urMus on iPhone and Android. It is noticeable that the default urMus interface automatically adjusts to Android's longer screen ratio and shows four rows rather than three.

5. DISCUSSION

5.1. Teaching with a cross-device architecture

One of the main advantage we are already experiencing with the design of the programming architecture of urMus is that it allows us to teach courses on mobile music making with reduced complications by the hardware requirements. For example iOS development necessitates access to MacOS computers with XCode and the proper SDK installed, whereas Android development is somewhat more flexible. Yet in either case provisioning steps are necessary. All these requirements are removed in urMus. For external programming it is only required to have access to any computer with wireless access and a web-browser. Programming then becomes possible by the editing environment provided by urMus itself accessible through a web browser, which is device independent. Hence it is possible to teach a course that can both utilize iOs and Android devices without the heterogeneity having an impact. The students are free to use whatever laptop or desktop computer they have available to program the device.

5.2. Latency Issues

iOS devices such as the iPhone have a high quality and low latency full-duplex audio pipeline. It is easy to achieve latencies well below 10ms. Sadly at the time of writing Android 2.2 on Google Nexus One has considerable latency in full-duplex operation. This is due to the current limitations on the lower-level sound support of Android. `AudioTrack` is the lowest level class that Android SDK 2.2 provides. Reason for the latency is the required size of the sound buffers of a minimum length is 8192 on Nexus One. Hence, even with 48000Hz sample rate, the minimum latency is about 170 milliseconds, which is clearly noticable delay. We expect that the OpenSL ES support of the latest version of Android NDK [2] will be able to help resolve this problem.

6. CONCLUSIONS

Cross-device use of mobile music environments is critical given the heterogeneity of mobile operating systems and hardware. We have discussed various aspects of cross-platform development of mobile music environment, including the benefits of modifiable code and code migration, as well as practical considerations of porting of urMus for the Android platform.

Future work include explicit support of code sharing and migration on the network, on-device programming and music instrument development to remove the need for external programming hardware completely, and the port of the environment to additional platforms, including Windows Mobile and RIM. With recentely released Android SDK 2.3 and Android NDK r5, Android is expected to have faster lower-level sound support as well as performance enhancements. This includes more efficient sound programming at native level using OpenSL ES API.

urMus, both iOS and Android version, can be found at the official website:

http://urmus.eecs.umich.edu/

7. REFERENCES

[1] "openframeworks, http://www.openframeworks.cc/."

[2] T. Bary, "Gingerbread sdk awesomeness," *Android Developers Blog*, 2011. [Online]. Available: http://android-developers.blogspot.com/2011/01/gingerbread-ndk-awesomeness.html

[3] G. P. Cook, Perry R.; Scavone, "The synthesis toolkit (stk)," *Proceedings of the International Computer Music Conference*, 1999.

[4] W. L. D. Turner, R. Wilhelm, "The freetype project," 1996.

[5] G. Essl, "Urmus-an environment for mobile instrument design and performance," *Proceedings of the International Computer Music Conference*, 2010.

[6] ——, "Mobile phones as programming platforms," *Proceedings of the First International Workshop on Programming Methods for Mobile and Pervasive Systems*, 2010.

[7] R. Ierusalimschy, *Programming in Lua, Second Edition*. Lua.org, 2006.

[8] J. McCartney, "Rethinking the computer music language: Supercollider," *Comput. Music J.*, vol. 26, no. 4, pp. 61–68, 2002.

[9] J. O'Dell, "5 cross-platform mobile development tools you should try," *Mashable*, 2010. [Online]. Available: http://mashable.com/2010/08/11/cross-platform-mobile-development-tools/

[10] V. G. Sarah Allen and L. Lundrigan, *Pro Smartphone Cross-Platform Development*. Springer, 2010.

[11] G. Wang and P. R. Cook, "Chuck: a programming language for on-the-fly, real-time audio synthesis and multimedia," in *ACM Multimedia*, 2004, pp. 812–815.