iSuperColliderKit: A Toolkit for iOS Using an Internal SuperCollider Server as a Sound Engine

Akinori Ito
Tokyo University of Technology
akinori@edu.teu.ac.jp

Kengo Watanabe
Watanabe-DENKI Inc.
kengo@wdkk.co.jp

Genki Kuroda
Tokyo University of Technology
g3115002e8@edu.teu.ac.jp

Ken'ichiro Ito
Tokyo University of Technology
itoken@stf.teu.ac.jp

ABSTRACT

iSuperColliderKit is a toolkit for iOS using an internal SuperCollider Server as a sound engine. Through this research, we have adapted the exiting SuperCollider source code for iOS to the latest environment. Further we attempted to detach the UI from the sound engine so that the native iOS visual objects built by objective-C or Swift, send to the internal SuperCollider server with any user interaction events. As a result, iSuperColliderKit makes it possible to utilize the vast resources of dynamic real-time changing musical elements or algorithmic composition on SuperCollider for iOS programmers.

1. INTRODUCTION

iSuperColliderKit is a development toolkit that adapts for the iOS7 later. It consists of two units, iSCKit and iSCApp. iSCKit generates three static libraries for building an iOS application using SuperCollider as a sound engine. iSCApp is a sample project which shows the usage of this toolkit. It has capability that programmers can develop their UI programming with iOS native API and programming language and sound designing with SuperCollider language simultaneously. In this paper, we present the improvement and testing process of it.

2. BACKGROUND / MOTIVATION

Smartphones and tablets become widely used as a music production environment, not only computer music research but popular one. In the computer music research field, the major development tools, Csound and stk have already been ported to iOS[1][2]. AudioKit[3] is a toolkit for iOS and MacOS based on Csound. Developers can make some synthesizers and effectors and control the parameters from Native iOS API.

One of similar computer music tools is SuperCollider[4]. It consists of two elements, synthesis server and editor client.

The editor client sends the OSC code-fragments to its server. Due to adopting the model, SuperCollider has feature that a programmer can dynamically change some musical elements, phrases, rhythms, scales and so on. The real-time interactivity is effectively utilized mainly in the live-coding field. If the iOS developers would make their application adopting the "sound-server" model, using SuperCollider seems to be a reasonable choice. However, the porting situation is not so good. SonicPi[5] is the one of a musical programming environment that has SuperCollider server internally. However, it is only for Raspberry Pi, Windows and OSX. The similar one is Overtone[6]. But it does not have the server internally. Overtone users have to install and run SuperCollider separately from Overtone itself on Linux or OSX. There is the iOS version of SuperCollider on Sourcefourge[7] but unofficial. It cannot be built for iOS7 and later smoothly. In this research, we attempt to improve the iOS version on GitHub on the assumption the following situation.

Use case:

- Building a native iOS application
- Building visual objects used by native iOS API
- Embedding some SuperCollider code fragment as a text
- Sending code fragments from iOS UI object including the embedded SuperCollider code fragments
- Changing musical elements in real-time

System requirements:

- Building SuperCollider for iOS7 and later
- Building on Xcode5 or later

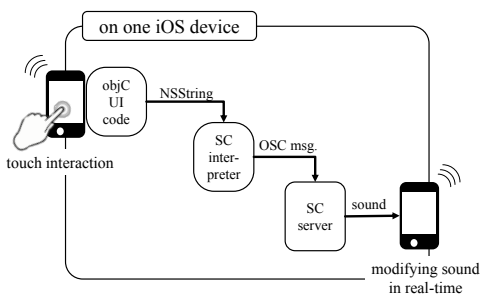

Figure 1. System flow

3. PILOT STUDY

At the start point of this research, we confirmed the GitHub repository and tried to build it. The succeeded environment with modest adjustments is below.

IDE	Xcode 4.6.2
Compiler	LLVM GCC4.2
Hardware	iPod touch 4th generation
iOS version	Version 6.1.5

Table 1. Build environment on GitHub repository version.

It was not the latest environment. We set the target for improvement as below.

IDE	Xcode 6.0
Compiler	Apple LLVM 6.0
Hardware	iPod touch 5th generation
iOS version	Version 7.0

Table 2. Target environment on this research.

3.1 Adjustment for the latest environment

3.1.1 Compiler issue

Most of build errors caused with compiler issue. We made modifications mainly three types of problems. First one is to adopt for Automatic Reference Counting mechanism. Second one is to stop using 32bit version of ARM NEON assembler language. We just changed to the traditional way, array copy but still fast on the new generation hardware. The last one is libsndfile issue. We solved it by utilizing the source code from the latest Csound GitHub repository.

3.1.2 Software architectural issue

The previous version's usage was the same of another OS version. It assumed that users write the SuperCollider code on the editor, select lines, region or file, then push 'exec' button to make sound. Any code send through the interpreter deeply connected the editor UI code internally.

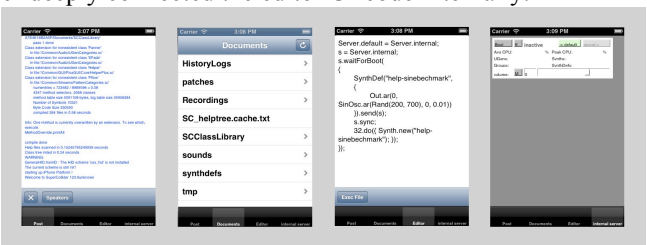

Figure 2. The four main UI panel on previous iOS version

In the aspect of MVC class architecture, the previous version was not enough segregated. Each function was gathered into its SCController class. It caused to be difficult to separate from SuperCollider synthesis engine to UI code of another OS. We have untangled the codes in the SCController class to separate MVC.

3.1.3 UI programming issue

Due to the problems on the version of InterfaceBuilder, the codes around that were replaced into the new ones. Further, we revised the usage of the application included this version as a sound engine. Our goal is to create the application that programmers can be embedded the SuperCollider code fragments in the iOS Native visual object reacting user actions and sending the code to internal SuperCollider server. The code does not go through the editor UI and interpreter system. These improvements were related to 3.1.2.

After doing those improvements, we reconstructed the project as 'wdkk/supercollider_ios' and opened on GitHub[8] before releasing iSuperColliderKit.

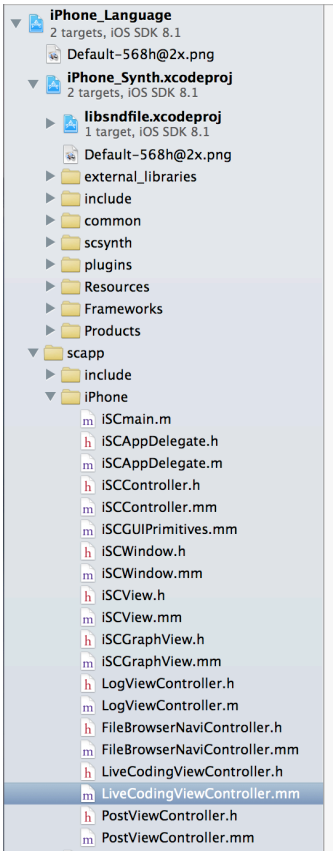

Figure 3. Project tree of wdkk/supercollider_ios

3.2 Sending SC code fragments test from Objective-C

At this phase, it has not been separated the sending some code fragments section from UI building block completely. However, we were able to test to send them from an iOS UIView object. The 'LiveCodingViewController.mm' is the SuperCollider interpreter windows. We tested to send some SuperCollider code fragments from UIView object instantiated in LiveCodingViewController object.

3.2.1 'interpret' method in iSCController

The code fragments are sent through the 'interpret' method in iSCController. In order to implement, the following objective-C code should be added into an interaction control method in LiveCodingViewController.mm.

```
{
 iSCController *scc=(iSCController*)(self.target);
 [scc interpret:@"a = {SinOsc.ar()}.play"];
}
```

The strings "a={SinOsc.ar()}.play" following @ mark is the SuperCollider code fragments reacting any interaction event. The 'self.target' in the first line delegates to another controller class a programmer should implement.

3.2.2 TouchView class and original behavior UI

In this test, we planed to instantiate four square UIView that have each color. In order to be easy to test, we separated the default UI action template to the other class named 'TouchView'. It is not edited from standard UI interaction methods. The following figure is the implemented UI.

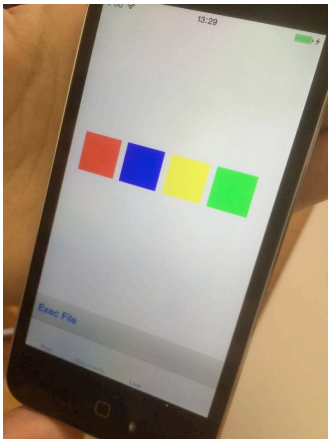

Figure 4. UI for testing to send some code fragments.

Next, four TouchView objects were prepared in 'LiveCodingViewController.h' as below.

```
@property TouchView *tv_red;
@property TouchView *tv_blue;
@property TouchView *tv_yellow;
@property TouchView *tv_green;
```

The example of implementing objective-C code for the red square is below. For blue, yellow, green are rewrote partly.

```
self.tv_red = [[TouchView alloc] initWithFrame:
     CGRectMake(20,200,60,60)]; //continue in 1 line
self.tv_red.backgroundColor=[UIColor redColor];
self.tv_red.delegate_touches=self;
self.tv_red.sel_touches_began=@selector(touchRed:withEvent:);
[self.view addSubview:self.tv_red];
```

The method selected by selector, in this case 'touchRed' is the method for the red square. The method implementation code is below that uses the code in 3.2.1

```
-(void)touchRed:(NSSet*)touches withEvent:(UIEvent *)event
{
   iSCController *scc = (iSCController*)(self.target);
   [scc interpret:@"a = {SinOsc.ar()}.play"];
}
```

Programmers can any kind of interaction and embedded a SuperCollider code fragment, booting and halting server, sending SynthDef, playing and stopping some phrases to each object by implementing each method.

3.3 Real-Time Phrase Modification

For our purpose to use SuperCollider, beyond sound file modifycation, we attempted the real-time modal change application on this system. It is easy to realize on OSX.

At first, we prepared the loop sequence to start on the booting the application. The pitches of loop sequence example were written in SuperCollider code excerpt.

```
~offset = 60;
~seq = [0,2,3,5, 7,9,10,12]
key1 = ~seq[0].wrapAt(~pnt) + ~offset;
```

This example plays a minor chord loop sequence. It plays by using TempoClock class. On another front, we prepared the vector of another scales as below.

```
~phrygian = [0,1,3,5,7,8,10,12]
~lydian = [0,2,4,6,7,9,11,12]
~com_dim = [0,1,3,4,6,7,9,10,12];
```

By preparing the scale vector, programmer can write the scale changing objective-C code in their controller method easily like below.

```
[scc interpret:@"~seq=phrygian;"];
```

The test was in success modifying the modal change in real-time on latest iOS environment by touch interaction onto a standard UIView object.

4. iSuperColliderKit

Following the success of this test, we proceeded in further functionality separation, refactoring and project tree arrangement. iSuperColliderKit is oriented more useful from native API version. By using that, iOS programmers can send the SuperCollider code fragments from Swift language. It has been opened on GitHub[9].

4.1 Content

iSuperColliderKit encloses two Xcode projects in 'project' folder, iSCKit and iSCApp. iSCKit generates three static libraries for building an iOS application using SuperCollider as a sound engine. iSCApp is a sample project which shows the usage of this toolkit in objective-C language.

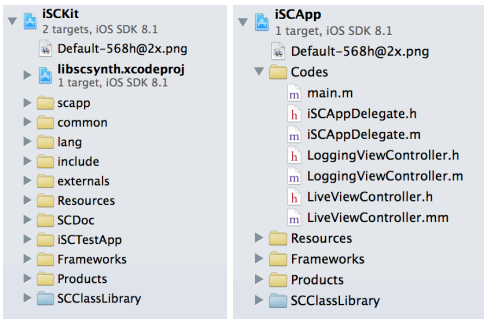

Figure 5. Project trees of iSCKit and iSCApp.

4.2 Usage of this packages

4.2.1 iSCKit

When iSCKit.xcodeproj runs, it generates 'libsndfile', 'libscsynth' and 'libiSCKit' on the projects 'lib' directory. Application programmers can easily develop by using these files including SuperCollider technology. When developers make their own iOS application including SuperCollider as a sound engine, the 3 libraries have to put on the 'lib' folder in their Xcode project folders or set the build settings appropriately according to the instruction in the 'README.md'.

4.2.2 iSCApp

This project is sample app using iSCKit. It is done getting ready to use iSCKit. Launching iSCApp.xcodeproj need to already generated 'libsndfile', 'libscsynth' and 'libiSCKit'. If this process succeeds, iSCKit is available.

4.3 Sending SC code fragments from Swift

The usage on Swift sending SuperCollider code fragment is the same of objective-C way. The codes go through the 'interpret' method on iSCController object. However, it is further simpler. When programmers use this from Swift, the example is in the below.

```
// initialization of the iSCController object
let scc = iSCController.sharedInstance()
scc.setup()

//call the interpret method and send some string
scc.interpretC("s.boot")
```

When developers make their own application from scratch, they can use the 3 libraries 'libsndfile', 'libscsynth' and 'libiSCKit' by the following procedure.

- Import "iSCKit.h"
- Set the Header Path to 'iSCKit' folder in the downloaded *iSCKit* project.
- Set the Library Path to 'lib' folder in the downloaded *iSCKit* project in which have been built in advance.
- Set *'SC_IPHONE'* on 'Debug' and *SC_IPHONE NDEBUG* on 'Release' on 'PreProcessor Macros'
- Add *'-lsndfile -lscsynth -liSCKit'*
- Copy and add *'SCClassLibrary'* in your project folder
- Add the 10 Frameworks below 'CoreMIDI','UIKit','Accelerate','MediaPlayer', 'CFNetwork', 'AVFoundation', 'CoreGraphics', 'CoreFoundation','AudioToolbox', and 'Foundation'.

5. CONCLUSION

We described the toolkit named iSuperColliderKit for iOS for developing native iOS7 or later applications using an internal SuperCollider server as a sound engine. It enable that the native iOS visual objects built by objective-C, even Swift, send to the internal SuperCollider server with any user interaction events.

For future work, we will explore not only the relation between musical elements and characteristic smartphone user-action but the true adaptive music or smartphone improvisation system applying the research achievements of algorithmic composition and music analysis.

6. REFERENCES

[1] Csound for iOS. http://www.csounds.com/shop/csound-for-ios/

[2] N. J. Bryan, J. Herrera, et al., "MoMu : A Mobile Music Toolkit", *Proceedings of the International Conference on New Interfaces for Musical Expression*, Sydney, 2010, pp. 174-177.

[3] AudioKit. http://audiokit.io/

[4] J. McCartney, "SuperCollider, a New Real Time Synthesis Language", *Proceedings of the 1996 International Computer Music Conference*, Hong Kong, 1996, pp. 257-258.

[5] Sonic pi. http://sonic-pi.net/

[6] S. Aaron, A. F. Blackwell, "From sonic Pi to overtone: creative musical experiences with domain-specific and functional languages", *Proceeding FARM '13 Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling & design*, ????, 2013, pp. 35-46.

[7] SuperCollider for iOS Sourceforge git repository git://supercollider.git.sourceforge.net/gitroot/supercollider/supercollider isc

[8] supercollider_ios. https://github.com/wdkk/supercollider_iOS

[9] iSuperColliderKit. https://github.com/wdkk/iSuperColliderKit