iHammond - A Csound based app-studio case

Alessandro Petrolati

www.apesoft.it - info@apesoft.it

Introduction

The developing process of an app at beginning, can be a significant problem because the passage from theory to the practical implementation, sometimes, results as a shot in the dark. In addition to knowledge of the programming language are required a series of additional information: rationally use of the IDE (Interface Developer Environment), know the stages of compilation, 'linking of library' and 'frameworks', configure the compiler properly through the flags such SDK (Studio Developer Kit), the CPU architecture etc… Furthermore, the certificate management, the 'provigioning files', 'entitlements', and also the publishing process on the app store etc...

All this is misleading and might discourage the neophyte developer.

But do not worry! The Apple philosophy has always paid particular attention to the simplicity and functionality and elegance, also the developing field follows this criterion and we have a set of powerful tools as, for instance, the xCode IDE. The xCode environment simplifies and manages for us all involved steps, the xCode templates are already executable applications on various devices (iDevices), even if not perform any particular task.

Think about it, when we begin to write a new app we start already from a very advanced level.

In general, we are led to a linear view of type 'Learn and then Make' but sometimes could be convenient a different approach that is 'Make while Learn'

Of course it is assumed a basic knowledge of the two programming languages

1) Csound for the (Digital Signal Processing)
2) Objective-C for the UI (User Interface)

Although the Objective-C language is not recommended by Apple, since in 2013 was introduced the new 'Swift' who should replace him, in the world of audio is still preferred language as it allows a greater integration of other languages such as 'C ++' and 'C'. In fact, 'C' language is the most common and efficient, yet used for audio as well as for graphics (example CoreAudio and CoreGraphics).

Regarding the programming of the graphical interface and additional functions, we have two options

1) Use the native language
2) Use a 3rd parties 'framework'

In the first case the graphics functions are written in the native language, for example Objective-C for iOS and then the app will run for only iOs, it will not be portable. We will have to completely rewrite the interface code, in Java if we want to run the app in Android.
To overcome this problem, the second case involves the use of a third parties cross-platform graphics library. Some of the most powerful for the audio apps are 'Juse' and 'Qt'. These libraries allow us to write the code once and make it native on the platforms it supports. This approach, however, is not without problems especially when we look for an portability between Desktop and Mobile OS. Any time the app is limited to a single system, for example IOS, we should always use the native language as Objective-C or Swift.

Advantages and disadvantages

Developing the audio DSP with Csound is very very powerful since all the DSP functions are performed by Csound and the graphics is developed 'outside' by Csound.

The main advantages of Csound are

1) High level audio programming
2) Automatic polyphony management, intrinsic to Csound
3) An arsenal of opcodes that cover practically all types of synthesis
4) Fully cross-platform

Some disadvantages

1) Setup non a run-time
2) Hard synchronization between OS and *ksmps / sr*
3) Performance not enough optimized, in some cases

However all of these 'limitations' are solvable through some workarounds and external implementations that will not be covered in this text.

In summary

1) Knowledge of Csound
2) Basic knowledge of OOP (Object Oriented Programming)
3) Minimal knowledge of the native language of the architecture on which we are programming. For instance, Objective-C for iOS or Java for Android.

As an alternative to the third point, the knowledge of the third-party frameworks used. Example C++ for both Juce and Qt.

There are several tutorials to set up from scratch an app for IOS, for using Csound inside an app the code required it is extremely simple and self explanatory, thanks to the classes 'wrapper' provided with the Csound for IOS.

Once we understood the get / set mechanisms with Csound, the main effort is the creation of the graphical interface and of course all the complementary functions necessary or even indispensable in a professional audio application.

As mentioned, thanks to Csound we can skip all the problems related to low-level audio programming, implementing a complete separation between the DSP and the graphical interface that assures us a great advantage in terms of strength.

Getting Started

The idea is to start from an 'Orchestra' (patches) of Csound fully functional and then programming the UI according to DSP features.
This leads to a steeper learning curve since, for UI programming, we will be guided by the requested DSP functions, that are closer to the way we think. This way of working is very similar to what we already know when we use our favorite editor, such as CsoundQt.

As an example we proposes the Csound patch 'DirectHammond v2' by Josep M. Comajuncosas (Barcelona, May / June 2001), which is easily available online. The patch is a Csound port of Istvan Kaldor´s TinyToneWheels for Sync Modular, this is a Hammond emulator with 2 drawbars, scanner, leslie, distortion, delay, echo and much more.
Please note that the Csound 'Orchestra' has been designed for a parallel distribution of Csound called DirectCsound which has been the first release oriented for the real time, thanks to the work of Gabriel Maldonado since the late 90s and now merged with Csound 6.
The same Maldonado has introduced a large number of opcodes, some of them based on the graphics library FL (Fast Light) to create graphical interfaces directly in Csound language. In fact, the patch of Comajuncosas has a control interface written in Csound, since it is thought to function in real time.

The patch has been adapted for iOs by making only a few minor changes to the file 'Orchestra'. It was commented the code for the FL GUI and it was added a tool to read values from the IOS through the software bus's mechanism.

```
instr 99 ;sampling from iOs

;Drawbars low
gkL_16ft_lvl  chnget "drawbar_1_low"
gkL_5_13ft_lvl chnget "drawbar_2_low"
gkL_8ft_lvl   chnget "drawbar_3_low"
gkL_4ft_lvl   chnget "drawbar_4_low"
```

The *chnget* and *chnset* opcodes are used to reads data from the software bus, please note that *invalue* and *outvalue* opcodes will not work with Csound API.

Another issue concerns the Csound's flags that have been modernised to make them compatible with Csound 6 and in particular with the Csound API, inhibiting rtmidi and rtaudio.

```
-o dac -+rtmidi=null -+rtaudio=null -d
```

For the rest, it was not modified the original code in no way and the patch magically plays the Hammond sound in IOS.

Csound for iOs Wrapper[1]

The CsoundObj class is an Objective-C 'wrapper' for the Csound API (Application Programming Interface) that are written in 'C', these files are distributed with the package 'Csound for IOS'.
The basic functions to communicate with Csound are exhibited in this class, but in specific cases it might be necessary to override the wrapper's functions and directly employ the Csound API.
As mentioned, we can write the 'C' code of Csound API in the same Objective-C class.

The CsoundUI class implements some methods in order to connect with Csound with the iOS native graphic objects

```
- (void)addLabel:(UILabel *)uiLabel    forChannelName:(NSString *)channelName;
- (void)addButton:(UIButton *)uiButton forChannelName:(NSString *)channelName;
- (void)addSlider:(UISlider *)uiSlider forChannelName:(NSString *)channelName;
- (void)addSwitch:(UISwitch *)uiSwitch forChannelName:(NSString *)channelName;
- (void)addMomentaryButton:(UIButton *)uiButton forChannelName:(NSString *)channelName;
```

or, directly through the method addBinding from CsoundObj

```
- (void)addBinding:(id<CsoundBinding>)binding;
```

---

[1] (from wiki) In software engineering, the adapter pattern is a software design pattern that allows the interface of an existing class to be used from another interface.[1] It is often used to make existing classes work with others without modifying their source code.
https://en.wikipedia.org/wiki/Adapter_pattern

thus we can link any not native object. For example if we want entirely program a Knob, you will have to adopt the protocol CsoundBinding and implement the functions required by the Protocol.

Ex:

```
@interface MyWidget : NSObject <CsoundBinding>
//…
@end
```

Since online we can get a huge amount of code written by thousands of programmers around to the world, we can use, for example, the inheritance mechanism in order to extend the widget to be compatible with Csound. This approach simplifies and allows to obtain, without effort, a more captivating app.

The interactions with Csound are of the type set / get, the first case is to pass data, for example, from an UI control to an oscillator Csound (a parameter of synthesis) and in the second case it to receive a value, for example, from Csound to an UI object as a label. As we will see the set / get methods are the **updateValuesToCsound** and **updateValuesFromCsound** functions required by CsoundBinding protocol.

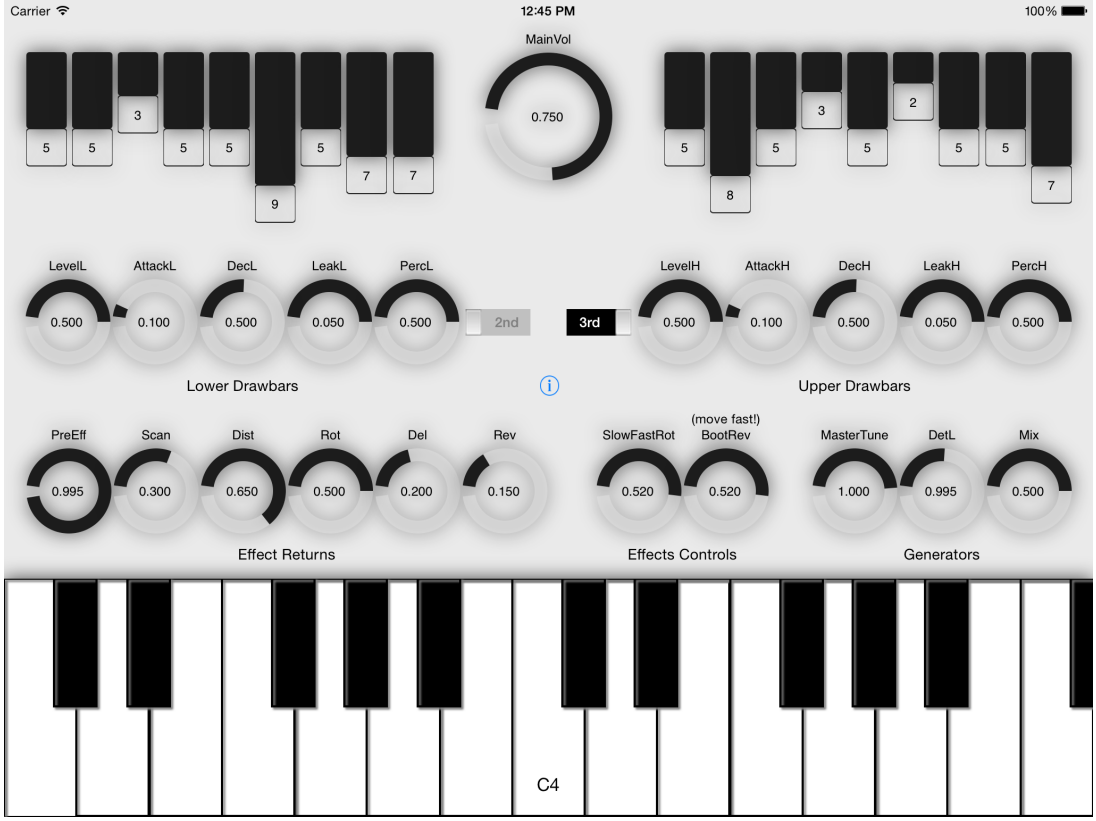
Furthermore through the low-level Csound API we can achieve more complex interactions, how to draw a waveform from a Csound *gen* function or write an array on a *gen* ect …
In the example we employs some not native widgets for the Drawbars and Knobs, all the the code has been obtained from internet and embedded in the project. Consequently the code has been specialised in order to work with Csound.
In some cases we might need to change the 'wrapper' code in the CsoundObj class, in order to resolve some issues

1) Some Audiobus and/or Inter-App Audio limitations
2) Fixed Csound *ksmps* at 64

However these code-level adjustments can be easily feasible.

User Interface Cordially

In the iHammond app-study we have four types of non-native iOS widgets, as you can see from the picture



The four types of controls are

1) Knobs
2) Drawbar
3) Switch
4) Keyboard

According to the fundamentals OOP (Object Oriented Programming)

1) Code reuse
2) Fast learning curve
3) Inheritance
4) Polymorphism
5) etc…

We tried to apply literally the first point for the custom widgets's creation, related to the Knobs and Drawbars. The code of these controls has been written by other programmers and obtained by the network, the classes and DCKnob DCSlider and DCControl (Copyright 2011 Domestic Cat. All rights reserved) are added to the xCode project as source files.

In the References section of this chapter we can find the download link.

The APEKnob and APESlider classes extend functions for DCKnob and DCSlider, making them compatible with Csound. The APESlider class implements some additional graphics functions for the Drawbars.
By discarding this additional implementation, we can focus on the APEKnob class declaration, the angle brackets syntax to indicate that a class adopts a protocol.

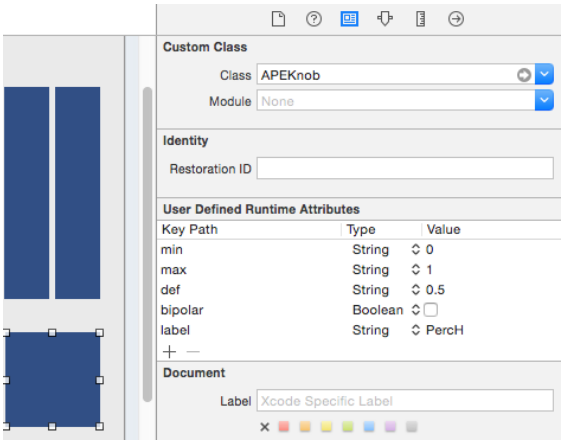
```
@interface APEKnob : UIControl  <CsoundBinding>  {}
```

The CsoundBinding protocol is the bridge between our code and Csound, therefore we will have to implement the protocol's functions locally i.e. in the APEKnob class.

```
- (void)setup:(CsoundObj *)csoundObj
{
        channelPtr = [csoundObj getInputChannelPtr:[label lowercaseString]
                                                channelType:CSOUND_CONTROL_CHANNEL];

    cachedValue = knob.value;
    self.cacheDirty = YES;
}

- (void)updateValueCache:(id)sender
{
        cachedValue = ((APEKnob*)sender).getValue;
        self.cacheDirty = YES;
}

- (void)updateValuesToCsound
{
        if (self.cacheDirty) {
           *channelPtr = cachedValue;
            self.cacheDirty = NO;
        }
}

- (void)updateValuesFromCsound { }

- (void)cleanup { }
```

In other words, the functions implemented will be called by CsoundObj in order to read (or write), the setup function is important since we get the software bus pointer then in updateValueCache, which is called every *ksmps*, we copy the Knob's value in the Csound software bus.

Notice how the NSString * label is declared as an attribute at runtime, directly on the IB (Interface Builder)



This approach is particularly convenient because it makes it possible to configure the widgets directly in IB, for instance the knob ranges and the default value also the control kind (uni / bi-polar).
Please focus on the NSString **label** (PercH) since this name is used as well for the channel-name of the software bus.
We have chosen to use only lowercase letters in order to avoid incompatibility issues with the capital letters, as found in some cases. In the setup function the method *[label lowercaseString]* assures us the correct interpretation of Csound. Now in the Csound orchestra we can read the knob's value through the *chnget* opcode.

```
gkH_perc_lvl chnget "perch"
```

N.B. “perch” must be in lowercase

Finally we need to hook to Csound the knob, for convenience the code was written in the class ViewController and mainly in the toggleOnOff function.

```
//…
[self.csound addBinding:PercH];
//…
```

The percussion (2nd / 3nd) switches are implemented through the UICustomSliderSwitch class (Created by Hardy Macia on 28/10/09. Copyright 2009 Catamount Software. All rights reserved). Also this code is obtained from internet and, in this particular case, we don't need to extend the class since inherits from an UISlider object which is a native type supported by the CsoundUI wrapper.

```
@interface UICustomSliderSwitch : UISlider {}
```

The code in toggleOnOff function, shows us how to add the two switches (UISlider)

```
CsoundUI *csoundUI = [[CsoundUI alloc] initWithCsoundObj:self.csound];

[csoundUI addSlider:orderL forChannelName:@"OrderL"];
[csoundUI addSlider:orderH forChannelName:@"OrderH"];
```

Please note that, starting from the latest releases of Csound for IOS, we need to instantiate the CsoundUI object on which we call the addSlider function.

Regarding the musical keyboard, the CsoundVirtualKeyboard class is already included in the Csound iOS Examples (Copyright 2011 Steven Yi), the ViewController class adopts the CsoundVirtualKeyboardDelegate protocol then we need to implement these functions

```
@interface ViewController : UIViewController <CsoundVirtualKeyboardDelegate> {}

-(void)keyUp:(CsoundVirtualKeyboard*)keybd keyNum:(int)keyNum {
    int midikey = 48 + keyNum;
    [mCsound sendScore:[NSString stringWithFormat:@"i-1.%003d 0 0 1", midikey]];
}

-(void)keyDown:(CsoundVirtualKeyboard*)keybd keyNum:(int)keyNum {
```

```
    int midikey = 48 + keyNum;
    [mCsound sendScore:[NSString stringWithFormat:@"i1.%003d 0 -2 %d 0 1", midikey, midikey]];
}
```

In this case the Hammond's notes are activated and deactivate through a 'score message' sent to the *instr 1*. To enable the polyphony, we use a fractional p1 for every instance, please refer to the Csound's documentation for further clarification.

Future developments

The proposed studio-app is just an example, has been developed in a few hours as programming exercise. However we can found some interesting convenient ideas, since the learning goes hand in hand with the practical development of the project. Some of the future developments could focus on the audio improvements and for the complementary features such as the presets

1) Enabling Audiobus and Inter-App Audio
2) Buffer frame and Csound *ksmps* synchronization
3) Save and Load UI state, presets

As an exercise of Objective-C or Swift programming, we may want to study the DCKnob class, or similar classes, in order to build a new knob with the following features

1) Double tap to reset default value
2) MIDI CC assignation
3) Save and load internal state
4) Editable value from numeric keyboard
5) Improve the widget's look
6) Curve mapping exp/log
7) etc…

References

Download Online Tutorial
http://www.apesoft.it/dev

Csound for iOS by Steven Yi and Victor Lazzarini
http://www.csounds.com

Josep M Comajuncosas
http://www.csounds.com/jmc/Instruments/instruments.htm

Gabriel Maldonado
http://www.csounds.com/maldonado/

Csound for iOS API - A Beginner’s Guide 1.0
http://www-users.york.ac.uk/~adh2/iOS-CsoundABeginnersGuide.pdf

Cocoa Controls
https://www.cocoacontrols.com/controls/dcknob

apeSoft
http://www.apesoft.it

How To Make a Custom Control - Colin Eberhardt

http://www.raywenderlich.com/36288/how-to-make-a-custom-control

JUCE is a powerful, open-source C++ audio engine
http://www.juce.com

Qt - cross-platform development
http://www.qt.io